Load Balancing in Cluster and Grid

This chapter deals with cluster and grid system load balancing problems. We have designed and implemented a MA assisted Load Balancing Algorithm (MALB) for I/O data intensive applications by considering three types of resources namely CPU, Memory and I/O. VF is designed, which is a combination of LI of CPU, memory and I/O disk resources. The performance of the developed algorithm is compared with two conventional CPU based load balancing (CLB) and Memory based load balancing (MLB) schemes. Rest of this chapter is organized as follows:

- Section 4.1 deals with load balancing issues in cluster and grid
- Section 4.2 gives problem formulation and assumptions, algorithm, and analysis
- Section 4.3 explores problem formulation and assumptions for multi level distributed tree algorithm, and analysis
- Section 4.4 implements the algorithms
- Section 4.5 gives result and discussion
- Section 4.6 summarizes the chapter

4.1 Issues

The existing load balancing schemes on cluster [8,12,13,14, 16, 17] are effective in maintaining high resource utilization, but these schemes suffer significant performance drop due to imbalance in disk I/O resources under I/O-intensive workload condition. In this situation, disk I/O resources
become a performance bottleneck, because performance gap between CPU and disk I/O is wide. But much of earlier work is either CPU specific or memory specific or combination of both. So much of the earlier approaches [16, 17] suffer a major performance drop for I/O applications under I/O workload on cluster.

Also there are applications which run in parallel mode demanding for resources which are available on distributed servers on grid. In accessing these resources, a large network delay occurs. These applications require updated load information for better utilization of the available resources without much network delay to improve the efficiency of grid. Therefore, the main issues regarding the load balancing on cluster and grid are - load balancing, resource allocation, resource utilization, response time, throughput, delay, and grid efficiency.

4.2 Load Balancing for I/O Data Intensive Applications on Cluster
Dynamic load balancing schemes on a cluster can improve system performance by assigning the load at run time to hosts with idle or under-utilized resources. Several distributed load balancing schemes have been presented in the literature, primarily considering CPU [8, 14], memory [10], or a combination of CPU and memory [12,163]. Although these load balancing schemes have been effective to increase the utilization of resources, but they have ignored one type of resource, namely disk I/O. The impact of disk I/O on overall system performance is becoming significant as more and more tasks with high I/O demand are running on cluster.

Therefore, we believe that for any dynamic load balancing scheme to be effective, it should be I/O load aware. Typical examples of I/O-intensive applications include multimedia and web based applications, remote sensing data etc. These applications share a common feature that their storage and computational requirements are extremely high. Therefore, the high
performance of I/O-intensive applications depends heavily on effective usage of global resources, in addition to that of CPU and memory.

4.2.1 Problem Formulation and Assumptions

Let the cluster consists of set of hosts \( N = \{N_1, N_2, \ldots, N_n\} \) that are heterogeneous in nature, LIA is the set of load index agent, LTA is the set of load transfer agent, RMA is the set of resource management agent, LQ is the set of local resources and GQ is the set of global resources with \( n \) tasks \( J = \{j_1, j_2, \ldots, j_n\} \) having task arrival rate \( \lambda \) and service rate \( \mu = \{\mu_1, \mu_2, \ldots, \mu_n\} \) competing for three resources namely CPU, Memory and Disk I/O. A task \( j \) is modeled as \( t_j, m_j, \lambda_j, \mu_j \) where \( t_j \) is the completion time required, \( m_j \) is the memory space required, \( \lambda \) is the task arrival rate, and \( \mu_j \) is the service rate.

We define LI for all the three resources as follows- LI of CPU is the remaining CPU lifetime of each task on a particular host. LI of memory is the sum of page fault processing time of all the tasks running in parallel mode \( LI^M = \sum_{j \in n} LI^M_j \)

Similarly LI for I/O is the combination of I/O requests produced by page faults and I/O requests generated from tasks resource from outside.

\[ LI^{IO} = \sum_{j \in n} page\_fault_j + \sum_{j \in n} IO\_load_j \]

The VF is the sum of LI of all the three resources i.e. \( VF = LI^{CPU} + LI^M + LI^{IO} \). This VF is used as a benchmark for load transfer by MA. We have assumed two types of resources in the system- dedicated and shared. The response time \( R_j \) of the task \( j \) is the fraction of time spent in
accessing the shared to dedicated resources, i.e., \( R_j = \frac{t_j^s + m_j^s + t_j^{1OS}}{t_j^d + m_j^d + t_j^{1Od}} \),

response time \( R \) for all tasks running in parallel mode is \( R = \sum_{j=1}^{n} R_j \). Thus it is required to design MALB to minimize the response time \( R \) of all tasks running in parallel, i.e., \( R = \min ( \sum_{j=1}^{n} R_j ) \).

