Journal of Computer Science 3 (1): 14-24, 2007
ISSN 1549-3636

A Framework for Distributed Dynamic Load Balancing in Heterogeneous Cluster

[1]Neeraj Nehra, [2]R.B. Patel, [3]V.K. Bhat
[1] School of Computer Science and Engineering, Shri Mata Vaishno Devi University,Katra(J&K), India
[2]Computer Engineering Department, M.M. Engineering College, Mullana(Ambala), Haryana, India
[3] School of Applied Physics and Mathematics, Shri Mata Vaishno Devi University, Katra(J&K), India

Abstract: Distributed Dynamic load balancing (DDLB) is an important system function destined to distribute workload among available processors to improve throughput and/or execution times of parallel computer in Cluster Computing. Instead of balancing the load in cluster by process migration, or by moving an entire process to a less loaded computer, we make an attempt to balance load by splitting processes into separate jobs and then balance them to nodes. In order to get target, we use mobile agent (MA) to distribute load among nodes in a cluster. In this study, a multi-agent framework for load balancing in heterogeneous cluster is given. Total load on node is calculated using queue length which is measured as the total number of processes in queue. We introduce types of agents along with policies needed to meet the requirements of the proposed load-balancing. Different metrics are used to compare load balancing mechanism with the existing message passing technology. The experiment is carried out on cluster of PC's divided into multiple LAN's using PMADE (Platform for Mobile agent distribution and execution). Preliminary experimental results demonstrated that the proposed framework is effective than the existing ones.

Key words: Dynamic load balancing, distributed systems, mobile agent, queue length, resource management

INTRODUCTION

Load balancing is an efficient strategy to improve throughput or speed up execution of the set of jobs while maintaining high processor utilization. Basically Load balancing is the allocation of the workload among a set of co-operating nodes. The demand for high performance computing continues to increase everyday. Load balancing strategies fall broadly into either one of two classes static or dynamic. A multi-computer system with static load balancing distributes tasks across nodes before execution using a priori known task information and the load distribution remains unchanged at run time. A multi-computer system with Dynamic Load balancing (DLB) uses no priori task information and satisfies changing requirements by making task distribution decisions during run-time. DLB in turn can be either centralized or distributed. Distributed Load balancing[1,2] is an active technology that provides the art of shaping, transforming and filtering the network traffic and then routing and load balancing it to optimal server node. By adding the concept of load balancer we can distribute the traffic for preventing from failure in any case by having capabilities such as scalability, availability, easy to use, fault tolerance and quick response time.

The computational need in areas like cosmology, molecular biology, nano-materials, etc., cannot be met even by the fastest computers available But with the availability of high speed networks, a large number of geographically distributed nodes can be interconnected and effectively utilized in order to achieve performances not ordinarily attainable on a single computing environment (CE). The distributed nature of this type of CE calls for consideration of heterogeneities in computational and communication resources. A common architecture is the cluster of otherwise independent nodes communicating through a shared network. An incoming workload has to be efficiently allocated to these nodes so that no single node is overburdened, while one or more other nodes remain idle. Further, tasks migration from high to low traffic area in a network may alleviate to some extent the network traffic congestion control problem. Workstation clusters are being recognized as the most promising computing resource of the near future.

A large-size cluster, consisting of locally connected workstations, has power comparable to a supercomputer, at a fraction of the cost. Furthermore, a wide-area coupling of workstation clusters is not only suitable for exchange of mail and news or the establishment of distributed information systems, but can also be exploited as a large meta-computer. Distributing the total computational load across available processors is referred to in the literature as load balancing. Effective load balancing of a cluster of

Corresponding Author: Neeraj Nehra, School of Computer Science and Engineering, Shri Mata Vaishno Devi University, Katra (J&K)

Nodes in a distributed computing system relies on accurate knowledge of the state of the individual Nodes. This knowledge is used to judiciously assign incoming computational tasks to appropriate Nodes, according to some load-balancing policy. In large-scale distributed computing systems in which the Nodes are physically or virtually distant from each other, there are a number of inherent time-delay factors that can seriously alter the expected performance of load-balancing policies that do not account for such delays.

One manifestation of such time delay is attributable to the computational limitations of individual Nodes. A more significant manifestation of such delay arises from the communication limitations between the Nodes. These include delays in transferring loads amongst Nodes and delays in the communication between them. Moreover, such delays not only fluctuate within each PE, as the amounts of the loads to be transferred vary, but also vary as a result of the uncertainties in the communication medium that connects the units. This kind of delay-uncertainty is frequently observed in systems for which the individual units are connected by means of a shared communication medium (e.g., the Internet, ATM, ad-hoc networks, wireless LANs).

Mobile agent (MA) [3] technology provides a new solution to support load balancing of this type. This approach consists of a number of different types of MAs in a cooperative way to fulfill the task of load balancing instead of single centralized component managing all load-balancing activities. Each type of agent implements one of the predefined policies of load balancing. Moreover, the MA paradigm supports the disruptive nature of wireless links and alleviates its associated bandwidth limitations.

We will use the concept of MA because MA technology offers a new computing paradigm in which an autonomous program can migrate under its own or host control from one node to another in a heterogeneous network. In other words, the program running at a host can suspend its execution at an arbitrary point, transfer itself to another host (or request the host to transfer it to its next destination) and resume execution from the point of suspension is called MA[6]. The migration of MA is associated with different movement costs viz, transmission time, round trip time, number of hops, etc. MA research evolved over the past few years from the creation of many monolithic MA systems (MASs), often with similar characteristics and built by research groups spread all over the world for optimization and better understanding of specific agent issues[3,4]. To improve the performance of MAs means to optimize their paths on the network. Furtermore, the agent uses a path through a network based upon known infrastructure characteristics (QoS). An agent optimizes its transmission between Agent hosts (AHs)[4] with the help of several migration strategies described in[5].

