ISSN:0975-9646

Centralized Modified Estimated Load Information Scheduling Algorithm with Reduced Communication delays

Amit Kumar Sharma

Assistant Professor, PSIT-COE Kanpur, U.P

Abstract- In this paper we have successfully reduced the total execution time of jobs submitted in a computational grid. We have addressed several important issues like cost, reliability and scalability. Centralized Melisa is a load balancing algorithm in which a task is scheduled by a central scheduler on several status parameters like job queue length, job arrival rate, service rate. MELISA, which is applicable to large-scale systems (that is, interGrid [1]), is a modified version of ELISA [2] in which we consider the job migration cost, resource heterogeneity, and network heterogeneity when load balancing is considered. A computational Grid is the cooperation of distributed computer systems where user jobs can be run on either local or remote computer systems In general, any load-balancing algorithm consists of two basic policies—a transfer policy and a location policy. The transfer policy decides if there is a need to initiate load balancing across the system. By using workload information, it determines when a node becomes eligible to act as a sender (transfer a job to another node) or as a receiver (retrieve a job from another node).

Index terms- **grid, melisa, cmelisa, elisa, load balancing, communication delays**

I. Introduction

In general, any load-balancing algorithm consists of two basic policies—a transfer policy and a location policy. The transfer policy decides if there is a need to initiate load balancing across the system. By using workload information, it determines when a node becomes eligible to act as a sender (transfer a job to another node) or as a receiver (retrieve a job from another node). The location policy determines a suitably underloaded processor. 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. In a static algorithm, the scheduling is carried out according to a predetermined policy. The state of the system at the time of the scheduling is not taken into consideration. On the other hand, a dynamic algorithm adapts its decision to the state of the system. Adaptive algorithms are a special type of dynamic algorithms where the parameters of the algorithm and/or the scheduling policy itself is changed based on the global state of the system.

Dynamic load balancing algorithms can be further classified into a **centralized approach** and a **decentralized approach**. In the centralized approach only one node in the grid acts as the central controller. It allocates jobs to each of the slave nodes. The slave nodes execute the jobs assigned by the controller. The centralized approach is a simple approach and is beneficial when the communication cost is less significant. It is mainly used for a small size grid. Although the centralized approach is used currently, it limits the scalability of the grid by becoming a bottle neck. Also failure of central controller can cause the entire system to fail. [5].

In the decentralized approach all nodes in the grid are involved in making the load balancing decision. The decentralized algorithms are scaleable and have better fault tolerance. The decentralized approach is preferred because elements of the network may vary in capacity or number during run time. Although the decentralized approach is suitable for dynamic heterogeneous resources it increases the communication overhead to a large extent. [5]

II. Metrics For Comparing Various Dynamic Load Balancing Algorithms For Heterogeneous Resources

The various metrics identified for comparing the load balancing algorithms are-

- **Communication overhead**- communication overhead is the status information which each node has to provide to other nodes in the grid.
- **Load balancing time** - Amount of time that elapses between the job arrival time and the time at which the job is finally accepted by a node.
- **Scalability** - It is the ability of the algoritm to perform load balancing for a grid with any finite number of nodes.
- **Fault tolerance** - It is the ability of the algorithm to perform uniform load balancing in spite of arbitrary node or link failure.
- **Reliability** - It is the ability of the algorithm to schedule job in predetermined amount of time.
- **Stability** - It is defined as the maximum job arrival rate which the load balancing algorithm.

