ISSN: 0067-2904

A Load Balancing Scheme for a Server Cluster Using History Results

Husam Ali Abdulmohsin*

Department of Computer Science, College of Science, University of Baghdad, Baghdad, Iraq

Abstract

Load balancing in computer networks is one of the most subjects that has got researcher's attention in the last decade. Load balancing will lead to reduce processing time and memory usage that are the most two concerns of the network companies in now days, and they are the most two factors that determine if the approach is worthy applicable or not. There are two kinds of load balancing, distributing jobs among other servers before processing starts and stays at that server to the end of the process is called static load balancing, and moving jobs during processing is called dynamic load balancing. In this research, two algorithms are designed and implemented, the History Usage (HU) algorithm that statically balances the load of a Loaded Web Server (LWS) and the Massage Passing Optimization (MPO) algorithm. HU algorithm is used at the Domain Name System (DNS) side is to minimize the time required to execute the client requests based on using history results available in one of the Web servers in the server cluster. The MPO is for optimizing the message passing between the Web servers and the DNS. This message optimization will lead to optimize the processing time in the DNS cluster required to find under-loaded Web servers that will perform the request.

Keywords*:* static load balancing, dynamic load balancing, Supporting Web Servers (SWS), History Usage (HU) Algorithm, Loaded Web Server (LWS), Massage Passing Optimization (MPO), Domain Name System (DNS).

توزيع مهام مجموعة خوادم بأستخدام النتائج القديمة

حسام علي عبدالمحسن*

قسم علوم الحاسوب، كلية العلوم، جامعة بغداد، بغداد، العراق

الخلاصة

موازنة الحمل من المواضيع التي جذبت انتباه اكثر الباحثين في العقد الاخير من الزمن. موازنة الحمل تؤدي الى عاملين، تقليل وقت معالجة البيانات و تقليل استخدام الذاكرة لذلك تعتمد الشركات على هذين العاملين في تقييم اي نظرية تخص موازنة الحمل. هنالك نوعان من نظريات موازنة الحمل، النوع الثابت، هذا النوع يقوم بتوزيع المهام من جهة الخادم المحمل بالمهام الى الخوادم الغير محملة بالمهام قبل بدء المهام، و هنالك النوع المتحرك، و هنا من الممكن ان يتم توزيع المهام حتى اثناء تنفيذها. استخدام مصادر و امكانيات خوادم غير مشغولة او غير مستخدمة سوف تكسب المنظومة وقت و تلغي الحاجة الى اي تطوير في المنظومة و التي تكلف الكثير. في هذا البحث تم تصميم خوارزميتين، خوارزمية استخدام التأريخ (HU) و خوارزمية تقليل مناقلة الرسائل (MPO). يتم استخدام نظرية ال HU عند جهة نظام مجال الاسم (DNS) و الذي يقوم بتوجيه المهام المرسلة من قبل الزبون الى اي خادم ضمن المدى و الذي قام بتنفيذ نفس المهام في الماضي و يتم استخدام خوارزمية ال MPO لغرض تقليل مناقلة الرسائل بين الخوادم و ال DNS و بالتالي تقوم الخوارزمية بتقليل وقت التنفيذ في ال DNS من خلال تقليل وقت ايجاد خوادم في حالة ركود او غير محملة و التي سوف تقوم بتنفيذ المهام.

*Email:husamex@yahoo.com

1. INTRODUCTION

Web servers in now days have been involved in many aspects of life for billions of users through the internet. The great increase in Web servers and clients require the achievement of reliability, availability and scalability of Web servers to produce fast service and high productivity for the clients' requests received at all time. The most approach used to improve the performance of the Web servers is the distributed Web server. A group of Web servers construct a cluster that represent replicated resources that can be used to serve the frequent requests from clients. When a client sends a request, that request can be distributed through the DNS cluster to every server in the cluster according to a load status distribution algorithm to serve the client in the best time.

The load balancing technique is important to enhance the outcomes of the distributed Web servers. To reach the distributed Web servers to the best performance, clients requests should be distributed between the Web servers in the cluster through the DNS according to a load balancing technique to serve the client within the best time [1, 2] as a static load balancing system, and the load of the over-loaded Web server has to be assigned to another under-loaded Web server to enhance the performance of the cluster and gain the full utilization of the resources. Any node in the network is to be flagged as over-loaded or under-loaded according to a load threshold determined by the designer of the load balancing system. The load threshold in this paper is 60%, any Web server load less than this threshold is flagged as under-loaded and flagged as over-loaded otherwise.