MA supports a variety of web based distributed applications namely: systems and distributed information management[7] and information retrieval[8]. Other areas where MAs are seen as offering potential advantages- wireless or mobile computing[3,4] dynamic deployment of code, thin clients or resource limited devices, personal assistants and MA-based parallel processing[5,9]. The idea of using MA in load balancing has been floating around for sometime in homogeneous telecommunication networks because traditional load balancing approaches are implemented based on message passing paradigm[1,10]. In message-passing based approaches, the nodes have to exchange messages of load information periodically in order to make decisions on load balancing. The mod_backhand[11] is such a load balancing module for the Apache web server. The message exchanges result in high communication latency and thus deteriorate the performance of the system. Differently, a MA can migrate to its target and interact to specified objects on the site. Moreover, a MA based approach is flexible to incorporate new load balancing polices for various systems. MAs produce low network traffic.

On each machine, the agent interacts with stationary service agents and other resources to accomplish its task[12]. Developments in wireless technology liberate network nodes from the constraint of being placed at a fixed physical location and enable the advent of the so-called mobile computing. The proliferation of mobile computing devices, which have the characteristics of low bandwidth and unreliable network connection, has lead to the increased use of MA since it supports disconnected operations[13]. MA paradigm provides a better conservation of bandwidth since only the final result returns back to the client. When the server lacks one of the services, the MA migrates to the server and performs the set of required operations locally. This, in turn, leads to reduction in total completion time. A MA provides an effective means for overcoming network latency, it monitors the network latencies and continually moves to the network location that minimizes the average latency between itself and its clients.

In this study we therefore proposed a framework consisting of distributed dynamic laod balancing (DDLB) strategies for minimizing the average completion time of applications running in parallel and improve the utilization of the nodes. We define the structure of each agent along with function of each layer for coordination among agents. Approach is distributed in the sense that each node has a task to be performed during overloaded situation.

Overview of PMADE: Figure 1 shows the basic block diagram of PMADE (Platform for Mobile Agent

Distribution and Execution). Each node of the network has an Agent Host (AH), which is responsible for accepting and executing incoming autonomous Java agents and an Agent Submitter (AS)[12], which submits the MA on behalf of the user to the AH. A user, who wants to perform a task, submits the MA designed to perform that task, to the AS on the user system. The AS then tries to establish a connection with the specified AH, where the user already holds an account. If the connection is established, the AS submits the MA to it and then goes offline. The AH examines the nature of the received agent and executes it. The execution of the agent depends on its nature and state. The agent can be transferred from one AH to another whenever required. On completion of execution, the agent submits its results to the AH, which in turn stores the results until the remote AS retrieves them for the user.

The AH is the key component of PMADE. It consists of the manager modules and the Host Driver. The Host Driver lies at the base of the PMADE architecture and the manager modules reside above it. It is the basic utility module responsible for driving the AH by ensuring proper co-ordination between various managers and making them work in tandem. Details of the managers and their functions are provided in[12]. PMADE provides weak mobility to its agents and allows one-hop, two-hop and multi-hop agents[12]. PMADE has focused on Flexibility, Persistence, Security, Collaboration and Reliability[13].

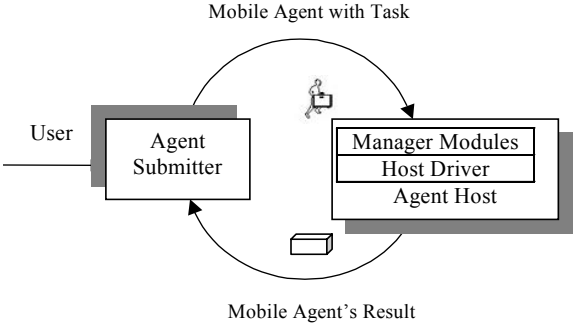

Fig. 1: Block architecture of PMADE

System architecture for load balancing: The various components for load balancing are arranged hierarchically as shown in Fig. 2. Load request comes from any node in Cluster whenever its load goes above or below threshold value. The various agents in the framework have a role to play.

Agent pool: It consists of various agents each having its own role. Te agents are:

* Job Scheduler Agent (JSA): Its main function is to act as a middleware. Whenever a request for load comes from any node in cluster, JSA passes the request to execution environment, which is PMADE execution engine. The request may be of information about system resources, load information and number of processes currently running etc.
* Task management: Task management is handled by Task Management Agent (TMA). Requests are passed to the task management module where they are queue for scheduling and execution. Each task is given a unique identification number and awaits the attention of the JSA scheduler. Task management also interfaces with the operations on the task queue, including adding, deleting, or inserting tasks.
* Resource management: The resources are managed by Resource Management Agent(RMA). The resource management is responsible for gathering information concerning the process nodes on which tasks may execute and pass this information to JSA. There is a proper coordination among the MA for information exchange using mobile group approach[14]. This information includes availability, load average and idle time. Resource management is also responsible for organizing the JSA scheduling and *Task execution*.

Policy selection: This section discusses the policies defined in the framework to be executed by the agents defined in agent Pool

* Information Gathering Policy specifies the strategy for the collection of load information including the frequency and method of information gathering. The frequency is determined based on a tradeoff between the accuracy of load information and the overhead of information collection.
* Initiation Policy determines who starts the load balancing process. The process can be initiated by an overloaded server (called sender-initiated) or by an under-loaded server (called receiver-initiated). Sender initiated policies are those where heavily loaded nodes search for lightly loaded nodes while receiver initiated policies are those where lightly loaded nodes search for suitable senders.

