CS213 Parallel and Distributed Processing
Project Report

Performance Study of Load Balancing Algorithms in Distributed Web Server Systems

Zhong Xu Rong Huang

Department of Computer Science and Engineering
University of California, Riverside
{zhong, rongh}@cs.ucr.edu

Abstract

The explosion of the WWW has triggered great interest on distributed web server systems. Among various distributed web server architectures, DNS-based dispatcher system is a promising solution in terms of performance, scalability and availability. In this report, we experimentally compare performance of load balancing algorithms of DNS-based dispatcher systems and the new proposed handoff system, in terms of load balance and average response delay. Simulation shows that Round Robin algorithm performs badly especially under non-uniform traffic and taking advantages of both servers load information and domain information could improve the performance. Simulation also shows that Handoff approach has excellent performance in terms of load balance but its average response delay depends a lot on the computation capacity of the Distributor.

1. Introduction

The popularity of the WWW has made web traffic the fastest growing component of Internet traffic. The congestion over the network and overloaded web servers cause users to spend more time on waiting response from web servers. To reduce the WWW traffic over the Internet as well as the response delay for accessing files, caching and replication are two most popular methods. While caching could efficiently reduce the network traffic, only replication could be helpful for reducing web server's response latency and increasing document availability [4].

For most popular web sites, the replication of information across independent or coordinated mirrored servers is becoming a common choice, and among various solutions, a distributed web server system is the most promising one [1]. The distributed web server systems do provide the

transparency for users, because users do not need to manually select URL sites or IP addresses, and they have much better scalability and availability. However, loads of servers should be carefully balanced; otherwise, the performance of web server systems would be degraded and then the benefits of this architecture would not be yielded.

An intuitive approach for load balancing among web servers is the IP-dispatcher approach. The main idea of this approach is to use a front-end dispatcher to distribute incoming requests among a set of clustered servers (Fig. 1). The IP dispatcher approach has full control over incoming requests and has a very good performance of load balancing. But with this approach, the dispatcher needs to modify the head of every packet to ensure the traffic integration. Thus, the dispatcher becomes the bottleneck of the whole system when the system is subject to heavy request load. Also, this approach can only be applied to locally clustered web servers.

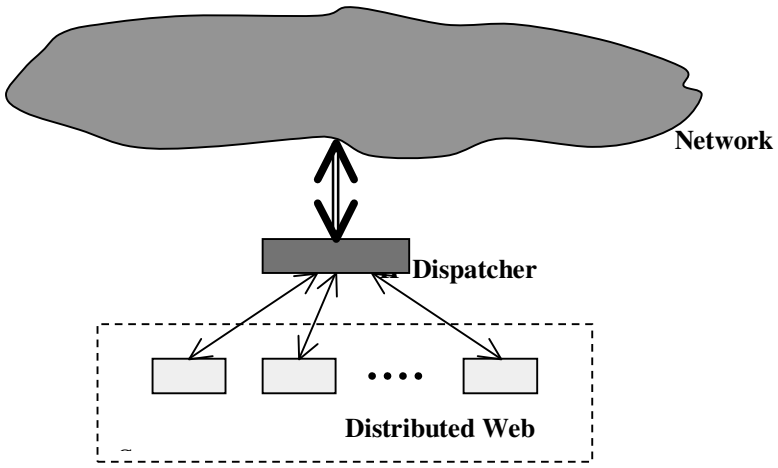


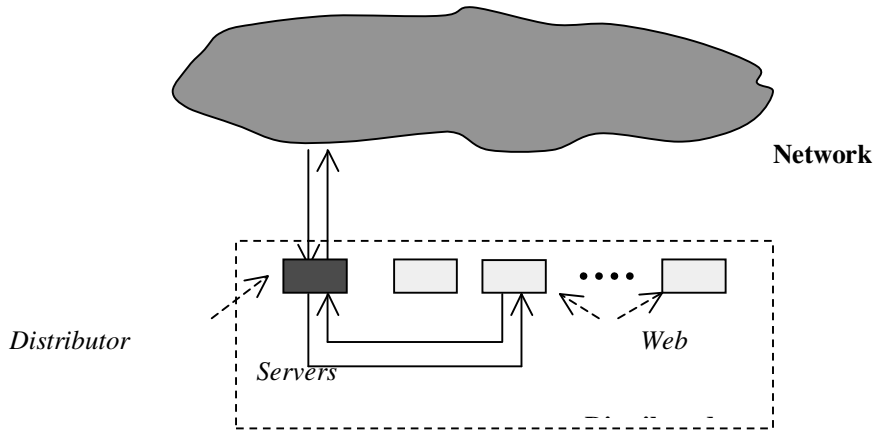
Fig.1 IP-dispatcher System

Similar to IP-dispatcher approach, [3] uses a server as distributor, which functionality is similar to IP-dispatcher. It receives all incoming requests, distributes requests to a set of web servers, receives responses from web servers, and forwards responses to clients (Fig.2). However, the distributor will be a bottleneck as in the IP-dispatcher case.

