American Journal of Applied Sciences 11 (5): 811-817, 2014
ISSN: 1546-9239
©2014 Science Publication
doi:10.3844/ajassp.2014.811.817 Published Online 11 (5) 2014 (http://www.thescipub.com/ajas.toc)

DISTRIBUTION OF LOAD USING MOBILE AGENT IN DISTRIBUTED WEB SERVERS

Vijayakumar G. Dhas and V. Rhymend Uthariaraj

Ramanujan Computing Centre, Anna University Chennai, India

Received 2012-06-14; Revised 2014-02-07; Accepted 2014-02-28

ABSTRACT

The continuing growth of the World-Wide Web is placing increasing demands on popular Web servers. Many of the sites are now using distributed Web servers (i.e., groups of machines) to service the increasing number of client requests, as a single server cannot handle the workload. Incoming client requests must be distributed in some fashion among the machines in the distributed Web server to improve the performance. In the existing work, reducing the high message complexity is a challenge. This study introduces a novel algorithm, which has low message complexity named Load Distribution by dynamically Fixing input to the server using Mobile agent (LDFM) which distributes the incoming request, as it arrives from the client world, to avoid overloading of the distributed web servers. LDFM uses prefetch techniques to balance the load among the distributed web servers. Mobile agents are susceptible to failure. This issue is also addressed to bring reliability to the algorithm. The simulation results confirm that the proposed method is reliable. The relative improvement in throughput, comparing with the exiting methods is appreciable.

Keywords: Mobile Agents, LDFM, Load Balancing, Distributed System

1. INTRODUCTION

The exponential increase in demand for web services makes distributed multiserver system the preferred choice of solution to handle the requests effectively (Colajanni *et al*., 1998). Balancing the workload in a distributed multiserver is critical to improve the performance (Jie and Kameda, 1998). In a distributed multiserver, load balancing is done by transferring jobs from the overloaded server to under loaded server. In order to successfully carry out load balancing, the load balancing policy is updated with load information of the servers in the distributed environment.

Dispatcher based load balancing in the distributed multiserver system improves the performance of the servers (Pao and Chen, 2006). In a dispatcher based system, the collection of load information about server and scheduling of job among servers is done by the dispatcher. Basically, the scheduling policy and scheduler functions are present in the dispatcher. In existing dispatcher based systems, the dispatcher collects load information from the servers. The collection of load information consumes a lot of network bandwidth as the dispatcher and the multi servers communicate with each other.

The unique contribution of the proposed work is to reduce the message complexity in the system due to load balancing. This results in conserving the bandwidth usage, which can be used effectively for carrying out other tasks like handling more user request. The throughput of the whole system improves.

Mobile agent is used to collect the load information of the server proactively and to communicate the load information to the dispatcher. The mobile management unit in the dispatcher receives the load information and computes the rank of the server. The least loaded server is given a rank of one. Based on the rank of the server, the mobile management unit updates the dispatcher table in the dispatcher. The dispatcher table is dynamically updated. The dispatcher uses the dispatcher table to dispatch the request to the server using round robin scheduling policy.

Corresponding Author: Vijayakumar G. Dhas, Ramanujan Computing Centre, Anna University Chennai, India

Mobile agents are used for load balancing of servers. However when mobile agent is lost the load balancing policy fails (Ciardo *et al*., 2001) and the reliability of the algorithm is compromised. In order to improve the reliability, the proposed work determines the state of mobile agent and creates a new mobile agent, if required.

The remainder of this study is organised as follows. In section 2, some of the related work is reviewed. Section 3, presents the proposed work for load balancing in distributed multiserver system. In section 4, the structure of the experiment set up is explained. Section 5, presents the results from the experimental study. Section 6 contains the conclusion and future work of our study.

2. RELATED WORK

Colajanni *et al*. (1998) have considered load balancing of multi servers and pointed out that overloaded servers and high bandwidth utilization leads the user to spend more time in waiting to access the documents. It is observed that when DNS scheduler is used for load balancing, the scheduler has limited control over scheduling of the job. Once the URL is translated to IP address, it is cached in the client’s browser. Subsequent requests are sent to the same web server resulting in skewed load on the server. In this case DNS cannot control caching.

Load balancing of multiservers in a distributed environment is a job scheduling policy, where a job as a whole is taken and assigned to a server (Jie and Kameda, 1998). In this approach only partial load information is used to determine scheduling of job and the message overhead is also high.

