# Scalable Load Balancing on Distributed Web Servers Using Mobile Agents

*Jiannong Cao, Yudong Sun, Xianbin Wang*

*Internet Computing and E-Commerce Lab*
*Department of Computing*
*The Hong Kong Polytechnic University*
*Hung Hom, Kowloon*
*Hong Kong*

*Sajal K. Das*

*Department of Computer Science*
*and Engineering*
*The University of Texas at Arlington*
*Arlington, TX 76019-0015*
*USA*

## Abstract

Distributed web servers on the immensely expanding Internet require high scalability and availability to provide efficient services to millions of clients on the web. To provide rapid responses to enormous number of client requests, load balancing is an important technique to evenly distribute the requests to web servers. In this paper, we propose a framework called MALD (Mobile Agent based LoaD balancing) that uses mobile agents technology to implement scalable load balancing on distributed web servers. The web servers can dispatch mobile agents to retrieve load information from and accomplish load redistribution on all servers. Various load balancing policies can be incorporated with versatile mobile agents in the framework. Compared with the traditional message-passing based load balancing methods, the mobile-agent based approaches have the merits of high flexibility, low network traffic and high asynchrony. The performance evaluation demonstrates that the MALD framework provides a foundation to develop efficient load balancing schemes on wide range of web server systems from cluster to the Internet.

**Key words**: Mobile agents, Distributed web servers, Load balancing

## 1. Introduction

Today, web services have been widely involved in all aspects of daily life for hundreds 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* 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 to the servers according to specific load distribution strategies and thus the requests can be processed quickly. The distributed web servers 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 *wide-area network*. Distributed web servers can present high scalability. They can readily scale up 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 is an important technique to enhance the performance of distributed web servers. Incoming client requests should be evenly distributed among the servers to achieve quick response [5, 18]. The load on an overloaded server should be transferred to an under-loaded server to enhance the system throughput. Thus, the system resources can get full utilization. Traditional load balancing approaches on distributed web servers are implemented based on *message passing* paradigm [5, 7].

In this paper, we propose a mobile agent enabled approach for load balancing. *Mobile agents* are a new technology being used to support a variety of web-based distributed applications such as information retrieval, telecommunication, and e-Commerce [3, 4, 10, 17]. A mobile agent is a software component that can move freely from one host to another on a network and transport its state and code from home host to other host and execute various operations on the site.

Mobile agent 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 mobile agents, on the other hand, the maintenance functions can be separated from the service module and be implemented separately in the mobile agents. Therefore, a mobile-agent based approach is flexible to incorporate new load balancing polices for various web server systems.

Mobile agents 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 is such a load balancing module for the Apache web server [13, 14]. The message exchanges result in high communication latency and thus deteriorate the performance of a web service system. Differently, a mobile agent 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.

Mobile agents support *asynchronous* and *autonomous operations*. The servers can dispatch mobile agents individually that travel independently between the servers to perform various operations. A mobile agent 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, mobile agents 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.

This paper describes a load balancing framework called MALD (Mobile Agent based LoaD balancing) that uses mobile agent technology to support load balancing on distributed web servers. The MALD framework provides a basis for developing load balancing schemes on varied systems from local-area cluster to wide-area network. The load balancing schemes based on the MALD framework can achieve better performance than the message-passing based approaches.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the MALD framework. Section 4 and Section 5 present the load balancing schemes developed based on the MALD framework for the web servers in cluster and in wide-area network respectively. Section 6 addresses the conclusions and future work.

## 2. Related Work

To improve the efficiency of web service, a group of web servers can be deployed to provide web service collectively to clients. Load balancing is indispensable for a web service system to assure even distribution of incoming requests on the web servers. There are various approaches for the load balancing on distributed web servers. The taxonomy in [5] classifies the load balancing approaches into four categories: client-based, DNS-based, dispatcher-based, and server-based approaches.

A *client-based approach* implements the server selection on the client side. The clients can select one of the web servers in random or choose a most appropriate server using an intelligent selection mechanism. The Netscape Navigator browser adopts a client-based approach to access its own website [15]. When a user browses the Netscape homepage, the Navigator randomly selects one of the servers and directs the user's request to it. 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 servers and the network delay [21]. Then, the most appropriate server can be selected and the request can be forwarded to the server. The drawback of the applet-based approach is the high network latency incurred in the detection.

A *DNS-based approach* is a system-side solution. A DNS (Domain Name System) server is the routing mechanism for distributed web servers. It can select one of the web servers to process a request by mapping the URL (Uniform Resource Locator) to the IP address of a web server. The mapping can implement different policies of server selection. However, the DNS-based approach has limited control on the request routing due to the intervention of intermediate name servers that cache the URL to IP address mapping between the clients and the DNS server. The DNS server may also become a bottleneck in the routing. Cisco DistributedDirector [6] is such a software product for load distribution between multiple geographically dispersed servers in which the DNS server counts on both the server availability and the client-to-server network latency in the server selection.