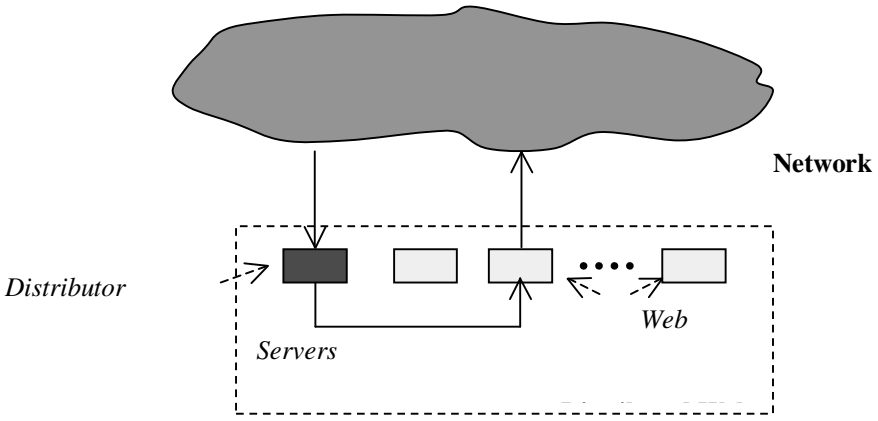
To minimize the bottleneck disadvantage in the distributor approach, a modified handoff approach has been proposed [8]. The main idea behind this approach is that, the distributor only receives and distributes incoming requests, and web servers send responses directly to clients (Fig.2). This approach also supports partitioning web servers into several groups for different sets of web pages or tasks. In this way, a faster average response is more likely [8]. However, the port controller parts of the distributor still need to modify the IP addresses of incoming and outgoing

packets and the distributor is still the bottleneck of the system although today's machines have much powerful computation capacities.

An alternative approach for load balancing is the DNS-based dispatcher approach [1][2]. This kind of distributed web server system, which consists of multiple, replicated WSs (web servers)



(a) Distributor System



(b) Handoff System

Fig.2 Two Distributed Web Server Systems

and a cluster DNS (Domain Name Server), is illustrated in Fig.3. The DNS translates the logical site name (URL) into the IP address of one of the web servers. If carefully designed, traffic load could be evenly distributed among the set of web servers. The advantages of this approach include: (1) The DNS-dispatcher system could be more scalable; and (2) This approach could be applied both locally and geographically distributed web server systems. However, this approach has its own difficulties on load balancing. There are a number of name servers on the path from the client to the DNS, and these name servers typically cache the name-to-address mapping

returned by the DNS. So, if a new name resolving request reaches the intermediate name server and the requested mapping is valid in the cache, the client will get the cached IP address instead of the IP address returned by the DNS. As a result of caching, a large number of clients in the domain behind the name server are mapped to the same WS, leading to a load imbalance among the servers, because clients are not uniformly distributed behind name servers [5]. The limited DNS control and the high load skew require more sophisticated DNS balancing algorithms to avoid web server overload [1].

