Two-Level Dynamic Load Balancing Algorithm Using Load Thresholds and Pairwise Immigration

Hojiev Sardor Qurbonboyevich
Department of IT Convergence Engineering
Kumoh National Institute of Technology,
Daehak-ro 61, Gumi, Gyeongbuk, South Korea
sardor_hq88@mail.ru

Tae-Young Choe
Department of Computer Engineering
Kumoh National Institute of Technology,
Daehak-ro 61, Gumi, Gyeongbuk, South Korea
choety@kumoh.ac.kr

***Abstract*—This paper proposes a two-level dynamic load balancing scheme for grid and distributed systems. We focus on reducing average task response time. In order to achieve the goals, efficient dynamic load balancing is required. What make difficult dynamic load balancing are decisions such that how much loads are migrated, and to which nodes the loads move. We simplified task immigration through pairwise immigration based on two load thresholds scheme. We compare the proposed scheme with HDLA algorithm proposed by B. Yagoubi et al. Experiments show that our algorithm reduces average response time to about 3% compared to that of HDLA algorithm.**

Keywords-dynamic load balancing; hierarchical cluster level; multiple thresholds; pairwise immigration

I. INTRODUCTION

Current computing architectures have allowed the popularity of grid computing. Grid technology is a type of distributed system which supports the sharing and coordinated use of resources, independently of their physical type and location, in dynamic virtual organizations that share the same goal [2]. The grid allows the use of data intensive applications, research of DNA sequences, molecular dynamics, protein biosynthesis, etc [3].

Two major parties in grid computing, namely resource consumers (users) who submit various applications and resource providers who share their resources, usually have different motivations when they join the grid. These incentives are presented by objective functions in scheduling. While grid users basically deal with the performance of their applications such as cost of CPU power to run a particular application, resource providers usually pay more attention to the performance of their resources such as the resource utilization in a particular period. Thus, objective functions can be classified into two categories [4]: application-centric and resource-centric.

Application-centric (application-level**)** objective function aims to optimize the performance of each individual application. Most of current grid applications concern about execution time, such as the makespan or response time.

Resource-Centric (system-level) objective functions aim to optimize the performance of resources. Resource-centric objectives are usually related to resource utilization, for example:

- Throughput which is the number of completed jobs in a given period.
- Utilization which is the percentage of time when the node is busy.

We focused on decreasing response time because the criterion is highly sensitive to users and the amount of resource is widely available in the case of grid or cloud computing systems.

It is known that a grid is composed of heterogeneous nodes and users dynamically submit tasks. Such environments require different QoS to grids. They lead to a situation that some nodes are overloaded and some of them are under-loaded during their operations. Overloading and under-loading ill-affect to average response time. In order to reduce the average response time, load balancing should be applied to grid system. In classical parallel and distributed systems, load balancing algorithms usually run on homogeneous and dedicated resources. Such traditional algorithms cannot work properly in grid architectures because a grid has a lot of specific characteristics like heterogeneity, autonomy, dynamicity, and scalability, which make the load balancing problem more difficult [5]. The goal of load balancing is to increase the overall system performance by

balancing loads at the resources or by forwarding loads to lightly weighted resources. Load balancing has been intensively studied for more than two decades as an interesting topic [6-9].

We propose a load balancing scheme that has following features:

1) The scheme runs on grid system with two-level hierarchical architecture. Hierarchical structure is a natural choice because a grid system is composed of thousands of nodes and tens of clusters.
2) In each level the proposed scheme uses two thresholds for deciding under-loaded and overloaded nodes. Two thresholds reduce the amount of immigrated tasks than schemes that make nodes have same loads.
3) We assume that grid system can expect exact processing times of tasks. Real-time systems satisfy the assumption. If a grid system executes small type of tasks repeatedly, it can expect processing time after some iteration.
4) The scheme uses pairwise immigration, which simplifies message transfer and reduces the amount of messages.

The rest of the paper is organized as follows. Section II presents previous works related to dynamic load balancing in distributed and grid systems. Section III describes a hierarchical grid model where our scheme is applied. Proposed dynamic two level load balancing strategy and simulation result are presented in Section IV and Section V, respectively. Finally Section VI gives short conclusion and some information about future works.

II. Related Works

Belabbas et al. proposed a hierarchical dynamic load balancing algorithm HDLA [1]. The objective of HDLA is to reduce average response time of tasks and their transfer cost. HDLA runs on two-level hierarchical architecture. A cluster is a group composed of nodes and a cluster manager, and a grid is a group composed of clusters and a grid manager. The first level is a cluster, and the second level is a grid.

In the algorithm, sum of task processing times of an entity is the workload index of the entity. The entity is a node if the workload index is used by cluster manager, and it is a cluster in the case of grid manager. The paper defines the workload index of an entity as $\frac{LOD}{SPD}$, where *LOD* is sum of tasks size in the entity and *SPD* is the CPU power of the entity. In order to decide whether an entity is balanced or not, two level thresholds on the workload index are suggested: higher threshold $T_h = TEX + \delta \cdot \varepsilon$ and lower threshold $T_l = TEX - \delta \cdot \varepsilon$, where *TEX* is an average workload index in the level, δ is a standard deviation of workload index in the level, and $\varepsilon \in$ [0-1] is a value that decides the range of balance. If load of a node is greater than higher threshold, the node is called overloaded. If load of a node is less than lower threshold, the node is called under-loaded. Otherwise, the node is called normally loaded.