### 4.2.2 Algorithm

The motivation for designing MALB is to reduce the average slowdown of all parallel tasks submitted to the cluster by balancing load of disk I/O resources. This, in turn, not only achieves the effective usage of global disk resources but also reduce response time of I/O data intensive parallel applications.

The MALB (Figure 4.1) consists of following four steps- initialization, calculation of VF, resource selection and allocation, agent communications and update of load status.

Step 1 is the initialization Step. In this step, we are considering a system of \( N \) hosts and agents of same type and two task queues- local and global.

Step 2 starts with calling of LIA which calculates the LI of each resource at each host for three defined resources - CPU, memory, and I/O. The sum of LI of each resource is the VF of that host. The response time of each task is calculated and the task with maximum response time is selected for transfer.

Step 3 starts with calling of RMA which calculates the resource requirement of each task in execution. If the resources are available in local queue then they are allocated to respective task otherwise global resources from global queue are to be allocated to the task in execution.

In Step 4, LI of each resource from the information provided by the respective agent is updated. For communication among agents, mobile group
Step 1: Initialization
Let set of hosts \( \text{LIA} = \{\text{LIA}_1,\text{LIA}_2,\ldots,\text{LIA}_n\} \) \( \text{LTA} = \{\text{LTA}_1,\text{LTA}_2,\ldots,\text{LTA}_n\} \)
\( \text{RMA} = \{\text{RMA}_1,\text{RMA}_2,\ldots,\text{RMA}_n\} \), \( \text{LQ} = \{\text{LQ}_1,\text{LQ}_2,\ldots,\text{LQ}_n\} \), \( \text{GQ} = \{\text{GQ}_1,\text{GQ}_2,\ldots,\text{GQ}_n\} \)
\( J = \{j_1,j_2,\ldots,j_n\} \)

Step 2: Calculation of VF
2. Call \( \text{LIA} \) from \( \bigcup_{i=1}^{n} \text{LIA}_i \).
3. Calculate \( \sum_{i=1}^{n} R_i \), where \( R_i = \{CPU_i, M_i, IO_i\} \), \( \forall \ i \in N \).
4. Calculate \( VF_i = \{LI_i^{CPU} + LI_i^{M} + LI_i^{IO}\} \), \( \forall \ i \in N \).
5. Calculate response time \( R_j \) of task \( j = \{j_1, j_2,\ldots,j_n\} \) for \( N_i \).
6. Select the tasks \( j_k \) from \( \bigcup_{i=1}^{n} \text{LQ}_i \) such that \( \sum_{j=1}^{n} R_j^k \) is maximum.

Step 3: Resource Selection and Allocation (Locally and Globally)
7. Call \( \text{RMA} \) from \( \bigcup_{i=1}^{n} \text{RMA}_i \) for task \( j_k \).
8. Select the resources task \( j_k \) demands.
9. If \( \xi^k_i \in \text{LQ}_i \), then \( \text{RMA} \) allocate the local resources to \( j_k \).
   Else
   Call \( \text{LTA} \) from \( \bigcup_{i=1}^{n} \text{LTA}_i \) for task \( j_k \), i.e., \( \xi^k_i \in \text{GQ}_i \), and allocate them to \( j_k \).
10. Execute the task \( j_k \).

Step 4: Agent Communication and Update of Load Status
11. Update load status for each \( i \in N \). Let \( l \) denotes the set of all possible locations (local and global). Let \( P \) be the set of all possible MAs \( \{MA_1, MA_2, MA_3,\ldots,MA_n\} \). These MAs are specific to the network selection. A mobile group is denoted by the set of agents \( g \)
\( = \sum_{i=1}^{k} MA_i \), \( g \subseteq P \). The following operations are used by MAs for coordination and communication:
   - join \( (g) \): issued by an agent, when it wants to join group \( g \).
   - leave \( (g) \): issued by an agent, when it wants to leave group \( g \).
   - move \( (g, l) \): issued when an agent wants to move from its current location-to-location \( l \).
   - send \( (g, m) \): issued by an agent when it wants to multicast a message \( m \) to the members of group \( g \).
   - receive \( (g, m) \): issued by an agent to receive a message \( m \) multicast from the group \( g \).

