Journal of Computer Science 2 (4): 337-346, 2006
ISSN 1549-3636

Load Balancing on Open Networks: A Mobile Agent Approach

R. B. Patel and Neetu Aggarwal
Department of Computer Engineering, M. M. Engineering College, Mullana-133203, Haryana, India

Abstract: In order to disperse the load on a Web server, generally the server cluster is configured to distribute access requests, or mirror servers are distributed geographically on different networks. Use of the Internet and the World-Wide-Web (WWW) has become widespread in recent years and mobile agent technology has proliferated at an equally rapid rate to evenly distribute the requests to web servers through load balancing. There are various loads balancing policies came into picture. Primitive one is Message Passing Interface (MPI). Its wide availability and portability make it an attractive choice, however the communication requirements are sometimes inconventior and inefficient when implementing the primitives provided by MPI. Mobile agent (MA) based approach have the merits of high flexibility, efficiency, low network traffic, less communication latency as well as highly asynchronous. In this study we present dynamic load balancing using mobile agent technology in which when a node is overloaded, task migrates to less utilized nodes so as to share the workload. However, the decision of which nodes receive migrating task is made in real-time by design and implementation of a framework called Platform for Load balancing (PLB). It is implemented on PMADE (A Platform for Mobile Agent Distribution and Execution). PLB integrated web servers can dispatch MAs to retrieve load information and accomplish load redistribution on all servers. The performance evaluation demonstrates that the PLB framework provides a foundation to develop efficient load balancing schemes on wide range of web server systems from cluster to open network and the results of a comparison of PLB, with some existing ones, is also reported.

Key words: Mobile Agents, Agent host, Agent Submitter, PLB, Load balancing.

INTRODUCTION

Today, web services have been widely involved in all aspects of daily life for more than six hundred millions users over the Internet. The proliferation of web services and users appeals for high scalability, availability and reliability of web servers to provide rapid response and high throughput for the client requests occurring at any time. Distributed web servers (DWSs) provide an effective solution for improving the quality of web services. A collection of web servers is used as a pool of replicated resources to provide concurrent services to the clients. The incoming requests can be distributed into the servers according to specific load distribution strategies and thus the requests are processed quickly. The DWSs can be organized in different scopes. They can be integrated into a cluster of web servers linked via local-area network to act as a powerful web server. They can also be deployed at different sites over open network (Internet). The DWSs can present high scalability by incorporating additional servers in response to the growing demands for web services as well as supporting fault tolerant service. When any server encounters failure, other servers can sustain the service.

Load balancing[24, 25] is a active technology that provides the art of shaping, transforming and filtering the network traffic and then routing and load balancing it to the optimal server node. If we take into an account the simple server granting services then at the time of peak traffic it can be vulnerable to failure but 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 tolerant, quick response time.

Mobile agent 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 mobile agent (MA)[2]. MA supports a variety of web-based distributed applications namely: systems and distributed information management[3] and information retrieval [4]. Other areas where MAs are seen as offering potential advantages- wireless or mobile computing[5,6] dynamic deployment of code, thin clients or resource-

Corresponding Author: R. B. Patel, Department of Computer Engineering, M. M. Engineering College, Mullana-133203, Haryana, India

limited devices, personal assistants, and MA-based parallel processing[7, 8].

Traditional load balancing approaches on distributed web servers are implemented based on message passing paradigm[1,24]. MA technology provides a new solution to support load balancing on distributed web servers. They can separate the functionalities in the design of a web service system. In traditional message-passing based load balancing approaches, the server service module mixes the main functionalities of web service with the maintenance functions such as load balancing. Whenever a new load balancing policy is introduced, the server module has to be rewritten. Using MAs, on the other hand, the maintenance functions can be separated from the service module and be implemented separately in the MAs. Therefore, a mobile agent based approach is flexible to incorporate new load balancing polices for various web server systems. MAs produce low network traffic. In message-passing based approaches, the web servers have to exchange messages of load information periodically in order to make decisions on load balancing. The mod_backhand[9] is such a load-balancing module for the Apache web server[9,27]. The message exchanges result in high communication latency and thus deteriorate the performance of a web service system. Differently, a MA can migrate to a target server and interact to specified objects on the site. The on-site interaction eliminates the direct message exchanges between the servers. The network traffic and communication latency can be largely reduced. MAs support asynchronous and autonomous operations. The servers can dispatch MAs individually that travel independently between the servers to perform various operations. A MA can encapsulate load balancing policies and travel to other servers where it can make decision on load distribution according to the up-to-date states of the servers. Due to the merits of low network traffic and quick response time, MAs can strengthen the scalability of a web server system. They can also improve the reliability of web servers by bringing client requests from a faulty server to an active one.

In this study we present a dynamic load-balancing framework called Platform for Load balancing (PLB). It uses MAs to implement reliable and scalable load balancing on distributed web servers. PLB is implemented on PMADE (A Platform for Mobile Agent Distribution and Execution)[2,11]. The load balancing schemes based on the PLB can achieve better performance than the message passing based approaches. The performance evaluation demonstrates that the PLB framework provides a foundation to develop efficient load balancing schemes on wide range of web server systems from cluster to the open network (Internet) and the results of a comparison of PLB, with some existing ones, is also reported.