Massage passing between the Web servers is the basic idea of implementing load balancing on distributed Web servers, those messages are used to exchange the load status between the Web servers in the cluster and with the DNS [1, 3]. A new technology in load balancing is called mobile agent technology that can separate the main functions execution time of the Web server from the time required to serve the client request. Mobile agents travel from one Web server to another collecting the load information required, therefore, mobile agents need low network traffic than packet-passing approach, because there will be no message passing between the Web servers anymore [4].

Using the resources of servers that are in idle mode or have less load than the loaded server, such servers will be referred to as Supporting Web Servers (SWS) in this research, will gain processing time to the system and avoid extra hardware expenses to develop the loaded server when there are servers not using their resources in full capacity. The Web servers can be connected in different network types, such as through a Local Area Network (LAN) to act as one powerful Web server or it can be connected as a Wide Area Network (WAN) each Web server at a different site. The idea of using the distributed Web server is to provide scalability, because they can easily engage in corporate additional Web servers as a solution for the growth of the clients requesting services. One of the advantages of the Web server cluster is that when a failure occurs in one of the servers, other servers in the same cluster can serve instead of the failed server [4].

Managing the resources in the Web server cluster in a professional way will improve the outcome of the overall system and prevents the inconvenient reactions of the system like the turn-around times of client requests [5].

Load balancing is classified into two types, static and dynamic. The concept of static load balancing is to balance the load before processing starts and the job remains at that server until the end of the process. The concept of dynamic load balancing is to balance the load by sending a job from the over-loaded server to another under-loaded server during runtime, because the load status of the current Server change to loaded [6, 7].

The main purpose of load balancing to transfer new requests to an idle node in a static load balancing system, or transfer already processed requests from the loaded node to the idle node in a dynamic load balancing system, both technologies will lead to enhance the performance of the overall system, therefore the main idea of load balancing to reach the perfect utilization of each resource in the environment [4].

The main drawback in any load balancing algorithm in a Web server cluster is the knowledge needed to determine the load status of each node in the cluster and the absence of scalability, both needed to achieve load balancing [4].

The HU uses the history search results in one under-loaded Web server to serve a client requests. The system enables the Web servers in the cluster to send packets between them and the DNS, to enable the DNS to determine the load status of each one of them at a certain period of time. In this research we used the habitual Packet-passing based load balancing mechanism, in this mechanism the

Web server merges its main functions with the maintenance functions like the load balancing function [4]. In the packet-passing approach, the Web server has to exchange packets holding load information with the DNS in order to take load balancing decisions. The exchange of the packets with the DNS requires high communication efforts, like time and size, and this can lead to the decadence of the overall system. Therefore, using the history results of one of the Web servers will avoid the overall system time execution needed to execute the request again.

This paper explores in details the load balancing algorithm designed and implemented in this research, the HU algorithm, and all the sub-functions related to the algorithm. Section 2 in this paper shows the related works. Section 3, shows the experimental side of our work. Sections 4, describes the HU Algorithm methodology in details. Section 5, shows the evaluation details of our work. Section 6, discusses the conclusion out of this work.

2. RELATED WORK

In 2003, Cao J., Sun Y., Wang X., Das S. K., proposed a scheme named MALD (Mobile Agent based LoaD balancing) that create mobile agents that can reach distributed Web servers to add scalability to the load balancing scheme designed. The duty of the mobile agents is to collect load information from all Webs servers and also perform load balancing in the scheme. One of the main obstacles of this scheme is the incorporation of the network environment with such versatile agents, they consider the agent unauthorized to move from one Web server to the other collecting information and performing tasks. The advantage of using the mobile agents in load balancing rather than message-passing, is the flexibility, low traffic in information movement and high synchronization between the Web servers, each Web server knows the status of all other Web server all the time. During the evaluation of the MALD scheme, it showed that the MALD provided an efficient load balancing results reaching a wide number of Web servers in the internet [4].