III. VARIOUS DYNAMIC LOAD BALANCING ALGORITHMS

The centralized and decentralized load balancing algorithms can be further classified into sender-initiated algorithms, receiver-initiated algorithms and symmetrically initiated algorithms according to their location policies. Sender initiated algorithms let the heavily loaded nodes take the initiative to request the lightly loaded nodes to receive the jobs; while receiver initiated algorithms let the lightly loaded nodes invite heavily loaded nodes to send their jobs. Symmetrically-initiated algorithms combine the advantages of both sender and receiver initiated algorithms. [5][1] Lee et al., [8] propose in the receiver initiated receiver broadcasting algorithm that whenever a processor in the grid becomes idle it broadcasts a request message. After receiving this request, the most heavily loaded node sends a task to the idle processor. This algorithm does not require an information process to collect the load information of other nodes, which incurs heavy communication traffic. This algorithm shows better total execution time of jobs and better node utilization with a smaller number of job migrations. The nodes which are located far away from each other can communicate quickly. This approach requires additional hardware for implementing the single bus structure and arbitration logic. Venu Gopalachari et al., [6] eliminate the time delay due to arbitration in the receiver broadcasting algorithm in their dynamic scheduling using weights algorithm by proposing that the jobs are assigned to those nodes which have minimum memory utilization or maximum elapsed time.

IV. RELATED WORKS

Numerous researchers have proposed scheduling algorithms [2], [12], [13] for parallel and distributed systems, as well as for Grid computing environment [14], [15], [16], [17], [18], [19], [20], [21], [22]. For a dynamic load-balancing algorithm, it is unacceptable to frequently exchange state information because of the high communication overheads. In order to reduce the communication overheads, Martin et al. [23] studied the effects of communication latency, overhead, and bandwidth in cluster architecture to observe the impact on application performance. Anand et al. [2] proposed an estimated load information scheduling algorithm (ELISA), and Mitzenmacher [24] analyzed the usefulness of the extent to which old information can be used to estimate the state of the system. Arora et al. [21] proposed a decentralized load-balancing algorithm for a Grid environment. Although this work attempts to include the communication latency between two nodes during the triggering process on their model, it did not consider the actual cost for a job transfer. Our approach takes the job migration cost into account for the load-balancing decision. In [15], [16], and [18], a sender processor collects status information about neighboring processors by communicating with them at every load-balancing instant. This can lead to frequent message transfers. For a large-scale Grid environment where communication latency is very large, the status exchange at each load-balancing instant can lead to large communication overhead. In our approach, theproblem of frequent exchange of information is alleviated by estimating the load, based on the system state information received at sufficiently large intervals of time. We have proposed algorithms for a Grid environment that are based on the estimation approach as carried out in the design of ELISA [2]. In ELISA, load balancing is carried out based on queue lengths. Whenever there is a difference in queue length, jobs will be migrated to the lightly loaded processor, ignoring the job migration cost. This cost becomes an important factor when the communication latency is very large such as for a Grid environment and/or the job size is large. Both of our algorithms balance the load by considering the job migration cost, which is primarily influenced by the available bandwidth between the sender and receiver nodes.

V. OUR WORK

We have proposed two dynamic, adaptive, and decentralized load balancing algorithms for computational Grid environments that are shown to be applicable in balancing loads depending on the size of the underlying Grid infrastructure. Thus, for smaller size Grids, one of our algorithms, Load Balancing on Arrival (LBA), is shown to be effective, whereas for large-scale Grid systems, the Modified ELISA (MELISA) is shown to have better control in balancing the loads. One of the key strengths of our algorithm is in estimating the system parameters and in proactive job migration. For large-scale Grid environments, resources are geographically distributed, and the communication latency between them is also very large due to the WAN through which they are usually connected. Therefore, the job migration cost, based on the estimate of the traffic and loading conditions, becomes an imperative factor for load balancing. Our proposed algorithms consider the job migration cost, which is primarily influenced by the available bandwidth between the sender and receiver nodes, when making a decision for load balancing. Further, Grid are dynamic in nature in the sense of resource availability and, hence, a changing network topology. Resource heterogeneity and network heterogeneity also exists in the Grid environment. We have also considered these facts into account by generating a random topology with nodes of varying capacities.