OVERVIEW OF PMADE

Figure 1 shows the basic block diagram of PMADE. 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)[10], 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[11]. 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[2].

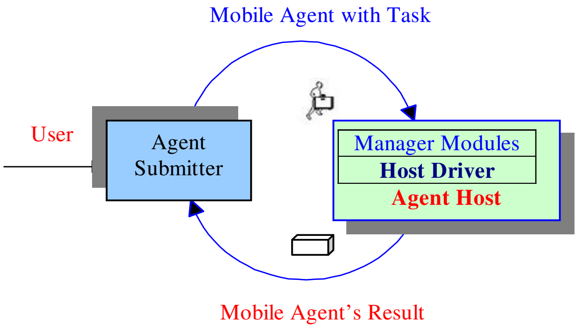

Fig. 1: Block Architecture of PMADE

ARCHITECTURE OF PLATFORM FOR LOAD BALANCING

Load balancing provides up to date information about the load on servers, which result in high network traffic in web servers. For providing reliable solution we have developed a MA based framework called Platform for Load Balancing (PLB) as shown in Fig. 2. PLB architecture is divided into four sections- interface, agents, policy and database. The interface is used for the communication with external world via PMADE. It maintains a buffer. The buffer is used for storing the messages temporarily whenever communication delayed arises. It also helps to provide fault tolerance to the message on system failure. For providing the security to messages it uses PMADE security[12]. We have considered few assumptions and identified some policies & agents in the development of the PLB which are discussed next.

ASSUMPTIONS

We have considered the following assumptions in the development of the PLB:

- Servers taken into an account are heterogeneous in nature. They can have different hardware configuration, operating system and processing power.
- Each sever is capable of processing the client request and cooperate with each other in order to share the workload.
- To provide dynamic capability to the server capacity can be changed due to the variation of workload.
- A MA can be proprietary to a server where it is created and perform dedicated operations for the owner.
- A MA can be shared among a group of servers to act on behalf of these servers. MAs can interact with each other by direct data exchange.
- A MA can interact using the stigmergy technique in which the MAs can collect the information from the traces left in the environment by one another. A MA can gather the information placed on a server by other MAs who have previously visited there. The stigmergy is an indirect method for the interaction between MAs, which can reduce the network traffic and achieve quick decision-making.

POLICY

We have defined four policies according to the need of agents we have founded. These policies are governed by system administrator and updated according to the load balancing schemes.

Information Gathering Policy (IGP) 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 (PI) 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).

Server Selection Policy (SSP) selects an appropriate server based on the load information to which the workload on an overloaded server can be reallocated. Different strategies can be applied to the selection. For example, the find-best strategy selects the least loaded server among all servers and this strategy selects the first server whose load is below a threshold. The least loaded server has been taken into best category as it has very less load and can be selected as an appropriate server for handling the request and responding. In find-first, we don't take into account less or overloaded factor but threshold value results in providing the appropriate selection of the server. The very first server who is having the less threshold value will be taken into consideration.

Job Movement Policy (JMP) determines when job reallocation should be performed and which job(s) (i.e., client requests) should be reallocated. Job reallocation is activated by a threshold-based strategy. In a sender-initiated method, the job movement is invoked when the workload on a server exceeds a threshold. In a receiver-initiated method, a server starts the process to fetch jobs from other servers when its workload is below a threshold. The threshold can be a pre-defined static value or a dynamic value that is assessed at runtime based on the load distribution among the servers. When job reallocation is required, the appropriate job(s) will be selected from the job queue on a server and moved to another server. Adequate administration is required for implementing this policy as threshold value is being determined statically and dynamically.

AGENTS

We have founded three agents, out of which two are mobile agents and one is stationary. A brief look of these agents is as follows:

Server Management Agent (SMA) is a stationary agent and sits at a server, responsible for monitoring the workload on local server and executing JMP if required. In sender-initiated policy, when the server is overloaded, SMA initiates the job reallocation process. It selects the jobs from the local job queue and dispatches them to the other servers. It works like a policy manager.

Load Managing Agent (LMA) is a MA responsible for information gathering. It travels around the servers, collects the load information, and meanwhile propagates the load information to the servers. It can be either a proprietary or a shared agent. A proprietary agent works for a single server and hence collect load information regarding one server only whereas shared server is associated to all. Its responsibility is to collect the load information about all the servers.

Job Managing Agent (JMA) activated by the SMA whenever an overload situation arises on web server. A

JMA executes the SSP to select another server to receive the reallocated job. In a message-passing based approach, such a negotiation involves multiple rounds of message exchange between the source and destination servers that result in high network traffic and job movement latency. On contrary, the job redirection on the fly conducted by MAs can efficiently accomplish job redirection. Then, the JMA carries the reallocated job to that server and negotiates with it for the acceptance of the job. JMA is an optional component in the PLB. In a load-balancing scheme without the JMA, the SMA is responsible for selecting the destination server and the job reallocation is fulfilled by direct interaction between the SMAs.

