A Load Balancing Algorithm on Heterogeneous Distributed WWW Servers

SingLing Lee Hann-Jang Ho Feng-Wei Lien

Department of Computer Science and Information Engineering
National Chung-Cheng University
Chia-yi, 621, Taiwan, ROC
Email:{lee,hjho,fwl88}@cs.ccu.edu.tw

Abstract

As the increasing of Web traffic, a distributed web cluster is needed to cope with growing client demands. In this paper, we propose a load-balancing algorithm to distribute client demands into several servers. Since the algorithm dispatches web documents to heterogeneous servers, each server does not contain all documents and only allows to access the local available documents. The aims of our algorithm include two-fold: (1) to make load balancing as good as possible, and (2) to respond a request as soon as possible. When a server is busy handling with CGI requests, it may delay the response time of a HTML request. That's why we have the second aim. Our algorithm can produce a 15% performance improvement on load balancing and a 10% performance improvement on average response time.

KEYWORDS: Distributed Web Cluster, Load-balancing Algorithm , Dispatch, Average Response Time

1. Introduction

A popular website must have a strong web system to cope with growing client demands. There are many kinds of architecture had been proposed for such a website. Most of them are based on a distributed or parallel architecture while preserving a virtual single interface. Such a system provides scalability and transparency, but needs to have some mechanisms that dynamically assign client requests to the web server [1]. The redirection mechanism can be done at IP level through some address packet rewriting mechanisms [2, 3, 4], or at the Domain Name System (DNS) level through mapping URL-name to the IP-address of one of the web servers [5, 6, 7, 8, 9].

Once a client attempts to establish a TCP connection with one of the servers, a decision is made as to whether or not the connection should be redirected to a different server. The IP-address in the packet will be changed to the destination server's if the redirection is required. The main disadvantage of all IP-level solutions is that they can be applied only to locally clustered web servers. One-IP [4] distributes requests to the different server in the clusters by dispatching packets at the IP-level. The dispatcher redirects requests to the different servers based on the source IP-address of the client. TCP-Router approach dispatches client requests to the appropriate server at the server site router which receives the requests. The router dispatches requests according to load information and changes the destination address of each IP-address before dispatching. In a server cluster, the router will have an acknowledgement of server failure. The router will know that which server is alive and will not redirect a request to a dead node. Therefore, both TCP-Router approach and ONE-IP approach can provide server fault-tolerance. And these two approaches also can provide load balancing since they both take account of load information.

Unlike the IP-dispatcher based solutions. The HTTP redirection does not require the modification of the IP-address of the packets reaching or leaving the web server cluster. What the DNS has to do is mapping the URL-name to the IP-address of one of the servers when a client request arrives. A simplest solution is using a DNS with round-robin (RR) scheduling algorithm on it. And the DNS redirects all requests to web servers by RR algorithm. But this solution will not perform very well on load balancing since it does not care about any load information and just redirects the client requests. SWEB [5] is a web system using distributed memory machines and network of workstations as web servers. In the SWEB, a server does not store all the documents but can go over the local-area network to fetch the document that is requested but not store in itself. A round-robin domain name resolution (DNS) is used to assign requests to back-end workstations. The weakness of SWEB is that failures are still a problem because of DNS name caching. NCSA's web servers [8] use a distributed file system to access document requested by the clients. A round-robin domain name services (RR-DNS) is used to multiplex requests to web servers. But, like SWEB, a high degree of load balance may not be achieved due to DNS name caching which is the main disadvantage of all DNS-level solutions.

In this paper, we will focus on an architecture that integrates the DNS dispatching mechanism with a redirection mechanism provided by HTTP protocol.The architecture we propose has a front-end domain name server and several back-end web servers. These web servers are assumed to have different capacities and hence this is a het-

erogeneous system. The front-end DNS receives all the HTTP requests, and uses HTTP to redirect the client request to the appropriate web server. Each server does not have all the documents, and can serve the requests only for it's locally available documents. We also propose an algorithm for document distribution. Our algorithm dispatches all documents to web servers and each server handles with only some specific kinds of documents. We demonstrate that the DNS dispatcher combined with suitable redirection mechanism provides good load control. Our algorithm will be compared with the RR algorithm and the binning algorithm [9]. According to simulation results, our system's average response time is 10% faster than the system that only uses DNS with RR to dispatch requests. For dispatching document requests, our algorithm could have 15% better performance on load balancing than dispatching by the binning algorithm, and about 100% better than dispatching by round-robin algorithm.