VI. SYSTEM MODEL AND PROBLEM DEFINITION

Our Grid system consists of M heterogeneous processors, P1; P2; . . . ; PM, connected via communication channels assuming an arbitrary topology (Fig. 1). We assume that each processor 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. The jobs are assumed to arrive randomly at the processors, the interarrival time being exponentially distributed with average $1/\lambda_i$. The jobs are assumed to require service time that is exponentially distributed with mean $1/\mu_i$. Each processor is modeled as an M|M|1 Markov chain, with the number of jobs queued up for processing at each processor representing the state of the system. Job size is assumed to have a normal distribution with a given mean and variance. This job size includes both the program and data sizes. Since in a Grid environment, the network topology is varying, our model captures this constraint as well by considering an arbitrary topology. The data transfer rate is not fixed and varies from link to link. The processors that are directly connected to a processor constitute its buddy set. We also assume that each processor has knowledge about its buddy processors and the communication latency between them, and load balancing is carried out within buddy sets only. It may be noted that two neighboring buddy sets may have a few processors common to each set. The job arrival rates and service rates are such that for some processor (say, Pi), $\lambda_i > \mu_i$ (that is, Pi is unstable), but the whole system always remains stable,

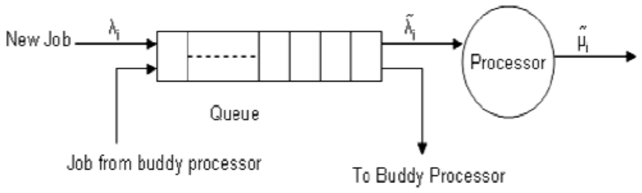


Fig 1. Job queue model

In this work, we have considered three performance metrics of relevance at three different levels. At the job level, we consider the ART of the jobs processed in the system as the performance metric. If N jobs are processed by the system, then

$$ART = 1/N\ (\Sigma\ (Finish_i - Arrival_i))$$

where Arrival $_i$ is the time at which the ith job arrives, and Finish $_i$ is the time at which it leaves the system. The delay due to the job transfer, waiting time in the queue, and processing time together constitute the response time. At the system level, we consider the total execution time as the performance metric to measure the algorithm's efficiency. It indicates the time at which all N jobs get executed. At the processor level, we consider resource utilization as the performance metric. It is the ratio between the processor's busy time to the sum of the processor's busy and idle time:

$U_i = Busy_i / (Busy_i + Idle_i)$

where $Busy_i$ indicates the amount of time Pi remains busy, and $Idle_i$ indicates the amount of time Pi remains idle during the total execution time of N jobs.

Thus, our objective is to design efficient load-balancing algorithms to minimize the ART of the jobs for computational Grid environments. We propose two different algorithms, LBA and MELISA, for small-scale and largescale Grid structures, as mentioned briefly in Section 1.2. Our algorithms will affect load balancing by careful estimation of the job arrival rates, CPU processing rates, and loads on the processor. Further, we take into account the resource heterogeneity, network heterogeneity, and job migration cost before a load-balancing decision.

A. Modified Estimated Load Information Scheduling Algorithm (Melisa)[19]

Although ELISA primarily works on estimates, it is mainly proposed for cluster-based supercomputing systems wherein the communication cost is not very large as resources are connected through a high-bandwidth network. However, for Grid-based supercomputing systems, the transfer delays are significantly high, and network heterogeneity also exists in terms of the varying available network bandwidth contributing to a large communication cost. Thus, the direct applicability of ELISA will yield an inferior performance that is unacceptable for Grid-based systems where heterogeneity exists in terms of resource and network. Hence, we revisit the design of ELISA and introduce the job transfer rate explicitly in a formulation that is more akin for Grids. In ELISA, at every status exchange time period T_s, each P_i communicates its status (queue length, estimate of the arrival rate) to all its buddy processors. At each estimation instant T_e, every processor calculates the queue length on buddy processors using the estimated arrival rate and exact service rate of a buddy processor. P_i will make a decision of job migration if its queue length is greater than the average queue length in its buddy set. In the design of MELISA, as shown in Fig. 3.3, each P_i estimates its arrival rate, service rate, and the load at each status exchange instant. At each estimation instant, P_i calculates the load on its all buddy processors using. Based on this calculated buddy load, each processor calculates the average load in its buddy set. P_i will make a decision of job migration if its load is greater than the average load in its buddy set and will try to distribute its load such that load on all buddy processors get finished at almost the same time, taking into account the node's heterogeneity in terms of processor speed. This average buddy load can be calculated using the following relationships. Let S_i denote the weight of a processor P_i, which is a normalized measure of its speed. Therefore, a value of 2 for S_i means that P_i will take half amount of time to execute a job than the time taken by the reference processor2 having a value of 1 for S_i. Here, each P_i will calculate the average