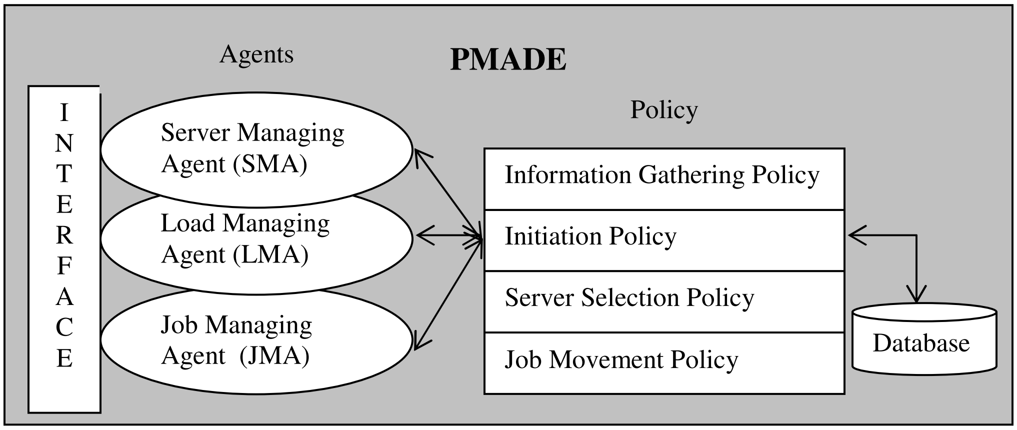

Fig. 2: Architecture of PLB

IMPLEMENTATION

We have assumed that the open network environment is divided into network domains, regions (sub networks) and agent hosts (local sites of the clients) as shown in Fig. 3. There is a domain management server (DMS) in each network domain, which has information about all other DMSs in the global network. It also has information about all the regions in the network domain. DMS is responsible for maintaining uniqueness of names of regions, which are part of that network and helps to identify the region in which an agent is present. Each DMS maintains a Domain Agent Database (DAD), for information about the current location of all agents which were created in that domain or transited though it. Each entry of DAD of the form (A, FD, r) represents that agent A (LMA or JMA) can be found in region r of the foreign network domain FD, or it has transited from that network domain or region. For DAD and RAD, the primary key is the agent name A (LMA or JMA). With the help of these naming schemes we check the fault tolerance by maintaining the status report of MA which keep the updated information of all the agents[26]. Agent migration from one network domain to another is always accomplished through the DMS. During inter domain migration the agent has to update location information in the DAD of the present domain and register in the DAD of the target network domain. Every region maintains information about all AHs that are part of that region. An AH can be a member of an existing region or can start in a new region. In each region, a Region Agent Database (RAD) is present at an AH which runs at the gateway of a sub network. It contains location information about each agent that was created in that region or transited through it. This host acts, as the Agent Name Server (ANS)[29], which manages the RAD. ANS, is responsible for maintaining uniqueness of names of all MAs, created in that region. When a new agent is created, the user assigns a name to it by registering in the RAD of its birth region. Each entry of RAD of the form (A, r, Nil) represents the region r where agent A (LMA or JMA) was found or transited through it. Similarly (A, Nil, AH) represents an agent A (LMA or JMA), which exists in that region at AH. For intra region migration, it has to update its location information in the RAD of that region. This is an Intra Region Location Update. During inter region migration, the agent has to update the location information in the RAD of present region and register in the RAD of the target region, specifying the host in that region to which it is migrating.

Architecture of the PLB based load-balancing scheme, which we need to install at the gateways for cluster use of two types of the agents specified in PLB, i.e., SMA and LMA. The scheme does not use JMA. Job reallocation is accomplished by the SMAs. In this scheme, every server within region runs an SMA and an LMA. The LMA is proprietary to a home server. In a cluster, an LMA can traverse all servers in short time latency. The SMA executes the IGP, PI, and JMP. It monitors local workload and dispatches the LMA to collect load information when required. The SMA maintains a log of global load information on local server. Each entry in the log records the load information of a server at a certain time. The load-balancing scheme adopts a sender PI. Once the load exceeds a threshold, the SMA activates the LMA to collect the load information. A dynamic threshold is specified, that is the average load on all servers calculated based on the log of global load information. The LMA traverses in the cluster to retrieve the updated values of cpu_load, $no\_connect$ and $freee\_mem$ on each server and calculate the load metrics. Meanwhile, the LMA selects an appropriate server (called best server) to accept the job reallocation from the home server. The PLB SSP adopts two strategies:

Best-fit: The LMA visits every server and selects the server who has the lowest load as the best server as well as the server who is fastest in processing the request than the slower one. When it returns to its home server, the LMA updates the log of global load information and reports the best server to the home server.

First-fit: The LMA pauses at the first server whose load is below a threshold. The LMA selects this server as the best server and reports this selection and the up-to-date load information to the home server. The LMA remains at the best server, not proceeding to successive servers. Later, the LMA will restart its travel from this server when the home server activates it again. This strategy allows the LMA to perform round-robin traversal in the cluster so that the workload can be fairly distributed to the servers.