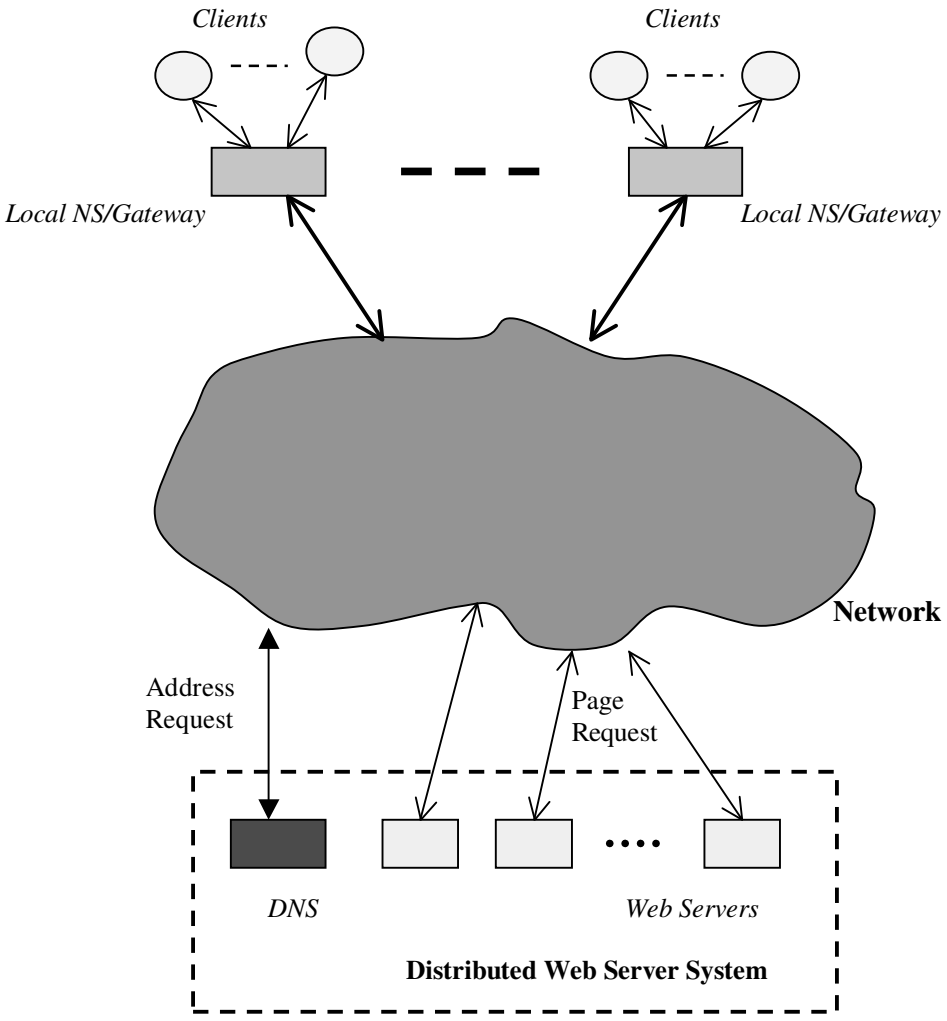


Fig.3 DNS-based Dispatcher System

In our project, one major objective is to compare the performance of load balancing algorithms in terms of load balance and average response delay. Another objective is to compare the performance of handoff approach with that of DNS-dispatcher approach. Our contributions are (1) Compare performance in terms of average response delay, in addition to load balance; (2)

Compare performance under different average traffic load; (3) Compare performance of DNS-based approach with that of Handoff approach.

The rest of this report is organized as follows. In section 2, we briefly discuss several typical load-balancing algorithms. In section 3, distributed web server system model, network model, traffic model and performance metrics in our simulation are discussed in detail. Simulation results are discussed in detail in section 4, and some conclusions and future works are suggested in section 5.

2. Load Balancing Algorithms

The simplest way to distribute requests among web servers is Round Robin approach. This algorithm alternatively assigns to address request the IP addresses of web servers in order. Although it is easy to implement, this algorithm has bad performance when the load is heavily skewed [1], or the service time happens to be unevenly distributed among web servers. An alternative algorithm is random algorithm, which randomly selects one IP address of web servers when the DNS resolve the address request. Although random algorithm might alleviate the performance degrade caused by uneven service time pattern, it has little effect on improving load balance when the clients are non-uniformly distributed.

The main reason is that the DNS only control a small portion of address resolution requests. To reduce the impact of address caching in the intermediate name servers, more information should be utilized to better scheduling algorithms. Distributed web server systems could take advantage of two kinds of information, one is the load information of web servers, and the other is the information of gateways, which can be seen by the DNS.

A number of algorithms have been proposed to enhance the basic load-balancing algorithm [1][2]. Two-Tier-Round-Robin (RR2) algorithm classifies the local gateways into two classes – normal or hot, and assigns to web servers separately according to gateway class. The main idea is to reduce the probability that the hot domains are assigned too frequently to the same web server. This algorithm utilizes domain information. In particular, the existing http and TCP/IP protocols allow the DNS and web servers to identify the origin of each address or http request. In a given period, we can count the number of address requests from each domain in the DNS and the sum of numbers of actual http connections from each domain in the web servers. The ratio of the http request sum over address request number is called *hidden load weight* in [1]. The RR2 algorithm partitions the local domains into two groups based on whether the hidden load weight of the corresponding local domain is less than a threshold. In our simulation, we determine the threshold

in this way: (1) Calculate the average hidden load weight; (2) Use this average weight as the threshold. If one hidden load weight is less than the threshold, the corresponding domain is normal domain; otherwise, it is hot domain. Within each group, we apply the round robin algorithm independently.

Another class of load balancing algorithms takes advantages of web server load information. They calculate load information based on web server's service information during current or the recent past time slots, and once the load meets some kind of criterion, they will either temporarily remove this web server from available web server list in the DNS (such as Asynchronous Alarms or Lowest Utilization Algorithm [1]) or redirect the traffic to other web servers (such as Redirection Algorithm [2]). In the Asynchronous Alarms (Alarms) algorithm, each web server calculates the current working load in each short period. Once the load exceeds a given threshold, the web server will send a message to the DNS, and the DNS will remove this server from the working list and will not assign its IP address to the address requests. After the working load goes down below the threshold again, the server will send another message to the DNS to resume to the working list and accept new IP assignments.

In addition, some algorithms take advantages of both the domain information and the load information to further improve the performance of load balancing. One simple example is that we can combine the RR2 algorithm, which is using domain information, with the Asynchronous Alarms algorithm, which is using the working load information of web servers.

In general, the main point behind these algorithms is how to use current information to estimate future loads. This problem involves the following questions: How to efficiently calculate traffic and load information? What kinds of information could best represent traffic patterns? In what condition web server tends to be overloaded? How much history information is enough for estimating future loads? These are open questions and need to be further investigated.