B. The Algorithm

- At the status exchange instant, for each processor:

1. Estimate the arrival rate and the service rate using the given equation. Also determine the load on a processor by dividing its queue length with estimated service rate using the given equations.
2. Communicate the status defined by a three tuple as <estimate load, arrival rate, service rate> to all processors in the buddy set.
3. Call TRANSFER

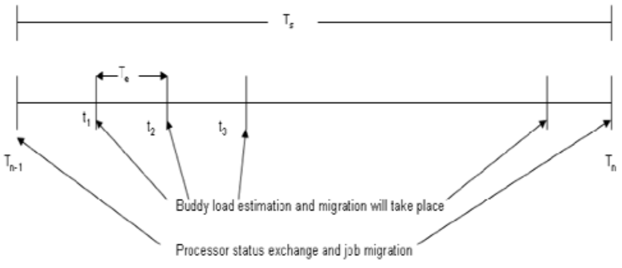
- At the estimated instant , for each processor

1. Estimate the load for each processor in the buddy set.
2. Call TRANSFER.

- By Pi (i=1,2,3…)

1. Estimate an average normalized buddy load.
2. If load of a processor is greater than average load , then
   i. Construct active set as follows: if a processor in the buddy set has load less than average normalized buddy load, include processor in active set.
   ii. Determine how much load can be transferred to buddy processors in active set such that load on all processors get finished at almost same time.

Attempt to migrate the load in excess over average buddy load to all buddy processors in active set by calculating EFT (average finish time) on destination processor in the active set and migrating the job only when $EFT_i < EFT_j$.



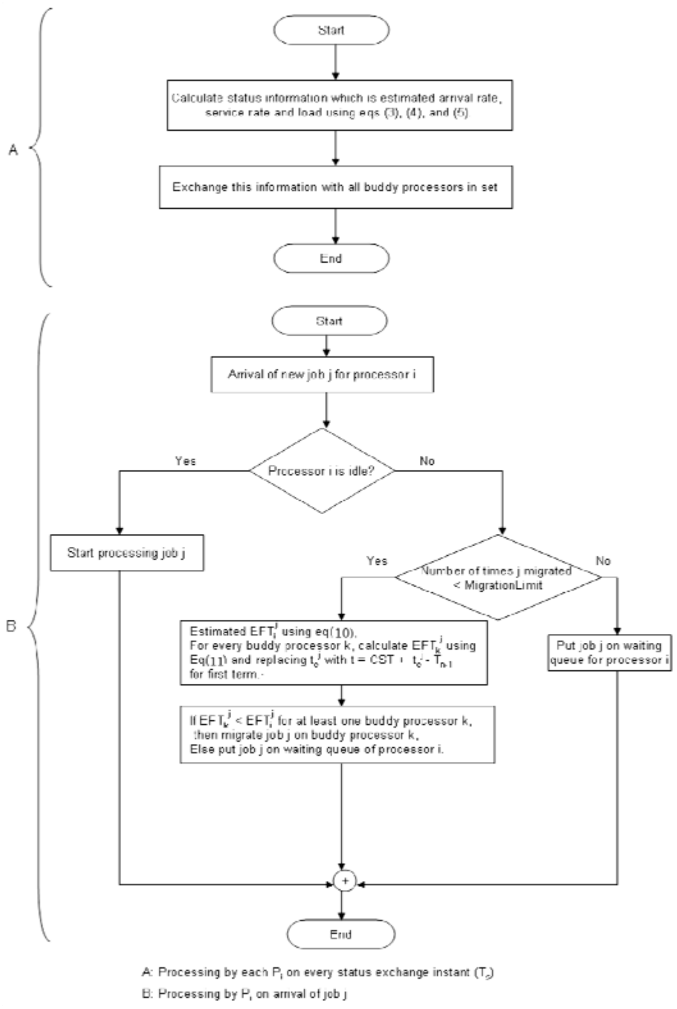
T_s
T_e
t_1
t_2
t_3
T_{n-1}
T_n
Buddy load estimation and migration will take place
Processor status exchange and job migration