Figure 4.1: Pseudo Code of MALB Algorithm on Cluster
approach [161] is used, which has above defined five operations to be used by MAs for communication.

MALB algorithm is developed specifically for I/O data intensive applications. To execute the MALB algorithm, the following agents and policies are selected- LIA, LTA, RMA, and Information Gathering Policy, and Initiation Policy respectively.

The main aim is to achieve the effective usage of global disk resources in cluster. This, in turn, minimizes the average slowdown of all parallel tasks running on cluster and reduces the average response time of the tasks. In addition to balance load of disk resources under I/O-intensive workload, the MALB improves the CPU and memory utilization under CPU- and memory-intensive workload. Consequently, MALB is able to maintain the same level of performance as two existing algorithms CLB [16] and MLB [17]. The performance improvements of MALB are due to the fact that CLB and MLB do not address the issue of balancing disk I/O load under I/O intensive workload conditions while MALB does.

4.2.3 Implementation
We have implemented the above defined MALB algorithm on 100 Mbps switched LAN that connects 5 networks domains each having 32 Pc’s and workstation. Hosts are grouped into 4 networks domain with their own server and each server is connected to main domain manager server. The AS host and AH have 256MB RAM while server host has 512 MB RAM. Cluster is implemented on group of PCs (P-4, 3GHz, 256MB RAM) using PMADE and J2sdk1.5.1. A multitasking Windows NT operating system is used. All Pc’s are P4, 3 GHz, 256MB RAM running on windows and Linux operating System.

A comparison between the average response time on the cluster when applying the load balancing using MA approach and the average response
time using traditional MPI has been considered. As mentioned above hosts for load transfer are selected by MA using VF. As the number of tasks increases average response time of the cluster decreases in MA approach as compared to MPI.

In Figure 4.2, we have considered data intensive application where the impact of task arrival rate on task service rate has been shown for parallel applications. Disk request arrival rate is varied from 0.8 to 1.2 no. /ms; disk request size is chosen randomly; memory reference rate is fixed to 0.8 no. /ms; number of agents are fixed to three. As soon as I/O request increases the task service rate decreases in all three approaches. But results obtained show that MALB algorithm is superior to CLB and MLB load balancing schemes. This is because the existing load balancing schemes are inadequate for I/O intensive workload conditions. CLB and MLB do not consider the issue of I/O workload for I/O data Intensive workload conditions.

In Figure 4.3, we have considered memory intensive application and shown the impact of memory reference rate on response time. Disk arrival rate is fixed to 0.02 no./ms; memory reference rate varies from 2.4 to 3.3 no./ms; number of agents are fixed to three. Figure 4.3 shows that MLB and MALB outperforms CLB under memory intensive load. This is because MLB and MALB are concerned with global memory usage in cluster by balancing the memory resources.
Figure 4.2: Task Arrival and Service Rate in CLB, MLB, and MALB approach

Figure 4.3: Comparison of Memory Reference and Response time in CLB, MLB, and MALB approach
Figure 4.4 shows the impact of multi agents on the service rate. Task arrival rate and memory reference rate is set to fixed value and only the LIAs are varied. Results show that as the number of LIA increases the response time decreases. This is because LIA takes less time for resource selection and allocation.

Figure 4.5 shows MALB approach and the case without load balancing. When MALB approach is applied response time reduces to a great extent as compared to the case without load balancing. This is due to fact that MAs takes less time to transfer the tasks to remote location and also it uses VF for transfer of tasks waiting in queue. Resource allocation and selection is done by MAs efficiently using MALB.

![Graph showing impact of number of agents on Task Service Rate](image)

**Figure 4.4:** Impact of number of agents on Task Service Rate
Figure 4.5: Comparison of MALB and the case without Load Balancing

4.2.4 Analysis
We qualitatively compared MALB with two conventional CLB and MLB algorithms. Results obtained show that the MALB algorithm is superior than the CLB and MLB algorithms w.r.t. time complexity.