2. System Model

Our system model has a front-end domain name server and several back-end web servers , for example, as shown in Figure-1. The system consists of heterogeneous

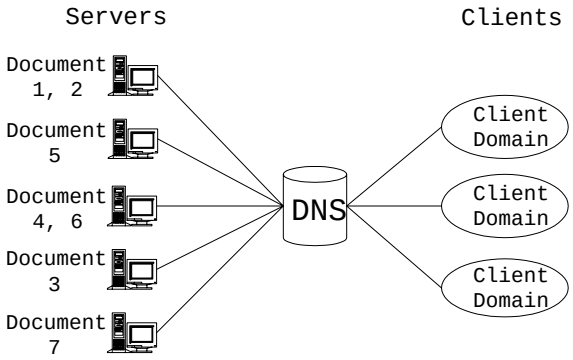

Figure 1. An example of the system model

distributed web servers that manage different sets of web documents and a DNS that translates the URL-name into the IP-address of one of the servers in the cluster. The DNS receives all the client requests as a single interface to access web documents, and uses HTTP to redirect each client request to an appropriate server. Each web server contains some of all documents and serves the requests only for locally available documents. Every server is independent to others, not like SWEB architecture, a server will not access the document which is stored in another server. The server processing capacity is different from one to each other, so the execution time of a request in one server may be shorter or longer than it in another server.

Given a set of N web servers S_1, S_2, ..., S_N, and a set of M documents D_1, D_2, ..., D_M. Each client demand requests for one of M documents. Assume that M documents have request arrival rate r_1, r_2, ..., r_M, and average execution time e_1, e_2, ... , e_M. That is, the load which is produced by requests to document j will be $r_j * e_j$, then the total system load L can be presented as $\sum_j (r_j * e_j)$, and the load probability of document j, $L_j = \frac{r_j * e_j}{L}$. A server i should bear the load equal to $\sum_j (a_{ij} * L_j)$ where a_{ij} is the partial of requests to document j that will be assigned to server i. If $a_{12} = 0.2$, that is 20% of requests to D_2 will be redirected to S_1.

We propose an algorithm called enhanced binning algorithm (EBA) to initial document distribution that achieves load balance in the heterogeneous server cluster. The algorithm will decide all values of a_{ij} initially. The DNS needs to maintain information about the location of documents and the values of all a_{ij} in order to determine which server should take the request just arrived.

3. Influences on Average Response Time

If now we have two web servers and two web documents at present: one of the documents is a HTML file and the other is a CGI file. We assume that the load produced by HTML requests is equal to the load produced by CGI requests, and the processing capacities of these two servers are also equal. The simplest solution is dispatching both two kinds of documents to each server, and uses a Round-Robin DNS to distribute all requests to web servers. This solution can make load balancing indeed, but it may not provide good performance on response time. The other method is: one server only takes care of the HTML requests, and the other one only takes care of CGI requests. Since the loads produced by HTML requests and by CGI requests are equal, this approach can also provide load balancing very well. We expect that this method has faster average response time and can save more disk capacity than the first solution since we don't have to store both kinds of documents on each server.

Suppose that both kinds of documents are stored in the same server and a Round-Robin DNS algorithm is used to distribute all requests to web servers. We will find that once the server is busy in handling HTML requests, it may delay the response time of CGI requests; if the server is busy in handling CGI requests, it may delay the response time of HTML requests. We design a simulation experiment to verify the impact of storing two kinds of documents in one server, as shown in Figure-2. In this figure, if a server deals with only document D_1 on it, the average response time for document D_1 is t_1. In the same way, when a server deals with only document D_2, the average response time for document D_2 is t_2. The horizontal axis of Figure-2 represents the value of t_2/t_1. If a server deals with both D_1 and D_2, the average response time of all requests is *t*. Then the vertical axis represents the value of t/t_1+t_2.