Fig. 2. Estimation and status exchange intervals

C. The Steps For Simulating Task Scheduling

In this section we present high-level steps to demonstrate how GridSim 5.2 simulator can be used to simulate a Grid environment to analyze scheduling algorithms.

a) First, we need to create Grid resources of different capabilities and configurations. (a single or multiprocessor with time/space-shared resource manager)
b) We also need to create users with different requirements (application and quality of service requirements).
c) Second, we need to model applications by creating a number of Gridlets and define all parameters associated with jobs. The Gridlets need to be grouped together depending on the application model.
d) Then, we need to create a GridSim user entity that creates and interacts with the resource broker scheduling entity to coordinate execution experiment. It can also directly interact with GIS and resource entities for Grid information and submitting or receiving processed Gridlets.
e) Finally, we need to implement a resource broker entity that performs application scheduling on Grid resources.



A
Start
Calculate status information which is estimated arrival rate, service rate and load using eqs (3), (4), and (5)
Exchange this information with all buddy processors in set
End
B
Start
Arrival of new job j for processor i
Processor i is idle?
Yes
No
Start processing job j
Number of times j migrated < MigrationLimit
Yes
No
Estimated EFT_i^j using eq(10). For every buddy processor k, calculate EFT_k^j using Eq(11) and replacing t_c^j with t = CST + t_c^j - T_{n-1} for first term.
Put job j on waiting queue for processor i
If $EFT_k^j < EFT_i^j$ for at least one buddy processor k, then migrate job j on buddy processor k, Else put job j on waiting queue of processor i.
+
End
A: Processing by each P_i on every status exchange instant (T_s)
B: Processing by P_i on arrival of job j

Fig.3. MELISA alogorithm

VII. RESULTS

Our result comprise of the following tables. These tables contains the execution details of the algorithms under defined circumstances. So above details can be easily used to understand the effect of communication delays occurred earlier in our algorithm ELISA and MELISA.

A. Comparison of ELISA, MELISA, IMELISA and CMELISA

We can compare our worked on algorithm with the original algorithm under the following points.

i) Simulation details for elisa

Note:

a) Number of gridlets – 03
b) Baud rate (communication speed) = 10 Mbps
c) Gridlet size 900, 600 and 200 respectively for gridlet 0 , 1 and 2.
d) Centralized scheduling with no status acknowledgement.

TABLE 1
GRIDLET EXECUTION DETAILS FOR ELISA

Gridlet. No.	**Gridlet submit time(sec)**	**Gridlet finish time(sec)**	**Gridlet Execution time (sec)**	**Waiting time (sec)**
0	105.0	759.545	654.545	
1	105.0	1195.909	1090.9	550.36
2	105.0	1341.364	1236.3	250.46
Total Job execution time = 1236.36 sec				

ii) **Simulation details for melisa**

Note:

a) Number of gridlets – 03
b) Baud rate (communication speed) = 10 Mbps
c) Gridlet size 900, 600 and 200 respectively for gridlet 0 , 1 and 2.
d) Distributed scheduling with status acknowledgement from all gridlets.

TABLE 2
GRIDLET EXECUTION DETAILS FOR MELISA

Gridlet. No.	**Gridlet Submit time(sec)**	**Gridlet Finish time(sec)**	**Gridlet Execution time (sec)**
0	115.6	835.60	720.0
1	842.0	1322.0	480.0
2	1568.4	1728.40	160.0
Total Job execution time = 1612.8 sec			

iii) **Simulation details for Improved melisa i.e Imelisa**

Note:

a) Number of gridlets – 03
b) Baud rate (communication speed) = 10 Mbps
c) Gridlet size 900, 600 and 200 respectively for gridlet 0, 1 and 2.
d) Distributed scheduling with status acknowledgement only for even numbered gridlets.