In 2007, Nehra N., R.B. Patel, V.K. Bhat, proposed a Multi-agent load balancing scheme in a heterogeneous environment. To enhance the performance of a parallel computing cluster and decrease the execution time needed, the Distributer Dynamic Load Balancing (DDLB) essential function is designed to distribute the load among the available processes. But instead of load balancing the jobs by migrating them from one process to another less loaded one as like in ordinary dynamic load balancing, they intend to split the jobs into sub-jobs and then load balance them to nodes in the cluster. In order to implement their approach, they designed a mobile agent that will be used to balance the load in the cluster. In this study, they used the number of jobs waiting in the queue to be processed represent the load of that node in the cluster. Many types of agents and their policies they searched in this work to choose the best type compatible with the requirements of their proposed scheme. Many mathematical functions and matrices were used to measure the performance of their proposed scheme with the performance of the existing message passing schemes. This scheme was implemented on a network of multiple LAN's using Platform for Mobile Agent Distribution and Execution (PMADE). The results obtained from experimenting this scheme showed that it is more efficient that existing ones [8].

In 2010, Eludiora S., Abiona O., Aderounmu G., Oluwatope A., Onime C., Kehinde L., studied the Un-regulated migration of jobs between the Web servers, and addressed the cause and solution of the problem and the frequencies lost in the bandwidth for this unnecessary migration. To achieve bandwidth optimization, it is necessary to design a policy that can determine the consumption occurring in the bandwidth because of the jobs migration from server to server. The aim of their research is to organize the migration of jobs between Web servers to decrease the bandwidth consumption. The designed policy in the work is called Cooperative Adaptive Symmetrical Initiated Dynamic/Diffusion (CASID), was programmed using Java Developer, to create a middle ware service medium as an agent based, to simulate the jobs distributed among the Web servers according to some criteria, such as, no job are allowed to migrate from Web server to another if all servers are busy and that job has to remain at the same Web server for processing. The results of this work was compared with the outcomes of the existing schemes, and the CASID scheme proved that its response time is better and the bandwidth needed is much lower [9].

In 2014, A. Paulin Florence, proposed an approach to enhance to resource utilization by providing a professional load balancing on all the resources in the server cloud environment. First, the load index of all resources are computed through a load model, using the memory usage, access time to that certain resource and CPU usage. Once all the load indexes are computed, all resources will be

assigned dynamically to a correspondent node in the cloud environment that needs to be load balanced. The process of assigning resources to over loaded nodes is an optimal distribution problem in load balancing studies and researches. Here comes the need for optimization algorithms, like Genetic and modified genetic, but these algorithms don't find the best neighbor solutions because it doesn't overcome the exploration problems. That why using the effective optimization procedure can lead to better results as load balancing decisions than the genetic algorithms. The firefly optimization algorithm was used to take the load balancing decisions. An indexed table and the index load computed previously are used by the firefly optimization algorithm, the table will represent the availability of the under-loaded Web servers and the queue of requests. Depending of the results obtained by the firefly algorithm, load balancing decisions are carried out. Based on analyzing the results obtained by this approach, it was noticed that the approach proposed is an efficient load balancing algorithm and efficient in schedule optimization [10].

3. EXPERIMENTAL Environment and setting

The HU algorithm is implemented in a multi LAN at the University of Baghdad, college of science, computer science department. The lab contained 21 nodes, one of the nodes acts as the DNS cluster, six of them were installed as Web servers, which represented the server cluster in our paper, and fourteen nodes were installed as clients, each one of those clients represented unknown number of virtual clients, the purpose of creating a big number of clients is to test the HU algorithm and check how much load will be balanced and determine the optimization time. Windows 10 was installed on all the nodes in both LAN's. The reason behind using windows 10 is because of the high security communication provided. The firewall of the operating system was on during the experiment of the project. Some configurations were needed to the firewall of the operating system to allow transferring the packets through the LAN's. We needed to fix the configuration of the LAN's, to create a Sub work group cluster (same subnet) for all six servers, so each packet flowing in the cluster will reach all the servers to the port specified in the packet. One Microsoft Access Databases was installed on all servers, the Database contains four tables, the first table is the Under-Loaded Status table (ULS), used to store the load status of the under-loaded Web servers, the second table is the Standard Table (ST), used to store 1000 SQL standard statements that will be used randomly as client requests, the third table is the History Table (HT), used to store the execution history, that will be used to refer to text files that contain the results of a certain SQL statement, store all the SQL requests executed at that server, the fourth table will be used to store the data set that is a Citizenship information Database, that contain 25 piece of information of each citizen (first name, second name, third name, tribe, height, weight, Eye color, etc.), this table will be stored at each server in the cluster to be used for the execution of the SQL statements received by the clients. The system will go through a self-learning phase, that represents the first 60min of the project life, then the evaluation process will start, this 60min will be used to fill in the history table and the under-loaded status table, so the DNS cluster can take the right decisions through this work.

