Decentralized Dynamic Load Balancing for Multi Cluster Grid Environment

Malarvizhi Nandagopal and V. Rhymend Uthariaraj

Anna University Chennai, Tamilnadu, Inida
nmv_94@yahoo.com, rhymend@annauniv.edu

Abstract. Load balancing is essential for efficient utilization of resources and enhancing the performance of computational grid. Job migration is an effective way to dynamically balance the load among multiple clusters in the grid environment. Due to limited capacity of single cluster, it is necessary to share the underutilized resources of other clusters. Each cluster saves the static and dynamic information about its neighbors including transfer delay and load. This paper addresses the issues in multi cluster load balancing based on job migration across separate clusters. A decentralized grid model, as a collection of clusters for computational grid environment is proposed. A Sender Initiated Decentralized Dynamic Load Balancing (SI-DDLB) algorithm is introduced. The algorithm estimates system parameters such as resource processing rate and load on each resource. The algorithm balances the load by migrating jobs to the least loaded neighboring resource by taking into account of transfer delay. The algorithm also considers the availability of selected resource before dispatching job for execution since the probability of failure is more in the dynamic grid environment. The main goal of the proposed algorithm is to reduce the response time of the jobs. The proposed algorithm has been verified through the GridSim simulation toolkit. Simulation results show that the proposed algorithm is feasible and improves the system performance considerably.

Keywords: Grid Computing, Load Balancing, Clusters, Scheduler, Transfer Delay.

1 Introduction

The grid is emerging as a wide-scale infrastructure that promises to support resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organization [1]. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. With rapid progress in computing, communication and storage technologies, grid computing has gained extensive interest in academia, industry and military. Grid computing provides the user with access to locally unavailable resource types. On the other hand, there is the expectation that a large number of resources are available.

A computational grid aggregates a plenty of computation resources. Computation resources may be low-end systems such as PCs and workstations, or high-end systems such as clusters, massively parallel processors (MPPs) and symmetric multiprocessors

N. Meghanathan et al. (Eds.): CCSIT 2011, Part III, CCIS 133, pp. 149–160, 2011.

(SMPs). In computational grid, user jobs can be executed on either local or remote computer systems. The computational grid [2] provides the opportunity to share a large number of resources among different organizations. As the number of resources increases, the probability of failure becomes higher than in a traditional parallel computing. The failure of resources affects the job execution fatally. With the multitude of heterogeneous resources, a proper scheduling and efficient load balancing across the grid is required for improving the performance of the system.

Grids have a lot of specific characteristics [3] such as heterogeneity, autonomy and dynamicity which are the obstacles for applications to harness conventional load balancing algorithms directly. One important advantage of grid computing is the provision of resources to the users that are locally unavailable. Users of the grid system submit jobs at random times. In such a system, some computers are heavily loaded while others have available processing capacity. The goal of load balancing is to transfer the load from heavily loaded computers to idle computers, hence balance the load to the computers and increase the overall system performance.

In general, any load-balancing algorithm consists of two basic policies—a transfer policy and a location policy. The transfer policy determines whether a job is processed locally or remotely. By using workload information, it determines when a resource becomes eligible to act as a sender (transfer a job to another resource) or as a receiver (retrieve a job from another resource). The location policy determines the resource to which a job, selected for possible remote execution, should be sent. In other words, it locates complementary nodes to/from which a node can send/receive workload to improve the overall system performance. Location-based policies can be broadly classified as sender initiated, receiver initiated, or symmetrically initiated [4], [5]. Sender initiated algorithms let the heavily loaded sites take the initiative to request the lightly loaded sites to receive the jobs; while receiver initiated algorithms let the lightly loaded sites invite heavily loaded sites to send their jobs. Symmetrically-initiated algorithms combine the advantages of these two by requiring both senders and receivers to look for appropriate sites.