Theorem 4.1: Given a cluster and a parallel application submitted to cluster, time complexity of MALB algorithm is $O(nm)$, where $n$ is the number of hosts in the cluster and $m$ is the number of tasks in the application and values of $n$ and $m$ should be larger than 2. \{ Proof of this theorem is given in Appendix C \}

4.3 Load Balancing for Parallel Applications over Distributed Servers
There are applications which run in parallel mode and demand for the resources that are located on distributed servers as a part of grid. For these applications, we have developed multilevel hierarchical scheme to provide
the inherent locality and to choose the available resources from distant locations called multi level distributed tree algorithm (MLDTA).

4.3.1 Problem Formulation and Assumptions

We have assumed an open network of $N$ servers. Each server contains an unbounded length First in First out (FIFO) queue of waiting tasks. We have considered two serving queues as the processing hosts with the following load balancing mechanism- when the difference in population of two queues is greater than or equal to a threshold $c$, tasks are transferred from heavily loaded queue to the lightly loaded at a constant rate $\beta$ (i.e. transfer time is exponential of mean length $1/\beta$). Waiting tasks are allowed to migrate into different queues. Each server has mean service rate $\mu_i$ and request are coming to the server queue with a mean arrival rate $\lambda_i$. Requests can migrate with a mean transfer rate $\beta_i$ to other server. Initially, for the sake of simplicity, we have assumed the identical transfer rate $\beta$ for all the servers.

Variance of load distribution is expressed in terms of difference in the population of waiting tasks. Heterogeneity in load distribution is expressed in terms of different arrival rates and/or different service rate. We have assumed that whenever the load balancing mechanism is initiated to transfer the load from heavily loaded host to lightly loaded, tasks of small size (e.g. packets) that are comparable to $\beta$ are moved. So if $\lambda_i$ and $\lambda_2$ are the average Poisson arrival rate for the two queues and $\mu_1$ and $\mu_2$ are the average service rate for the two servers, and $\beta_1(n_1,n_2)$ and $\beta_2(n_1,n_2)$ are the rates with which tasks are transferred from one queue to another when the difference in the population of two queues exceeds $c$. Then

$$\beta_1(n_1,n_2) = \beta \text{ if } n_1 - n_2 \geq c$$

$$= 0 \text{ otherwise}$$

$$\beta_2(n_1,n_2) = \beta \text{ if } n_1 - n_2 \geq c$$

$$= 0 \text{ otherwise}$$
4.3.2 Multi Level Distributed Tree Algorithm (MLDTA)

To address the problem of load balancing in System of N servers, MLDTA is executed by MAs. The MLDTA creates a tree structure over the actual network. In this tree, the leaves (level 0 hosts) have a 1–1 correspondence to the servers of actual network. At each level of the tree the pairs of conjugate hosts (hosts with the same father) exchange their load {Figure 4.6}, when $\lambda_i \geq \mu_i$. Also $\delta_i$ is the ratio of task arrival and service rate at each level.

![Tree Diagram]

**Figure 4.6:** A tree arrangement of servers and hosts in Grid

One main characteristic of the MLDTA is multilevel construction in contrast to the two level DLB algorithm [162]. The two-level structure is suitable for large distance transitive networks, but MLDTA creates a scheme applicable to a wide variety of topologies. This multilevel construction can help in modeling efficiently the inherent locality of real distributed systems. Furthermore, this algorithm is suitable to heterogeneity, i.e., the case where load varies significantly among servers.

The MLDTA has two steps {Figure 4.7} - In the first Step, actual binary tree of the servers and hosts is created. The ratio of task arrival and service rate is calculated. Based upon this ratio servers are sorted at each level of the tree. Then based upon the load on each server at each level one of the servers is treated as leader at that level. In the second Step, load is balanced among
the conjugate hosts, i.e., hosts with same father in the binary tree. A predefined threshold $c$ is set for each level. As soon as the difference in load at two conjugate hosts exceed $c$, exactly $t_k$ tasks are transferred at a constant rate $\beta$ from heavily loaded host to lightly loaded one using LTA. This LTA executes S-I policy.