Job Transfer Policy determines when job reallocation should be performed and which job(s) should be reallocated. Job reallocation is activated by a threshold-based strategy. In a sender-initiated method, the job transfer is invoked when the workload on a node exceeds a threshold. In a receiver-initiated method, a node starts the process to fetch jobs from other nodes

when its workload is below a threshold. The threshold can be a pre-

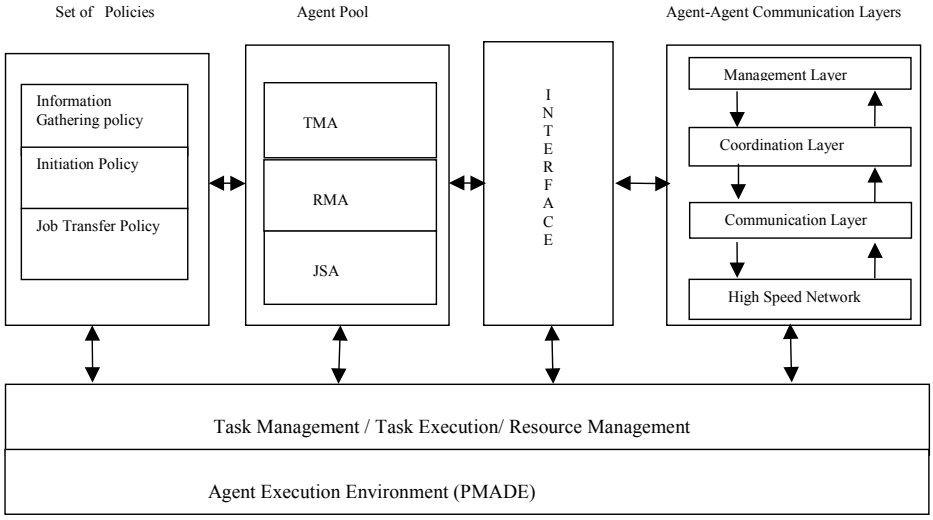

Fig. 2: System architecture for load balancing along with various components

defined static value or a dynamic value that is assessed at runtime based on the load distribution among the nodes. When job reallocation is required, the appropriate job(s) will be selected from the job queue and transferred to another node

Inter-agent communications: The framework for load balancing consisting of multi-agent with each agent has a specific role to play and have facility for inter agent communication as shown in Fig. 2. Each agent is implemented for managing hosts processors of a Cluster resource and scheduling incoming tasks to achieve load balancing. The functions of various layers are:

* Communication and Coordination Layers. Agents in the system communicate with each other or with users using mobile group approach for coordination of MA. The request an agent receives from the communication layer should be explained and submitted to the coordination layer, which decides how the agent should act on the request according to its own knowledge. We assumed a distributed system as a collection of agents, locations and communication channels. A location represents a logical place in the distributed environment where agents execute. When a MA migrates, it moves from a location to another. Agents communicate by exchanging messages through reliable communications channels, i.e., transmitted messages are received uncorrupted and in the sequential sent order, as long as the message sender does not crash until the message is received (reliable channels can be implemented over unreliable channels by tagging transmitted messages with sequential numbers, delivering such messages according to the sequential order and asking for retransmission in case of missing messages). As implied by reliable channel assumption, we assume that network partitions do not occur or, when they occur, they are repaired within a finite amount of time and communication reestablished. No bounds on message transmission or relative agent execution times are assumed. Agents and locations are assumed to fail only by crashing (without producing any further action) and the agents of a faulty location are assumed to have crashed. The failure of a given location is not directly handled. Instead, it is only detected when the associated agents are detected faulty. An agent that never crashes is named correct. Let L denote the set of all possible locations. Let P be the set of all possible agents. A mobile group is denoted by the set of agents g = {p1, p2, …, pn }, g$\subset$ P. On a mobile group, five operations are defined:
* *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*.

An agent pi of a group g also installs views, named $v_i(g)$. In mobile groups a view $v_i(g)$, $v_i(g) \subset \{(p, l) \mid p \in g \text{ and } l \in L\}$, is a mapping between agents of group g and locations l. A view represents the set of group members that are mutually considered operational in a given instant of the group existence and indicates the

locations where these members are, (a pair (p, l) in a view indicates that agent p is currently at location l). This set can change dynamically on the occurrence of agent crashes (suspicions) or when agents deliberately leave, join, or move to another location[14]. In this way these agents communicate with each other using mobile group communication defined above for updated information about all the system resources and other valuable information.

- Management Layer: This layer is responsible for submitting local service information to the coordination layer for agent decision making. In a Cluster Computing environment, the composition of nodes is dynamic, every node is likely to enter a busy state at any time and thus lower its performance, so when selecting nodes for load sharing, CPU utilization cannot be the sole factor of consideration for load sharing among participating nodes. Other factors affecting the nodes are the node's past completion rate, possibility of the resource utilization, job queue length, memory utilization etc. Thus, a value function is proposed to evaluate the value of each node and provide reference for selecting nodes. In this value function, relative value of each resource including CPU memory, size of available memory, transmission rate, past completion time is treated as a decision variable in calculation of value function and we calculate the relative value by score of each variable, i.e., this value function is the benchmark to select or reject a particular node. In addition, to search for a node that demand most for load sharing, different weight values will be given to the nodes in accordance with the level of preference for the task, so as to select the nodes most suitable for the execution of the task. Therefore, the value function is shown as given by following equation

$$Z_i = \sum_{i=1}^{n} Wi\ f\ (x_{i,j}),\ 1 \le j \le N$$

$$= w_1 f(x_{1,j}) + w_2 f(x_{2,j}) + \ldots + w_n f(x_{n,j})$$

$$\text{Where} \sum_{i=1}^{n} Wi = w_1 + w_2 + w_3 + \ldots w_n = 1;$$

$$1 \le j \le n; 0 \le f(x_{i,j}) \le 1$$

Whereas

$F(x_ij)$ Score of decision variable i in node j.
Z_i The estimated value of node i.
i The decision variable in the value function and there are total n decision variables.
j The node j in cluster, there are n nodes in cluster.
wi The weight value of each decision variable.