Further, while balancing the load, certain types of information such as the number of jobs waiting in queue, job arrival rate, CPU processing rate, memory availability are exchanged among the resources for improving the overall performance. Based on the information that can be used, load-balancing algorithms are classified as static, dynamic, or adaptive [5], [6], [7]. According to another classification, based on the degree of centralization, load-scheduling algorithms could be classified as centralized or decentralized [5], [7]. In a centralized system, a single resource acts as a central controller to perform load scheduling. Many authors argue that this approach is not scalable, because when the system size increases, the central controller may become a system bottleneck and the single point of failure. Such algorithms are bound to be less reliable than decentralized algorithms, where load scheduling is done by many, if not all, resources in the system. However, decentralized algorithms have the problem of communication overheads incurred by frequent information exchange between resources.

The rest of the paper is organized as follows:

Section 2 presents related work. Section 3 presents the decentralized grid system model. Section 4 describes in detail the design of the proposed SI-DDLB algorithm. Section 5 discusses simulation environment. Section 6 elaborates experimental results and discussion. Finally, this paper is concluded in Section 7.

2 Related Work

Numerous researchers have proposed scheduling and load balancing algorithms for grid computing environment [8], [9], [10].

The authors in [11] provided a dynamic solution for distributed load-balancing, with the load defined as the number of jobs currently running. However, they ignored the effects of network latency. An agent based system is used in [12], where work is always moved from the busiest nodes to the least busy nodes. However, they assumed that the nodes in the system were homogeneous. In [13], authors analyzed and compared the effectiveness of dynamic load balancing and job replication by means of trace-driven simulations. Agent-based approaches have been tried to provide load balancing in cluster of machines [14].

The authors in [15] proposed a new decentralized dynamic job dispersion algorithm that is capable of dynamically adapting to changing operating parameters. This distributed load-balancing algorithm is dynamic, decentralized, and it handles systems that are heterogeneous in terms of node speed, architecture and networking speed. The algorithm allows individual nodes to leave and join the network at any time and have jobs assigned to them as they become available. Each node saves information about its neighbors including the network bandwidth available between the local resource and its neighbor; the current CPU utilization; and the current I/O utilization. This status information is exchanged periodically. Knowing the status information, each node can choose when to send jobs to its neighbors. This algorithm is fault tolerant and takes job size into account. It is scalable but at the cost of increased communication overhead. The limitation in this algorithm is large buffer space is required on each node to store status information of all nodes. Also, the algorithm should only handle a grid of computers on the same local area network.

The main objective of this study is to propose a SI-DDLB algorithm for computational grid environment that can cater the following unique characteristics:

- Heterogeneous grid clusters
  There may be a difference in the hardware architecture, operating systems, computing power and resource capacity among clusters. In this study, heterogeneity only refers to the processing power of cluster.
- Effects from considerable communication delay
  The communication overhead involved in capturing load information of clusters before making a job dispatching decision can be a major issue negating the advantages of job migration. In this work, the communication overhead is reduced by means of mutual information feedback.

3 Decentralized Grid Model

A decentralized grid system model, comprising of clusters (resources) C is proposed. The set C contains n clusters c_1, c_2,.... c_n. The components in each cluster are shown in Fig. 1. Each of these components has its own independent functionalities that help in

grid management and job scheduling and thus serve the purpose of grid. The computational nodes at each cluster are typically interconnected by a high-speed local network and protected by firewalls from the outside world.

A decentralized job scheduling and load balancing approach is used since the jobs generated by users are submitted to the scheduler where the job originates. The scheduler determines whether to send the jobs to the nodes in local cluster or to a neighboring cluster. The clusters in the grid system may have different processing power. The processing power of a cluster c_i is measured by the average CPU speed across all computing nodes within c_i, denoted by $APPW_i$. For $i \neq j$, APPWi may be different from APPWj. GIS is responsible for collecting and maintaining the details of memory, CPU utilization and load value along with registration information of the resources. The remote manager in each cluster saves information about its neighbors including the transfer delay available between the local cluster and its neighbor and the current CPU utilization. Knowing this, each cluster can choose the neighbor, to which the scheduler has to send job.