4. METHODOLOGY

In this work we proposed two new algorithms, the HU algorithm that can load balance the load in the DNS cluster by getting advantage of the Web server history search results stored. The load balancing will be done through forwarding the instant request received by the DNS cluster to all under-loaded Web servers that each one will check if it has served the same request in the past or not. The second algorithm is the MPO algorithm for the message passing between the Web servers and the DNS cluster, in this algorithm, all the load packets received by the Web servers that are over-loaded will be neglected by the DNS cluster. The use case diagram of the project is shown in Figure-1.

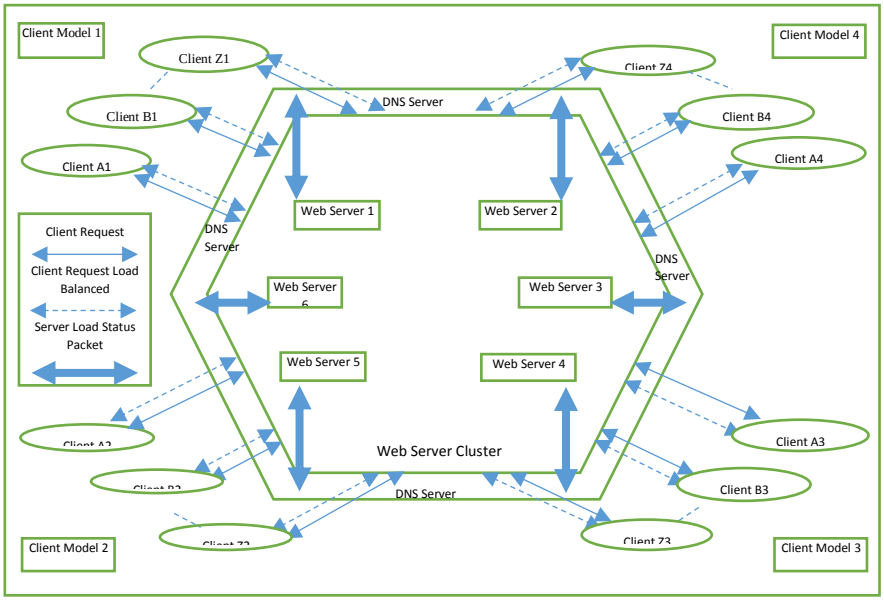

Figure 1- The use case diagram of the HU algorithm

There are 1000 request in the ST as mentioned previously, The 1000 requests are selected randomly, but each will be executed for a fixed number of times controlled by a counter in the ST, each request that will reach its maximum execution times will be deleted from the ST to insure that it will not be selected randomly any more. This fixed number of execution times is to insure fair evaluation for the overall system with and without the proposed algorithms.

When a client sends a request, the DNS cluster will recieve the request and search for a Web server in the Web server Cluster, for example Web Server Y, to balance the load with under two conditions, first Web server Y has to be under-loaded, second, it has performed such a request in its history results. The DNS will send the client request and IP to Web Server Y, that will inturn execute the request and send the results directly to the client. To achieve the load balancing in this designed HU algorithm, many funtions have to be performed, those functions will be described in details.

4.1 Load Status Calculation Function

This function exists in each web server in the cluster is to compute the load value that will be from 0 to100 as shown in the Pseudo code in Figure-2. The main idea of computing the load of any node in the network is to compute its accumulated load in a period of time and to not calculate the load in a specific second or milisecond because unrealiable results will be gained, like the CPU is busy this milisecond while its not the milisecond after or vise-versa.