Now based upon the above defined value function for each node an effective node is chosen for load sharing among the participating nodes. Score of decision variable defines how effective that resource is with respect to available resources[15].

Mechanism for load transfer between different nodes: For load transfer among different Nodes each node maintains its own list of participating nodes to which it wants to communicate for load sharing. Each node maintains its own job queue along with some predefined threshold values to initiate load transfer. Let t be the time when tasks were last executed and $a(t_j)$ be the arrival time of task t_j and $e(t_j)$ be time when it starts executing. Then the jobs in the queue are those being executed and ready to be executed are given by $\{ t_j / a(t_j) \le t \text{ t}, e(t_j) \le t \}$ and $\{ t_j / a(t_j) \ge t, e(t_j) \ge t \}$ as shown in Fig. 3. Detailed working of load sharing is as follows:

A. At the beginning of each time interval, each node calculates its load from previous interval. Let us call this quantity the difference in load (DL). Each node may calculate this quantity independently from other Nodes in the system. Second, the length of time interval may vary with time for a given node, depending; for instance, on the number of load requests received or network traffic. As a consequence different Nodes may use different intervals at any given time. It calculates the number of time intervals it will take to reach an idle state (no tasks to process). If the number of intervals (times its duration) is less than the network delay (ND), then the node will initiate a migration request.

   For the sake of ease, let us introduce the three thresholds upper threshold value (UTV), lower threshold value (LTV), critical threshold value (CTV). Former two are used to determine the load status of the processor. If a PE's load is greater or equal to HTV it is considered a Source PE. If on the other hand it is less or equal to LTV it is considered a sink PE. If its load lies between these two thresholds then the node is in a neutral state. If however a PE's load falls below a critical threshold (CTV) the node immediately initiates a request for load regardless of the predicted future load based upon the current DL value. The node responsible for initiation of load transfers and performs the following actions according to below defined algorithm.

Algorithm1 (Request by under loaded node for load Transfer)

1. Begin
2. Set number of tasks being requested to appropriate value.
3. Selects the source node from whom the message will be sent. Removes the entry associated with the selected node from its source table.

4. Changes its status to *Waiting*, preventing it from issuing another request before receiving a Reply.
5. Add id of the sender node to its source table after receiving reply.
6. End