3. Models and Simulation

3.1 Distributed web server system and client distribution

The distributed web server system we simulated is illustrated in Fig.3. The DNS will process all address resolving requests from local gateways, and the page requests can be served by any web server since information are replicated across web servers.

Study shows that, on average, 75 percent of the client requests come from only 10 percent of the domains [6]. A good approximation that fits this non-uniform distribution is the Zipf function. The Zipf distribution can be represented by

$$N_i = c / i^{(1-x)}$$

where N_i is the number of clients in i-th domain, x is a parameter, and c is a constant and given by

$$c = \left[\sum_{i=1}^{L} 1/i^{(1-x)} \right]^{-1}$$

where L is the number of domains (gateways). The pure Zipf function is the Zipf function with $x=0$. Another approximation for non-uniform client distribution is geometric distribution, which is represented by

$$N_i = p(1-p)^{i-1}$$

where p is the parameter which is between 0 and 1.

In our simulation, we use pure Zipf distribution to approximate the client distribution associated with gateways. A sample pure Zipf client distribution with 20 gateways and 1500 clients is shown in Fig. 4. With this graph, it is easy to understand why there is a skewed load over the web server system.

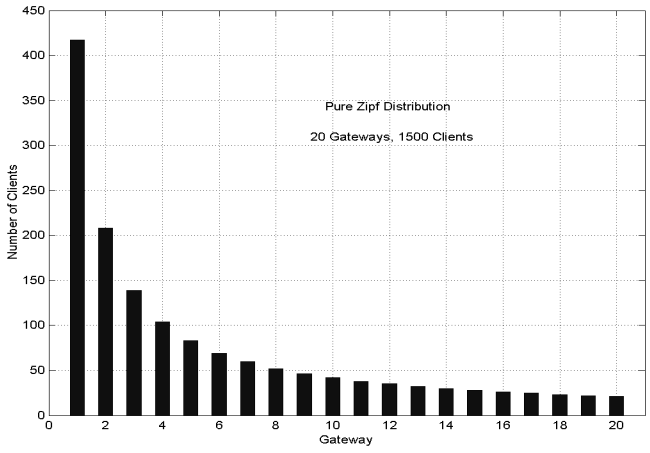


Fig.4 Pure Zipf Client Distribution

3.2 Network Model and Traffic Model

The local name servers (gateways) will cache name-to-address mapping and associated TTL time, and if the TTL time expires, gateways will invalidate this address in the cache. When a client sends a name resolution request, if the gateway associated with the client has valid mapping, the client will get the address cached in the gateway. Otherwise, the request will be sent to the DNS of the distributed web server system. The DNS will reply with the IP address of one of the web servers and an associated TTL value. The intermediate gateway will store the IP address and TTL value into the cache.

Although accurate WWW traffic models are still under study, some characteristics of Internet traffic have been recovered. The number of page requests per session and the inter-request time are exponentially distributed [6]. The service time for each hit are modeled as an exponential random variable with mean of several milliseconds. And the number of hits per page is roughly 7 hits/page and this mean value tends to increase [1]. We model the number of hits per page as a uniformly distributed number over [5, 15].

Some parameters used in our simulation are listed in the following table.

Table 1. Simulation Parameters

Parameters	***Values***
Number of web servers	7
Average utilization	0.667
Number of gateways	20
Number of clients	1500
Client distribution	Pure Zipf
Web page requests per session	Exponential (mean 20)
Hits per request	Uniform [5, 15]
Hit service time	Exponential (mean 4.5 milliseconds)
Inter-request time	Exponential (mean 15 seconds)

3.3 Performance Metrics

We compare the performance of several typical DNS-based dispatcher systems and the handoff system in terms of load balance and average response delay. Similarly to [1], we use cumulative

density function (CDF) of the maximum utilization among all web servers as the performance metric of the load balance. For each time slot, we record the maximum utilization among all utilization values of web servers, and calculate the corresponding cumulative density function. The value of the CDF on a given maximum utilization is the probability of the number of time slots, whose maximum utilization values are no more than the given threshold, over the total number of time slots. This value could tell us more about the instance behavior of the system.

In our project, we also compare the performance in terms of average response delay. Because we don't take account of the delay caused by network infrastructure, which has little to do with our objective, we calculate the response delay for each hit by subtracting the arrival time of hit packet from the time when the web server sends out the response packet. This delay includes the time when the request packet waits in the server queue and the service time that web server serves the request. For the handoff approach, since an incoming packet has to be served first by the distributor, we have to include the delay caused by the distributor into the response delay. Since the distributor only distribute incoming short request packets to web servers, it is reasonable that service time of the distributor is much shorter than normal web server service time, although they all conform to exponential distribution. With no doubts, clients prefer approaches with short average response delay.

3.4 Simulation Tool

We use CSIM18 as our simulation software. CSIM18 is a process-oriented discrete-event simulation package for use with C or C++ programs [7]. It is implemented as a library of routines, which implement all of the necessary operations. A CSIM program models a system as a collection of CSIM processes and produces estimates of time and performance. We have successfully used this package to write networking congestion control simulator. Also, several similar simulations are written with CSIM18 [1][2].