A situation that a workload exceeds the capacity of a group, that is CPUs are fully utilized and some tasks are waiting in queues, is called saturated. In the case, HDLA does not start load balancing in the group since its elements will remain overloaded. Terms Supply and Demand are introduced to decide immigration. Supply is $\sum_{n \in C_u}(T_l - load_n)$, and Demand is $\sum_{m \in C_o}(load_m - T_h)$, where C_u is a set of under-loaded entities in a group *C*, and C_o is a set of overloaded entities in a group *C*. When $Supply/Demand > \rho$ then tasks immigration start, where threshold ρ is set to 0.75.

HDLA has some limitations. First, it does not manage heavily overloaded nodes when its group is saturated. Second, if the amount of supply is not sufficient for the amount of demand, HDLA algorithm does not provide load balancing.

Malarvizhi and Rhymend proposed another hierarchical load balancing algorithm [11]. When a task is submitted by a user, a local scheduler defines maximum share of every node in its cluster, where maximum share of a node is the number of tasks that the node can execute without being overloaded. That is, for node *s* in cluster *m* maximum share is

$$R_{share}(s) = \frac{N \cdot R_{sC}}{C_{mC}} , \quad (1)$$

where *N* is number of jobs arrived at cluster *m*, and R_{sC} and C_{mC} are capacity of node *s* and cluster *m*, respectively. Here the capacity refers to the number of jobs a node or a cluster can process per second.

The number of tasks in a node is used as a workload index in the paper. To decide load state of the node, the maximum share and the number of tasks in the node are compared. Unfortunately, the number of tasks is not a good workload index when weights of tasks are different. A node can be overloaded even when the number of tasks in the node is smaller than that in under-loaded node.

III. Proposed Grid Model

A *grid* is composed of multiple clusters and a *cluster* is composed of multiple nodes. A worker node or a *node* is a processing entity. Nodes in a cluster are connected by high speed low latency communication links, while communication links between clusters have low speed high latency property. So it is natural to construct two-level hierarchical grid model as shown in Fig. 1.

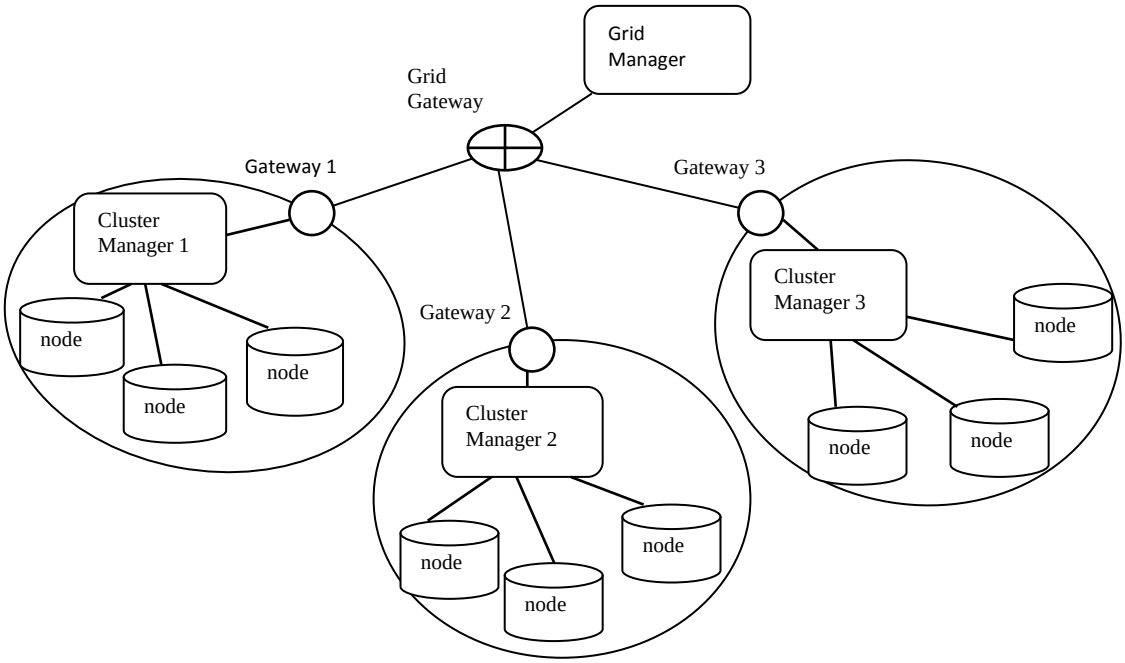


Figure. 1. Proposed grid model

A cluster is mapped to local level and the grid is mapped to grid level in the architecture. For each cluster there is a *cluster manager* that takes responsible for local load balancing. A *grid manager* is connected to a grid gateway and it is responsible for grid level load balancing. Multiprogramming is applied to each worker node for performance, and the maximum multiprogramming degree is pre-defined. Assume that a task is submitted to a node. If the multiprogramming degree of the node is less than the maximum degree, the task is loaded into memory of the node. Otherwise, the task waits in a queue of the node.

Each node is responsible for

- processing tasks, returning results,
- sending workload information to its cluster manager periodically,
- running received tasks or storing to queue, and
- migrating tasks to the cluster manager.

Each cluster manager is responsible for

- receiving workload information from nodes, estimating workload its cluster, sending it to the grid manager,
- receiving immigration order from the grid manager, redistributing immigration orders to nodes,
- forwarding migrated tasks from nodes to target nodes in the cluster,
- making a decision to begin local load balancing, and ordering nodes to make task immigration.