TABLE 3
GRIDLET EXECUTION DETAILS FOR IMELISA

Gridlet. No.	**Gridlet submit time(sec)**	**Gridlet finish time(sec)**	**Gridlet Execution time (sec)**
0	115.6	835.60	720.0
1	842.0	1322.0	480.0
2	842.0	1428.0	586.0
Total Job execution time = 1366.4 sec			

iv) **Simulation details for centralized melisa i.e Cmelisa**

Note:
a) Number of gridlets – 03
b) Baud rate (communication speed) = 10 Mbps
c) Gridlet size 900, 600 and 200 respectively for gridlet 0 , 1 and 2.
d) Distributed scheduling without status acknowledgement.

TABLE 4
GRIDLET EXECUTION DETAILS FOR CMELISA

Gridlet. No.	**Gridlet submit time(sec)**	**Gridlet finish time(sec)**	**Gridlet Execution time (sec)**
0	115.6	835.60	719.4
1	115.6	1315.60	1200.0
2	115.6	1475.60	1360.0
Total Job execution time = 1360.0 sec			

Total Execution times of all variants:

TABLE 5
JOB EXECUTION DETAILS OF ALL ALGORITHMS

Execution times			
Elisa	**Melisa**	**Imelisa**	**Cmelisa**
1236.36	1612.8	1366.4	1360.0

VIII. RELATIVE COMPARISONS OF ALGORITHMS FOR EXECUTION TIME.

Melisa > Imelisa > Cmelisa > Elisa

TABLE 6
JOB EXECUTION DETAILS FOR MELISA AND MELISA

Execution time (sec)	
Melisa	1612.8
Imelisa	1366.4
Reduced communication	246.4 sec

TABLE 7
JOB EXECUTION DETAILS FOR CMELISA AND IMELISA

Execution time (sec)	
Cmelisa	1360.0
Imelisa	1366.4
Reduced communication	6.4 sec

TABLE 8
COMPARISON ON PARAMETERS

Parameters	**ELISA**	**MELISA**	**IMELISA**	**CMELISA**
Communication overhead	less	More	Less	less
Load balancing time	less	more	less	less
Scalability	Limited	Highly	Highly	moderate
Fault tolerance	less	Highly	Highly	moderate
Reliability	more	more	less	more
Buffer size	large	less	less	more

Graphical comparison on results:

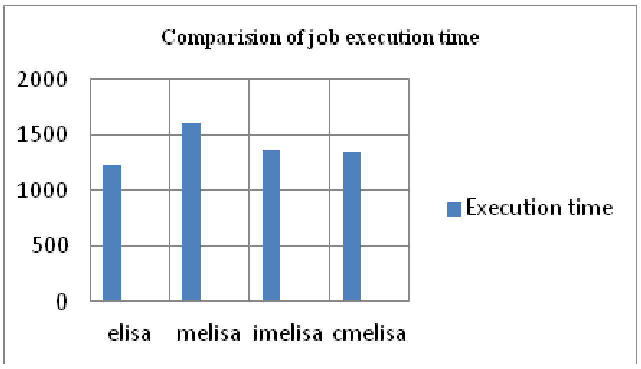


Fig 4 . Comparison of execution times

CONCLUSIONS

In this paper, we presented decentralized, scalable, adaptive, and distributed algorithms for load balancing across resources for ata-intensive computations on Grid environments. The objective is to minimize ART and the total execution time for jobs that arrive at a Grid system for processing. Several constraints such as communication delays due to the underlying network, processing delays at the processors, and an arbitrary topology for a Grid system are explicitly considered in the problem formulation. Our algorithms are adaptive in the sense that they estimate different types of strongly influencing system parameters such as the job arrival rate, processing rate, and load on the processor and use this information for estimating the finish time of job on a buddy processor. Through this study, we demonstrate the usefulness and effectiveness of the load estimation approach to devise adaptive and dynamic load-balancing strategies for data hungry essential computational Grid structures.