Using one virtual URL name for the clustered web system, the problem of load balancing was approached (Hong *et al*., 2006). In this method http requests are assigned to the server with least load by the IP-address dispatcher. The load information is collected and broadcast to all the web servers, leading to high message complexity.

Pao and Chen (2006) used dispatcher based web server load balancing architecture to improve the performance of popular web sites. The current loading on the back end server is taken in to account before forwarding the request to the least loaded server. The load information is advertised by back end servers and is collected by the dispatcher. This approach also leads to high message complexity.

Reallocation of request from server queue leads to delay in execution. Considerable time is spent in collecting the load information negotiating with the server to accept the job, if the job is not accepted then find an alternate server. The number of hops required before the job is accepted adds to the delay in processing the request. The problem of redistribution of job request is observed in the study of Zhang *et al*. (2005).

Mobile agent enabled approach for load balancing distributed web servers was proposed by Cao *et al*. (2003). They show that load balancing approaches involve frequent message exchanges between the dispatcher and the servers to detect and exchange load information. The message exchange increases the bandwidth utilization and reduces the availability of network bandwidth for other useful purpose. They have pointed out that for effective load balancing, comprehensive and up to date load information should be available.

Aramudhan and Uthariaraj, (2006) (Aramudhan *et al*., 2008) have proposed an approach for load balancing using mobile agents. This method has certain drawbacks. First, consider that the mobile agent is lost then the scheme waits for twice the Round trip time before processing the request. It can be inferred that loss of a mobile agent cannot be left as it is, as this results in low throughput and steps must be taken to correct this situation.

Karjoth *et al*. (1997) have pointed out that mechanisms must be in place to prevent attacks. A host can implant its own tasks or modify the agents’ state. If the agents’ state is changed by a selfish host, then behaviour of the agent can be counterproductive.

In the existing work, it is clear that message complexity and throughput are parameters to be considered for improving the performance of the distributed multiserver environment. In addition, loss of mobile agent, overloading of server, reallocation of request from queue are the other factors that need to be considered.

3. PROPOSED SYSTEM

The literature survey reveals that for effective load balancing, comprehensive and current load information should be available. In the pursuit of achieving the above, excessive bandwidth of the network is utilised in the existing methods for collection and exchange of load information. The limited availability of bandwidth hampers in carrying out other useful work. There is contention between collecting load information and dispatching user request for using the limited bandwidth.

This contention for bandwidth usage must be streamlined for effective utilization of resources. The request is also migrated from server to server and this contributes to the latency in processing the request. Also in the existing system, when mobile agents fail the reliability of the load balancing mechanism is reduced and thereby increasing the possibility of system overload.

The proposed system of Load Distribution by dynamically Fixing input to the server using Mobile agent (LDFM) address the problem of excessive utilization of network bandwidth for collecting and exchange of load information. When mobile agents fail, the problem of reduced reliability of the load balancing mechanism and system over load is also considered. The proposed system consists of a set of clients and a network of servers. The LDFM framework defines two worlds, namely: Client world and server world. The client world is an aggregation of all the clients in the physical world and the server world is an aggregation of the clustered web servers, which are called replicas.

The client world communicates with the server world through the dispatcher. In this system the number of servers in the multiserver environment is configured as fixed number of servers. The fixed number is referred as L. The block diagram of the proposed system is shown in **Fig. 1**. In this system two data structures are deployed, one for collection of the load information from the server and the other organising the same for distribution. The data structure deployed for collection of data from the server is called the queue and organized as:

```
Struct qitem {
int load;
int ipaddr [4];
} mmu_queue [L];
```

The data structure deployed for receiving the ranked data from the queue is called the dispatcher table and defined as:

```
Struct ipdata {
char servr_id [L];
int ipaddr [4];
int rank;
} dispatcher_table [L];
```

The function of the mmu_queue is to receive the server load information according to the IP address of the server. The MMU uses this information to sort the server load in ascending order. Once the server loads are sorted, it is used to update dispatcher table. The dispatcher table now contains the IP address of the servers according to the ascending order of server load. The least loaded server will be the first entry in the table and the heavily loaded server will be the last entry in the table. The table is dynamically updated on collection of server load information proactively. The dispatcher table is shown in **Table 1**.