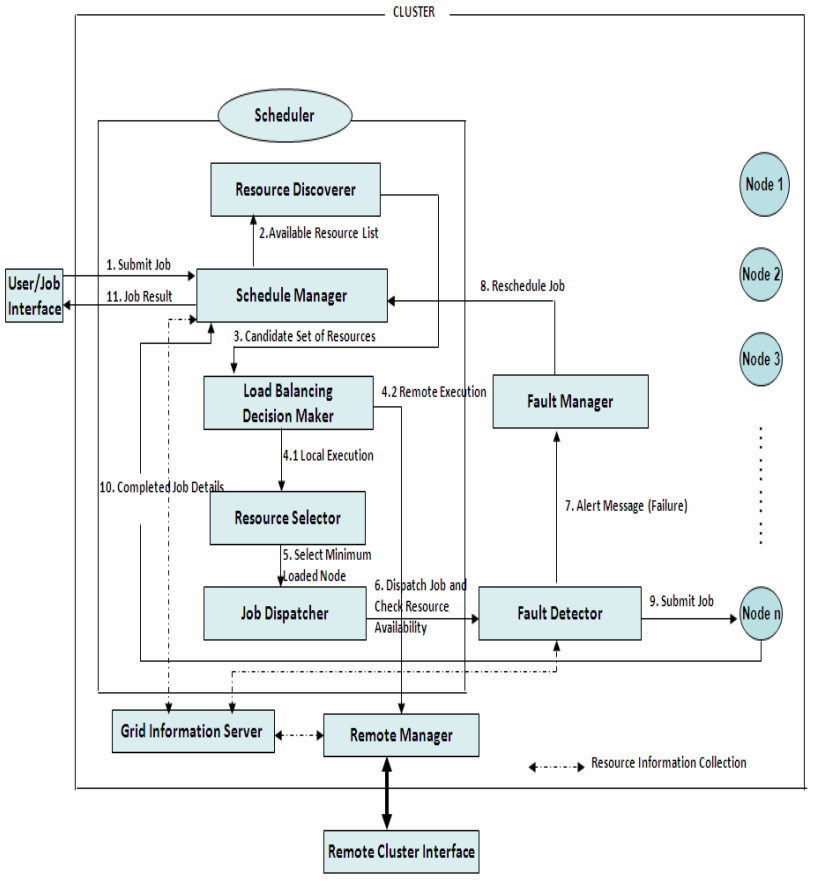

Fig. 1. Components and their interactions in each cluster

When the schedule manager receives job from a user, it gets information about available resources from GIS. It then passes the available resource list to the resource discoverer. Resource discoverer discovers the candidate set of resources based on job requirements and resource characteristics and send the list of candidate resources to the load balancing decision maker. Decision maker make a decision that whether the job is to be executed in the local or remote cluster and transfers the job accordingly. Resource selector selects the computational node with minimum load. A job dispatcher dispatches the jobs one by one to the fault detector.

The availability of selected resource is important for job execution that determines the computing performance. A fault detector is responsible for monitoring the state of resources and detecting an occurrence of resource failure before dispatching a job to the resource. If the resource failure or system performance degradation occurs, the fault detector sends an alert message to a fault manager. If the fault manager receives an alert message, it requests the schedule manager to allocate new resource and restarts execution using a checkpoint.

The clusters in C is fully interconnected, meaning that there exists at least one communication path between any two clusters in C. For any cluster $c_i \in C$, there are jobs arriving at c_i. The jobs are assumed to be computationally intensive, mutually independent, and can be executed at any cluster. At each cluster, there exists a global job waiting queue (GJWQ), which holds those jobs waiting to be assigned for execution. GJWQ (c_i) denotes the global job-waiting queue of the cluster c_i. The jobs in the GJWQ are processed in "First-Come-First-Serve" order. It is assumed that each cluster has an infinite capacity buffer to store jobs waiting for execution. This assumption eliminates the possibility of dropping a job due to unavailability of buffer space.

There is a small non-zero probability that a job can shuttle between clusters. This can be prevented in various ways. The approach used in this work is, the entire simulation makes the job join not at the end of the queue, but at the position where it should have been if the job had arrived at that queue. This means that, keep track of the time at which the job left the last cluster. This can considerably reduce the probability of the job being transferred once again and can guarantee minimizing the response time of that job. The other approach is, set the migration limit as 1 since several researchers assume a migration limit of one, as job migration is often difficult in practice and there are no significant benefits of higher migration limits [16], [17].