At the grid level, grid manager responsible for

- receiving workload information of clusters from cluster managers,
- deciding to start grid load balancing, and ordering to cluster managers.

IV. PROPOSED LOAD BALANCING SCHEME

Before explain our algorithm we give some notations used in the paper.

A. Some notations:

- Cluster *m* is a set of nodes.
- Grid *G* is a set of clusters.
- Node *s* is an entity where tasks run. There could be one or multiple CPUs and I/O devices.
- *Task i* is an entity that is executed in a node.
- N_G is the number of clusters in grid *G*.
- N_m is the number of nodes in cluster *m*.

B. Local load balancing scheme

Expected processing time $PT_s(i)$ for a given task *i* in node *s* is computed as follows:

$$PT_s(i) = \frac{L_i}{P_s} + t_s(i), \quad (2)$$

where $L(i)$ is length of task i in node s, P_s is a processing power of node s (in MIPS), and $t_s(i)$ is executed time of task i in node s at the moment. $EPT(s)$ is an expected processing time of node s. Thus $EPT(s)$ is equal to the sum of processing times of tasks in node s as follows:

$$EPT(s) = \sum_{i \in s} PT_s(i). \quad (3)$$

Periodically, each node sends its *EPT* to its cluster manager.

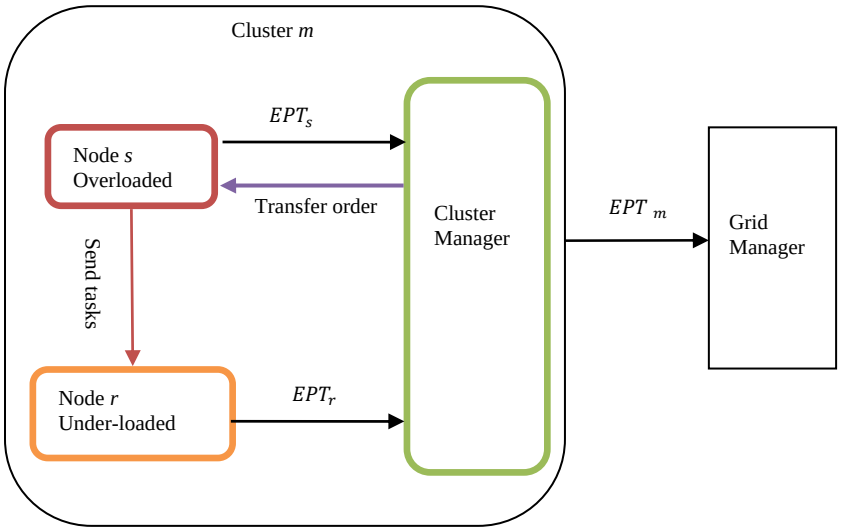


Figure 2. Local load balancing

When a cluster manager receives *EPT* from all nodes in its cluster m, it calculates following values:

-EPT_m, average of the *EPT* values from nodes in cluster m:

$$EPT_m = \frac{1}{N_m} \sum_{s \in m} EPT(s). \quad (4)$$

-σ_m, standard deviation of the *EPT*s from nodes in cluster m.

-upper and lower thresholds for cluster m :

$$T_h(m) = EPT_m + \sigma_m \cdot \varepsilon, \quad (5)$$

$$T_l(m) = EPT_m - \sigma_m \cdot \varepsilon, \quad (6)$$

where ε is a parameter that determines distance of two thresholds.

A cluster manager decides load state of each node by comparing *EPT* of the node with the thresholds. A state for a node can be under loaded, normally loaded, or overloaded. If *EPT* of a node is less than lower threshold, the node is under loaded. If *EPT* of a node locates between two level thresholds, it means that the node is normally loaded. Otherwise it is overloaded.

Next, the cluster manager computes *supply* of each under-loaded node and *demand* of each overloaded node in the cluster. $supply_m(s)$ is a difference between $EPT(s)$ of an under-loaded node s and the lower threshold $T_l(m)$. $demand_m(r)$ is a difference between $EPT(r)$ of an overloaded node r and the higher threshold $T_h(m)$. Thus, for under-loaded node s and overloaded node r in cluster m, supply and demand are computed as follows:

$$supply_m(s) = T_l(m) - EPT_m(s), \quad (7)$$

$$demand_m(r) = EPT_m(r) - T_h(m). \quad (8)$$

From Equations (7) and (8) supply and demand for cluster m are computed as follows:

$$supply_m = \sum_{s \in under(m)} supply_m(s), \quad (9)$$

$$demand_m = \sum_{r \in over(m)} demand_m(r), \quad (10)$$

where *under*(*m*) is the set of under-loaded nodes, and *over*(*m*) is the set of overloaded nodes in cluster *m*. The cluster manager sorts overloaded nodes by descending order of their *EPT*s and under-loaded nodes by ascending order of their *EPT*s. Since the proposed scheme uses pairwise immigration, the number of immigration pairs is the smaller number between the number of overloaded nodes and the number of under-loaded nodes. For example, if there are *c* overloaded nodes and *d* under-loaded nodes in the cluster, then min(|*under*(*m*)|, |*over*(*m*)|) pairs are constructed for immigration. Pairing is done in the sorted order of *EPT*s. The most overloaded node is paired with the most under-loaded node. Given an immigration pair (*s*, *r*), the immigration size is min($supply_m$(*s*), $demand_m$(*r*)). The cluster manager sends an *immigration order* with (immigration *size*, id of under-loaded node *s*) to the overloaded node *r*. After sending all orders to nodes in a cluster, the cluster manager *m* sends its EPT_m to the grid manager. If an overloaded node receives an immigration order (*size*, *s*) from its cluster manager, the node transfers tasks in its queue to the under-loaded node *s*. The amount of emigrated tasks should not exceed *size*.