The CPU load of the Web server shown at Figure-2 (#4), is calculated through the Java function shown in Figure-3. To be mentioned that the result of this function is how many ms have the processor being processing the curent process regardless of the multiprocessing and all the paralell operation the CPU performs. As we mentioned previously in the experimental section, that the system will be operated for 60min then the load balancing will start operating, one of the reasons is to calculate the CPU_LOAD. This function operates in a loop to find the summation of the CPU_LOAD for the last 60ms, then devide it by 60. The purpose of looking at the last 60 milliseconds of the CPU performance will give us more reliable results about the CPU load status.

```
#1 The CPU load of the last 60 milliseconds are accumulated in variable CPU_LAOD;
#2 While (True)
#3 Wait for 2 ms;
#4 CPU_LOAD = (New_CPU_LOAD + OLD_CPU_LOAD) / 2; Calculate the CPU usage at the Server
#5 Calculate Memory Usage at the Server;
#6 Calculate the Active Requests at the Server;
#7 Calculate the Load of the Server;
#8 Send the Load of the Server to the DNS;
#9 End of while
```

Figure 2- The Pseudo code of the function calcualting the web server load status.

```
CPU_LOAD = OperatingSystemMXBean.getSystemLoadAverage() /
OperatingSystemMXBean.getAvailableProcessors() (load average per cpu)
```

Figure 3- The CPU usage Java function

The memory usage of the Web server is calculated through the two Java functions shown in Figure-4. The output of the two functions is in Byte.

```
TOTAL_MEM = OperatingSystemMXBean.getTotalPhysicalMemorySize();
FREE_MEM    = (OperatingSystemMXBean.getFreePhysicalMemorySize() * 100) /
            TOTAL_MEM;
```

Figure 4-The Memory usage Java function

The function used to calculate the estimated number of Active requests is shown in Figure-5.

```
Public double getEstimatedAverageNumberOfActiveRequests(){
return getRequestPerSecondRetirementRate() * (getAverageServiceTime() +
getAverageTimeWaitingInPool()); }
```

Figure 5-The Number of Active Requests in a Java function

The maximum number of connections to the Server was fixed in the program to the value 150 as shown in Figure-6.

```
ServerSocket Server = New ServerSocket (System.out, 150);
```

Figure 6-Fixing the Maximum Number of Connections to a Server in Java function

The function used to calculate the load of the Web server is shown in Figure-7.

```
Load = w1 * CPU_LOAD + w2 * NAR / MC + w3 * (TOTAL_MEM - FREE_MEM)
```

Figure 7-The Web server load calculation function

Where CPU_LOAD is the workload on the server, measured in the length of job queue, Number of Active connections on the Server (NAR) is the number of threads created at each milliseconds, Maximum number of Connections (MC) allowed to the server, which equals 150 in our work.

FREE_MEM is the percentage of free memory space, w1, w2, w3 are the weights of the parameters, w1+w2+w3 = 1, [7]. Here we decided to equal the weights of w2 and w3 because both parameters are similarly important, and the value of w1 will be 0.4 for its higher importance in load balancing. At the end of calculating the load, it will be sent to the DNS as shown in Figure-2 (#8).

4.2 Updating Under-Loaded Status (ULS) Table function

Under-loaded Web server sends a packet called Load Packet (LP) to the DNS Cluster to updating its load status, the DNS will check instantly the load value stored in the received packet, through the MPO algorithm propossed, if it is above 60, the packet will be neglected, if not, the ULS will be updated as shown in the Pseudo code in figure 8. The MPO was supported with a new function called the monitoring function (MF), the job of this function is to scan the ULS every 2ms, and delete the entry of any Web server in the cluster that has not updated its status in the last 2ms, because that means that the Web server of the entry deleted is newly over-loaded, and thats the reason for not sending its LP. The ULS at the DNS will be updated all the time, Figure 9, shows the ULS at a certain moment. The LP sent to the DNS stores the load and IP of that Web server sending the LP, the load status is a value calculated by a function that will be discussed later in this work. To be mentioned, the Web servers status in the ULS are sorted in an accending order, so the DNS can take priority decisions according to the less under-loaded Web server.

```
#1 While (True)
#2 The DNS is Listening at the port 2100 to receive any load packet;
#3 if the load value in the packet received > 60
#4 Neglect packet;
#5 else Update ULS table
#6 End of while
```

Figure 8- The Pseudo code of function updating the under-load status table.

ULS

ID	IP	Load Status	Server Nan	Click to Add
5	192.168.1.3	46	S1	
6	192.168.1.4	36	S5	
7	192.168.1.5	47	S6	
(New)				

Figure 9- The Under-Loaded Status Table (ULS) at a certain moment.