As shown in Figure-2, the more difference between two documents' average response time, the longer total av-

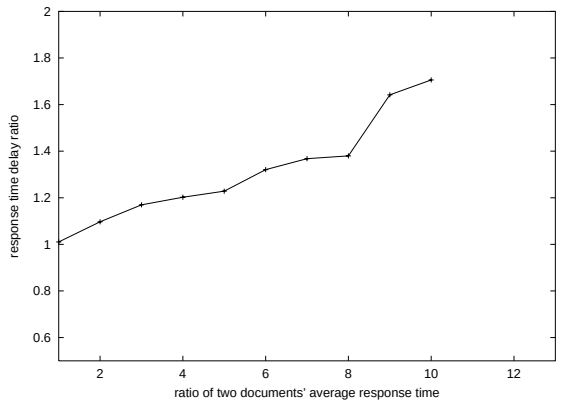

Figure 2. A variation of the response time

erage response time will be. In that case, we will dispatch all documents to web servers in order of their execution times since the response time of a request almost equals to the execution time if the server starts executing right away when the request arrives. We hope that the server could respond a request as soon as possible, so we have to avoid putting two documents on the same server if their execution times are very different.

Table 1. Notations

N	Number of back-end web servers.
M	Number of web documents.
P_i	Processing capacity of server i.
P	Total processing capacity.
C_i	Capacity probability of server i, i.e. $C_i = \frac{P_i}{P}$.
r_j	Arrival rate of request to document j.
e_j	Average execution time of request to document j.
L	Total system load: $\sum_j (r_j * e_j)$.
L_j	Load probability of document j. i.e. $L_i = \frac{(r_j * e_j)}{L}$.

4. Related Works

Using a DNS with Round-Robin algorithm to distribute requests among web servers is the simplest solution for a website. The Round-Robin algorithm dispatches a request just according to the incoming order of a request. In this architecture, each web server should contain all documents and can serve requests for all documents. If there are N servers S_1, S_2, ..., S_N. The first request will be redirected to server S_1, the second request will be redirected to server S_2, and the N_{th} request will be redirected to server S_N. That is, the j_{th} request will be redirected to server S_i, where $i = j \ mod \ N$. Without considering the factors that might effect the performance of load balancing, this solution can not achieve load balance.

To improve the performance of the Round-Robin algorithm, a binning algorithm was proposed in [9]. This algorithm dispatches requests according to access rate, which equals to request arrival rate in our algorithm. The binning algorithm works as follows. A random server S_i is picked each time and a random document j is picked and placed on that server. After the access probability of the document (can be obtained by scaling the access rate) is assigned to the server, if there is still some residual capacity left in the server then another document k is randomly picked and placed on this server, this goes on until the total access rate to the server upon the placement of a document m exceeds the capacity of S_i. At this stage, another server S_j is randomly picked and document m is replicated on S_j. Continue doing these stages until all access rates are assigned to servers. Thus, if we consider each of the servers to be a bin with a certain capacity, then the algorithm fills every bin completely before another bin is chosen to be filled.

We consider an example for a heterogeneous cluster with one redirection server, three document servers with capacity probabilities 0.3, 0.6 and 0.1, and five documents with access probabilities 0.35, 0.5, 0.05, 0.04 and 0.06. Please take notice of that access probabilities here are not equal to load probabilities in our algorithm. The binning algorithm chooses documents and servers at random. For this example, we assume that the documents are picked in order D_1 through D_5 and the servers are also picked in numerical order. Then, all documents initially are distributed as following steps:

1. $S = \{0.3, 0.6, 0.1\}$,
   $R = \{0.35, 0.5, 0.05, 0.04, 0.06\}$.
2. $c_i = c_1 = 0.3$, and $r_j = r_1 = 0.35$.
3. Assign r_1 to $c_1 \Rightarrow S_1$ contains the copy of D_1.
4. $0.3 < 0.35 \Rightarrow r_j = 0.05$, $S = \{0.6, 0.1\}$, and $c_i = c_2 = 0.6$.
5. Assign r_1 to $c_2 \Rightarrow S_1$ contains the copy of D_5 and S_2 also contains the copy of D_1.
6. $0.6 > 0.05 \Rightarrow c_i = 0.55$, $R = \{0.5, 0.05, 0.04, 0.06\}$, and $r_j = r_2 = 0.5$.
7. Assign r_2 to $c_2 \Rightarrow S_1$ contains the copy of D_1 and S_2 contains copies of D_1 and D_2.
8. $0.55 > 0.5 \Rightarrow c_i = 0.05$, $R = \{0.05, 0.04, 0.06\}$, and $r_j = r_3 = 0.05$.
9. Assign r_3 to $c_2 \Rightarrow S_1$ contains the copy of D_1 and S_2 contains copies of D_1, D_2 and D_3.
10. $0.05 = 0.05 \Rightarrow S = \{0.1\}$, and $c_i = c_3 = 0.1$, $R = \{0.04, 0.06\}$, and $r_j = r_4 = 0.04$.
11. Continue the assignment until both S and R are empty.