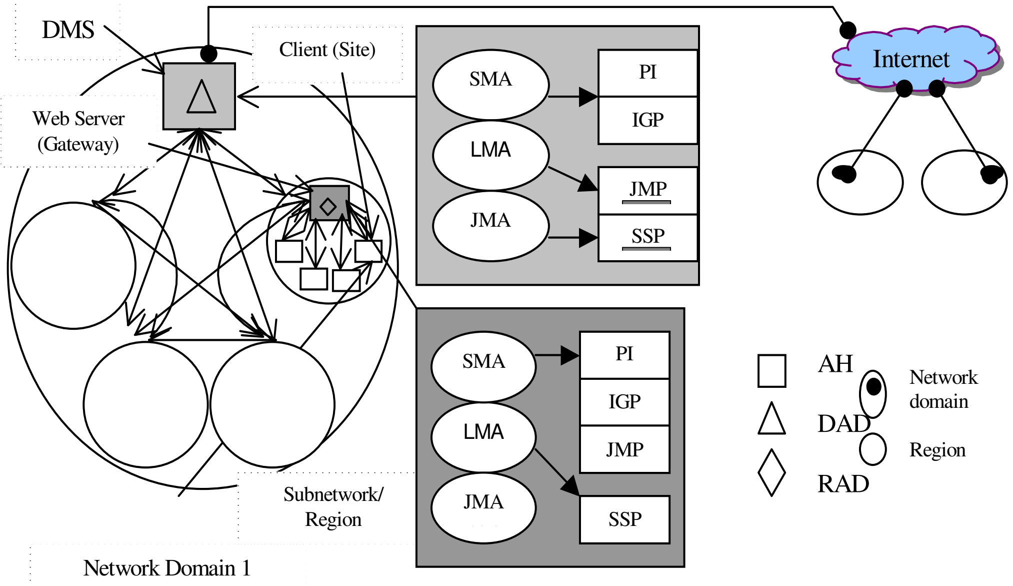

Fig. 3: Load Balancing of Cluster & Open Network

In the open network, the web servers are geographically distributed at different sites as shown in Fig. 3. If the load balancing scheme for such a system uses the same strategy as for a cluster, the proprietary LMA needs to spend high time latency to traverse all servers and the load information may become obsolete when the LMA reports to the home server. Thus, the log of global load information on each server cannot precisely reflect the current states of all servers. Consequently, the load reallocation cannot assure load balancing in the system. When a home server transfers a job to the best server, the latter probably has been overloaded and has to redirect the job to another best server. Due to the stale load information, the job redirection may be transferred across a chain of servers over a long distance until reaching an appropriate server to accept it. Thus, the response time will be greatly prolonged. To resolve these problems, a different load-balancing scheme is designed for open network. For balancing the load on open network, PLB utilize shared LMA to collect and update the load information and dispatches the JMA to deliver jobs to the best server for this scheme. The JMA can perform job redirection on the fly in case the target server is overloaded. Every server within network domain runs a SMA. The SMA is responsible for executing the PI and JMP, but the SSP is handed over to the JMA. Each server receives and processes client requests independently. When a server is overloaded, the SMA initiates the load balancing process by dispatching a JMA to transfer some job to the best server. The SMA selects the client requests from local job queue for reallocation.

In open network, the job selection criterion should also take into account the latency of job transfer. A job on server S_1 will be reallocated to a remote server S_2 only when $t_{S_1} > t_{S_1S_2} + t_{S_2}$, where t_{S_1} and t_{S_2} are the expected waiting times of the job on S_1 and S_2;

$t_{S_1S_2}$ is the communication latency of delivering the job from server S_1 to server S_2. The latency is estimated based on the distance between the servers. The JMA executes the SSP to select a best server to receive the job using the log of global load information. The server selection can adopt the find-best or find-first strategy. Then, the JMA delivers the job to the best server and negotiates with it on the site. If the best server has also been overloaded, the JMA can find another server to receive the job on the fly using the up-to-date load information that it has collected on the way. The JMA conducts the negotiation and selection of alternate receiver (if required) on behalf of the home server. The homes server can proceed to process other jobs after it has dispatched the JMA. This feature can improve the flexibility and efficiency of the load-balancing scheme. A shared LMA keeps on traveling around the servers to collect and propagate the load information. When it arrives on a server, the LMA collects the load information of the server and updates the log of global load information on the server using the load information that it has collected on the way. Therefore, the LMA can continuously propagate the load information to distributed servers. A PLB integrated web server system can dispatch one or more LMAs to collect load information in the system. An LMA takes long time to collect load information on all servers on open network. Multiple LMAs can be used to speed up the global information collection and updating. Each LMA travels within a domain of the network. The load information in different domains can be exchanged at the intersection points between the domains. Each LMA traverses in one of the domain. They exchange load information on the two common servers on the border of domains using the stigmergy technique and then propagate the global information to all servers. The use of multiple LMAs can accelerate the information update and improve the accuracy of global load information.