4.3 The History Function (HF)

When the DNS recieves a request, it will operate the HF to determine which Web servers in the ULS has performed the client request in its history. The HF will start sending packets called the Client Request Packet (CRP) to all under-loaded Web servers in the ULS to check if one of them has performed the client request in the past or not. The DNS will send a Request Packet (RP) to the first under-loaded Web server that will respond to the DNS CRP, that will inturn reply to the client request and send the results directly to the client. When the CRP is sent, all under-loaded Web servers that received the CRP will look in its History table (HT) to check if it has performed such a request in the past or not, if not, the under-loaded Web server will send a Negative Packet (NP) that contains the value (0) to the DNS, If it did find the client request in its HT, it will send a positive message that contains the value (1) to the DNS notifying him that it has served such a request and he will take the load and serve that client on behalf, at this moment the DNS will send a packet called the Neglect Packet (NOP) to all web servers that didn't respond till that moment to ignore the CRP that holds the IP of the client. The response packet regardless if it is negative or positive, will be sent to port (2200) at the DNS cluster side. If the DNS didn't receive any packet from any Web server in the ULS tabel (no under-loaded Web server has performed the same request before), it will go back to the first Web server in the ULS (the least under-loaded Web server) and send the request to that Web server for manupulation and finding results and sending the results directly to the client. Figure 10, shows the pseudo code of this function.

#1 While (True)
#2 The DNS is listening at the port 8080 to receive any client request;
#3 wait until client request received, then send a packet called the Client Request Packet (CRP) that contains the client request, to all under-loaded Web servers in the ULS; all web servers will check in their HT, and send a positive or negative response packet;
#4 DNS wait for any under-loaded Web server response at port 2200 for 4ms maximum; if the DNS didn't receive any response packet during the 4ms, go to #10.
#5 when a response packet received at port 2200, DNS will check if it is a negative packet or not;
#6 if the response packet is negative; go back to #4;
#7 If the response packet is positive;
#8 The web server started responding to the client request.
#9 The DNS will send the NOP to all web servers that didn't respond to the CRP, to neglect the CRP; go to #11;
#10 Send the CRP again to the the least under-loaded Web server in the ULS to perform the client request.
#11 End of while

Figure 10-The Pseudo code of function updating the under-load status table.

5. EVALUATION

We have used the same 1000 requests every time we evaluate the system with and without the HU algorithm. The 1000 requests are selected randomly but each are executed for a 100 number of times to equally evaluate the system with and without the HU algorithm every time. So the total number of requests that will be executed will be 100,000 requests. The total number of requests served with and without the HU algorithm is equalled. We can get from the results obtained from experiment number 1, that the experiment total time required to execute the 100,000 requests using the HU algorithm is 2.41.02 h/m/s and 2.58.16 h/m/s without using the HU algorithm. Experiment number 2, shows that the experiment total time required to execute the 100,000 requests using the HU algorithm is 2.39.42 h/m/s and 2.56.23 h/m/s without using the HU algorithm. Experiment number 3, shows that the experiment total time required to execute the 100,000 requests using the HU algorithm is 2.41.57 h/m/s and 2.57.09 h/m/s without using the HU algorithm. As we can see from the results that the performance of the system is faster with the HU algorithm.

Another kind of evaluation was done to the system with and without the HU algorithm. We choose a 20 second random period of time, we noticed that the number of requests processed in that period using the HU algorithm is higher than without using the HU algorithm as shown in Figure 11. The total number of requests served using the HU algorithm in the 20 seconds is 202 and 198 without using the HU algorithm, taking in consideration the same sequence in process execution. Figure 11 shows that the Average CPU and Memory usage when using the HU algorithm is 5152 kB/s which is higher than without using it which is 5085 kB/s because of storing the old requests results and the extra message passing between the servers.