To take full control on the routing of client request, a *dispatcher-based approach* performs the address mapping at the IP level. It specifies a group of web servers with a *single virtual IP address* (IP-SVA) which is the IP address of the dispatcher that acts as a centralized scheduler and has complete control on the routing. The dispatcher identifies each server with a private address and, if needed, reroutes a client-to-server packet by rewriting the IP address (known as *packet rewriting* [2]). In the basic *TCP router* mechanism [7], the TCP router acts as a dispatcher. The major disadvantage of the packet rewriting approach is the overhead of address rewriting. Another scheme for packet rerouting is the *HTTP redirection* in which the dispatcher distributes the incoming requests to the servers through the redirection mechanism provided in HTTP. The dispatcher can redirect a request by sending a response containing a server address to the client. Then the client resends the request to the server directly.

The *server-based approach* uses a two-level dispatching mechanism. The DNS server initially assigns the client requests to the web servers. Then, each server may reassign a request to another server. This is a decentralized load reallocation strategy where all servers are allowed to participate in the process of load balancing. The mod_backhand [13, 14] module provides a server-based solution for the Apache web server that allows for seamless reallocation of HTTP requests from heavy-loaded servers to under-utilized servers on a per-request basis. The mod_backhand passes a request through a chain of decision-making functions called *candidacy functions*. Each candidacy function reorders the set of available servers and chooses the method of request reallocation (HTTP redirect or HTTP proxy) based on the resource information. The first server in the resulted list is selected to serve the request.

From the above discussion, we can find that all load balancing approaches for distributed web servers involve frequent message exchanges between the request distributor (DNS server or dispatcher) and the servers or clients to detect and exchange load information. The message exchanges increase the network traffic in a web service system. Most of the approaches also exhibit the problem of bottleneck in the routing and redirection of the requests. However, the mobile agent enabled load balancing approach presented in this paper can resolve these problems with respect to the advantages described in Section 1.

Mobile agents have been used to support load balancing in parallel and distributed computing [9]. For example, TRAVELER [20] is a Java-based mobile agent infrastructure to support wide area parallel computing. It allows users to dispatch compute-intensive jobs as mobile agents via a resource broker. The mobile agents can move autonomously from one machine to another for load balancing. TRAVELER has been used to implement parallel applications such as LU factorization and sorting. MESSENGERS [8] is a system for general-purpose distributed computing based on mobile agents. It supports load balancing and dynamic resource utilization [11]. It has been used to implement distributed applications in master-slave model and parallel applications like matrix multiplication. FLASH [16] is a framework for the creation of load-balanced distributed applications in heterogeneous cluster systems. It supports the transfer of subtasks of a parallel application to mobile agents, which travel autonomously through a network to search for free resource. A ray tracer was implemented to test the performance of the framework. So far to our knowledge, mobile agent technology has not been widely used to support the load balancing on distributed web servers.

## 3. MALD: A Mobile Agent Enabled Framework

Distributed web servers can be deployed in different geographical scopes. They can be organized into a *cluster of web servers* linked through local-area network (LAN) to provide high processing power and reliability. They can also be loosely coupled and distributed over a wide-area network (WAN) to share information or services and collectively serve the clients from different locations.

The MALD framework is built on a network of web servers ranging from LAN to WAN. To simplify our discussion, all web servers are considered equivalent in their capabilities. In general, the servers can be heterogeneous in terms of hardware configuration, operating systems, and processing power. The capacity of a server may change at runtime due to the variation of workload. Every server can process client requests individually and meanwhile cooperate with each other to share the workload.

Load balancing can be realized only when comprehensive and up-to-date load information is available. However, the collection of load information will increase the network traffic and interfere with the services for the clients. The load balancing on wide-area network is more time-consuming that involves the interactions between remote servers for gathering load information, negotiating on load reallocation and transporting the workload. Consequently, the load balancing may not improve but degrade the performance of a web service system.

Mobile agents provide a novel technology for implementing load balancing mechanism on distributed web servers. A server can dispatch mobile agent to the system if required. The mobile agent can embed policies of load balancing and travel to other servers and interact with them on the site to acquire latest load information. The embedded policies can deal with the load balancing requirements according to the up-to-date states of the servers and the client requests. The mobile-agent based approach can minimize the network traffic and enhance the flexibility of a load balancing mechanism.

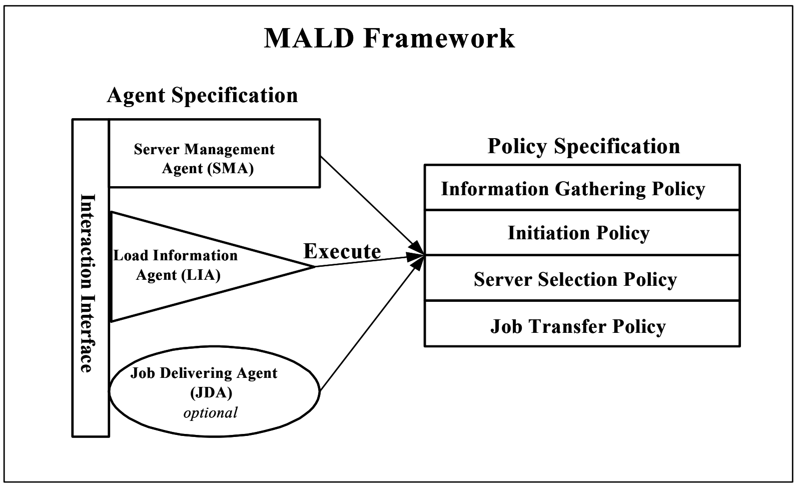


Figure 1 The MALD framework

The mobile-agent enabled framework MALD can be used to specify the behaviors of mobile agents and the policies of load balancing for implementing efficient load balancing mechanisms on different web server systems. The MALD framework consists of a suite of load balancing policies and a collection of cooperative mobile agents as shown in Figure 1. The suite of polices specify the strategies of load information gathering, the conditions for activating load balancing operations, and the metrics for server selection and job reallocation. It defines four *policies*:

(1) *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. Different schemes can be designed for this policy. Each server can dispatch its own mobile agent to collect load information. Or, the servers can share the information collected by a common mobile agent.