The dispatcher is a front end machine, has a dispatcher table and a Mobile Management Unit (MMU). The server to which the http request is to be sent for processing by the dispatcher is available in the dispatcher table. When the http requests sent by the user arrives at the dispatcher the dispatcher increments the count value by one. The http request is associated with a Request Identification number (RID). The RID is generated for each request using the 'mod' function. The mod function computes the RID using count value and the number of servers available in the distributed system for serving http requests from the user. The RID value is computed as RID = count mod (L+1). When the http request is associated with the RID, the http request is sent to the server using the IP address of the server available in the dispatcher table by the dispatcher. The dispatcher uses round robin policy for distributing the http requests.

When the request with RID is received at the server, the incoming RID is compared with the previous RID in the queue. The comparison result shows that if the present RID is lesser than the previous RID then the load of the server is communicated to the MMU. The request is processed by the server.

The MMU initiates the mobile agent to the server. Mobile agents are used to collect load information from the server world and communicate to the MMU running in the dispatcher. This enables reduced communication overhead between the mobile agent and MMU. The MMU ranks the server according to the load information received from the mobile agent. The MMU then updates the dispatcher table according to the rank of the server. The MMU is responsible for the mobile agent and make sure that it is active and has not crashed.

The mobile agent periodically sends an I Am Alive (IAA) signal to the MMU. The MMU based on the IAA signal checks the status of mobile agent and creates a new mobile agent when needed. The MMU sets the time out timer on receipt of IAA. When the time out occurs a new mobile agent is initiated from the MMU for the server.

An event based algorithm is developed to handle the various events in the system. The algorithm is presented here.

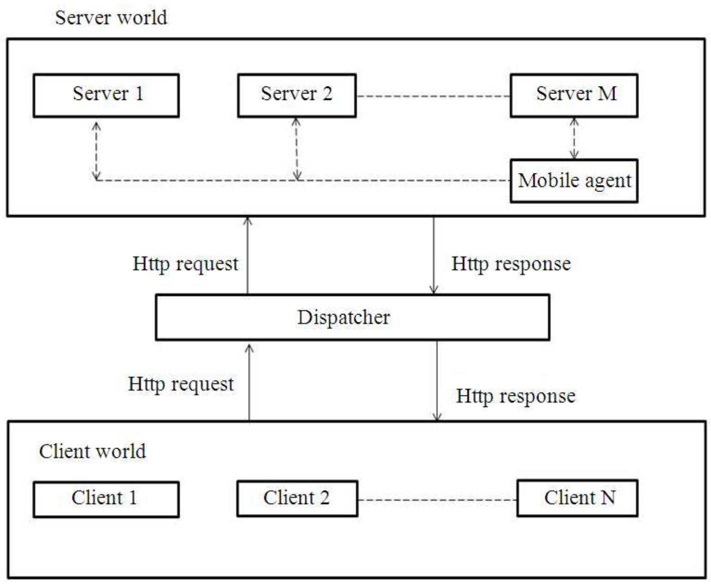

Fig. 1. Proposed system

Table 1. Dispatcher table

Server ID	IP address	Rank
P	192.168.1.100	1
Q	192.168.1.101	2
R	192.168.1.102	3

```
    Algorithm for load allocation
On receiving http request from client at dispatcher
        count = count+1
        RID = count mod (L+1)
        send request with RID to server

On receiving request with RID at server from dispatcher
        if (incoming RID < present RID in the queue)
        send load of server using mobile agent to dispatcher
        running Mobile Management Unit (MMU)
        process the request
On receiving server load from mobile agent at dispatcher
running MMU
        Accept the load of server
        Compute the rank of server
        Update the dispatcher table according to the rank of
        server
On receiving I Am Alive (IAA) signal from mobile agent
at dispatcher running MMU
        Set time out timer
On time out at MMU
        create mobile agent
        move mobile agent to corresponding server
On receiving server response at dispatcher
        Send server response to corresponding client
```

3.1. Updating of Dispatch Table

Each server in the server world has a mobile agent initiated from the MMU so that polling of server by the mobile agent is avoided. This reduces the latency in collecting the load information. This also helps in avoiding unnecessary usage of network bandwidth for communicating load information. At any point in time only one mobile agent will be communicating the load information to the MMU. Initially the dispatcher assigns the user request with RID's to the servers 1, 2, 3….N in a round robin fashion.

This process continues as long as the incoming RID is greater than the present RID. Due to the cyclic property of the mod function the above condition will fail, initiating the mobile agent. This arrangement enables staggered communication between the mobile agent and the MMU. The mobile agent collects the load of the server proactively and sends it to the MMU. The mobile management unit after receiving the information from the mobile agent computes and updates the dispatcher table with IP addresses along with ranks 1, 2, 3… of N servers with lightest load. The ranking of the server is according to the load. The list of IP address of the server in the dispatcher table will be different every time the load information is received and the dispatcher table is updated dynamically. The table is used by dispatcher for allocation of server for new http request.