4 The SI-DDLB Algorithm

The SI-DDLB algorithm mainly consists of three procedures: Neighbor Selection, Inter Cluster Information Exchange and Instantaneous Job Migration.

4.1 Neighbor Selection

The SI-DDLB algorithm assumes that the communication delay between pairs of clusters can be estimated. The scheduler in each cluster gets the details of neighboring clusters from GIS and then uses this information to select a neighbor cluster for processing new arriving job. Neighbors for each cluster are formed in terms of transfer delay (td). For a grid cluster c_i, it measures the relative distances (e.g. transfer delay)

to a set of neighboring clusters. For a cluster c_i, a cluster c_j is considered as its neighboring cluster as long as the transfer delay between the cluster c_i and c_j is within ε times of the transfer delay between the cluster c_i and the nearest cluster. For each cluster, the chosen neighboring clusters are sorted by load in ascending order. After this process, the first-ranked cluster is chosen as the destination cluster for job migration. This is described as follows:

$$\varepsilon = \frac{td_{ij}}{td_{nearest}}$$

Where, td_{ij} denotes the transfer delay from cluster c_i to cluster c_j. $td_{nearest}$ denotes the transfer delay from the nearest cluster of cluster c_i to itself. It is found that $\varepsilon = 1.5$ yields very good results and this value is used throughout the simulation. The procedure for neighbor selection is described in Algorithm 1.

Algorithm 1: Neighbor Selection

For each cluster c_i
  Find all clusters c_j (i#j) whose $APPW_j > APPW_i$
  Denote this set as Q_i
  Sort Q_i in ascending order based on td
  Choose the first ranked cluster as $c_{nearest}$ and note down the
    transfer delay between c_i and $c_{nearest}$ as $td_{nearest}$
  Find all clusters $c_j \in Q_i$ (i#j) where $td_{ij}=\varepsilon * td_{nearest}$
  Denote this set as $Nbor_i$
  Sort $Nbor_i$ in ascending order based on load
  Choose the first ranked cluster c_j as destination for job migration

4.2 Inter Cluster Information Exchange

An important issue in designing a dynamic load balancing algorithm is to identify the load index that measures the current load of a cluster. A good load index must be easily obtained and calculated with minimum overhead. The authors in [18] report that the simple CPU queue length is the most effective load index. Each cluster c_i maintains the state information of other clusters by using a Cluster State Object CSO_i. CSO helps a cluster to estimate the load of other clusters at any time without message transfer. The state object CSO_i of a cluster $c_i \in C$ is an n-dimensional ArrayList object maintained by c_i. Each item $CSO_i[j]$ is a state object and has a property list (LD, LT):

$CSO_i[j].LD$ denotes the load information of cluster c_j.
$CSO_i[j].LT$ denotes the cluster c_j's local time when the load status information is reported.

Each cluster collects and maintains the state information of only the neighboring clusters. For any cluster $c_i \in C$, c_i maintains its state information in its state object element $CSO_i[i]$. $CSO_i[j].LD$ and $CSO_i[j].LT$ are maintained through message exchanges with the neighboring clusters.

Thus, in order to reduce/minimize the overhead of cluster state information collection, state information exchange is done by mutual information feedback. Algorithm 2

outlines the procedure when c_i transfers a job j_x to a neighbor cluster c_j for processing. Cluster c_i appends the load information of itself and ω_p (a small positive integer) random neighbors to the job transfer request sent to c_j by piggybacking. c_j then updates the corresponding load information in its state object by comparing the timestamps, if the clusters contained in the transfer request belong to its neighbors. Similarly, c_j inserts the current load information of itself and ω_p random clusters from its $Nbor_j$ in the job acknowledge reply or job completion reply to c_i, so c_i can update its state objects.