(2) *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*).

(3) *Server Selection Policy* 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 the *find-first* strategy selects the first server whose load is below a threshold.

(4) *Job Transfer Policy* 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 transfer 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 transferred to another server.

Different schemes of these policies can be defined and encapsulated in mobile agents. The mobile agents carry the policies to other servers and execute the policies on the sites in response to the states of the servers. The mobile agents can be specified with varied properties. A mobile agent can be *proprietary* to a server where it is created and perform dedicated operations for the owner. It may also be *shared* among a group of servers to act on behalf of these servers. The mobile agents can interact with each other by direct data exchange. They can also interact using the *stigmergy* technique in which the mobile agents can collect the information from the traces left in the environment by one another [12]. A mobile agent can gather the information placed on a server by other mobile agents who have previously visited there. The stigmergy is an indirect method for the interaction between mobile agents, which can reduce the network traffic and achieve quick decision-making. The MALD framework specifies three types of *agents*:

(1) *Server Management Agent* (SMA) is a stationary agent that motionlessly sits at a server, responsible for monitoring the workload on local server and executing job transfer policy if required. In sender-initiated policy, when a server is overloaded, the SMA on it initiates the load reallocation process. The SMA selects the jobs from local job queue and dispatches the job to other servers.

(2) *Load Information Agent* (LIA) is a mobile agent responsible for information gathering. It travels around the servers, collects the load information, and meanwhile propagates the load information to the servers. An LIA can be either a proprietary or a shared agent. A proprietary LIA belongs to one server and works for it. A shared LIA serves for a group of servers and provides load information to them.

(3) *Job Delivering Agent* (JDA) is activated by the SMA on an overloaded server. A JDA executes the server selection policy to select another server to receive the reallocated job. Then, the JDA carries the reallocated job to that server and negotiates with it for the acceptance of the job. The JDA can also perform *job redirection on the fly*: if the destination server refuses to accept the job because it is also overloaded, the JDA can find an alternate server to receive the job using the up-to-date load information it just collected on the way. The job redirection on the fly is a significant merit of the load balancing scheme based on mobile agents. 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 transfer latency. On contrary, the job redirection on the fly conducted by mobile agents can efficiently accomplish job redirection. JDA is an optional component in the MALD framework. In a load balancing scheme without the JDA, the SMA is responsible for selecting the destination server and the job reallocation is fulfilled by direct interaction between the SMAs.

The MALD framework creates a foundation for developing decentralized load balancing schemes on distributed web servers. Varied load balancing schemes can be implemented by specifying the relevant policies and the behaviors of the agents. We have designed two schemes based on this framework, one for local-area cluster (see Section 4) and the other for wide-area network (see Section 5).

## 4. Load Balancing Scheme on Cluster

### *4.1 The Scheme*

A cluster of web servers is interconnected by a local-area network. The load balancing scheme for cluster makes use of two types of the agents specified in MALD, i.e., SMA (server management agent) and LIA (load information agent). The scheme does not use JDA. Job reallocation is accomplished by the SMAs.

In the scheme, every server runs an SMA and an LIA. The LIA is proprietary to a server (called home server). It executes the information gathering policy to visit other servers and collect load information for its home server. In a cluster, an LIA can traverse all servers in short time latency. The SMA executes the information gathering policy, initiation policy, and job transfer policy. It monitors local workload and dispatches the LIA to collect load information when required. The load on a server is measured as:

$$load = w1 * cpu\_load + w2 * no\_connect / MAXCON + w3 * free\_mem$$

where *cpu_load* is the workload on the server, measured in the length of job queue;
*no_connect* is the number of active connections on the server;

*MAXCON* is the maximum number of connections allowed to the server;
*free_mem* is the percentage of free memory space;
*w1*, *w2*, *w3* are the weights of the parameters, *w1*+*w2*+*w3* = 1.

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-initiated policy. Once the load exceeds a threshold, the SMA activates the LIA 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 LIA traverses in the cluster to retrieve the updated values of *cpu_load*, *no_connect* and *free_mem* on each server and calculate the *load* metrics. Meanwhile, the LIA selects an appropriate server (called *best server*) to accept the job reallocation from the home server. As mentioned in Section 3, the server selection policy can adopt two strategies:

- *Find-best:* the LIA visits every server and selects the server who has the lowest load as the best server. When it returns to its home server, the LIA updates the log of global load information and reports the best server to the home server.
- *Find-first:* the LIA pauses at the first server whose load is below a threshold. The LIA selects this server as the best server and reports this selection and the up-to-date load information to the home serve. The LIA remains at the best server, not proceeding to successive servers. Later, the LIA will restart its travel from this server when the home server activates it again. This strategy allows the LIA to perform round-robin traversal in the cluster so that the workload can be fairly distributed to the servers.