The redirection mechanism will now work as follows. Requests to D_1 will be redirected to S_1 with probability 0.3/0.35 = 0.86 and to S_2 with probability 0.05/0.35 = 0.14. All requests to D_2 and D_3 will be redirected to S_2 and all requests to D_4 and D_5 will be redirected to S_3. This is shown in Table-2.

Table 2. Redirection Probabilities in the Example (BA)

	D_1	D_2	D_3	D_4	D_5
S_1	0.86				
S_2	0.14	1	1		
S_3				1	1

5. The Enhanced Binning Algorithm

We propose an enhanced binning algorithm to improve the average response time. A formal description of the enhanced binning algorithm is given as follow:

1. Sort the capacity probabilities in decreasing order,

   assume the ordered set $S = \{c_1, c_2, ..., c_N\}$.

2. Sort the load probabilities in decreasing order by their execution times,

   assume the ordered set $D = \{l_1, l_2, ..., l_M\}$.

3. Set $i = 1$ and $j = 1$.

4. Assign l_j to c_i and document j is replicated on server i.

5. If $(c_i > l_j)$ then $c_i = c_i - l_j$, $D = D - \{l_i\}$, and $j = j + 1$.

   else if $(c_i < l_j)$ then $l_j = l_j - c_i$, $S = S - \{c_i\}$, and $i = i + 1$.

   else if $(c_i = l_j)$ then $S = S - \{c_i\}$, $i = i + 1$, $D = D - \{l_i\}$, and $j = j + 1$.

6. Repeat step 4 and step 5 until all load probabilities are assigned to servers.

As the same example for BA algorithm, we consider a heterogeneous cluster with one DNS, three document servers S_1, S_2, S_3 and five documents D_1, D_2, ..., D_5. Assume that the load probabilities of the five documents are 0.35, 0.5, 0.05, 0.04 and 0.06, and execution times of five documents are $e_5 > e_4 > e_1 > e_2 > e_3$. Also assume that the capacity probabilities of servers are 0.3, 0.6, and 0.1. According to the enhanced binning algorithm, all documents initially are distributed as following steps:

1. Sort the capacity probabilities of servers in decreasing order, $S = \{0.6, 0.3, 0.1\}$.

2. Sort the load probabilities of documents in decreasing order by their execution times, $D = \{0.06, 0.04, 0.35, 0.5, 0.05\}$.

3. Set $i = 1$ and $j = 1 \Rightarrow c_i = c_1 = 0.6$ (server S_2) and $l_j = l_1 = 0.06$ (document D_5).

4. Assign l_1 to $c_1 \Rightarrow S_2$ contains the copy of D_5.

5. $0.6 > 0.06 \Rightarrow c_i = 0.54$, $D = \{0.04, 0.35, 0.5, 0.05\}$, and $l_j = l_2 = 0.04$ (document D_4).

6. Assign l_2 to $c_1 \Rightarrow S_2$ contains copies of D_5 and D_4.

7. $0.54 > 0.04 \Rightarrow c_i = 0.5$, $D = \{0.35, 0.5, 0.05\}$, and $l_j = l_3 = 0.35$ (document D_1).

8. Assign l_3 to $c_1 \Rightarrow S_2$ contains copies of D_5, D_4 and D_1.

9. $0.5 > 0.35 \Rightarrow c_i = 0.15$, $D = \{0.5, 0.05\}$, and $l_j = l_4 = 0.5$ (document D_2).

10. Assign l_4 to $c_1 \Rightarrow S_2$ contains copies of D_5, D_4, D_1 and D_2.

11. $0.15 < 0.5 \Rightarrow l_j = 0.35$, $S = \{0.3, 0.1\}$, and $c_i = c_2 = 0.3$ (server S_1).

12. Continue the assignment until both S and D are empty.

Table 3. Redirection Probabilities in the Example (EBA)

	D_1	D_2	D_3	D_4	D_5
S_1		0.6			
S_2	1	0.3		1	1
S_3		0.1	1		

After the document distribution is done, server S_1 contains copy of D_2, server S_2 contains copies of D_1, D_2, D_4 and D_5 and server S_3 contains copies of D_2 and D_3. The redirection mechanism will use the probabilities shown in Table-3 to redirect all requests to the appropriate server. An entry for row S_1 and column D_1 in the table indicates the probability with which the request to document D_1 will be redirected to server S_1.