Let us show an example of local load balancing of the proposed scheme. Assume we have five nodes node 1, node 2, node 3, node 4, and node 5 with *EPT* 60, 65, 43, 48, and 40, respectively in cluster *m* as shown in Figure 3. When cluster manager *m* receives these EPTs, it calculates average EPT_m 51.2 and standard deviation σ_m 10.85 for the cluster. From Equations (5) and (6), if ε = 0.5, upper and lower thresholds are 56.6 and 45.7, respectively. Node 1 and node 2 are overloaded because their EPTs are greater than upper threshold 56.6. Node 4 is normally loaded. Node 5 and node 3 are under-loaded as shown in Table I. The cluster manager *m* estimates "Supply" and "Demand" for cluster *m*. From Equation (7), supplies of node 3 and node 5 are 2.7 and 5.7, respectively. Demands for overloaded node 1 and node 2 are 3.4, and 8.4, respectively. Supply of node 5 is 5.7 and it is less than 8.4 demand of node 2, so node 2 emigrates tasks of amount 5.7 to node 5. Demand of node 1 (3.4) is greater than supply of node 3 (2.7), hence node 1sends tasks to node 3 while EPT of tasks sent lower than 2.7 supply of node 3.

TABLE I. MAKING A DECISION IN LOCAL LEVEL LOAD BALANCING

Node	**EPT**	**state**	***supply***	***demand***	**target node**	**Immigration size**
1	60	Overloaded		3.4	3	2.7
2	65	Overloaded		8.4	5	5.7
3	43	Under-loaded	2.7			
4	48	Normally loaded				
5	40	Under-loaded	5.7			

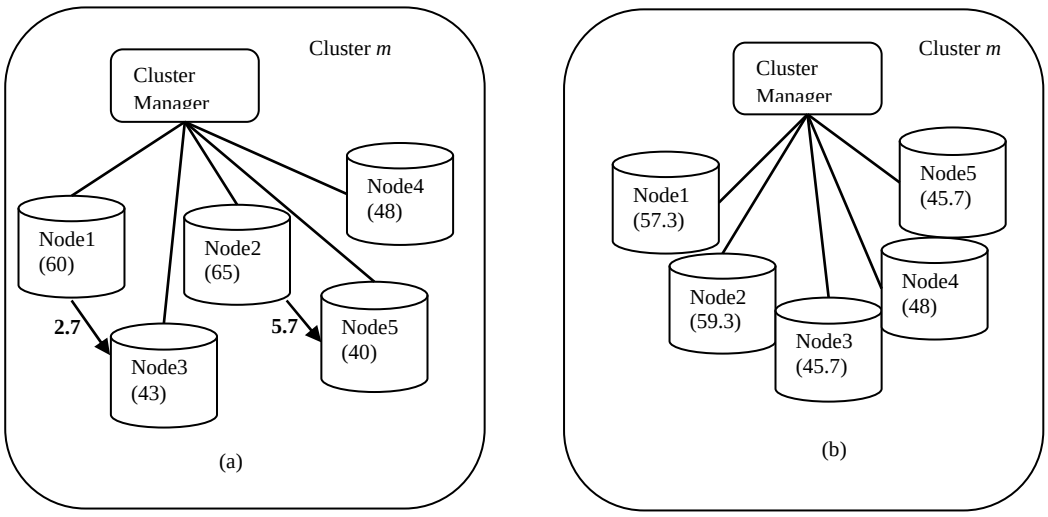


Figure 3. Load migration in local load balancing, (a) before immigration, (b) after immigration.

C. Grid load balancing

After the local load balancing, a cluster manager *m* sends cluster workload EPT_m to the grid manager. The grid manager computes EPT_G average of *EPT*s from cluster managers as follow:

$$EPT_G = \frac{1}{N_G}\sum_{m \in G} EPT_m, \quad (11)$$

where N_G is the number of the clusters in grid. The grid manager also computes standard deviation σ_G of *EPT*s and thresholds for grid level as follows:

$$T_h(G) = EPT_G + \sigma_G \cdot \varepsilon, \quad (12)$$

$$T_l(G) = EPT_G - \sigma_G \cdot \varepsilon. \quad (13)$$

Then it partitions the set of clusters to normally loaded, under-loaded and overloaded clusters. It computes supplies of under-loaded clusters and demands of overloaded clusters. It also pairs clusters according to their EPTs as cluster managers do. If there is a cluster pair (C_s, m), the grid manager sends an order including (immigration size, C_s) to overloaded cluster m. When cluster manager m receives the order (*size*, C_s), it computes immigration amount $P_m(s)$ for overloaded or normally loaded node s in cluster m as follows:

$$P_m(s) = size \cdot \frac{EPT(s)}{\sum_{s \in over(m)} EPT(s)} \quad \text{if } over(m) \neq \Phi\,, \quad (14)$$

$$P_m(s) = size \cdot \frac{EPT(s)}{\sum_{s \in normal(m)} EPT(s)} \text{ if } over(m) = \Phi\,, \quad (15)$$