4. EXPERIMENTAL SETUP

The experiment was conducted using computers in a LAN network. The processors used were of Intel Pentium 4 or core or core 2 duo. The memory capacity was in the range of one to four GB. Simulation was done using Java in LINUX environment. Aglets software development kit AGLETS SDK 2.0.1was used for mobile agents.

The web server processes only http requests. The server processes the request in FIFO (First in First out) manner. User requests are handled only by the dispatcher. In our approach it is assumed that the IP-address dispatcher is not overloaded by http requests. The queue at the dispatcher has a finite buffer and a tail drop discard policy. The dispatcher is highly reliable and has sufficient capacity to process the requests. Mobile agent was implemented using aglets.

4.1. Methodology

Initial values were assigned. The number of server (L) was chosen to be 4. The requests were generated from the clients and sent to the dispatcher. One mobile agent was initiated from the MMU to each of the server. The dispatcher table was initialized with IP address of computers. The load request sent to the computer was assumed to be the same. The initial order of dispatch of request was 1, 2, 3 and 4. The dispatcher was assigned one virtual address. The request from the user will have uniform execution time. The servers are heterogeneous having varying CPU, Memory capacity and configurations. The experiment was conducted by gradually increasing the number of users in the system and the corresponding throughput was taken. The experiment was repeated a number of times and the average value used. The failure of mobile agent was simulated by disabling the IAA signal. On time out the creation of new mobile agent was observed. Similar experiment is done for the system without LDFM and the average was taken.

5. RESULT

Among various performance parameters, system throughput i.e., the total number of request processed per second by the given system is considered. LDFM algorithm is simulated for various numbers of requests at different frequencies and the system throughput is measured.

The system throughput of the server (the total number of request processed per second) is plotted against the number of request from client world. The result of the experiment using LDFM algorithm and without LDFM algorithm is shown in **Fig. 2**.

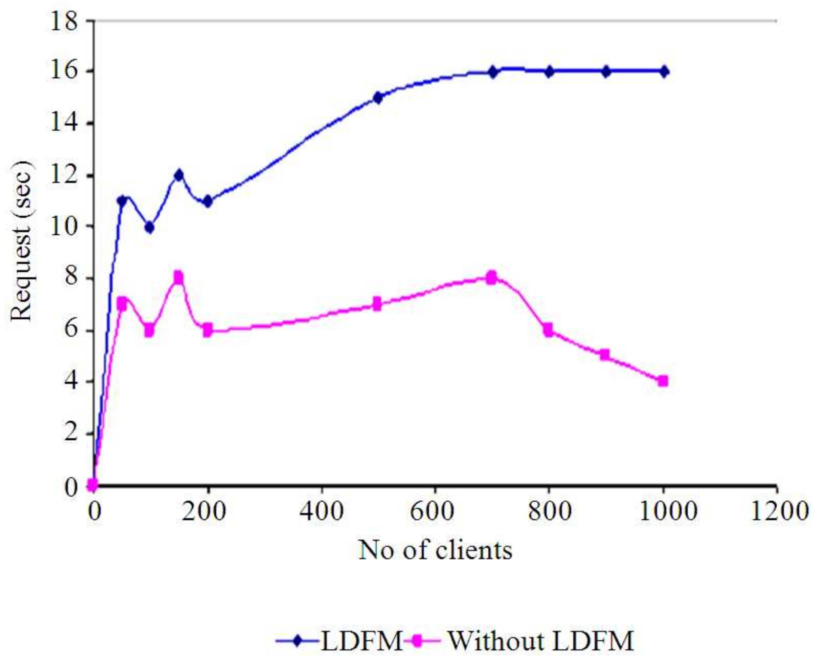

Fig. 2. System throughput

It was observed that without LDFM the throughput of the system gradually gets degraded. With LDFM the system throughput comparatively increases. The result also shows a performance degradation when mobile agent failure was simulated.