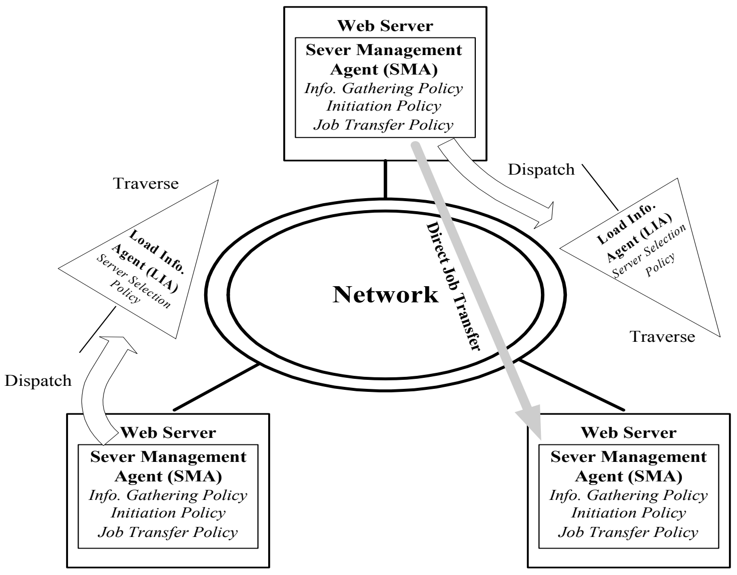


Figure 2 Load balancing scheme for cluster

Figure 2 illustrates the load balancing scheme for cluster. As every server can dispatch proprietary LIA on demand, more than one LIA may be traveling in the cluster to collect load information independently. When a server receives a client request, the SMA on the server can immediately determine a server to process the request. If local load is below a dynamic threshold, the home server processes the incoming request itself. Otherwise, the SMA will reallocate the request to the best server that the LIA recently reported.

The scheme of proprietary LIA is suitable to local-area cluster as an LIA can traverse over all servers in short time latency and report the latest load information to the home server. The log of global load information on each server can be frequently updated so that the SMA can make proper decision on load distribution. In case that the best server is overloaded when job is reallocated to it, it can redirect the request to another server found by its LIA.

## *4.2 Performance Evaluation*

A prototype of mobile-agent enabled web server cluster is implemented on a cluster of PCs using IBM Aglets Software Development Kit [1] and JDK 1.1. The IBM Aglets provides Java API for programming mobile agents (called *aglets*) and an environment for running mobile agents in Java. The prototyped web server cluster consists of eight PCs installed with Linux and Apache web server. All PCs are linked by a 100Mbps switching hub. Among them, three PCs are configured as web servers and other PCs are assigned as clients.

The MALD-based load balancing scheme is evaluated on the cluster by comparing its performance with the *mod_backhand* module [13, 14] that is a load balancing approach based on message-passing paradigm. The WebStone benchmark [19] is used to generate client requests. The WebStone is used to measure the performance of web server software and hardware products by simulating the activity of multiple web clients to create a load on each web server. In the test, the 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 load on a server is denoted by the length of its job queue. 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.

In the test, 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 MALD scheme and the mod_backhand module on three servers at different moment. Table 1 also includes the average deviation of load on the three servers. It shows that the MALD scheme has lower load deviation than the mod_backhand module in most of the cases. That means MALD can distribute client requests more evenly onto the web servers. The *overall average deviation* in Table 1 is the mean of average deviations at all moments. The overall average deviation of the MALD scheme is lower than the mod_backhand module that verifies the better performance of the MALD scheme in supporting load balancing.

Table 1 Load distribution on three servers

| | **Time (minutes)** | **5** | **15** | **25** | **35** | **45** | **55** | **65** | **75** |
|---|---|---|---|---|---|---|---|---|---|
| **MALD** | Server 1 | 65 | 63 | 37 | 74 | 68 | 42 | 30 | 33 |
| | Server 2 | 58 | 79 | 76 | 55 | 48 | 50 | 37 | 34 |
| | Server 3 | 37 | 60 | 75 | 78 | 56 | 51 | 16.6 | 48 |
| | Average deviation | 10.89 | 7.78 | 17.11 | 9.33 | 7.11 | 3.78 | 7.51 | 6.44 |
| | Overall average deviation | 8.74 | | | | | | | |
| | | | | | | | | | |
| | **Time (minutes)** | **5** | **15** | **25** | **35** | **45** | **55** | **65** | **75** |
| **mod_backhand** | Server 1 | 50 | 42 | 7 | 56 | 46 | 30 | 12 | 6 |
| | Server 2 | 43 | 70 | 62 | 46 | 25 | 33 | 20 | 22 |
| | Server 3 | 12 | 40 | 60 | 50 | 40 | 31 | 3 | 41 |
| | Average deviation | 15.33 | 12.89 | 24.0 | 3.56 | 8.0 | 1.11 | 5.78 | 12.0 |
| | Overall average deviation | 10.33 | | | | | | | |

Figure 3 shows the system throughput of two approaches. The throughput of the MALD scheme is close to the mod_backhand in all case. As the prototype of MALD scheme is implemented in Java, the high execution overhead of Java program results in the lower throughput of the MALD scheme when the number of clients is less than 500. However, the mobile-agent based interaction in the MALD scheme can reduce the network traffic, especially when a large number of clients are involved. The MALD scheme can eventually outperform the mod_backhand module above 500 clients.