Future enhancements:

1. Study of the effect of job file size on the execution times of tasks and total execution times.
2. Increase in the bandwidth of the networks links to minimize the execution times.
3. Increasing the internetworks between the processing elements to increase the fault tolerance property.
4. Categorization of buffer to increase the reliability.

REFERENCES

[1] Ruchir Shah, Bhardwaj Veeravalli, Senior Member, IEEE ,On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Compu tational Grid Environments. *IEEE transaction on parallel and distributed computing*, vol. 18, no. 12, December- 2007

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid:Enabling Scalable Virtual Organizations," *Int'l J. High Performance Computing Applications,* vol. 15, no. 3, pp. 200-222, 2001.

[3] L. Smarr and C.E. Catlett, "Metacomputing," *Comm. ACM*, vol. 35, no. 6, pp. 44-52, June 1992.

[4] K. Lu, R. Subrata, A. Zomaya, An Efficient Load Balancing Algorithm for Heterogeneous Grid Systems Considering Desirability of Grid Sites,25th IEEE International Conference on Performance, Computing and Communication, April 2006, pp 320 – 328.

[5] A. Abed, G. Oz, A. Kostin, Competition-Based Load Balancing for Distributed Systems, *Proceedings of the Seventh IEEE International Symposium on Computer Networks* (ISCN' 06),pp 230 – 235.

[6] W. Lee, S.Hong, J.Kim, Dynamic Load Distribution on a Mesh with a Single Bus, *IEEE International Conference on Parallel and Distributed systems*, Dec. 1997, pp 368 – 375.

[7] M. Venu Gopalachari, P. Sammulal, Dr. A. Vinaya Babu, Correlating Scheduling and Load balancing to achieve optimal performance from a cluster,2009 *WEE International Advance Computing Conference* (IACC 2009), March 2009,pp 320 – 325

[8] A. Abed, G. Oz, A. Kostin, Competition-Based Load Balancing for Distributed Systems, Seventh *IEEE International Symposium on Computer Networks* (ISCN' 06),pp 230 – 235.

[9] R. Shah, B. Veeravalli, M. Misra, On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Computational Grid Environments *IEEE Transactions on Parallel and Distributed systems*, Vol. 18, Dec.2007, pp 167 – 1687.

[10] D. Acker, S. Kulkarni, A Dynamic Load Dispersion Algorithm for Load-Balancing in a Heterogeneous Grid System, *Sarnoff Symposium IEEE*, May 2007, pp 1- 5.

[11] L. W. McKnight, J. Howison, S. Bradner, "Wireless Grids: Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices*", IEEE Internet Computing*, 2004, Vol.8, No.4, pp.24-31.

[12] G. Li, Y.B. Han, J. Wang, Z.F. Zhao, R. M. Wagner, "Facilitating Dynamic Service Compositions by Adaptable Service Connectors", *International Journal of Web Services Research, Vol.* 3, No.1, 2006, pp.68-84.

[13] G. Li, H.M. Sun, "RESTful Dynamic Framework for Services in Mobile Wireless Networks", *Proceedings of 2009 International Conference on E-business and information System Security*, 2009, pp.156-160.

[14] G. Li, X.J. Ma, Y.B Han, J. Wang "Transprent service composition in dynamic networks", *Chinese Journal of Computers*, 2007, Vol.30, No.4, pp. 579-587

[16] M. Arora, S.K. Das, and R. Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. *In Workshop on Scheduling and Resource Management for ClusterComputing*, pages 499–505, 2002.

[17] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. On advantages of grid computing for parallel job scheduling*. In 2nd International Symposium on Cluster Computing and the Grid,* pages 39–46, 2002

[18] G.Coulouris, J.Dollimore, T.Kindberg, Distributed Systems concepts and Design 4th Edition, Addison Wesley, 2005.