In the enhanced binning algorithm, when a document is placed on a server, either all it's load probability can be completely assigned onto that server or the server's cumulative load probability will exceed the capacity. A document is replicated on next server only when the server is not able to accept the load probability of that document completely. Each server can be exhausted once at most and exhaustion will not produce more than one replica of the document. Consequently, with N servers, there can be N replications at most. Since the sum of capacity probabilities of the servers is equal to the sum of load probabilities

of documents, all load probabilities of documents can be assigned exactly when the last server is exhausted.

In our system, each server doesn't have to store all web documents and only has to serve requests for it's local documents. Documents which are stored in memory will not be replaced frequently. The cache mechanism will provide higher cache utilization efficiency than the server which is capable of serving all documents. So use our algorithm to dispatch documents will improve the cache utilization and memory utilization of the whole system.

6. Simulation Results

In the distributed system, jobs are assigned to different servers to execute. What we need is to shorten the total execution time of these jobs. The job distribution algorithm is the key of the whole system. If we use two job distribution algorithms, A and B, in the same distributed environment. Fe can easily determine that which algorithm has better performance on load balancing by their total execution times. That is, if the total execution time using A is shorter than the total execution time using B, then we can say that the system using algorithm A will have better performance on load balancing.

We use simulation to analyze our algorithm and compare our algorithm with algorithm BA and algorithm RR. Except for the dispatch algorithm, the platform and other conditions include parameters shown in Table-1 are all the same. In this situation, we can make out if our algorithm is better than others directly. If the total execution rounds (a round means a unit time) of system using our algorithm EBA is smaller than the system using another algorithm, we can say that dispatch requests by our algorithm will have better performance on load balancing than dispatch by another algorithm. In the beginning of each simulation, we have to define the values of all parameters. Some parameters will be generated in the way shown in Table-4.

Table 4. Parameters

Processing capacity P_i	Random number: $100 \sim 1000$
Avg. execution time e_j	Random number: $1000 \sim 5000$
Arrival rate r_j	Rnd. number: $10000 \sim 100000$
Total capacity P	Can be obtained by $\sum_i (P_i)$
Capacity probability C_i	Can be obtained by $\frac{P_i}{P}$
Total load L	Can be obtained by $\sum_j (r_j * e_j)$
Load probability L_j	Can be obtained by $\frac{(r_j * e_j)}{L}$

Since we use program to simulate actions of web servers, the simulation platform won't have any effect on our results. A simulation has only one total execution rounds whether it executes on a PC with 800MHz CPU or on a PC with 500MHz CPU. Do simulation on different PC will only cause different elapsed time, but the total execution rounds will be the same.

First, we consider that the number of servers is fixed and the number of documents is from 1 to 20. In this situation we can find out the difference between three algorithms when the number of documents is getting more and more. Assume that there are 5 servers, and the simulation results are shown in Figure-3. Each point in the figure is the average value of fifty times simulations. The horizontal axis represents the number of documents, and the vertical axis represents the total execution rounds of the algorithm. As the Figure-3 shows, our algorithm EBA has the least total execution rounds; the algorithm RR has the worst results, and algorithm BA performs between EBA and RR.

By comparing EBA with the other two algorithms RR and BA, it seems that more number of documents is requested the better result we gain. We present simulation results in ratios of one algorithm's execution rounds to RR's execution rounds, so the ratio of execution rounds of RR will be always 1. Results are shown in Figure-4. As what we can see in the Figure-4, the ratio of execution rounds that dispatch requests by RR is at least double of ours, and ratio of execution rounds that dispatch requests by BA is 1.2 times of ours in average.

Now, we consider another situation that the number of documents is fixed and the number of servers is from 1 to 20. Assume we have 10 documents, the simulation results are shown in Figure-5. Each point in figure is also the average value of fifty times simulations. The horizontal axis represents the number of servers, and the vertical axis represents the total execution rounds of the algorithm. In this situation, our algorithm also performs better than other two algorithms, especially when the number of servers is double of the number of documents. As the increasing of the number of servers, the total execution rounds of three algorithms are getting close. Normally, it should be so since our parameters are generated in a fixed range, total load will not increase as the increasing of number of servers because of the number of documents is fixed. We also present these results in ratio form and shown in Figure-6. When number of servers is less than 10, our dispatch algorithm has better performance than other two algorithms clearly.