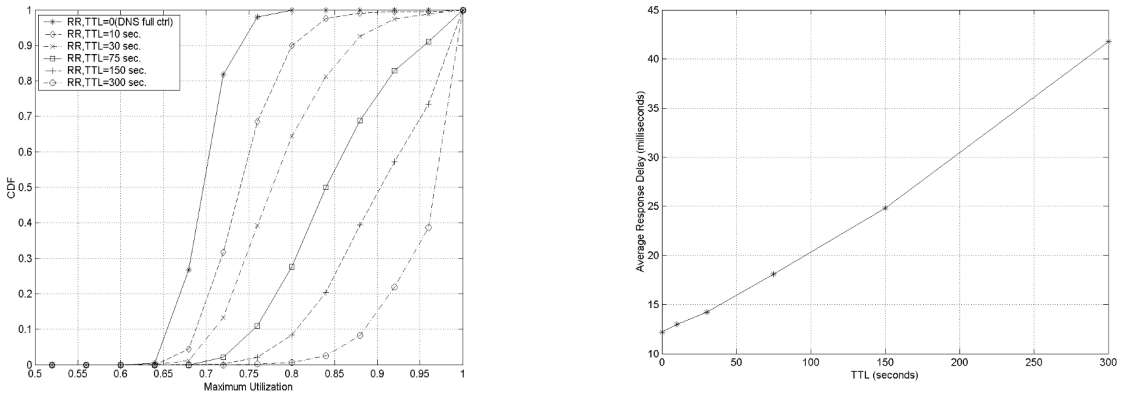
4. Results and Discussion

In our simulation, we have implemented the following load balancing algorithms: Round Robin (RR), Two Tier Round Robin (RR2), Asynchronous Alarms (Alarms), RR2-Alarms, and Handoff algorithms. These algorithms include one representative from each category of DNS-based load balancing algorithms. Some parameters used in our simulation are shown in Table 1. We will discuss the algorithm performance in the following aspects: (1) Effects of TTL values; (2)

Performance comparison of first 4 algorithms; (3) Performance under different average traffic load; and (4) Performance of Handoff algorithm.

4.1 Effects of TTL values

Fig. 5 illustrates the CDF of maximum utilization and average response delay of Round Robin algorithm under different Time-To-Live (TTL) values. From Fig.5, we can see that with the increase of TTL value, the load balance among web servers tends to degrade and the average response time tends to increase. For the worst case (TTL=300 seconds), maximum utilizations of more than 75% time slots are larger than 96%. Recall that average system utilization is set at 67.7%. That means at most time, some web servers have much high traffic load than others and the loads are highly un-balanced. On the other hand, when the TTL value is set to 0 (in this case, all name mapping requests are sent to the DNS of the system), Round Robin could distribute traffic evenly among web servers. We know that TTL=0 is an ideal case, which is impossible in the real world, and DNS could control every address request and yield much better load balances. The reason why we give the curve of DNS full control is to give a comparison base, and the closer the curve is to that of DNS full control, the better the load balancing performance is. With non-uniformly distributed traffic, Round Robin algorithm has very bad performance. As to the average delay, Round Robin yields similar performance with load balance. The average delay increases linearly with TTL value.



(a) CDF of maximum utilization

(b) Average response delay

Fig.5 Performance of Round Robin algorithm under different TTL values

Because TTL value is widely used in the Internet system, performance of Round Robin algorithm is bad especially under large TTL values. We mentioned above that address caching is one of the reasons why the traffic loads over web servers are non-uniform. Another conclusion is that average response time of Round Robin goes up with the increase of the TTL value. The possible reason for this is that, we know that if traffic is uniformly distributed, the average response time will be the smallest, and large TTL values cause large non-uniform of the traffic.

4.2 Performance Comparison of RR2, Alarms and RR2-Alarms

Fig. 6 - 9 illustrate the performance of these algorithms in terms of load balance and average response time under different TTL values. Other simulation parameters are shown in table 1.