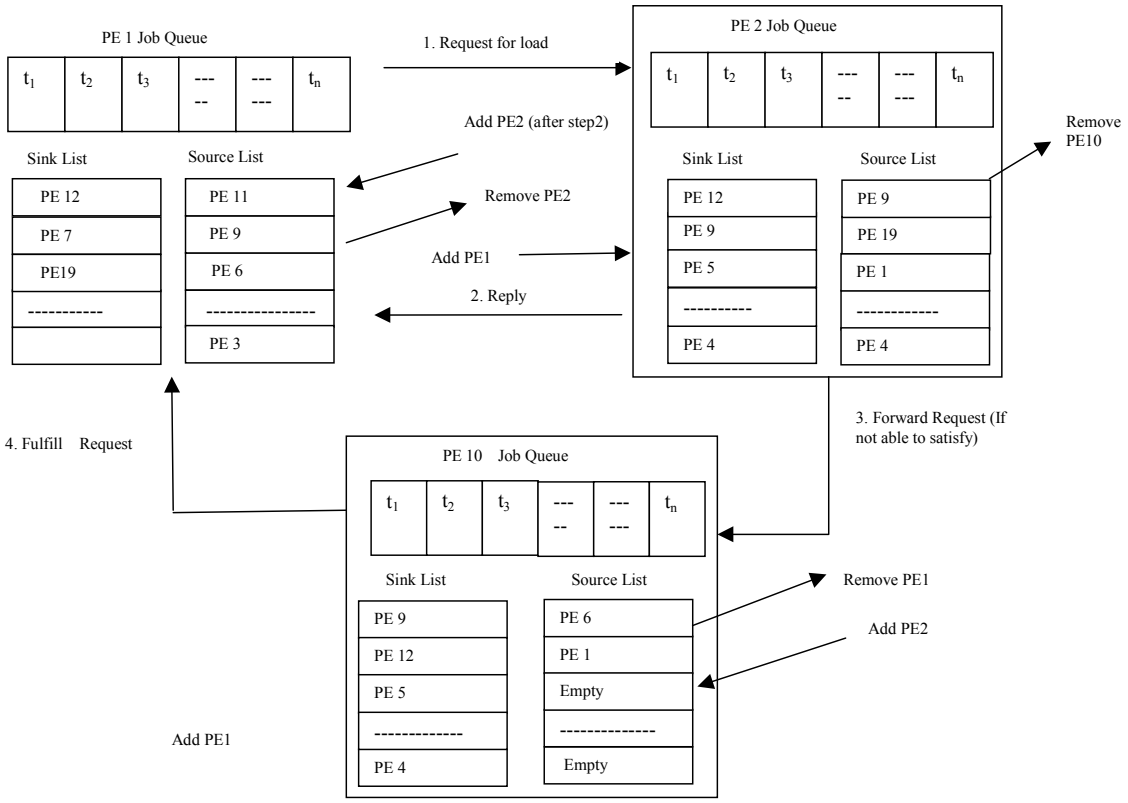

Fig. 3: Scenario of load request and transfer between different nodes

B. Each node keeps two local tables containing system load information. One contains information regarding the location of sink Nodes (under loaded nodes), called sink table shown in Fig. 3, the other of source Nodes (overloaded nodes), called source table. Any node that initiates a request for load is considered to be a sink by the receiving node(s). The sink node (request initiator) selects a source node from its source table (the first entry in the table) and sends a message, requesting for load transfer to it. Initially the table is empty since no information is available regarding the state of the node is known and therefore a node is chosen at random. It only means that when no information is known regarding the load of any node in the system then every node is as likely to be considered a source node as any other and therefore we chose one among all possible ones at random.

Algorithm 2 (Transfer of load from Destination PE)

1. Begin
2. Updates its own local tables by adding ID to its sink PE's table and removing it from the source PE's table if present.
3. Checks its own load status.
4. If it can completely fulfill the load transfer request it sends the load to corresponding node and removes the message from the system.
5. If it can only fulfill the load transfer request partially decrements amount of load transfer value by the amount of load units that is able to satisfy.
6. If it is not a source node or cannot completely fulfill the Load, transfer the request to appropriate node.

   End

C. At the beginning of each time interval, each node calculates its DL. Using DL it computes how many tasks it would have, assuming DL would stay constant, after a length of time equal to network delay (ND), let us call this quantity predicted load value (PLV). If PLV is greater or equal to zero then the node does not expect to become idle within the next ND period and therefore does not initiate a request for load transfer. If on the other hand PLV is lesser than zero the node requests load. On the receiver side, a node will only transfer tasks if its load is above the UTV level, in which case

it transfers tasks above this value up to the requested transfer amount. Each node maintains two local tables with information representing its view of the system's load distribution. Following parameters would be changed during transfer of load

D.

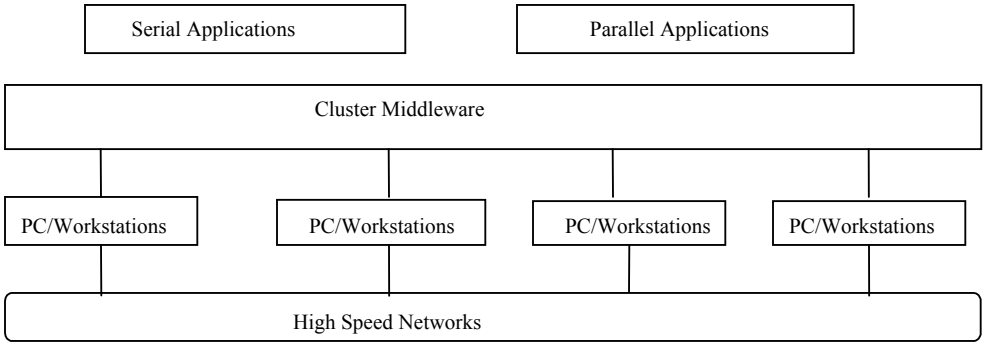

Fig. 4: Various components in typical cluster

* Number of load units requested.
* Each PE's local tables every time the message is forwarded.

D. Finally when the request for load is completed then the corresponding information is sent to original node making the request and it updates its tables accordingly.

Implementation: Following procedure in sequence is adopted while calculating the load on cluster.

* Define estimated load of each node with threshold level (by means of value function defined above) estimated threshold Upper and Lower value.
* The agents collects predictable load from the others.
* Repeat the following two steps repeatedly during a particular time interval, until no overloaded node exists.
* All nodes in a cluster exchange their current workload information using agents communication and coordination to elect the heaviest overload node.
* The heaviest loaded node dispatches the client agent to migrate workload to the node who has light workload.
* This process is repeated until the node's workload is below the estimated high threshold level.

Typical components of cluster are shown in Fig. 4. We have implemented the above defined load balancing scheme on 100 Mbps switched LAN that connects 10 networks each having 100 PC's and workstation. Nodes are grouped into 10 networks with their own server and each server is connected to main server. The AS node and AH node have 512 MB RAM. MA cluster is implemented on cluster 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.

As shown in Fig. 5, a comparison between the average response time of the cluster when applying the load balancing using MA approach and the average response time using traditional message passing approach (MPI). Nodes are selected to execute tasks by the value function. The value of each node is estimated with the value function and serves as the basis for task assignment. This method first divides the task into several independent subtasks and takes minimum resource demand of each node as the threshold value (Upper threshold value and Lower threshold value). After the value of each node is estimated with the value function, the nodes for the execution of the task are selected by the order of their values. In the value function, decision variables can be given a different setting according to the factor focused in the actual application. In the experiment, the available CPU capacity, size of available memory, transmission rate and the past completion rate were the four factors regarded as the threshold for the value function to select nodes and the decision variables for estimating node values. As shown in the Fig. 5, as the number of tasks increases average response time of the cluster decreases in MA approach as compared to traditional message passing approach.