According to simulation results of Figures 3-6, our system's average response time is 10% faster than the system that only uses DNS with RR to dispatch requests. For dispatching document requests, our algorithm could have average 15% better performance on load balancing than dispatching by the binning algorithm, and about 100% better than dispatching by round-robin algorithm.

At the beginning of this paper, we mentioned that we wish our system could respond a request as soon as possible. We select a document and assign it's load probability to a server in decreasing order of execution times of documents. When load balance is achieved, our algorithm will always dispatch the document with longer execution time to the server that has higher processing capacity. For this reason, we think that our algorithm should have better performance on average response time. In order to prove our

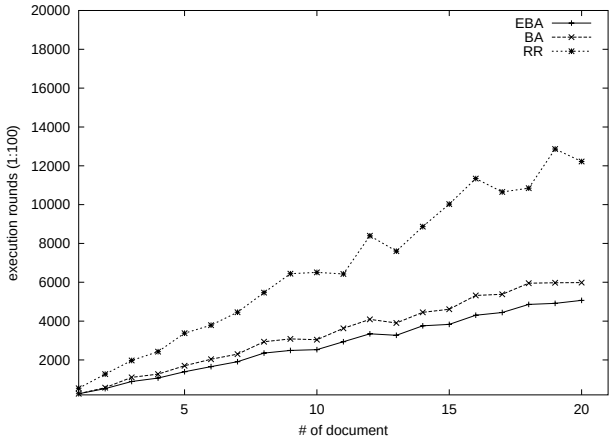

Figure 3. 5 Servers and M Documents

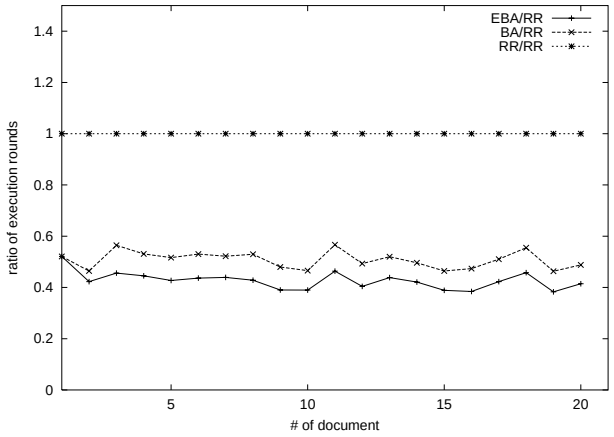

Figure 4. 5 Servers and M Documents (ratio to RR)

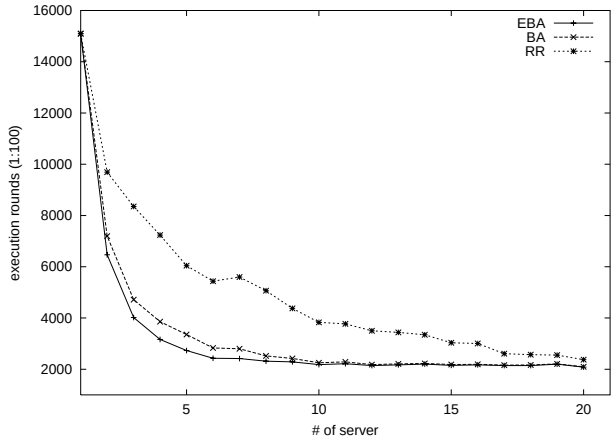

Figure 5. N Servers and 10 Documents

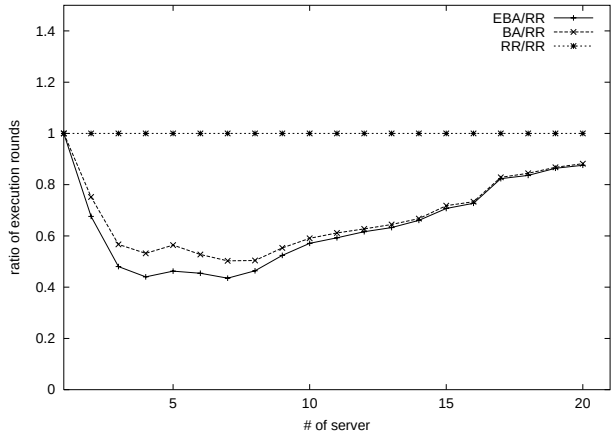

Figure 6. N Servers and 10 Documents (ratio to RR)

point of view, we have some simulations to estimate average response time of our algorithm and compare it with other two algorithms. The number of documents is 5 and the number of servers is from 1 to 10 and Figure-7 shows the scaled results.