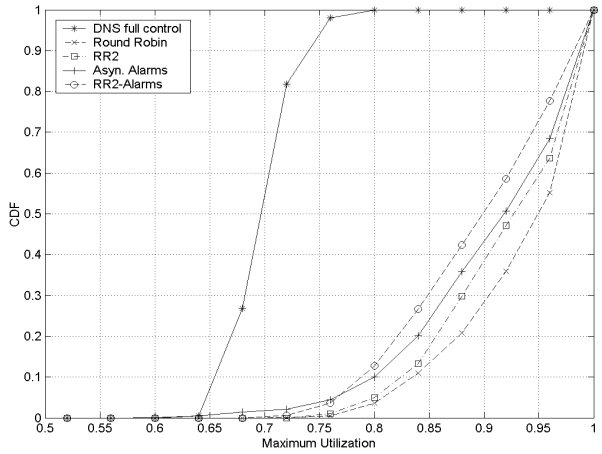


Fig.6 CDF of maximum utilization, TTL = 240 seconds

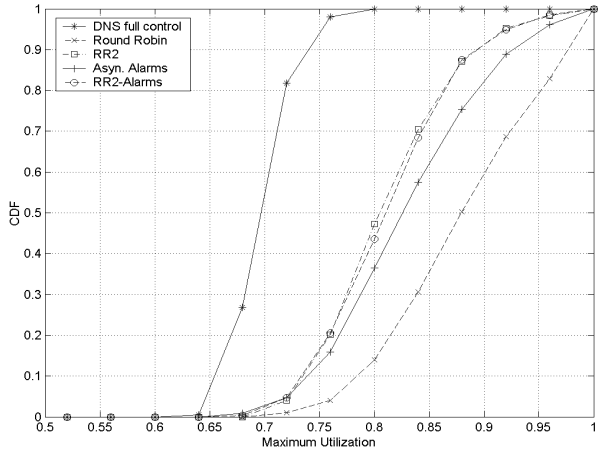


Fig.7 CDF of maximum utilization, TTL = 120 seconds

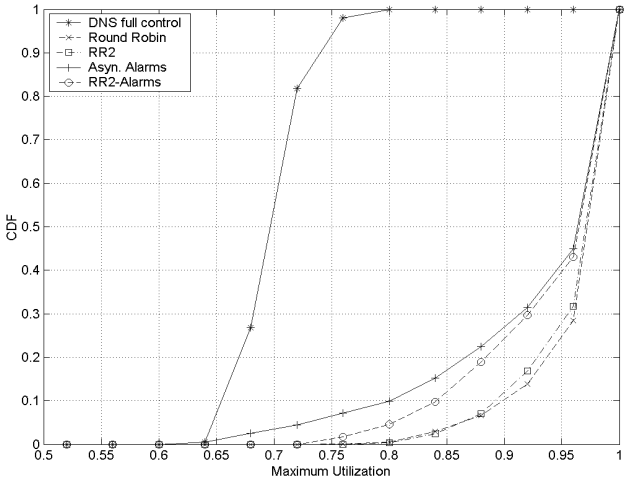


Fig.8 CDF of maximum utilization, TTL = 360 seconds

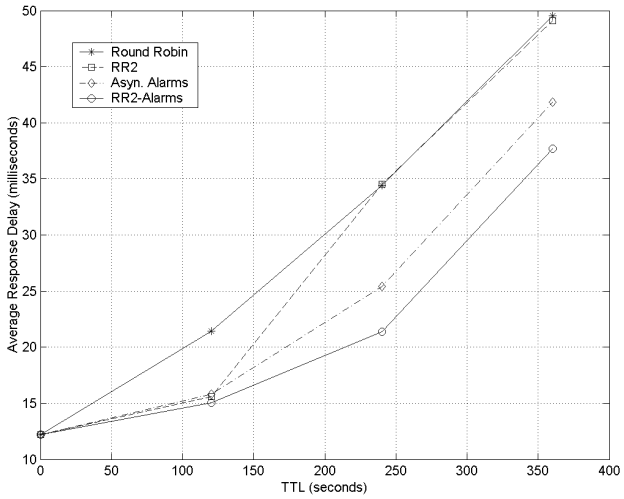


Fig.9 Average Delay

First, based on comparison of the curves of same algorithm, we can find that load balancing performance of each algorithm becomes better with the decrease of TTL values. This is easy to understand, because no matter what enhancements are applied, DNS can only control a small portion of address requests if the TTL is large. Second, under smaller TTL values, Two Tier Round Robin performs better than Asynchronous Alarms, but under larger TTL values, Asynchronous Alarms has much better performance than RR2. The possible reason is that under smaller TTL values, the degree of non-uniform of traffic load is smaller, and RR2 has larger

ability in adjusting traffic load in this case; but when the TTL values are larger, although RR2 has larger adjusting ability, it could not prevent large degree of traffic non-uniform. On the other hand Asynchronous Alarms could prevent the increasing of working load of web servers by removing them from the active list of DNS. In the case of larger TTL values, this ability has more advantages over RR2.

The RR2-Alarms algorithm has both advantages of RR2 and Alarms, so it performs best or second best among these algorithms. In addition, the average response delay time of RR2-Alarms is smallest among those of RR, RR2, Alarms, and RR2-Alarms. It can be concluded that taking advantage of both domain information (like RR2) and working load information (like Alarms) could yield best performance in terms of load balance and average response time.

4.3 Performance under different average traffic loads

[1] and [2] didn't give the performance trends under different average traffic loads. In our report, we study the algorithm performance under different traffic loads. We select three algorithms – RR, RR2 and RR2-Alarms, and three average traffic loads in our experiments – 55% (light), 67%(moderate) and 80%(heavy). In our simulation, we adjust client numbers in order to adjust traffic loads. The corresponding client numbers are 1237, 1500 and 1800, and TTL value is 240 seconds. Other simulation parameters are listed in table 1. The CDFs of maximum utilization are given in Fig.10 – Fig.12 respectively and average delays are illustrated in Fig.13.