Algorithm 2: Inter Cluster Information Exchange

Steps processed in c_i:

1. $Y \leftarrow c_i + \{\omega_p$ random clusters from $Nbor_i - c_j\}$
   /* c_i select neighbors for state information exchange */
2. $\forall c_y \in Y$, c_i appends ($CSO_i[y].LD$, $CSO_i[y].LT$) to the Job Transfer Request JTR
3. c_i sends message JTR to c_j

Steps processed in c_j:

Upon receiving JTR:

1. $\forall c_y \in Y$: If ($CSO_i[y].LT > CSO_j[y].LT$) AND ($c_y \in Nbor_j$) then
   $CSO_j[y] \leftarrow CSO_i[y]$
   /* c_j updates the state object using c_i's information */
   Endif
2. $Z \leftarrow c_j + \{\omega_p$ random clusters from $Nbor_j - c_i\}$
3. $\forall c_z \in Z$, c_j appends ($CSO_j[z].LD$, $CSO_j[z].LT$) to the Job Ack. Reply JAR
4. c_j sends message JAR to c_i

Upon completion of job j_x:

1. $Z \leftarrow c_j + \{\omega_p$ random clusters from $Nbor_j - c_i\}$
2. $\forall c_z \in Z$, c_j appends ($CSO_j[z].LD$, $CSO_j[z].LT$) to the Job Completion Reply JCR
3. c_j sends message JCR to c_i

Steps processed in c_i:

Upon receiving JAR or JCR:

$\forall c_z \in Z$: If ($CSO_j[z].LT > CSO_i[z].LT$) AND ($c_z \in Nbor_i$) then
$CSO_i[z] \leftarrow CSO_j[z]$
/* c_i updates the state object using c_j's information */
Endif

4.3 Instantaneous Job Migration

When a new job arrives at cluster c_i, the load balancing algorithm decides whether it is to be sent to the global job waiting queue of cluster c_i or to any one of the neighboring clusters $Nbor_i$. Algorithm 3 describes the procedure for instantaneous job migration in cluster c_i. If there are two neighboring clusters with the same minimum load, the cluster that gives minimum response time is chosen.

Algorithm 3: Instantaneous Job Migration

$\forall j_x \in J$ *in* $c_i \in$ C:
Let $LD_{min} \leftarrow$ Min $\{CSO_i[k].LD \mid c_k \in c_i + Nbor_i\}$
/* the minimum load among cluster c_i and its neighbors $Nbor_i$ */
If ($CSO_i[i].LD - LD_{min} < \theta$) then
/* θ is a positive real constant close to zero */
GJWQ (c_i) $\leftarrow$ enqueue(j_x)
/* put the job j_x in the global job waiting queue of cluster c_i */
Else
Transfer the job j_x to the neighbor cluster c_j having LD_{min}
Update $CSO_i[j].LD$
Endif

5 Simulation Environment

The simulation is based on the excellent grid simulation toolkit GridSim ToolKit 4.0 [19] which allows modeling and simulation of entities in grid computing systems-users, applications and resources. In GridSim simulation, the user creates the experiment specifying the job (gridlet), QoS requirements (including deadline and budget) and optimization strategy.

In GridSim based simulations, the interaction between different GridSim entities takes place through events. In GridSim, each gridlet is defined in terms of the size (in MI) of gridlet, the file size (in byte) of gridlet before execution, the file size (in byte) of the gridlet after execution. The experiments are performed on a PC (Core 2 processor, 3.20GHZ, 1GB RAM). Table 1 shows the values of the parameters used in the simulation.

Table 1. Simulation Parameters

Simulation Parameters	Value
Number of Clusters	10
Number of nodes in each cluster	2
Processing power of each cluster	500-5000
Job Length(Gridlet size)	7000-10,000
Input and Output file size	500 – 700
Total Number of Queue/Cluster	1
Number of users	8
Number of jobs	100-800
E	1.5
Θ	0.9
Number of clusters in Nbor	Varied based on transfer delay
ω_p	20% of Nbor

6 Simulation Results and Discussion

In this section, the performance of the proposed SI-DDLB algorithm is evaluated using the simulation setup described above by varying the number of jobs for different scenarios. A set of experiments are conducted and the performance of proposed algorithm is compared with the Non Migration (NM) or local execution algorithm. In NM algorithm, jobs originate at any cluster in the grid and are processed at the originating cluster itself.