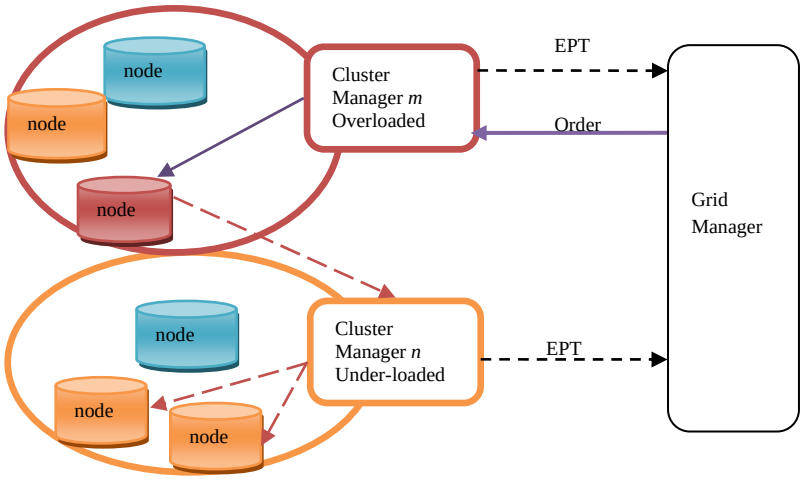
where *normal*(m) is the set of normally loaded nodes in cluster m. Cluster manager m forwards an immigration order with $P_m(s)$ to each overloaded node s in m. Overloaded node s transfers tasks of the amount $P_m(s)$ to cluster manager C_s. A cluster manager merges the immigrated tasks and computes temporary share amount $Q_n(r)'$ for each task r in cluster n as follows:

$$Q_n(r)' = size \cdot \frac{EPT(r)}{\sum_{r \in under(n)} EPT(r)} \quad \text{if } under(n) \neq \Phi, \quad (16)$$

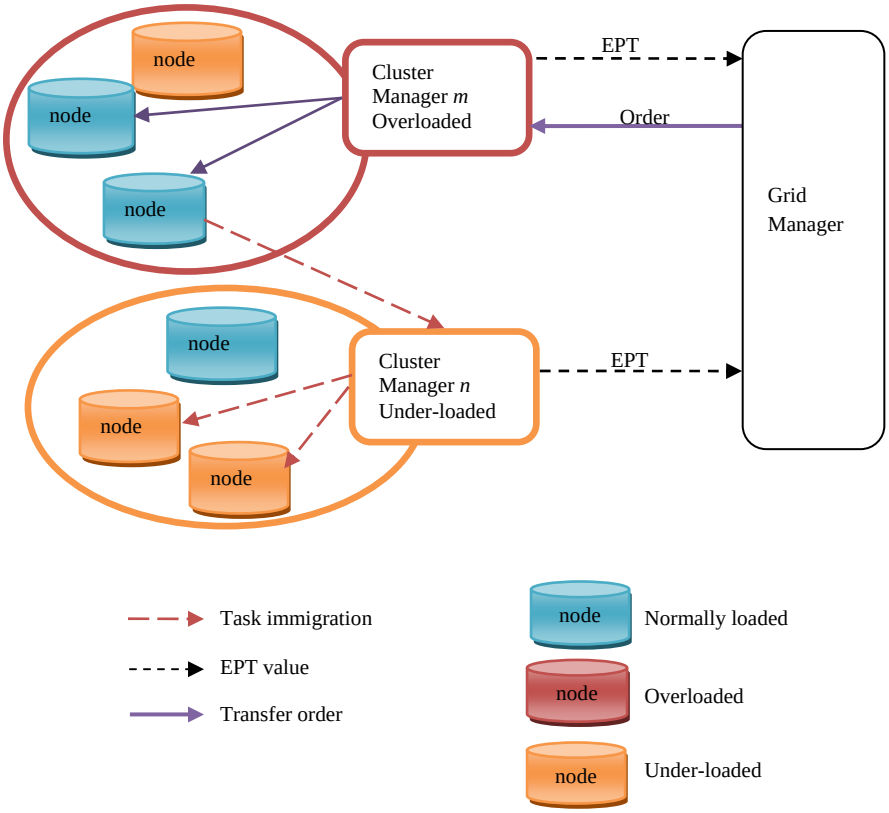
$$Q_n(r)' = size \cdot \frac{EPT(r)}{\sum_{r \in normal(n)} EPT(r)} \text{ if } under(n) = \Phi. \quad (17)$$

Next, the share mount values are reverted in the order of $EPT(r)$. If *EPT*s of under-loaded nodes $EPT(r_1)$, $EPT(r_2)$,…, $EPT(r_R)$ are in ascending order, $Q_n(r_1)'$, $Q_n(r_2)'$ … $Q_n(r_R)'$ are also in ascending order. $Q_n(r_i)'$ is allocated to node $r_{R\text{-}i+1}$. Thus share amount $Q_n(r) = Q_n(r_{R\text{-}i+1})'$. The cluster manager divides the immigrated tasks to the amount $Q_n(r)$ for each under-loaded or normally loaded task r and sends amount $Q_n(r)$ of tasks to node r.

Fig. 4 shows two cases of grid level load balancing. Red node is overloaded, blue node is normally loaded, and orange node is under-loaded. In Fig. 4 (a), cluster manager m orders overloaded node to immigrate tasks to cluster n, and the tasks are divided and forwarded into two under-loaded nodes. Fig. 4 (b) shows a case that an overloaded cluster has only normally loaded nodes. In the case, the amount of tasks are computed based on (15) and are distributed to the normally loaded nodes. The immigrated tasks are gathered at cluster manager n and are distributed to two under-loaded nodes as shown in Fig. 4 (b).