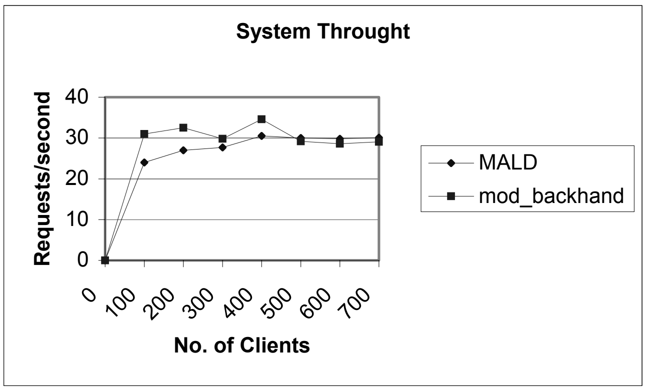


Figure 3 System throughput

Figure 4 compares the network traffic of two schemes. It confirms the lower communication overhead of mobile agents in the MALD scheme than the message passing in the mod_backhand module.

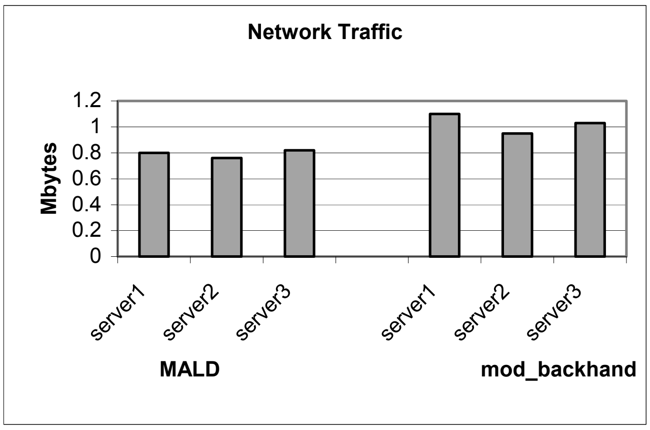


Figure 4 Network traffic

## 5. Load Balancing Scheme on Wide-area Network

### *5.1 The Scheme*

In wide-area network, the web servers are geographically distributed at different sites. If the load balancing scheme for such a system uses the same strategy as for a cluster, the proprietary LIA needs to spend high time latency to traverse all servers in wide-area network and the load information may become obsolete when the LIA 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 wide-area network. The scheme utilizes the *shared* LIA to collect and update the load information and dispatches the job delivering agent (JDA) to deliver jobs to the best server. The JDA can perform job redirection on the fly in case the target server is overloaded.

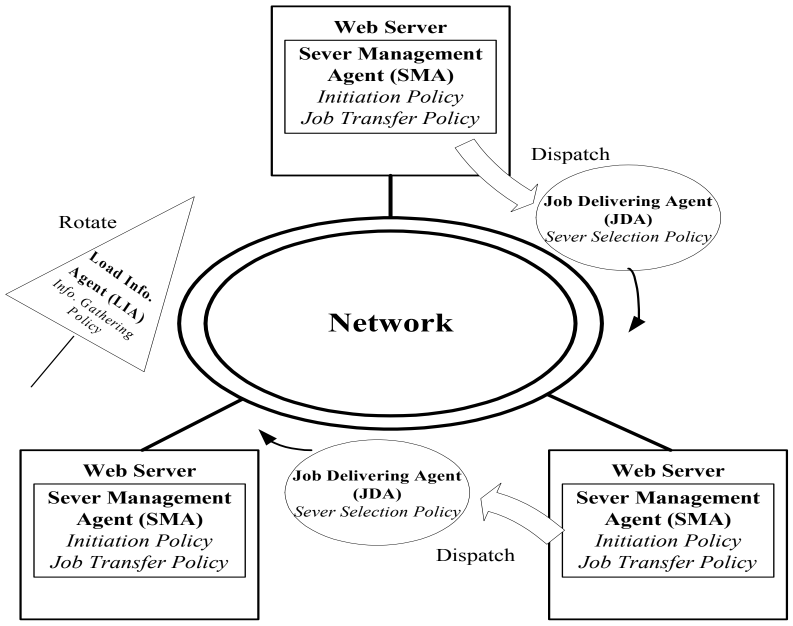


Figure 5 Load balancing scheme for wide-area network

Figure 5 illustrates the load balancing scheme for wide-area network. Every server runs a SMA as the scheme on cluster. The SMA is responsible for executing the initiation policy and job transfer policy, but the server selection policy is handed over to the JDA. Each server receives and processes client requests independently. When a server is overloaded, the SMA initiates the load balancing process by dispatching a JDA to transfer some job to the best server. The SMA selects the client requests from local job queue for reallocation. In a wide-area network, the job selection criterion should also take into account the latency of job transfer. A job on server *A* will be reallocated to a remote server *B* only when

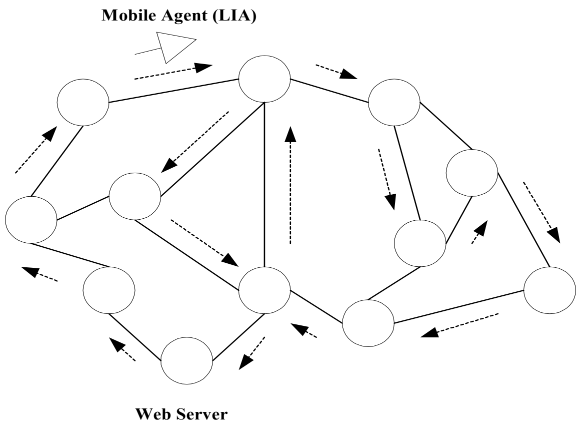
$$t_A > t_{AB} + t_B$$

where $t_A$ and $t_B$ are the expected waiting times of the job on server *A* and server *B*; $t_{AB}$ is the communication latency of delivering the job from server *A* to server *B*. The latency is estimated based on the distance between the servers.