6.1 Performance Improvement in Response Time (SI-DDLB vs NM)

The performance of SI-DDLB is compared in terms of response time by varying the number of jobs. The system load is varied by varying the number of jobs submitted. The higher the load, the higher is total response time of both algorithms as depicted in Fig. 2. When comparing the results of SI-DDLB and NM algorithm, it is observed that the response time of the system that results from applying SI-DDLB is lower than the response time of the NM algorithm under all loads. SI-DDLB has an average improvement factor of 39.43 percent than NM algorithm when 800 jobs get completed. The reason is, SI-DDLB algorithm exhibits more load balancing when system workload is high. NM simply schedules the jobs to the originating cluster without considering whether the cluster is underloaded or overloaded. On the other hand, SI-DDLB algorithm balance the load by migrating jobs to the least loaded neighbor resource by taking into account of transfer delay thus resulting in overall performance improvement.

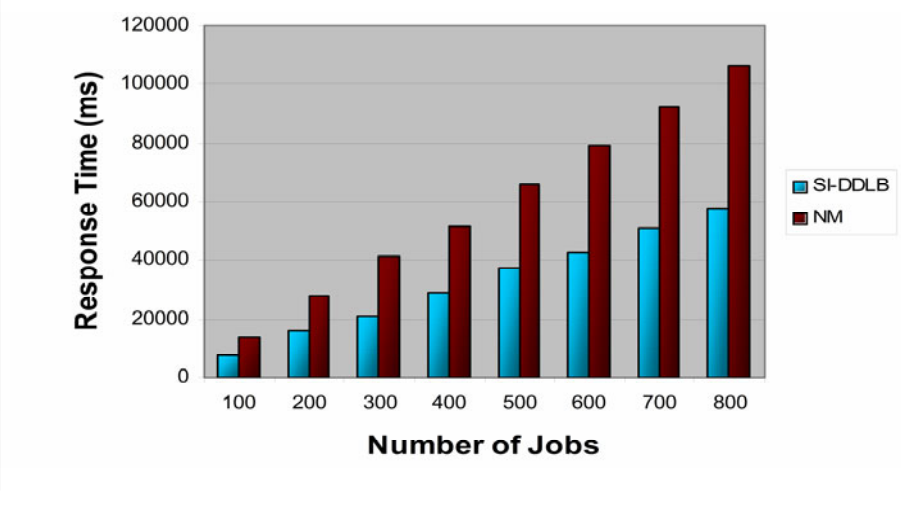

Fig. 2. Response time comparison for varying number of jobs

6.2 Performance Improvement in Waiting Time (SI-DDLB vs NM)

The performance of SI-DDLB is compared in terms of waiting time by varying the number of jobs. The waiting time against the number of jobs is plotted in Fig. 3. Both algorithms take almost the same amount of time in waiting to start execution when the number of jobs is less. It is observed that there is a considerable reduction of waiting time in SI-DDLB than NM when the number of jobs is more. SI-DDLB has an average improvement factor of 10.52 percent than NM algorithm when 800 jobs get completed.