Figure 6 shows a comparison between the variance of the load over the cluster in case of load balancing

using MA and variance of load in case of MPI. The Cluster environment is composed of heterogeneous systems, so the structures of each system may greatly vary, so this metric of variance becomes important. In other words, the computing capability provided by the

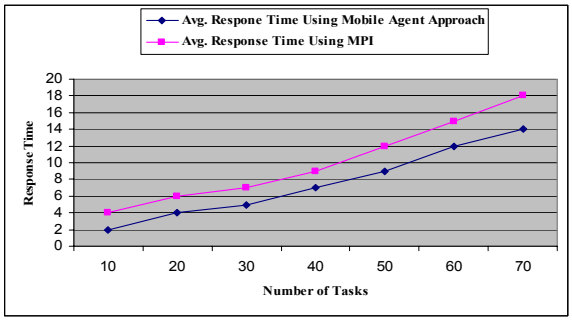

Fig. 5: Average response time of the cluster

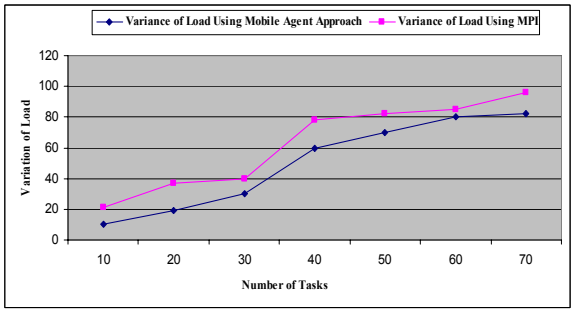

Fig. 6: The variance of the load over cluster

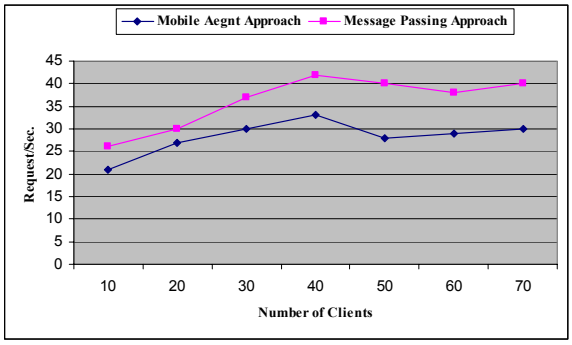

Fig. 7: System throughput

CPU and the available size of memory are different. In addition, Cluster Computing utilizes idle resources of each node, so the available resource of each node may vary in a busy condition. From the perspective of task completion time, the available CPU capacity and size of available memory are the two decisive factors for the duration of execution. Thus, in this experiments, the available CPU capacity and the size of available memory were taken as the threshold for value function and decision variables for estimating node values. The variance is measured for different workload of 100,200,400,500,600,700 tasks.

Figure 7 compares System Throughput of MA approach with traditional message passing approach. It is clear from Fig. 7 that MA approach is better than Message Passing approach in terms of system throughput. Also as the number of clients are increasing rapidly then system throughput decreases with message passing approach as compared with MA approach.

Figure 8 shows that load distribution has to be dynamically adjusted in accordance with variation of node status. The variation of the node status can be identified in two conditions; firstly, when the overloaded node receives the message that a certain node can no longer provide resources and secondly, when the execution of a certain node exceeds the expected time. When any of the above situations occurs and is detected then the agent is sent to collect the related data of all the nodes in the table of effective nodes and compare the collected data with historic ones, in order to confirm if the node is still effective so that load can be distribute to it. If the node remains effective, the distribution of the task will not be re-adjusted, but the time required for the node's execution of the task will be estimated again. If the node is confirmed ineffective, a node with the highest value will be selected from the waiting aggregate and the existing task will be transferred from the ineffective node to this new effective node.

Figure 9 shows that when demand for computing resource is large and amount of data transmission is small, the available CPU capacity memory usage provided by the node will obviously affect the completion of the task and the impact of transmission rate of the node on the task completion time is not significant. As in case of simple FIFO and FCFS applied to job queue, the task re-distribution and re-execution constantly occur because the selected nodes often cannot complete the task in effective time, thus prolonging the task completion time. In the aspect of value functions (VF), due to the considerations for the various factors of the nodes (such as the available CPU capacity, available size of memory, transmission rate and past task completion rate), nodes with a better performance are chosen and as shown in Fig. 9 task completion time is less using VF compared to CPU based approach.

Figure 10 compares the execution time of centralized and distributed strategies; it is clear from the figure that as the number of agents in distributed strategy increases execution time decreases.

It is clear from all these results with various parameters (defined above) that MA approach is far more better than traditional message passing approach and when applying this strategy in distributed manner, response time, system throughput and variance in load decreases as shown in above results.

Related work: Load balancing is indispensable for a group of cluster system to assure distribution of workload on each Cluster. But one of the most difficult problems that arise on Cluster system is the selection of

an efficient load balancing policy. The load balancing policy should aim for evenly utilized Cluster and a

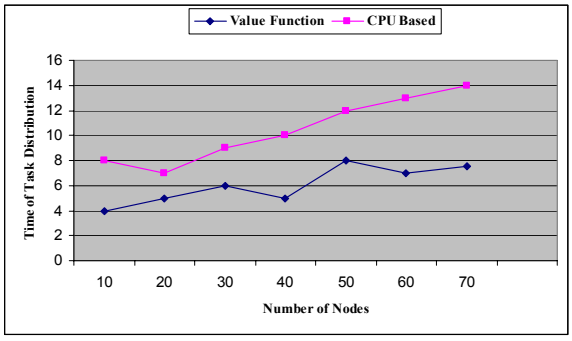

Fig. 8: Time taken for Load redistribution in Value function and CPU based approach