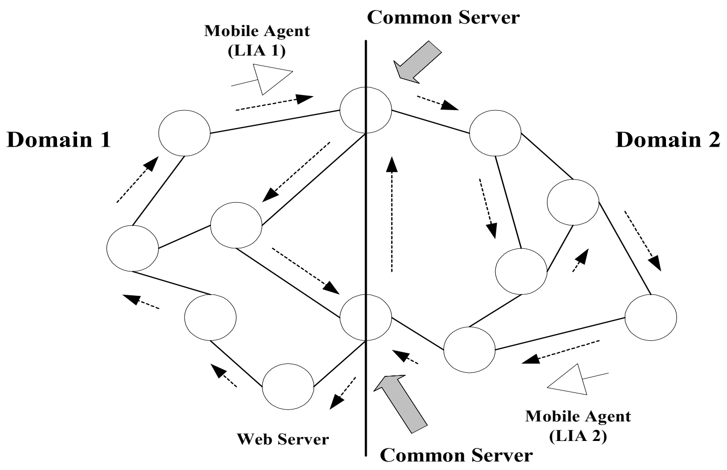
The JDA executes the server selection policy to select a best server to receive the job using the log of global load information. The server selection can adopt the find-best or find-first strategy. Then, the JDA delivers the job to the best server and negotiates with it on the site. If the best server has also been overloaded, the JDA 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 JDA 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 JDA. This feature can improve the flexibility and efficiency of the load balancing scheme.

The load balancing scheme on wide-area network uses shared LIA for load information gathering. A shared LIA keeps on traveling around the servers to collect and propagate the load information. When it arrives on a server, the LIA 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 LIA can continuously propagate the load information to distributed servers.

A web server system can dispatch one or more LIAs to collect load information in the system. Figure 6a shows a single LIA traveling over all servers in a network. For a wide-area network, an LIA takes long time to collect load information on all servers. Multiple LIAs can be used to speed up the global information collection and updating. Each LIA travels within a domain of the network. The load information in different domains can be exchanged at the intersection points between the domains. Figure 6b shows two LIAs traveling in the network that is divided into two domains. Each LIA traverses in one of the domain. They exchange load information on the two common servers on the border of domains using the stigmergy technique (see Section 3) and then propagate the global information to all servers. The use of multiple LIAs can accelerate the information update and improve the accuracy of global load information.



6a. One LIA traveling in whole network



6b. Two LIAs traveling in separate domains

Figure 6 LIA traveling in the network

### *5.2 Performance Evaluation*

To study the performance of the MALD scheme on wide-area network, we have developed a simulation environment using the IBM Aglets 2.1.0 and J2SDK (Java 2) 1.4.1. The environment simulates the work of mobile-agent enabled distributed web servers by running multiple aglets on a single PC. Each aglet represents an SMA on a server. The aglet dispatches a mobile agent of JDA to another aglet when job reallocation is required. The mobile agent of LIA goes on traveling around the aglets to collect load information. The environment provides a GUI for user to specify the parameters of a web server system in the simulation and to display the execution procedure and the performance of the load balancing scheme on the simulated system.

The system in simulation ranges from 1 to 30 fully-connected web servers. Client requests are randomly generated and sent to the servers. The arrival interval of client requests to each server follows exponential distribution with a mean of 200 ms. The processing time for a client request also follows exponential distribution with a mean of 1000 ms. In each run of simulation, totally five hundred requests are generated and processed. The incoming requests may queue at the servers and thus the load balancing scheme will be invoked.

We have simulated three cases of a web server system: using the MALD-based load balancing scheme, using a load balancing approach based on message passing, and running without load balancing. The performance of the simulated system is measured in s*ystem throughput* and n*etwork traffic*. In the mobile-agent based scheme, the network traffic is measured in the number of hops that the mobile agents pass through per second. In the message-passing approach, the network traffic is measured in the number of interactions between the servers per second.

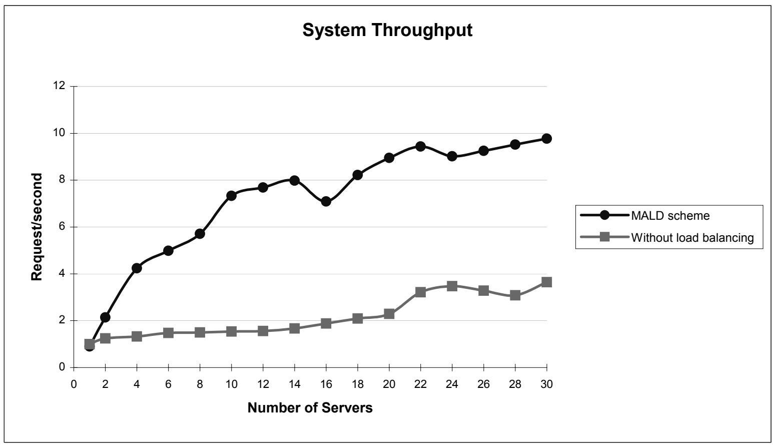


Figure 7 System throughputs of the MALD scheme and the case without load balancing

Figure 7 compares the system throughputs of the MALD load balancing scheme using one shared LIA and the case without load balancing. The result shows that the MALD scheme can obviously improve the system throughput when increasing the number of servers. On contrary, there is not obvious improvement of the throughput in the case without load balancing. In the latter case, the processing capacities of the servers are wasted.