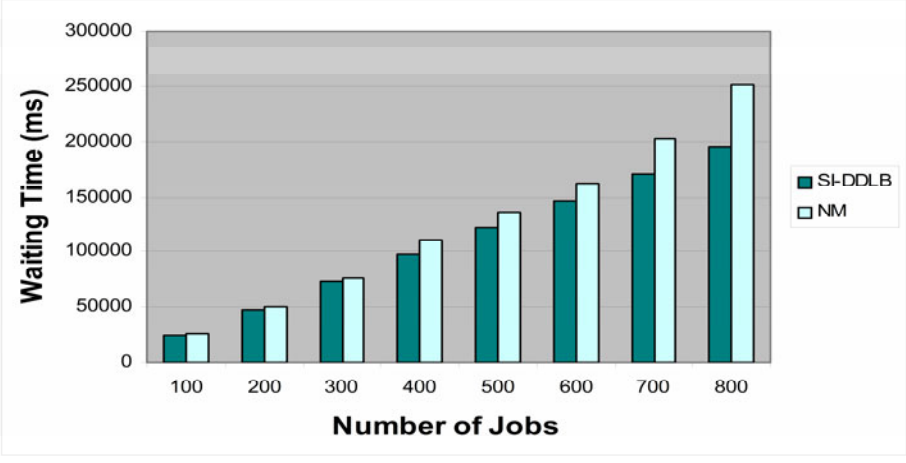

Fig. 3. Waiting time comparison for varying number of jobs

6.3 Performance Improvement in Waiting Time (Load Based NN vs Distance Based NN)

The neighbor selection algorithm plays a major role in SI-DDLB to optimize the performance of the system. The performance of the proposed load based Nearest Neighbor (NN) selection algorithm is compared with the distance based NN algorithm in terms of waiting time. As depicted in Fig. 4, when the number of jobs is increased, the waiting time is also increased in both algorithms. It is concluded from Fig. 4 that the distance based NN algorithm behaves poorly when compared with the load based NN algorithm. Load based NN algorithm has an average improvement factor of 20.81 percent over distance based NN algorithm. In the distance based NN algorithm, a job is migrated to a nearest cluster which is identified based on distance without considering the load of that cluster. Therefore the length of the waiting queue is increased and the incoming job needs to wait for more time in the queue for execution. This increases the response time of the jobs. In the load based NN algorithm, a nearest neighbor is selected based on transfer delay and load. The job is migrated to the minimum loaded nearest neighbor. Thus the waiting time of the job is reduced considerably.

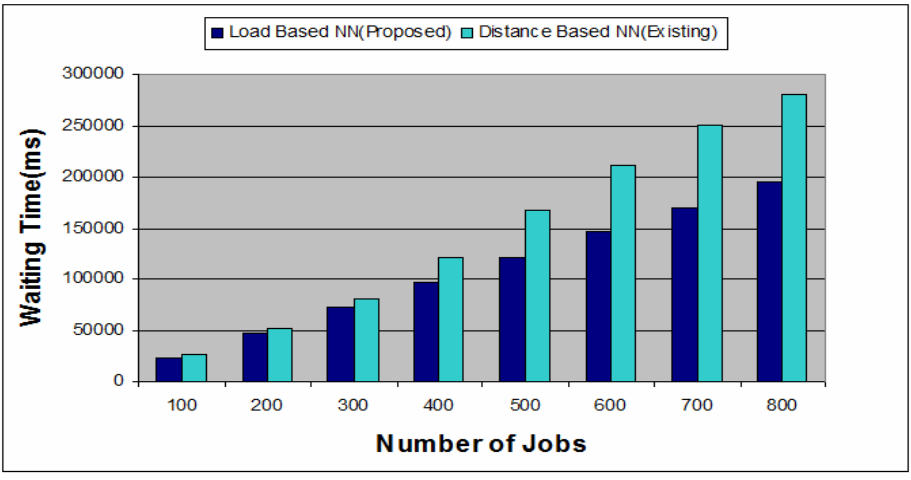

Fig. 4. Effect of Neighbor Selection

7 Conclusion and Future Work

In this study, architecture for sender initiated dynamic and decentralized load balancing algorithm is presented. The proposed algorithm schedules jobs and balances the load across the clusters in the grid environment. The grid is considered as a collection of clusters which differs in terms of processing power and transfer delay. The objective of the algorithm is to minimize the response time of the job that arrive at a grid system for processing. The algorithm also considers overheads of job migration due to the large communication latency between grid clusters. To reduce the communication overhead, the algorithm uses mutual information feedback for inter cluster information exchange. A fault detection and management is also provided in the cluster so that submitted job is executed reliably and efficiently. Various metrics are used to discuss the results obtained including response time, waiting time and resource utilization under varying load. The proposed algorithm is compared with the non migration algorithm with respect to the above defined metrics. From the simulation results it is observed that non migration algorithm results in low efficiency and remarkably takes more time to complete the jobs than the proposed one. In the proposed algorithm load can be shared across different clusters and thus the job response time is greatly reduced.