In this experiment, our program simulates handling web requests as a real web server to compute the average response time. The average response time can be obtained by the amount of execution times of servers divided by the number of total requests. As shown in Figure-7, we gain about 10% faster than other two algorithms on average response time. We also have simulations in the environment that has 5 servers and the number of documents is from 1 to 10. Results are not very different from results of previous simulations and the diagram is similar to Figure-7. Since our algorithm dispatches documents by their execution times, the binning algorithm dispatches documents in random and the Round-Robin DNS dispatches documents by round-robin algorithm, we think that our algorithm can gain more benefits under the two situations listed below:

1. Server capacities are very different.

2. Documents' execution times are very different.

If all documents are dispatched by BA, the more different server capacities are, the longer average response time of a document will be. Because a document with longer execution time may be dispatched to a server has lower processing capacity. If dispatches by RR, the more different server capacities are, the longer average response time of a document will be. Because the average response time will lower due to requests with longer execution times will be equally dispatched to each server. Both BA and RR will have the same results in the second situation as in the first one.

7. Future Works

In this section, we will introduce two problems that might occur, and we bring up two simplest solutions to

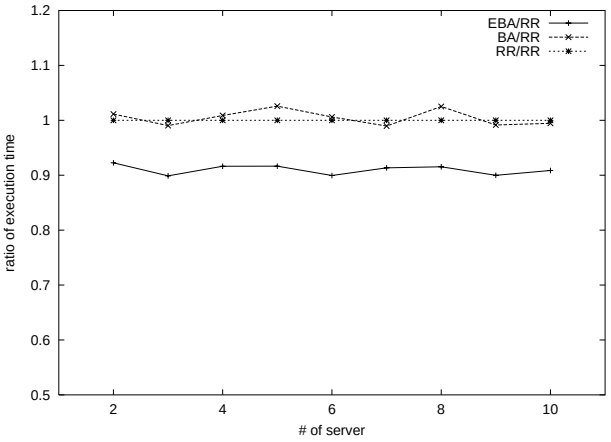

Figure 7. N Servers and 5 Documents (ratio to RR)

these two problem. For further investigations, it is worthwhile to look into ways of making better mechanism for obtaining the optimal solution.

First, we consider the DNS bottleneck. There is only one dispatcher (DNS) in the architecture. Because of the DNS with EBA is not like a common DNS that just needs to return a corresponding IP-address, it should spend more time on running dispatch algorithm and chooses an appropriate server to handle the request. Hence, for a popular website, client demands may crowd into web server very fast, DNS might become a bottleneck in this kind of architecture.

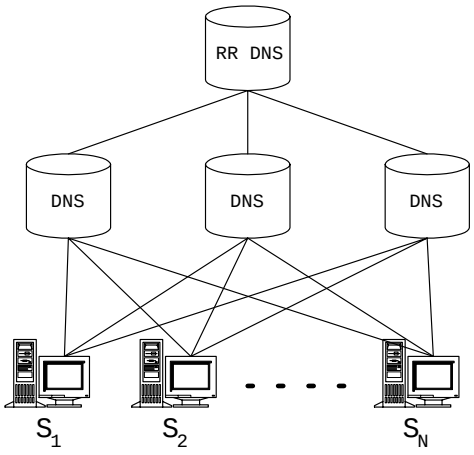

Figure 8. Multi DNS Architecture

To solve this problem, we can use a two-level DNS architecture that is shown in Figure-8. A Round-Robin DNS in the first level and several DNS with EBA which we call dispatchers in the second level. When Round-Robin DNS receives a HTTP request, it just needs to pass the request to one dispatcher by round robin scheduling algorithm. And each dispatcher does the same jobs as what the DNS does in One-DNS-Architecture.

Next, we consider the problem about the fault tolerant. Document distribution using our algorithm guarantees load balance, and has better performance on average response time. But the system can't stand any server failure because each server only maintains the documents it is dispatched. If a server fails, the requests to documents on that server will not be served. To save capacity of hard disk and to avoid server failure will become a trade off. If we spend more disk capacity to store all documents in each server, the fault tolerant problem could be solved. When a server is fault, redirection probabilities are recomputed to exclude capacity probability of that server.

Consider the example which we mentioned in Chapter 3. The redirection probabilities after document distribution is finished, shown in Table-5.