PERFORMANCE STUDY

To study the performance of the PLB we have implemented it on 10/100/1000 Mbps switched LAN that connects 850 workstations and personal computers, and is used by about 500 hundred researchers and students. Machines are grouped into eight different networks with their own servers and servers of each network are connected to the main server of the institute. For each network there are 100 nodes which are running clients, eight web servers running nodes and one DMS equipped. MA enabled web server cluster is implemented on a cluster of PCs (P-4, 3 GHz machines) using PMADE and j2sdk1.5.1. The AS node and agent host nodes have 256 MB main memory, while the web server host has 512 MB, we have used j2sdk 1.5.1 Java Virtual Machine with native thread support. Among them, eight PCs are configured as web servers and other PCs are assigned as clients. **LOAD**

BALANCING METRICS

The load balancing metrics is the parameter that determines how to balance the client load across servers. We can fine-tune how traffic is distributed across multiple real servers by selecting one of the following load balancing metrics.

Least Connections: Sends the request to a server that currently has the fewest active connections with clients. For sites where a number of servers have similar performance, the least connections option smoothes distribution if a server gets bogged down. For sites where the capacity of various servers varies greatly, the least connections option maintains an equal number of connections among all servers. This results in those servers capable of processing and terminating connections faster receiving more connections than slower servers over time.

Round Robin: Directs the service request to the next server, and treats all servers equally regardless of the number of connections or response time. For example, in a configuration of four servers (S_1, S_2, S_3, S_4), the first request is sent to S_1, the second request is sent to S_2, the third is sent to S_3, and so on. After all servers in the list have received one request, assignment begins with S_1 again. If a server fails, PLB avoids sending connections to that server and selects the next server instead.

Weighted: Assigns a performance weight to each server. Weighted load balancing is similar to least connections, except servers with a higher weight value receive a larger percentage of connections at a time. We can assign a weight to each server and this weight determines the percentage of the current connections that are given to each server. The default weight is 0.

Server Response Time: Selects the server with the fastest response time. The SI (Server Iron), the server that acts as an intermediate for forwarding the requests among multiple servers unseen by end user calculates the response time based on TCP SYN and TCP SYN ACK packets[28].

EVALUATION

The PLB-based load-balancing scheme is evaluated on the cluster by comparing its performance with the *mod_backhand* module[9] (a load balancing approach based on message-passing paradigm). The AS is used to generate client requests. It is used to measure the performance of web server software and hardware products on multiple web clients to create a load on each web server. Performance of a load-balancing scheme is assessed in the following criteria:

Load distribution: the load on each of the servers is measured at different time instants. The length of its job queue denotes the load on a server. The average deviation of the load distribution over all servers is calculated to show the effect of load balancing.

System throughput: the overall throughput of the web server cluster, measured in the number of requests processed per second.

Network traffic: the overall communication overhead in the cluster, measured in the total number of data (bytes) transferred in the communication.

Every server receives the client requests independently. If a server is overloaded, it can redirect an incoming request to another server. Table 1 compares the load distribution generated by the PLB scheme and the mod_backhand module on eight servers at different moment. Table 1 also includes the average deviation of load on the eight servers. It shows that the PLB scheme has lower load deviation than the mod_backhand module in most of the cases. It means PLB can distribute client requests more evenly onto the web servers. The average of overall mean deviation in Table 1 is the average of mean deviations at all moments. The average of the overall mean deviation of the PLB scheme is lower than the mod_backhand module that verifies the better performance of the PLB scheme in supporting load balancing.

Table 1: Load distribution on eight servers

Mod_backhand (message passing load balancing)									
Mean Deviation	Server 1	Server 2	Server 3	Server 4	Server 5	Server 6	Server 7	Server 8	Time in min
25.88	81	49	56	91	93	95	64	64	5
14.12	68	93	93	67	79	68	83	88	10
24.38	79	48	89	48	82	96	66	73	15
21.88	71	54	54	74	83	56	90	56	20
16.75	92	93	69	70	95	63	74	78	25
20.62	53	78	89	75	95	99	73	65	30
11.75	99	76	94	75	85	87	95	80	35
3.25	97	92	97	98	97	98	97	98	40
13.88	79	96	71	76	79	78	99	71	45
7.62	93	97	99	91	80	85	91	95	50
15.5	92	79	97	84	70	95	92	83	55
1.38	97	98	99	97	95	97	99	99	60
Over all mean deviation =14.75083333									
PLB based load balancing									
Mean Deviation	Server 1	Server 2	Server 3	Server 4	Server 5	Server 6	Server 7	Server 8	Time in min
13.62	90	97	89	82	90	74	77	76	5
17	71	64	66	92	91	72	88	99	10
21.62	79	76	74	72	92	87	66	57	15
5.38	85	92	97	96	92	87	90	87	20
17.88	89	76	80	88	64	83	71	98	25
12.38	74	76	76	97	78	94	80	83	30
7.38	92	81	84	84	98	96	95	95	35
9.62	97	92	88	86	98	82	91	89	40
19.38	78	96	78	81	71	59	69	89	45
21.25	63	94	63	64	57	63	56	67	50
4.12	98	96	98	98	88	88	95	90	55
6.75	85	99	92	83	85	91	90	99	60
Over all mean deviation =13.03166667									

Figure 4 shows the system throughput of two approaches. The throughput of the PLB scheme is close to the mod_backhand in all case. As the prototype of PLB scheme is implemented in Java, the high execution overhead of Java program results in the lower throughput of the PLB scheme.