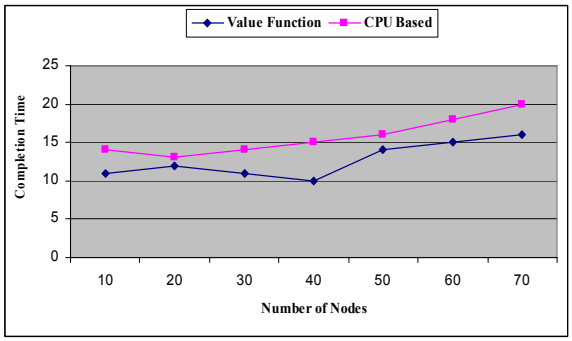

Fig. 9: Task Completion time using value function and CPU based approach using MA

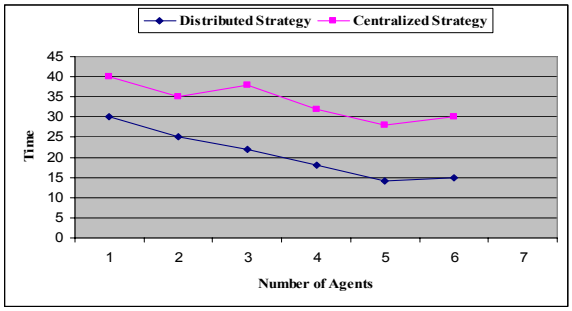

Fig. 10: Comparison of total execution time between centralized and distributed strategies

minimum response time for the processed requests. Under standard methodology load selection is done randomly. The random selection cannot guarantee load balancing. Round robin is widely used because it is easy to implement and implies only a minimum overhead. A variation of round robin policy is the weighted round robin policy[16]. With weighted round robin the incoming requests are distributed among the participating nodes on a round robin fashion, weighted by some measure of the load on each of the node.

Another techniques, which is called dispatching techniques which when implemented by network address translation or other methods (such as HTTP redirection), introduce higher overhead than does network load balancing. This limits throughput and restricts performance. SUNSCALAR[17] provides load balancing by using both approaches, i.e., Dispatcher and Round Robin.

When we consider load balancing in a system, there are four levels to apply: (1) hardware level; (2) system software level; (3) middleware software level; and (4) application software level. The hardware level load balancing is used in Layer 4 (L4) switches. Based on traffic distribution information or service-level checking information, the L4 switches perform server load balancing. The Alteon Web Switch with WebOS[21] and the CISCO Switches with Local Director[22] are the major commercial products. The system software level load balancing can be found in the Linux Virtual Server (LVS)[23,24]. LVS is an open software project to provide Linux OS-based load balancing. In Linux OS kernel level, LVS delivers NAT (Network Address Translation), IP Tunneling, Direct Routing schemes with several scheduling algorithms, such as Round-Robin (RR), Weighted RR (WRR), Least Connection (LC) and Weighted LC (WLC). The commercial products based on LVS load balancing are Turbo Cluster Server[25] and Red Hat HA Server[26].

Meanwhile, MS provides two load balancing solutions[27], the Network Load Balancing (NLB) and the Cluster Service, which can be used with only MS Windows. NLB distributes traffics on the network layer; the Cluster Service balances loads on the service layer. Those are employed in the MS Application Center and the Windows Data Center Server.

The middleware software level load balancing is used by WAS (Web Application Server) vendors. WAS delivers web traffic to application servers through a dispatcher or Web server Plug-in. The BEA Web Logic Cluster[28] and the IBM Web Sphere Edge Server[29] are the commercial products. The application software level load balancing is an approach that is used on most of application servers. For DNS (Domain Name Server), RR-DNS is used for balancing. For Web service, HTTP Redirection is used for Web browser to reconnect to other URLs. Also IP Redirection is used on other most of application software with their dedicated GUIs.

In addition to above defined system, we also have three well-known MA systems, namely Voyager[18], Aglets[19] and Concordia[20]. Which are used for different applications. A framework for load balancing using MA named EALBMA (Efficient and Adaptive Load Balancing based on MA)[30] has been made in which a novel algorithm for updating load information partially based on MA which is called ULIMA.

MA support load balancing in parallel and distributed computing[5,9], e.g., Traveller[31] using resource broker. It implements parallel application such

as L. U. Factorization and sorting. MESSENGERS[32] is a system for general-purpose distributed computing based on MAs. It supports load balancing and dynamic resource utilization. Flash[33] is a framework for the creation of load balanced distributed application in heterogeneous cluster system.

The load balancing approaches for distributed Nodes or nodes involve frequent message exchanges between the request distributors and clients to detect and exchange load information. These message exchange leads to network traffic. But the multiagent framework presented in this study can resolve these problems. In this framework whenever load on a Nodes exceeds from a threshold value, agents are activated dynamically for load balancing on overloaded Nodes.

CONCLUSION AND FUTURE WORK

In this study we have presented design and implementation of multiagent framework for load balancing, which is implemented on PMADE. This framework is a flexible foundation to implement different load balancing schemes for distributed applications. The performance evaluations show that the multiagent based approach outperform in comparison to message passing paradigm when large number of client requests are involved. The performance evaluation shows that multi MA based approach is better than traditional message passing paradigm on heterogeneous cluster. A Value Function have been proposed which evaluate the effectiveness of a particular node in cluster. This value function is compared with CPU based approach which only takes CPU memory into account for load distribution, but value function takes various parameters into account.

Further, we are in the process of implementing this system on grid computing environment fro measuring the load and computing power of a grid. Based on result we will suggest and test some performance improvement policy. In this system we have considered that each node is equipped with PMADE environment. Performance metrics we have considered data size, fault tolerance, throughput vs. jobs, communication cost, etc.