In general, LTA use local information to check the balancing condition (transfer policy), and remote information to identify the server to transfer a task (location policy). This means that messages are exchanged only when the load balancing condition is satisfied. The characteristics of algorithm are dynamic and symmetrical behavior, accurate knowledge of the load conditions, limited exchange of messages and scalability, adaptive, and distributed. It is a multilevel hierarchical scheme which considers the idea of multilevel hierarchical constructions creating a binary tree structure over the actual network topology. It introduces the basic concept of conjugate hosts in multiple levels of the tree. The application of the algorithm is more effective for network topologies where the communication cost can be assumed to be the same for all the servers in the network.
**Initialization Step**

1. Construct a binary tree with ancestors and descendants of each server, where servers of the network are at level 0 of tree and hosts are from level 1 to $\log_2 N$ are subsets of leaves.
2. While ($K \neq \log_2 N$), $K = 1, 2, \ldots, \log_2 N$ are the levels of tree.
3. Let $N = \{N_0, N_1, \ldots, N_{n-1}\}$ are the number of servers and $\lambda_i$ is the task arrival rate and $\mu_i$ is the task service rate.
4. Calculate $\delta_i = \frac{\lambda_i}{\mu_i}, i = 0, 1, 2, \ldots, n-1$
5. Sort the servers in increasing order of their workload at level 0 as follows
   $$\delta_0 \leq \delta_1 \leq \delta_2 \leq \ldots \leq \delta_{n-1}$$
6. Select the servers with lightest and heaviest workload and remove from the list.
7. Repeat Step 6 until list becomes empty.
8. Choose the leader at this level in terms of heaviest and lightest workload.
9. Calculate the sum of workload of leader at each level.
End While

**Load Balancing Step**

10. if ($\lambda_i \leq \mu_i$) then the system is balanced else calculate $\delta_i$
11. While ($K \neq \log_2 N$), $K = 1, 2, \ldots, \log_2 N$ are the levels of tree.
12. if ($\delta_i \geq \tau_i$), where $\tau_i = 2^{K+1}$ is the predefined threshold value of messages transferred at each level which varies according to level of the tree. i) Then call $LTA$ from
    $$\bigcup_{i=1}^{n} LTA_i$$ for tasks $t_k$, where $t_k \leq \frac{\beta}{\text{size}}$, $\beta$ is the task transfer rate. ii) Execute S-I policy.
    iii) Transfer $t_k$ from heavily loaded host to light one.

   Else  i) Call $RMA$ from $\bigcup_{i=1}^{n} RMA_i$ for tasks $t_k$. ii) Execute R-I policy.

End While

**Figure 4.7**: Pseudo Code for MLDTA for Load Balancing on Grid
4.3.3 Implementation
We evaluated the quality of load balancing achieved by MLDTA theoretically and by using simulation. In the simulations, we have considered the three factors- impact of homogeneous and heterogeneous arrival streams, and message processing cost.

(a) Impact of Homogeneous Arrival stream
Figure 4.8{(a)-(d)} shows the impact of homogeneous arrival stream with different arrival rate on the response time. For different values of task arrival rate, MLDTA has less response time compared to DLB algorithm under medium and heavy load conditions and case of no load balancing. Also the performance of MLDTA is not significantly affected by increasing the size of the network. MLDTA is effective to enable the activation of load balancing mechanism asynchronously at multiple levels of the binary tree.

![Graph showing impact of homogeneous arrival stream](image)

\[ \lambda = 0.5, \mu = 2.0, \beta = 20.0 \]
(b)

(c)

\( \lambda = 1.0, \mu = 2.0, \beta = 20.0 \)
Figure 4.8{(a)-(d)}: Response Time for homogeneous arrival rate with \( \lambda = \{0.5 \text{–} 1.8\} \) for DLB, MLDTA and no load balancing

(b) Impact of Heterogeneous Arrival Stream

Figure 4.9{(a)-(d)} compares the impact of heterogeneous arrival stream on the response time. MLDTA succeeds to get better response time compared to DLB algorithm and no load balancing case. Although it exchanges more messages but still it is able to balance the load more effectively as shown by the variance of load. MLDTA achieves better response time, since it takes most of load balancing decisions at various levels of the binary tree, making the best load redistribution at each level.
(a) Variance of Load vs. Response Time (sec.)

(b) Variance of Number of Tasks vs. Load Balancing

Graph (a) shows the response time (sec.) against the variance of load for different load balancing methods: DLB, No Load Balancing, and MLDTA. The graph is labeled with the average values of λ, μ, and β.