Figure 5 compares the network traffic of PLB and message based schemes. It represents load of on eight servers, each dot for one server. PLB MAs generate lower communication overhead than the message passing in the mod_backhand module.

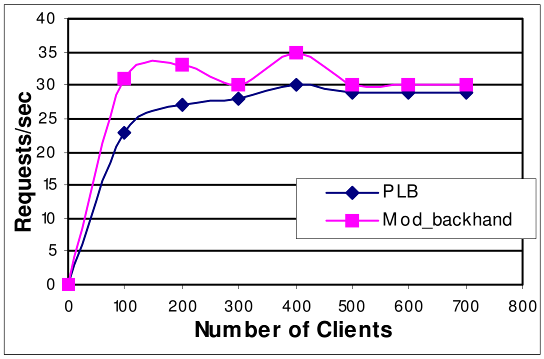

Fig. 4: System throughput

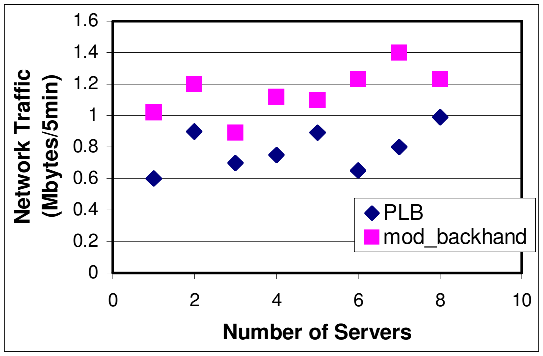

Fig. 5: Network Traffic (Average of 1 hour)

Figure 6 compares the system throughputs of the PLB load-balancing scheme using one shared LMA and the case without load balancing. The result shows that the PLB scheme can obviously improve the system throughput when increasing the number of servers. In the latter case, the processing capacities of the servers are wasted and no improvement.

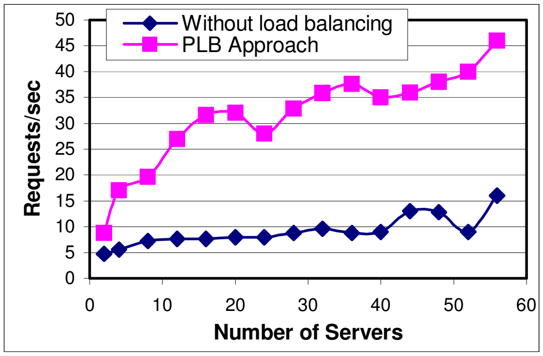

Fig. 6: System throughputs of the PLB scheme and the case without load balancing

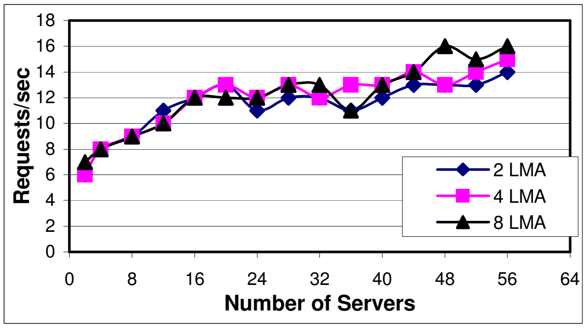

Fig. 7: System throughputs with two LMA's are working in one network.

Figure 7 depicts the system throughputs using two to eight LMAs the effect of which becomes apparent on 56 servers and more. It shows that the use of multiple LMAs can enhance the throughput on larger number of web servers. Among the three cases, eight LMAs can achieve the highest performance and two LMA presents the lowest performance.

ANALYSIS

The SMA maintains a log of global load information on local server. Each entry in the log records the load information of a server at a certain time. Once the load exceeds a threshold, the SMA activates the LMA to collect the load information. A dynamic threshold is specified, that is the average load on all servers calculated based on the log of global load information. The LMA traverses in the cluster to retrieve the updated values of server workload, number of active connections and free memory space, on each server and calculate the load metrics. Meanwhile, the LMA selects an appropriate server (called best server) to accept the job reallocation from the home server. The load on a server is measured as:

$$Load = w_1 * cpu\_load + w_2 * n/T + w_3 * free\_mem \qquad (1)$$

Where

cpu_load is the workload on the server, measured in the length of job queue;

n is the number of active connections on the server;

T is the maximum number of connections allowed to the server;

$free\_mem$ is the percentage of free memory space;

w_1, w_2, w_3 are the weights of the parameters, $w_1 + w_2 + w_3 = 1$.

The performance can be increased if it consists of minimum required transaction throughput and maximum acceptable transaction latency. The minimum throughput (transaction rate) of transaction j for service i required to support this usage profile is given by equation (2).

$$T_{ij} = n_{ij} * N_i / t_i \qquad (2)$$

Where

T_{ij} is the transaction rate of transaction j for service i.

n_{ij} is the number of transactions (j) per user session for service i.

N_i is the number of users concurrently using service i at the peak time.

t_i is the user session time for service i.