6. DISCUSSION

The improvement in performance is attributed to the reduced message complexity of mobile agents. The reduced message complexity compared with other method has contributed to processing more user requests. The reduction in message complexity is achieved by using mobile agent to communicate with the MMU for collection of load information. The mobile agent communicate only when the RID condition fail. The failure of the condition is achieved by the use of 'mod' function. This arrangement helps the mobile agent to communicate always in a sequential manner. In the previous work of Hong *et al*. (2006) the load information was broadcast to all the servers. Also Pao and Chen (2006) have used the advertisement technique to convey the load information to dispatcher from backend servers. The communication is not at all regulated. In the experiment there is no distribution of job request from the one queue to another. The load information was proactively collected, ranked and dispatcher table updated. The sequence of distribution of user request by the dispatcher to the server is altered whenever the dispatcher table is updated.

New mobile is created after timeout, when mobile agent fails. However a performance de gradation was observed and this is due to the fact that in the intervening period the mobile agent is not communicating the load information and the dispatcher table is not updated. The dispatcher table is not update with the current information of load on the server, resulting in the performance de gradation. The same phenomenon was observed in the other works, but the failure condition was restored by our intervention. If the failure condition was not restored, the performance will be poor.

7. CONCLUSION

A load-balancing algorithm that improves throughput and reduces communication overhead due to collection of load information was presented and discussed. The method uses mobile agent and mobile management unit for load information collection, ranking of servers and dynamic update of dispatch table. Reallocation of job request from queue is eliminated. If there is a loss of mobile agent, there is a possibility that all the request end up reaching a single server, leading to adverse effects due to overloading. To avoid this, a fault tolerance system is introduced to keep track of the mobile agent and to restore the mobile agent. Our analysis shows that improvement in throughput is due to reduced message complexity. In case of failure of mobile agent, the reliability of the algorithm is restored with creation of new mobile agent.

It is assumed that the dispatcher is not overloaded. However if the dispatcher is overloaded there is a performance impairment. The mobile agents are prone to attacks, have security issues and needs to be addressed. Additionally, more parameters apart from load of server which help to make the dynamic change of input to the server more optimal can be introduced. More over some priority can be introduced in processing of request in the queue. The improvement to the above limitations can be considered for future work.

8. REFERENCES

Aramudhan, M. and V.R. Uthariaraj, 2006. LDMA and WLDMA: Load balancing strategies for distributed. IJCSNS Int. J. Comp. Sci. Netw. Security, 6: 76-84.

Aramudhan, M., S. Karthikeyan, K. Mohan and V.R Uthariaraj, 2008. ELDMA: Enhanced load balancing decision making using decentralized mobile agent framework. Proceeding of the International Conference on computer and Communication Engineering, May, 13-15, IEEE Xplore Press, Kuala Lumpur, pp: 11-14. DOI: 10.1109/ICCCE.2008.4580559

Ciardo, G., A. Riska and E. Smirni, 2001. EquiLoad: A load balancing policy for clustered web servers. Performance Eval., 46: 101-124.

Colajanni, M., P.S. Yu and D.M. Dias, 1998. Analysis of task assignment policies in scalable distributed web-server systems. IEEE Trans. Parallel Distributed Syst., 9: 585-600. DOI: 10.1109/71.689446

Hong, Y.S., J.H. No and S.Y. Kim, 2006. DNS-based load balancing in distributed web-server systems. Proceedings of the 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, Apr. 27-28, IEEE Xplore Press, Gyeongju. DOI: 10.1109/SEUS-WCCIA.2006.23

Cao, J., Y. Sun, X. Wang and S.K. Das, 2003. Scalable load balancing on distributed web servers using mobile agents. J. Parallel Distributed Comput., 63: 996-1005. DOI: 10.1016/S0743-7315(03)00099-6

Jie, L. and H. Kameda, 1998. Load balancing problems for multiclass jobs in distributed/parallel computer systems. IEEE Tran. Comp., 47: 62-76. DOI: 10.1109/12.660168

Karjoth, G., D.B. Lange and M. Oshima, 1997. A security model for aglets. IEEE Internet Comput., 1: 68-77. DOI: 10.1109/4236.612220

Pao, T.L. and J.B. Chen, 2006. The scalability of heterogeneous dispatcher-based web server load balancing architecture. Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies, Dec. 4-7, IEEE Xplore Press, Taipei, pp: 213-216. DOI: 10.1109/PDCAT.2006.110

Zhang, Q., A. Riska, W. Sun, E. Smirni and C. Gianfranco, 2005. Workload-aware load balancing for clustered web servers. IEEE Tran. Parallel Distributed Syst., 16: 219-233. DOI: 10.1109/TPDS.2005.38