REFERENCES

1. Dias, D., W. Kish, R. Mukherjee and R. Tewari, 1996. A scalable and highly available web-server. Proc. 41st Intl. Computer Conf. (COMPCON'96), IEEE Computer Society, SanJose, CA, pp: 85-92.
2. Tang, W. and M. Mutka, 2000. Load distribution via static scheduling and client redirection for replicated web servers. Proc. 1st Intl. Workshop on Scalable Web Services (in conjunction ICPP 2000), Toronto, Canada, pp: 127-133.
3. Chess, D., B. Grosof, C. Harrison, D. Levine, C. Parris and G. Tsudik, 1995. Itinerant agents or mobile computing. IEEE Personal Commun. Mag., 2: 34-49.
4. Imielinsky, T. and B.R. Badrinath, 1994. Wireless computing: Challenges in Data management. Commun. ACM, 37: 18-28.
5. Al-Jaroodi, J., N. Mohamed, J. Hong and D. Swanson, 2003. A middleware infrastructure for parallel and distributed programming models on heterogeneous systems. IEEE Trans. Parallel and Distributed Systems, Special Issue on Middleware, 14: 1100-1111.
6. Patel, R.B., 2004. Design and implementation of a secure mobile agent platform for distributed computing. Ph.D. Thesis Department of Electronics and Computer Engineering, IIT Roorkee, India.
7. Dale, J., 1997. A mobile agent architecture for distributed information management. Ph.D. Thesis, Univ. of Southampton.
8. Haverkamp, D.S. and S. Gauch, 1998. Intelligent information agents: Review and challenges for distributed information sources. J. Am. Soc. Inform. Sci., 49: 304-311.
9. Al-Jaroodi, J., N. Mohamed, J. Hong and D. Swanson, 2002. An agent-based infrastructure for parallel java on heterogeneous clusters. Proc. IEEE Intl. Conf. Cluster Computing, IEEE.
10. Cardellini, V. and M. Colajanni, 1999. Dynamic load balancing on web-server systems. IEEE Internet Computing, 3: 28-39.
11. Schlossnagle, T., 2000. The backhand project: Load balancing and monitoring apache web clusters. Proc. Apache Con Europe2000, mod_backhand, http://www.backhand.org/mod_backhand
12. Patel, R.B. and K. Garg, 2001. PMADE - A Platform for mobile agent Distribution & Execution, in Proceedings of 5th World Multi Conference on Systemics, Cybernetics and Informatics (SCI2001) and 7th International Conference on Information System Analysis and Synthesis (ISAS 2001),Orlando, Florida, USA, July 22-25, 4: 287-293.
13. Patel, R.B. and K. Garg, 2004. A new paradigm for mobile agent computing. WSEAS Trans. Computers, 3: 57-64.
14. Raimundo, J., A. Macêdo, F.M. Assis Silva, 2005. The mobile groups approach for the coordination of mobile agents. J. Parallel Distributed Computing, 65: 275-288.
15. Yan, K.Q., S.C. Wang and C.P. Chang, 2006. A hybrid load balancing policy underlying grid computing environment. J. Computer Standard and Interfaces.
16. CiscoSystemsInc.LocalDirector. http://www.cisco.com
17. Singhai, A., S.B. Lim and S.R. Radia, 1998. The sun SCALR framework for internet servers. IEEE Fault Tolerant Computing Systems.

18. Silva, M., J. Mira da Silva and J. Delgado, 1998. An overview of AgentSpace: a next-generation mobile agent system. Proc. Mobile Agents Second Intl. Workshop, Springer, Berlin, Germany, pp: 148-159.
19. Lange, D. and M. Oshima, 1998. Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley, Boston, MA.
20. Kobliak, R., 1999. Concordia [Java mobile agent]. Comm. ACM, 42: 96-97.
21. WebOS 10.0 Application Guide. <http://www.nortelnetworks.com/cgi->-bin/eserv/cs/main.jsp.
22. Cisco Local Configuration and Command Line Reference Guide.
23. http: //www. cisco.com /univercd /cc/td/ doc/ product/>-iaabu/ localdi r/ldv42 /421 guide/index. htm.
24. Zhang, W., S. Jin and Q. Wu, 2000. Scaling internet service by linux director. Proc. Fourth Intl. Conf./Exhibition on High Performance Computing in the Asia-Pacific Region, 1: 176-183.
25. LVS documents. http://www.linuxvirtualserver.org/Documents.html
26. TurboLinux, Turbo Linux Cluster Server 6 User guide <http://www.turbolinux.com>.
27. RedHatLinux, Piranha white paper. http://www.redhat.com/support/wpapers/piranha/index.html
28. Gamache, R., R. Short and M. Massa, 1988. Windows NT clustering service. IEEE Comput., 31: 55-62.
29. BEA WebLogic Server Clustering White Paper. http://www.bea.com/products/weblogic/server/clustering.pdf.
30. IBM WebSphere Edge Server Redbook. http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg246511.pdf.
31. Server Iron Chassis L4-7 Software Configuration Guide. <http://www.foundrynet.com/services/documentation -/sichassis/management.html>
32. Xu, C.-Z. and B. Wims, 2000. Mobile agent based push methodology for global parallel computing. Concurrency and Computation: Practice and Experience, 14: 705-726.
33. Obeloer, W., C. Grewe and H. Pals, 1998. Load management with mobile agents. Proc. 24th EUROMICRO Conf. (EUROMICRO98), Vasteras, Swedan, 2: 1005-1012.