From equations (1) and (2) we are able to compute total load from all the clients connected with web server at any time t, i.e.,

$$S_{L/t} = Load * T_{ij} \qquad (3)$$

By putting values of $load$ from (1) and T_{ij} from (2) into (3) we get (4) which is used for finding load on any server at any time.

$$S_{L/t} = \left(w_1 * cpu\_load + w_2 * {}^{n}\!/_{T} + w_3 * free\_mem\right) * \left(n_{ij} * N_i / t_i\right) \qquad (4)$$

RELATED WORK

To improve the efficiency of web services, a group of web servers can be deployed to provide web service collectively to the clients. Load balancing is indispensable for a web service system to assure even distribution of incoming requests on the web servers. One of the most compelling problems that arise on distributed web server clusters is the selection of an efficient load balancing policy. The load balancing policy should aim for evenly utilized servers in the cluster and a minimum response time for the processed requests. Under standard methodology for load balancing server selection is done randomly[1]. The client can select one of the web server in random or choose an appropriate one using an intelligent selection mechanism, e.g., Netscape Navigator browser[13]. The random selection cannot guarantee load balancing and server availability. Intelligent server selection can be implemented by java applets running on the client side to detect the states of the server and the network delay.

A DNS (Domain Name Server) is the routing mechanism for distributed web servers. It selects one of the servers by mapping the URL to the IP address of a web server. It leads to bottleneck and has limited control on the request routing due to the intervention of intermediate servers that cache the URL to IP address mapping between the clients and the DNS, e.g., Cisco Distributed Director[14]. 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[15]. With weighted round robin the incoming requests are distributed among the servers on a round robin fashion, weighted by some measure of the load on each of the servers. In[16] author presented Tomcat 5 server which facilitates a rules-based load balancer application based on round robin and random algorithms. Dispatching techniques whether 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. Also, the entire cluster's throughput is limited by the speed and processing power of the dispatch server. SUNSCALAR[17] provides load balancing by using both approaches, i.e., Dispatcher and Round Robin. In[18] author presented locality aware policy. This policy improves the cluster's performance if considering the cache content of the servers, which does the distribution of requests. A request is served much faster if it is fetched from the cache of a server than from the local disk.

Other policies incorporate knowledge of the workload variability into the load balancing policy [19,20,21]. In [19] author presented SITA-E (Size Interval Task Assignment with Equal Load), which assigns the incoming requests into servers based on the request size, assuming that the requested file sizes follow a bounded Pareto distribution.

DC-Apache system[27] demonstrated its ability to achieve high performance and scalability by effectively distributing load among a group of cooperating Apache servers and by eliminating hot spots and performance bottleneck with replicated documents. A framework for load balancing using MA named EALBMA (Efficient and Adaptive Load Balancing based on MA)[28] has been made in which a novel algorithm for updating load information partially based on MA called ULIMA.MA support load balancing in parallel and distributed computing[7,8], e.g., Traveller[22] using resource broker. It implements parallel application such as L. U. Factorization and sorting. MESSENGERS[5] is a system for general-purpose distributed computing based on MAs. It supports load balancing and dynamic resource utilization Flash [23] is a framework for the creation of load balanced distributed application in heterogeneous cluster system.

The load balancing approaches for distributed web servers involve frequent message exchanges between the request distributors and the servers or clients to detect and exchange load information. These message exchange leads to network traffic. But PLB presented in this paper can resolve these problems. In PLB running web servers whenever load on a server exceeds from a threshold value, agents are activated dynamically according to the topology of the network for the load balancing on overloaded servers.

CONCLUSION AND FUTURE WORKS

In this study we have presented design and implementation of PLB, which is implemented on PMADE. The PLB is a flexible foundation to implement different load balancing schemes for scalable distributed web server systems. The performance evaluations show that the PLB based approach outperform in comparison to message passing paradigm when large number of servers and client requests are involved. The objectives of this paper is to develop an analytical model for the stochastic dynamics of delay-limited distributed systems and utilize it to develop load-balancing policies that mitigate the performance degradation (or failure) caused by communication and load-transfer delays. In the future we define more accurate measurements of load to provide an accurate assessment for the load balancing schemes and improve the performance of the PLB and find a comparison with more existing system and develop a benchmark. The measurements will take into account the data size, fault tolerance, and communication cost. We will combine the properties of the client requests and the heterogeneous features of web servers to determine a load distribution strategy. For example, if a request needs memory-intensive service, the MAs will switch to use a load measurement that puts higher weight on memory space. Thus, a server with large spare memory space can be selected to process the request.