(a) Overloaded cluster has overloaded node



(b) All nodes of overloaded cluster are normally loaded

Figure 4. Grid level load balancing

Let us show an example of grid level load balancing using the proposed scheme. Assume a grid system has three clusters: node 1, node 2, node 3, node 4, and node 5 are belong to cluster 1 with loads 30, 35, 32, 40, and 45, respectively; node 6, node 7, and node 8 are belong to cluster 2 with load 50, 60, and 70, respectively; node 9, node 10, node 11, node 12, and node 13 belong to cluster 3 with load 70, 83, 92, 95, and 85, respectively. Average *EPT* for each cluster is shown in Table II. Higher and lower thresholds for grid level are 72.6 and 48.3, respectively. Thus cluster 1 is under-loaded and cluster 3 is overloaded. Demand of overloaded cluster 3 is 12.4. Supply of under-loaded cluster 1 is 11.9. Since the supply value is less than the demand value, grid manager orders cluster 3 to immigrate 11.9 amount tasks to cluster 1 as shown in Fig. 5.

TABLE II. AN EXAMPLE OF CLUSTER STATE

Cluster	EPT	state	Supply	Demand	Target cluster	Immigration size
1	36.4	Under-loaded	11.9		3	
2	60	Normally loaded				
3	85	Overloaded		12.4	1	11.9

When receiving the order from the grid manager, cluster 3 rechecks load states of all nodes. In our example cluster 3 has two overloaded nodes, node 11 and node 12. Cluster manager 3 divides 11.9 amount tasks using Equation (14), and it orders node 11 to immigrate load of 5.85 and node 12 to immigrate load of 6.05 to cluster 1.

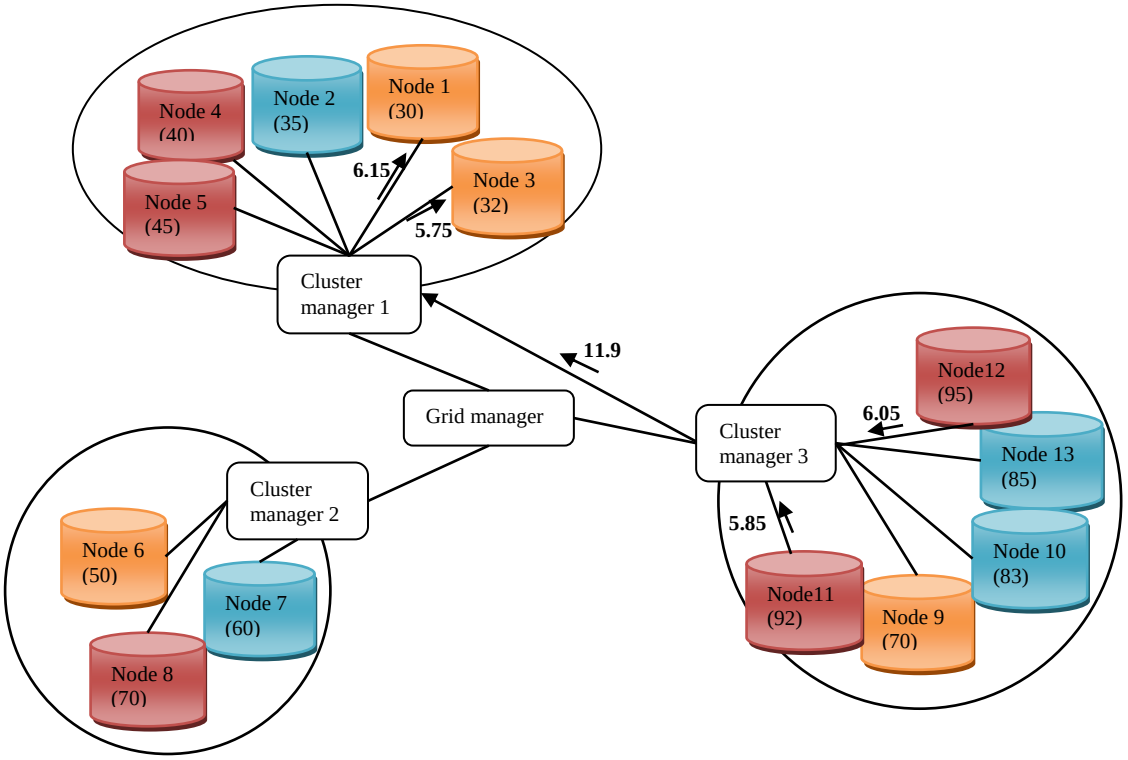
After receiving immigration tasks from overloaded cluster 3, under-loaded cluster manager 1 distributes the tasks to two under-loaded nodes, node 1 and node 3. The amount of share computed from Equation (16) is 5.75 for node 1 and 6.15 for node 3, and distributes these share loads in reverse order. As a result, node 1 gets load of 6.15 and node 3 gets load of 5.75.

V. SIMULATION RESULTS AND ANALYSIS

In order to present the performance of the proposed scheme, we constructed a simulation on GridSim 5.0 which is Java based simulation tool [12]. All experiments have been performed on 2.2 GHz P4 Intel Pentium with 2 GB main memory, running on Windows 7. In order to obtain reliable results, the same experiments have been run 10 times. Simulated grid model includes 4 clusters and 16 nodes. Each cluster has 4 nodes.