We have also compared the performance of the MALD scheme with different number of LIAs. Figure 8 depicts the system throughputs using one to three LIAs. It shows that the use of multiple LIAs can enhance the throughput on larger number of web servers. In Figure 8, the effect of multiple LIAs becomes apparent on twenty servers and more. Among the three cases, three LIAs can achieve the highest performance and one LIA presents the lowest performance.

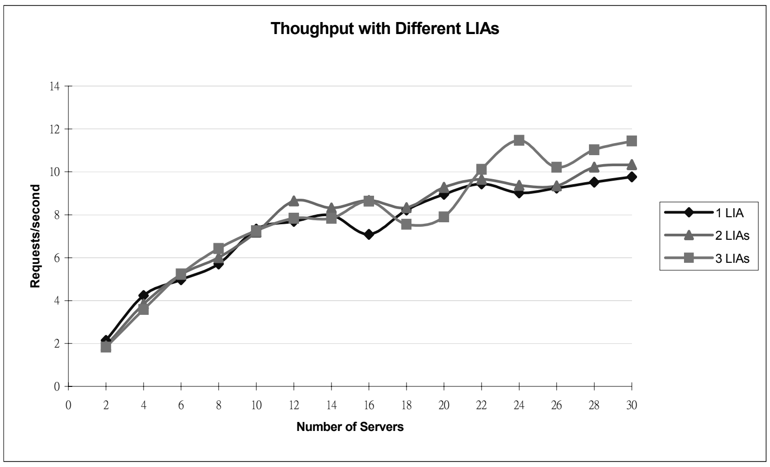


Figure 8 System throughputs with different number of LIAs

Finally, we have compared the network traffic of the mobile-agent based MALD scheme with the message-passing based approach. The MALD scheme uses one LIA to collect and propagate load information. The LIA travels around the servers in constant pace. As Figure 9 shows, the network traffic of the MALD scheme remains constantly at a low level that is independent from the number of servers. In the message-passing based approach, however, every server needs to periodically broadcast load information to other servers. For totally $N$ servers, the broadcast incurs the communication overhead as high as $O(N^2)$. The high network traffic largely restrains the performance of a web server system. From the results of simulation, we can conclude that the mobile-agent based MALD scheme is a communication-efficient approach to support load balancing in wide-area network.

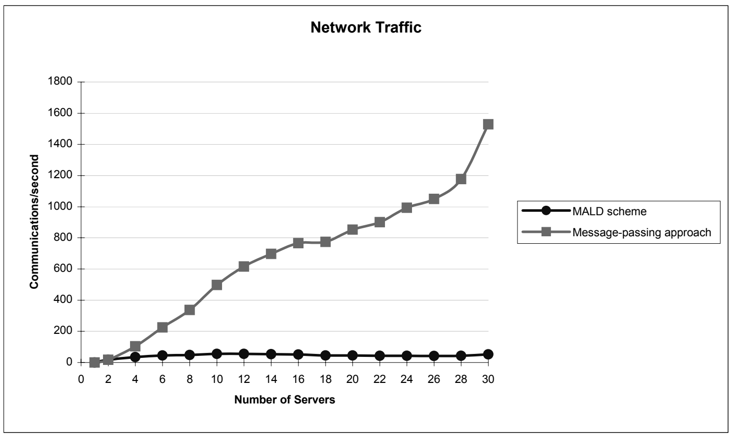


Figure 9 Network traffic in the MALD scheme and the message-passing based approach

## 6. Conclusions

In this paper, we have discussed a MALD framework based on mobile agent technology to support load balancing on distributed web servers. The MALD framework is a flexible foundation to implement different load balancing schemes for scalable distributed web server systems. Two load balancing schemes are developed based on the MALD framework for the web servers in cluster and in wide-area network. The performance evaluations show that the two schemes can outperform the load balancing approaches based on message passing paradigm when large number of servers and client requests are involved.

In the future work, we plan to implement and test the MALD load balancing scheme in a real wide-area environment. We will also define more accurate measurements of load to provide an accurate assessment for the load balancing schemes. The measurements will take into account the load of client requests, including the CPU workload, data size, 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 mobile agents 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.

## Acknowledgements

This work is partially supported by the University Grant Council of Hong Kong under the CERG Grant B-Q518 and the Hong Kong Polytechnic University under HK PolyU ICRG grant A-P202. The authors are most grateful to Lo Siu and Wilson W.S. Yip for their contribution to the implementations of the schemes and the performance evaluation.