REFERENCES

1. Cardellini, V. and Colajanni, M., Dynamic Load Balancing on Web-server Systems, IEEE Internet Computing, 3, pp. 28-39, 1999.
2. Patel, R. B., Design and Implementation of a Secure Mobile Agent Platform for Distributed Computing, PhD Thesis Department of Electronics and Computer Engineering, IIT Roorkee, India, Aug. 2004.
3. Jonathan Dale, A Mobile Agent Architecture for Distributed Information Management, Ph.D. thesis, Univ. of Southampton, Sept. 1997.
4. Haverkamp, D. S. and Gauch, S., Intelligent Information Agents: Review and Challenges for Distributed Information Sources, in Journal of the American Society for Information Science, 49(4): 304-311, 1998.
5. Chess, D., B. Grosof, Harrison, C., Levine, D., Parris, C. and Tsudik, G., Itinerant agents or mobile computing, IEEE Personal Communications Magazine, 2, pp. 34-49, Oct. 1995.
6. Imielinsky, T. and Badrinath, B. R., Wireless Computing: Challenges in Data Management, Communication of the ACM, 37(10): 18-28, 1994.
7. Al-Jaroodi, J., Mohamed, N., Jiang Hong and Swanson, D., An Agent-Based Infrastructure for Parallel Java on Heterogeneous Clusters, in Proceedings of the IEEE International Conference on Cluster Computing, IEEE, Nov. 2002.
8. Al-Jaroodi, J., Mohamed, N., Jiang Hong and Swanson, D., A Middleware Infrastructure for Parallel and Distributed Programming Models on Heterogeneous Systems, IEEE Transactions on Parallel and Distributed Systems, Special Issue on Middleware, 14(11): 1100-1111, Nov. 2003.
9. Schlossnagle, T., The Backhand Project: Load balancing and Monitoring Apache Web Clusters, in Proceedings Apache Con Europe 2000, London, Britain, mod_backhand, <http://www.backhand.org/mod_backhand/>
10. Patel, R. B. and Garg, K., A New Paradigm for Mobile Agent Computing, WSEAS Transaction on Computers, Issue 1, Vol. 3, pp. 57-64, Jan. 2004.
11. Patel, R.B. and Garg, K., PMADE – A Platform for mobile agent Distribution & Execution, in Proceedings of 5th World MultiConference on Systemics, Cybernetics and Informatics (SCI2001) and 7th International Conference on Information System Analysis and Synthesis (ISAS 2001), Orlando, Florida, USA, July 22-25, 2001, Vol. IV, pp. 287-293.
12. Patel, R.B. and Garg, K, A FLEXIBLE SECURITY FRAMEWORK FOR MOBILE AGENT SYSTEMS, Control and Intelligent Systems, 33(3): 175-183, 2005.
13. MoseDale, D., Foss, W. and McCool R., Lesson Learned Administering Netscape's Internet Site, IEEE Internet Computing, 1, pp. 28-35, 1997.
14. CiscoDistributedDirector, <http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml>
15. CiscoSystemsInc.LocalDirector. <http://www.cisco.com>.
16. Clustering and Load Balancing in Tomcat 5, Part 1 by Srini Penchikala, <http://www.onjava.com/pub/au/1418> 03/31/2004.
17. Singhai, A., Lim, S. B. and Radia S. R., The SunSCALR Framework for Internet Servers, IEEE FaultTolerant Computing Systems, Jun 1998.
18. Cherkasova, L., FLEX: Design and Management Strategy for Scalable Web Hosting Service. HP Laboratories Report No. HPL-1999-64R1, May 1999.
19. Harchol-Balter, M., Crovella, M.E. and Murta, C. D. , On Choosing a Task Assignment Policy for a Distributed Server System, in Proc. of Performance Tools '98, LNCS 1469, pp. 231-242, 1998.
20. Zhu, H., Tang, H. and Yang, T., Demand-driven Service Differentiation for Cluster-based Network Servers, in Proc. of INFOCOMM'2001, April Alaska.
21. Andersen, D., Yang, T. and Ibarra, O., Towards a Scalable Distributed WWW Server on Workstation Clusters, Journal of Parallel and Distributed Computing, 1997.
22. C. -Z. Xu and Wims, B., Mobile Agent Based Push Methodology for Global Parallel Computing, Concurrency and Computation: Practice and Experience, 14 (2000), pp. 705-726.
23. Obeloer, W., Grewe, C. and Pals, H., Load Management with Mobile Agents, in Proc. 24th EUROMICRO Conference (EUROMICRO 98), Vol. 2, Vasteras, Sweden, 1998, pp. 1005-1012.
24. Dias, D., Kish, W., Mukherjee, R. and Tewari, R., A Scalable and Highly Available Web-Server, in Proc.41st International Computer Conference (COMPCON'96), IEEE Computer Society, San Jose, CA, 1996, pp. 85-92.
25. W. Tang, M. Mutka, Load Distribution via Static Scheduling and Client Redirection for Replicated Web Servers, in Proc. 1st International Workshop on Scalable Web Services (in conjunction ICPP 2000), Toronto, Canada, 2000, pp. 127-133.
26. Patel, R. B. and Mastorakis, Nikos, FAULT-TOLERANT MOBILE AGENTS COMPUTING, WSEAS Transactions on Computers, Issue 3, Vol. 4, March 2005, pp. 287-314.
27. Quanzhong Li, Distributed Cooperative Apache web server, WWW10, May 1-5, 2001, Hong Kong, ACM 1-58113-348-0/01/0005.
28. Server Iron Chassis L4-7 Software Configuration Guide. <http://www.foundrynet.com/services/documentation/sichassis/management.html>
29. Terry, D.B., Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments, Ph.D. thesis, University of California, Berkely, 1985. Available as UCB/CSD Tech. Rep 85-228 and as Xerox PARC Tech. Rep. CSL-85-1.