Graph (b) illustrates the variance of the number of tasks against the variance of load for the same load balancing methods. The graph also includes the average values of λ, μ, and β.
Figure 4.9{(a)-(d)}: Results for heterogeneous arrival rate for DLB, MLDTA and no load balancing
(c) Impact of Message processing cost

Figure 4.10 {(a)-(d)} shows the impact of message processing cost on response time, messages per task, system utilization and % of task moved. MLDTA has a better mean response time when the values of message processing time are low. Also MLDTA has better utilization of resources compared to other algorithms, although it transfers more tasks, and exchanged more messages than other algorithms.

All these results prove that the aim of load balance algorithm succeeds, as it makes the right number of load balancing decisions in most of the cases.
Message Processing Cost

![Graph showing Message Processing Cost](b)

Utilization

![Graph showing Utilization](c)

\[ \text{Avg} \lambda = 1.5, \mu = 2.0, \beta = 20.0, N = 32 \]
Figure 4.10{(a)-(d)}: Results for different Message Processing Cost with $Avg \lambda = 1.5$

Figure 4.11 compares the task migration policies S-I, R-I, Sy-I and Central. S-I policy improves the average response time (seconds) significantly under varying system load. Under light load all policies are almost identical in terms of response time, but the key difference is under heavy load where S-I policy outperforms in comparisons of R-I and central.

In Figure 4.12 there is almost no change in grid efficiency for lightly loaded case. But for heavily loaded case grid efficiency is increased to 82% using S-I compared to 70% and 75% using R-I and Sy-I policy, respectively. Even though there is a little change in grid efficiency for light load but individual site utilization depends upon specific task migration scheme and the algorithm used for task migration.
**Figure 4.11**: Average Response Time under varying network Load

**Figure 4.12**: Grid Efficiency under varying Load


4.3.4 Analysis
We have investigated the quality of developed algorithm by establishing the maximum expected difference between the number of tasks at any two servers and the number of exchanged messages and is compared with DLB algorithm [162]. The theoretical results show that MLDTA achieves a very good distribution of load in the system, without exchanging many more messages than other efficient algorithms. We have evaluated various performance indicators for the algorithm, e.g., average response time and resource utilization. The results obtained show that MLDTA reduces the average response time for a task, especially under medium and heavy load conditions and uses less resources. Furthermore, the performance of MLDTA is not significantly affected by increasing the number of tasks in the system.

Theorem 4.2: Consider a network of \( N \) servers with same mean arrival \( \lambda \) and service time \( \mu \) at each server, \( N \) is a power of 2, then average number of messages that would be sent after the occurrence of arrival or departure event is \( \log_2 N/2 \). \{ Proof of this theorem is given in Appendix C \}

4.4 Result and Discussion
For load balancing on cluster, we have considered I/O data intensive application running in parallel mode and designed MALB algorithm for load balancing. This algorithm consists of defining VF which is combination of three different resources namely-CPU, memory and I/O resources. This VF is set as a bench mark to select or reject a particular host for load balancing. Results obtained show that MALB algorithm is superior than CLB and MLB algorithms with respect to selected metrics.

For load balancing on grid, we have considered applications demanding for resources located on distributed servers. For these applications, we have used MLDTA. In this algorithm, servers and hosts are arranged in a binary tree structure in hierarchical form. MLDTA reduces the average response
time for a task, especially under medium and heavy load conditions and uses fewer resources. The average number of messages that would be sent after the occurrence of arrival or departure of an event is \( \log_2 N/2 \), where \( N \) is number of servers in the network with same mean arrival \( \lambda \) and service time \( \mu \) at each server, \( N \) is a power of 2. The performance of MLDTA is better than the two level DLB algorithm [162] with respect to metric chosen. It is found that the designed algorithm is able to balance the load across wide variety of topologies and is able to deal with heterogeneous arrival.

4.5 Summary

In this chapter, we have tested SFLBMS for load balancing on cluster and grid. On cluster, I/O data intensive applications are implemented in the form of MALB algorithm. In this algorithm, two types of resources namely local and global are considered and VF is used for load transfer by the respective agent. The performance of MALB is compared with existing algorithms with respect to selected metrics. The performance of MALB is found better than the existing algorithms with respect to chosen metrics. On grid, applications running parallel on distributed servers are considered. MLDTA is tested on the system with respect to message complexity. MLDTA behaves slightly better than the existing algorithms under heterogeneous environment, and has low message processing time. Also the grid efficiency improves using MLDTA.

In the next chapter, we will present the load balancing in P2P networks.