In the future, it is planned to investigate more complex models and to study additional factors that may affect the performance of the algorithm. Also, it is planned to explore the potential of these load balancing strategies by embedding them into real world grid computing environments.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing applications 15(3), 200–222 (2001)

2. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure (2004)
3. Baker, M., Buyya, R., Laforenza, D.: Grids and grid technologies for wide-area distributed computing. International Journal of Software:Practice and Experience (SPE) 32(15) (2002)
4. Feng, Y., Li, D., Wu, H., Zhang, Y.: A Dynamic Load Balancing Algorithm Based on Distributed Database System. In: Proc. Fourth Int'l Conf. High-Performance Computing in the Asia-Pacific Region, pp. 949–952 (2000)
5. Shivaratri, N., Krueger, P., Singhal, M.: Load Distributing for Locally Distributed Systems. Computer 25(12), 33–44 (1992)
6. Watts, J., Taylor, S.: A Practical Approach to Dynamic Load Balancing. IEEE Trans. Parallel and Distributed Systems 9(3), 235–248 (1998)
7. Zaki, M.J., Parthasarathy, W.L.S.: Customized Dynamic Load Balancing for a Network of Workstations. J. Parallel and Distributed Computing 43(2), 156–162 (1997)
8. Shah, R., Veeravalli, B., Misra, M.: Estimation based load balancing algorithm for data-intensive heterogeneous grid environments. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2006. LNCS, vol. 4297, pp. 72–83. Springer, Heidelberg (2006)
9. Murata, Y., Takizawa, H., Inaba, T., Kobayashi, H.: A Distributed and Cooperative Load Balancing Mechanism for Large-Scale P2P Systems. In: Proc. Int'l Symp. Applications and Internet (SAINT 2006) Workshops, pp. 126–129 (2006)
10. Zeng, Z., Veeravalli, B.: Design and Analysis of a Non-Preemptive Decentralized Load Balancing Algorithm for Multi-Class Jobs in Distributed Networks. Computer Comm. 27, 679–693 (2004)
11. Lüling, R., Monien, B.: A Dynamic Distributed Load Balancing Algorithm with Provable Good Performance. In: Proc. of the 5th ACM Symposium on Parallel Algorithms and Architectures (SPAA 1993), pp. 164–173 (1993)
12. Liu, J., Jin, X., Wang, Y.: Agent-Based Load Balancing on Homogeneous Minigrids: Macroscopic Modeling and Characterization. IEEE Transactions on Parallel and Distributed Systems 16(7), 586–598 (2005)
13. Dobber, M., Mei, R., Koole, G.: Dynamic Load Balancing and Job Replication in a Global-Scale Grid Enviornment: A Comparison. IEEE Transaction on Parallel and Distributed Systems 20(2), 207–218 (2009)
14. Cao, J., Spooner, D.P., Jarvi, S.A., Nudd, G.R.: Grid Load Balancing using Intelligent Agents. Future Generation Computer Systems 21(1), 135–149 (2005)
15. Acker, D., Kulkarni, S.: A Dynamic Load Dispersion Algorithm for Load-Balancing in a Heterogeneous Grid System. In: Sarnoff Symposium IEEE, pp. 1–5 (2007)
16. Lu, K., Subrata, R., Zomaya, A.Y.: On the performance driven load distribution for heterogeneous computational grids. Journal of Computer and System Sciences 73(8), 1191–1206 (2007)
17. Shah, R., Veeravalli, B., Misra, M.: On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments. IEEE Transactions on Parallel and Distributed Systems 18(12), 1675–1686 (2007)
18. Kunz, T.: The influence of different workload descriptions on a heuristic load balancing scheme. IEEE Transactions on Software Engineering 1991 17(7), 725–730 (1991)
19. Buyya, R., Murshed, M.: GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency Computation Practice and Experience (CCPE) 14(13-15), 1175–1220 (2002)