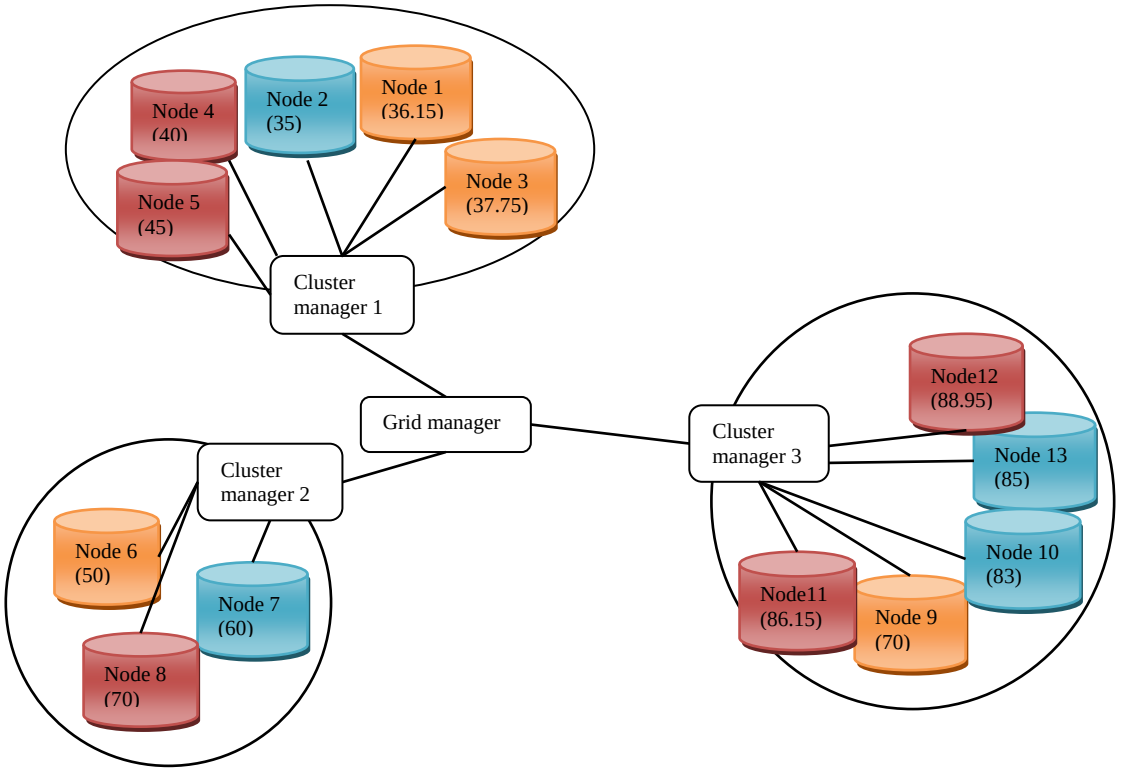
Average arrival rate of tasks is 4 (per second) for each node. The characteristics of nodes, jobs, and network link are shown in Table III. The experiment focuses on jobs which are computationally intensive tasks as it is more common in today's real life applications [13]. The maximum multi-programming degree of each node is 10, and each node sends *EPT* every 10 seconds.

TABLE III. SIMULATION PARAMETERS



(a)

Node Characteristics	
Number of machines per node	1
Number of processing element per machine	1
Processing element rating	10 MIPS
Bandwidth	8000-40000 bits/sec
Job Characteristics	
Length	0-50000 MI
File Size	100+ (10% to 40%) Bytes
Output Size	250+ (10% to 50%) Bytes
Network Link Bandwidth	
LAN	8000-40000 bits/sec
WAN	5000-10000 bits/sec



(b)

Figure 5. Load migration in grid load balancing, (a) before migration, (b) after migration.

We implemented no load balancing scheme and HDLA [1], and the proposed scheme in order to compare performance. First, response times of three schemes are compared on uniformly distributed random workloads. According to the Fig. 6 our algorithm has the best performance on response time.

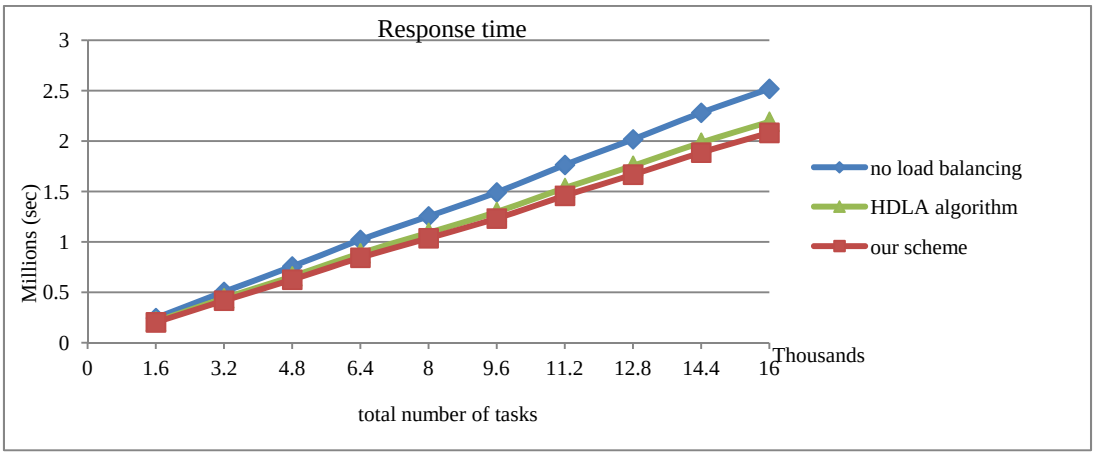


Figure 6. Average response time

Second, we created a hot spot node in a cluster where the arrival rate of the node is greater than that of other nodes. Fig. 7 illustrates comparison result when arrival rate of the hot spot node is 5 times higher than that of other nodes. The hot spot is node 1 in the simulation. According to results as shown in Fig. 7, proposed scheme makes system load always balanced fairly among nodes.