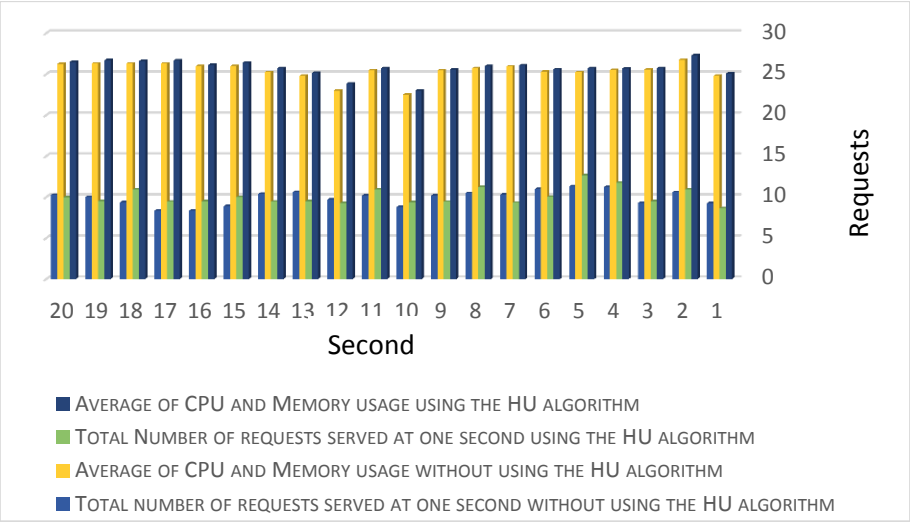

Figure 11- The System Specification Comparison With and Without Using the HU Algorithm

6. CONCLUSION

Reaching high performance results in DNS clusters load balancing is a hard challenge for most of the researchers. In this work we aimed to minimize the response time required to meet the clients requests by using the history results in the Web servers available in the cluster to optimiz the time required to manupulate the clients request, becuase most of the clients request in reality are redundent.

Through the experiments in the lab two conclusions were reached, first, the time required to server the clients was optimized by using the HU algorithm, second, it was noticed that many packets were used to perform the algorithms designed, this message passing caused extra data movement in the network trafic, that caused some delay in the network.

Performing the load balancing algorithm using the massage exchange technology uses more processing time and resources, it will be better to add mobile agents that move from one Web server to another to decide which Web server is better to execute the client requestand to reduce time and information exchnaging between the servers in the cluster. The message passing between the Web servers and the DNS can be reduced also by using message optimization methods.

REFERENCES

1. Cardellini V. and Colajanni M. **2000**. Dynamic Load Balancing on Web-server Systems, *IEEE Internet Computing*, 3 (1999), pp:28-39.
2. Tang W. and Mutka M. **2000** 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, pp:127-133.
3. Dias D., Kish W., Mukherjee R. and Tewari R.**1996**. A Scalable and Highly Available Web-Server, in: Proc. 41st International Computer Conference (COMPCON'96), IEEE Computer Society, San Jose, CA, pp:85-92.
4. Cao J., Sun Y., Wang X. and Das S. K. **2003**.Scalable Load Balancing on Distributed Web Servers Using Mobile Agents, Internet Computing and E-Commerce Lab, Department of Computing, The Hong Kong Polytechnic University, *Journal of Parallel and Distributed Computing*, 63(10), pp:996-1005.
5. Arora M., Das S. K.**2002**. A De-centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments. Proc. of Int. Conf. Parallel Processing Workshops, 499, IEEE, Washington.
6. Gan B. P., Low Y. H. and Jain S. **2000**. Load Balancing for Conservative Simulation on Shared Memory Multiprocessor Systems, Gintic Institute of Manufacturing Technology 71 Nanyang Drive Singapore 638075{bpgan,yllow,sjain}@gintic.gov.sg.
7. Cardellini V., Janni M. C. and Yu P. S. **1999**. Dynamic Load Balancing on Web-server Systems, *IEEE Internet Computing*, 3(3), pp:28-39.

8. Nehra N., Patel R.B. and Bhat V.K. **2007**. A Framework for Distributed Dynamic Load Balancing in Heterogeneous Cluster, *Journal of Computer Science*, 3 (1), pp: 14-24, 2007 ISSN 1549-3636, Science Publications.

9. Eludiora S., Abiona O., Aderounmu G., Oluwatope A., Onime C. and Kehinde L. **2010**. A Load Balancing Policy for Distributed Web Service, *Int. J. Communications, Network and System Sciences*, 3, pp:645-654 doi:10.4236/ijcns.2010.38087 Published Online August 2010, IJCNS.

10. Florence A. P. and Shanthi V. **2014**. A Load Balancing Model Using Firefly Algorithm In Cloud Computing, *Journal of Computer Science* 10 (7), pp: 1156-1165.