## References

[1] Aglets Software Development Kit: http://www.trl.ibm.com/aglets/index_e.htm

[2] A. Bestavros, M. Crovella, J. Liu, D. Martin, Distributed Packet Rewriting and Its Applications to Scalable Web Server Architectures, in: Proc. 6th International Conference on Network Protocols, IEEE Computer Society, Austin, TX, 1998, pp. 290-297.

[3] M. Bui, S. Das, A. Datta, D. Nguyen, Randomized Mobile Agent Based Routing in Wireless Networks, International Journal of Foundations of Computer Science, 12 (2001), pp. 365-384.

[4] J. Cao, G. Chan, W. Jia, T. Dillon, Checkpointing and Rollback of Wide-Area Distributed Applications Using Mobile Agents, in: Proc. 2001 International Parallel and Distributed Processing Symposium (IPDPS2001), IEEE Computer Society, San Francisco, USA, April 23-27, 2001, p. 14.

[5] V. Cardellini, M. Colajanni, Dynamic Load Balancing on Web-server Systems, IEEE Internet Computing, 3 (1999), pp. 28-39.

[6] Cisco DistributedDirector, http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml

[7] D. Dias, W. Kish, R. Mukherjee, R. Tewari, 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.

[8] M. Fukuda, L. Bic, M. Dillencourt, F. Merchant, A Hierarchical Mapping Scheme for Mobile Agent Systems, in: Proc. 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), IEEE Computer Society, Tunis, Tunesia, 1997, pp. 66-71.

[9] J. Gomoluch, M. Schroeder, Information Agents on the Move: A Survey on Load-balancing with Mobile Agents, Software Focus, 2 (2001), pp. 31-36.

[10] D. Lange, M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley Longman, Reading, Mass., 1998.

[11] F. Merchant, L. Bic, M. Dillencourt, Load Balancing in Individual-Based Spatial Applications, in: Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT'98), IEEE Computer Society, Paris, France, 1998, pp. 350-357.

[12] N. Minar, K. Kramer, P. Maes, Cooperating Mobile Agents for Dynamic Network Routing, in: A. Hayzelden, (Ed.), Software Agents for Future Communication Systems, Springer-Verlag, 1999.

[13] mod_backhand, http://www.backhand.org/mod_backhand/

[14] T. Schlossnagle, The Backhand Project: Load balancing and Monitoring Apache Web Clusters, in: Proc. ApacheCon Europe 2000, London, Britain, 2000.

[15] D. MoseDale, W. Foss, R. McCool, Lesson Learned Administering Netscape's Internet Site, IEEE Internet Computing, 1 (1997), pp. 28-35.

[16] W. Obeloer, C. Grewe, H. Pals, Load Management with Mobile Agents, in: Proc. 24th EUROMICRO Conference (EUROMICRO 98), Vol. 2, Vasteras, Sweden, 1998, pp. 1005-1012.

[17] V. Pham, A. Karmouch, Mobile Software Agents: An Overview, IEEE Communications Magazine, 36 (1998), pp. 26-37.

[18] 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.

[19] WebStone, the Benchmark for Web Servers, http://www.mindcraft.com/webstone/

[20] C.-Z. Xu, B. Wims, Mobile Agent Based Push Methodology for Global Parallel Computing, Concurrency and Computation: Practice and Experience, 14 (2000), pp. 705-726.

[21] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, D. Culler, Using Smart Clients to Build Scalable Services, in: Proc. Usenix, 1997, pp. 105-117.

**Jiannong Cao** received the B.S. degree from Nanjing University, China, in 1982, and the M.S. and Ph.D. degrees from Washington State University, USA, in 1986 and 1990, all in computer science. From 1982 to 1983, he studied as a M.S. student in the Graduate School of the Chinese Academy of Science in Beijing. Before joined the Hong Kong Polytechnic University in 1997, where he is currently an associate professor, he has been on faculty of computer science in James Cook University and the University of Adelaide in Australia, and the City University of Hong Kong. His research interests include parallel and distributed computing, networking, mobile computing, fault tolerance, and distributed software architecture.

**Yudong Sun** received the B.S. and M.S. degrees from Shanghai Jiao Tong University, China, in 1985 and 1988, and Ph.D. degree from the University of Hong Kong in 2002, all in computer science. From 1988 to 1996, he was a teaching staff at Shanghai Jiao Tong University. From 2002 to 2003, he had a research position at the Hong Kong Polytechnic University. At present, he is a research associate at University of Newcastle upon Tyne, UK. His research interests include parallel and distributed computing, Grid computing, and bioinformatics.

**Xianbin Wang** received the B.S. degree in computer science from Wuhan University, China in 1992, the M.Eng degree in computer application from Wuhan University of Hydraulic and Electric Engineering, China in 1997, and the Ph.D. in computer application from Wuhan University in 2001. Currently, he is a lecturer at Wuhan University. His current research interests include mobile agent, distributed fault-tolerant computing, distributed algorithms, mobile computing, and Grid computing.

**Sajal K. Das** is a professor in the Department of Computer Science and Engineering and founding director of the Center for Research in Wireless, Mobility and Networking (CReWMaN) at the University of Texas at Arlington. His research interests include resource and mobility management in wireless networks, mobile and pervasive computing, mobile Internet architectures and protocols, distributed processing, and Grid computing. Das received a Ph.D. in computer science from the University of Central Florida, Orlando. He is a member of the IEEE Computer Society, the IEEE, the ACM, the New York Academy of Sciences, and Sigma Xi.