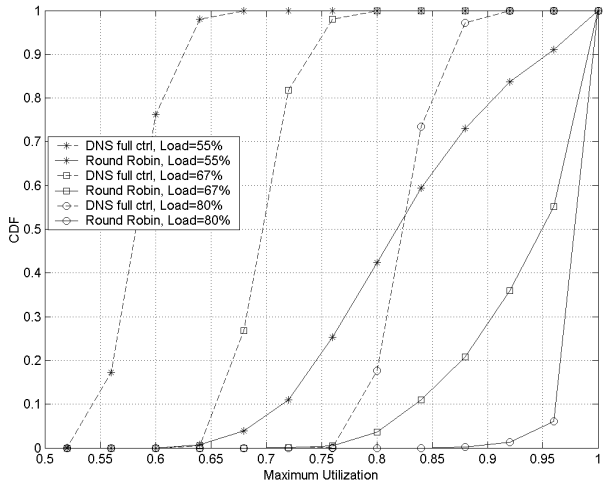


Fig.10 Performance of RR under different traffic loads

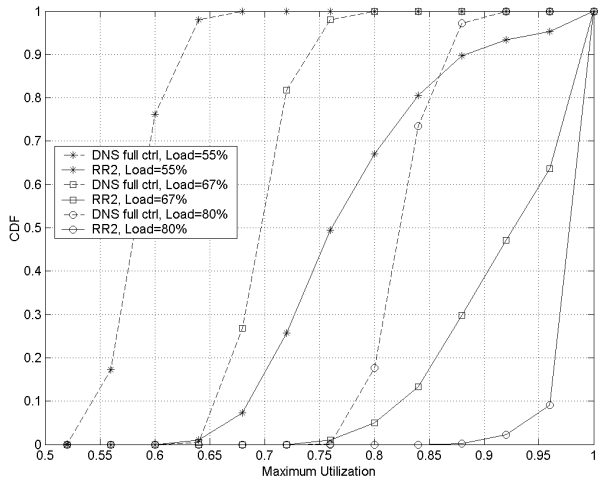


Fig.11 Performance of RR2 under different traffic loads

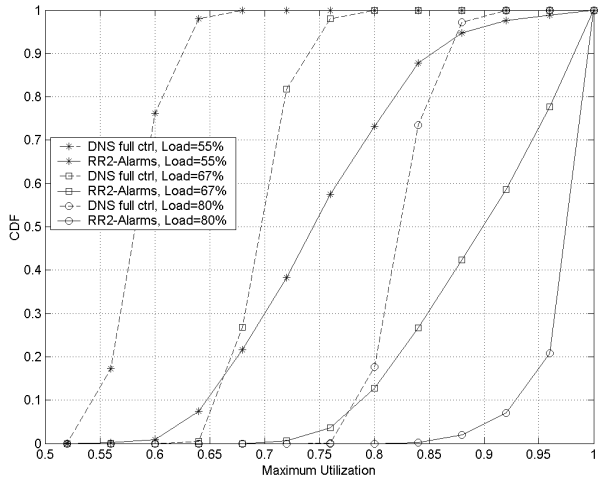


Fig.12 Performance of RR2-Alarms under different traffic loads

All algorithms perform better when traffic load becomes lighter. When TTL=240 seconds, the performance relationships between RR, RR2 and RR2-Alarms remain same no matter how much the average traffic load is. The RR2-Alarms is a little better than RR2 and RR2 is much better than RR in terms of load balance. At same time, RR2-Alarms is much better than RR2 and RR in terms of average response delay. So far, we can conclude that RR2-Alarms algorithm has the best performance in terms of average response delay and good performance in terms of load balance.

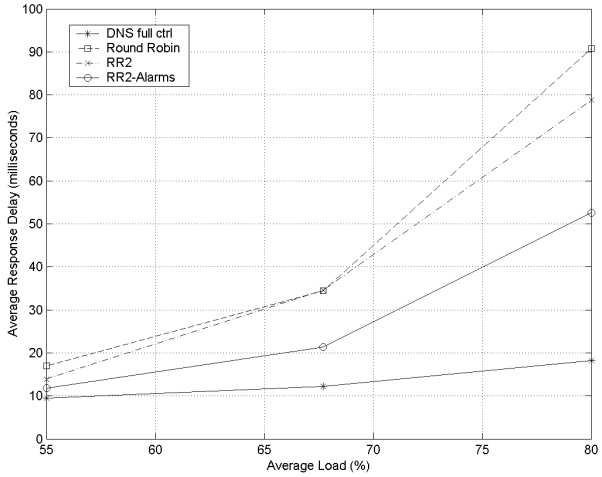


Fig.13 Average response delay under different traffic loads

4.4 Performance of Handoff Approach

Handoff approach is more like a switch-based approach and quite different from DNS-base approach. In our simulation, we use a constant time to approximate the handoff service time. Fig.14 shows the load balance performance of handoff approach and Fig.15 shows the delay performance of handoff approach. In both figures, traffic load is 67%, server number is 14 and client number is 3000.