Table 5. Original Redirection Probabilities

	D_1	D_2	D_3	D_4	D_5
S_1		0.6			
S_2	1	0.3		1	1
S_3		0.1	1		

If server S_2 is fault, the capacity probabilities of two residual servers are 0.75 and 0.25 since the processing capacity of S_1 is three times of S_3's. Once the document distribution is completed, redirection probabilities will be changed as shown in Table-6.

Table 6. New Redirection Probabilities

	D_1	D_2	D_3	D_4	D_5
S_1	1	0.6		1	1
S_3		0.4	1		

Compare our algorithm with the other two algorithms, the system still has better performance on load balancing and average response time after redirection probabilities are recomputed.

8. Conclusion

In this paper we have proposed a load-balancing algorithm to distribute client demands into a heterogeneous web server cluster. Since client requests are handled by HTTP redirection on DNS, our solution belongs to the DNS-level solutions. The DNS dispatching function is integrated with a redirection mechanism and assigns requests to servers according to redirection probabilities. We use simulations to analyze our algorithm EBA and have compared our algorithm with two alternative dispatching algorithms. The experimental results indicate that our algorithm performs well on load balancing. Furthermore, the load balance performance of the system using our algorithm has increased by

15% as compared with the algorithm proposed in [9] and also has a little performance improvement on average response time. Although our solution has DNS bottleneck problem and server failure problem, the system will provide higher cache utilization efficiency and can save more disk capacity.

References

[1] M. Colajanni, P.S. Yu, and D.M. Dias, "Analysis of task assignment policies in scalable distributed web-server systems," *IEEE transaction on parallel and distributed systems*, Vol 9, No. 6, June, 1998

[2] L. Aversa, and A. Bestavros, "Load balancing a cluster of web servers using distributed packet rewriting," *Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International*, pp. 24-29, 2000.

[3] A. Bestavros, M. Crovella, J. Liu, and D. Martin, "Distributed packet rewriting and its application to scalable server architectures," *Network Protocols, 1998. Proceedings. Sixth International Conference on*, pp. 290-297, 1998.

[4] O.P. Damani, P.-Y. Chung, Y. Huang, C. Kintala, and Y.-M. Wang, "ONE-IP: Techniques for hosting a Service on a Cluster of Machines," *International World Wide Web Conference*, Santa Clara, Apr. 1997.

[5] D. Andresen, Y. Tao, V. Holmedahl, and O.H. Ibarra, "SWEB: Towards a scalable World Wide Web Server on Multicomputers," *IEEE International Symp. on Parallel Processing*, pp. 850-856. April, 1996.

[6] M. Colajanni, P.S. Yu, and V. Cardellini, "Dynamic load balancing in geographically distributed heterogeneous Web servers," *International Conference on Distributed Computing Systems*, pp. 295-302, 1998.

[7] Z. Huican, S. Ben, and Y. Tao, "Scheduling Optimization for Resource-Intensive Web Requests on Server Clusters," *Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures*, pp. 13-22, 1999.

[8] T.T. Kwan, R.E. McGrath, and D.A. Reed, "NCSA's World Wide Web Server: Design and Performance," *IEEE Computer*, pp. 68-74, 1995.

[9] B. Narendran, S. Rangarajan, and S. Yajnik, "Data distribution algorithms for load balanced fault-tolerant Web access," *Proceedings of The Sixteenth Symposium on Reliable Distributed Systems*, pp. 97-106, 1997

[10] C.S. Yang, and M.Y. Luo, "A content placement and management system for distributed web-server systems," *International Conference on Distributed Computing Systems*, 2000.

[11] M. Castro, M. Dwyer, and M. Rumsewicz, "Load balancing and control for distributed World Wide Web servers," *IEEE Conference on Control Applications*, vol. 2, pp. 1614-1619, 1999.

[12] V. Holmedahl, B. Smith, and Y. Tao, "Cooperative caching of dynamic content on a distributed Web server," *High Performance Distributed Computing*, The Seventh International Symposium, 1998.

[13] V. Cardellini, M. Colajanni, and P.S. Yu, "Redirection algorithms for load sharing in distributed Web-server systems," *International Conference on Distributed Computing Systems, 19th IEEE*, pp. 528-535, 1999.

[14] Z. Huican, S. Ben, and Y. Tao, "Hierarchical resource management for Web server clusters with dynamic content," *International conference on Measurement and modeling of computer systems*, 1999.