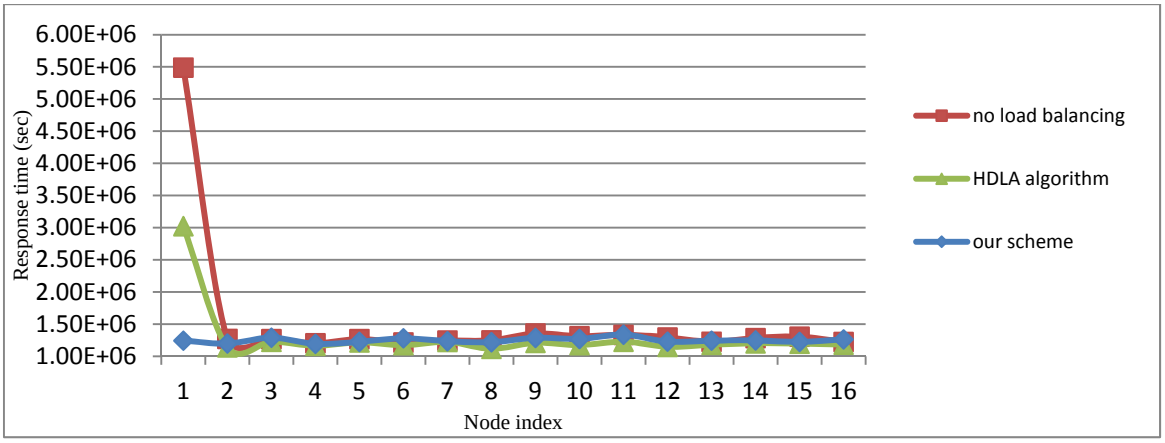


Figure 7. Response time in hot spot case. Node 1 has 5 times arrival rate of others.

VI. CONCLUSION

This paper addresses pair-wise dynamic load balancing scheme for computational grid systems with hot spot. According to results, the proposed load balancing strategy has good responsible time against system imbalance. In the future works we will improve our load balancing scheme with fault tolerant features to increase the reliability of our algorithm.

ACKNOWLEDGMENTS

This paper was supported by Research Fund, Kumoh National Institute of Technology.

REFERENCES

[1] B.Yagoubi, H. Nadia, and H. T. Lilia, "Dynamic load balancing in Grid computing," African Journal of Research in Computer Science and Applied Mathematics, (ARIMA), vol. 7, pp. 1–19, 2007.
[2] I. Foster, C. Kesselmen, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," IEEE Journals &Magazines, Computer, vol. 35, pp. 37-46, 2002.
[3] I. Foster and C. Kesselman, "The grid2: Blueprint for a new computing infrastructure," The Elsevier Series in Grid Computing, second edition, 2004.
[4] I. Foster and T. S. Chervenak, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," Journal of Network and Computer Applications, vol. 23, pp. 187-200, 2001.
[5] C. Xu and F. M, "Load balancing in parallel computers: theory and practice" in International Journal of Network and Computer Applications, 2000.
[6] A. J. H. Fran Berman, Geoffrey Fox, "Grid computing: Making the global infrastructure a reality," Willey Series in Communications Networking and Distributed Systems, 2008.
[7] C. K. H. Kameda, J. Li and Y. Zang, Optimal load balancing in distributed computer systems, Springer, 1997.
[8] S. R. Malarvizhi and Rhymend, "Resource scheduling in hybrid grid environment," IJCSNS International Journal of Computer Science and Network Security, vol. 4, pp. 1471-1479, 2010.
[9] A. B. Saxena and D. Sharma, "Analysis of threshold based centralized load balancing policy for heterogeneous machines," International Journal of Advanced Information Technology (IJAIT), vol. 1, pp. 39-53, October 2011.
[10] B.Yagoubi and M. Meddeber, "Distributed loacd balancing model for grid computing," ARIMA journal, vol. 2, January 2007.
[11] S. R. Malarvizhi and Rhymend, "Hierarchical status information exchange scheduling and load balancing for computational grid environment," International Journal of Computer Science and Network Security, vol. 10, pp. 177-185, February 2010.
[12] R. Buyya and M. Murshed, "Using the GridSim Toolkit for Enabling Grid Computing Education," Proc. of International Conference on Communication Networks and Distributed Systems Modeling and Simulation, January 27-31, 2002.
[13] A. Moallem, "Using swarm intelligence for distributed job scheduling on the grid," Master thesis, University of Saskatchewan, Canada, 2009.

AUTHORS PROFILE

Hojiev Sardor Qurbonboyevich, received the BSc degree in telecommunication from Tashkent University of Information Technologies, Uzbekistan. Currently, he is MS student in the Department of Computer Engineering, Kumoh National Institute of Technology, Gumi City, South Korea. His research interests include dynamic load balancing and resource allocation in Grid and Cloud Computing.

Tae-Young Choe, is working as Associate Professor in the Department of Computer Engineering, Kumoh National Institute of Techonology, Gumi City, South Korea. Currently, his research interests are load balancing in Cloud Computing and parallel algorithms using graphic devices.