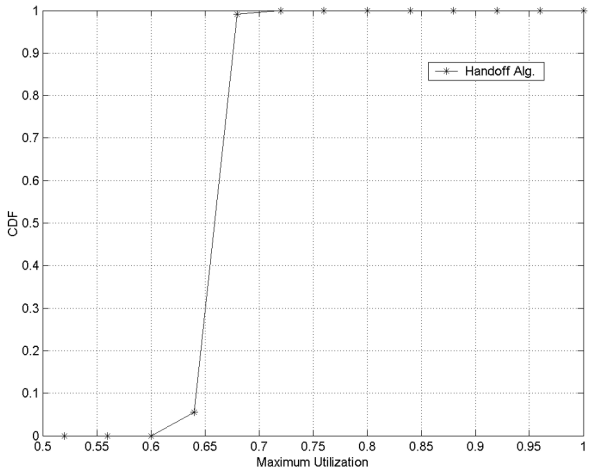


Fig.14 Performance of Handoff Approach (Load Balance)

From CDF of handoff approach, we can say that handoff approach has excellent performance in terms of load balance. Because Handoff approach controls the distribution on the all in-coming packets, it has much control than DNS-based approach and therefore performs much better than DNS-based approach in terms of load balance. However, the response delay goes up with the increase of handoff service time. When the average handoff service time approaches 0.6, the system approaches the maximum processing ability of handoff. In this case, the average responses delay increase exponentially. So, we can say that Handoff approach still has the bottleneck problem if the traffic is too heavy.

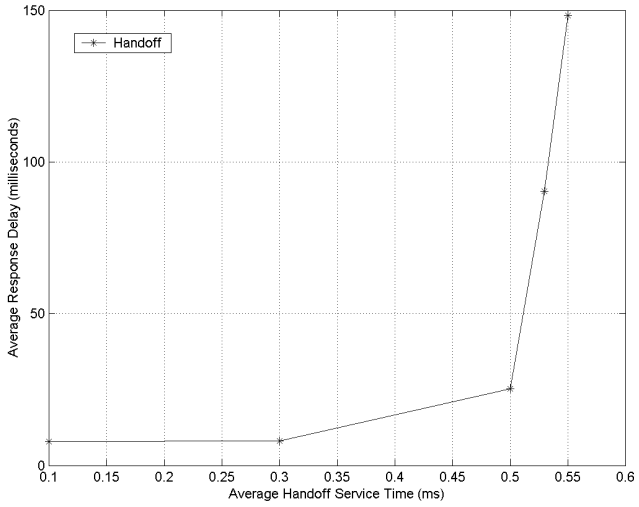


Fig.15 Performance of Handoff Approach (Average Response Delay)

5. Conclusion

Through extensive simulation, we have some conclusion: (1) The Round Robin algorithm has bad performance especially under large-degree non-uniform traffic; (2) Taking advantages of both servers load information and domain information could improve the performance under most situations; (3) Handoff approach has excellent performance in terms of load balance but its average response delay depends a lot on the computation capacity of the Distributor.

However, our simulation is not complete. We suggest the following future works: (1) How to use domain information and load information to accurately and efficiently estimate the future load. (2) The simulation only considers the address caching in the local gateways. It needs to be further investigated the performance of DNS-based approach if hierarchical caching (intermediate name server caching, switch caching, etc.) is applied.

6. References

[1] Michele Colajanni, Philip S. Yu, and Daniel M. Dias, "Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems", *IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No. 6, Jun. 1998*

[2] Valerua Cardellini, Michele Colajanni, and Philip S. Yu, "Redirection Algorithms for Load Sharing in Distributed Web-Server Systems", *Proc. IEEE 19th Int'l Conf. on Distributed Computing Systems* (ICDCS'99), Austin, TX, pp. 528-535, June 1999

[3] C.S. Yang, and M. Y. Luo, "Design and Implementation of an Administration System for Distributed Web Servers", *Proc. of the 12th Systems Administration Conference (LISA'98), Boston, Massachusetts, Dec. 6-11, 1998*

[4] Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel, and Peter Sturm, "Enhancing the Web's Infrastructure – From Caching to Replication", *Internet Computing, Vol.1, No.2, pp.18-27, Mar.-Apr. 1997*

[5] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella, "Characteristics of WWW Client-based Traces", *Technical Report BU-CS-95-010, Computer Science Department, Boston University, Apr. 1995*

[6] Martin F. Arlitt, and Carey L. Williamson, "Web Server Workload Characterization: The Search for Invariants", *IEEE/ACM Trans. on Networking, Vol.5, No.5, Oct. 1997*

[7] Mesquite Software, Inc., "CSIM18 User Guide", *http://www.mesquite.com/htmls/guides.htm*

[8] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, "Design, Implementation and Performance of a Content-Based Switch", *proceedings of INFOCOM'00, Tel Aviv, March 2000*