Document Distribution Algorithm for Load Balancing on an Extensible Web Server Architecture*

Ben Chung-Pun Ng and Cho-Li Wang
Department of Computer Science and Information Systems
The University of Hong Kong
Pokfulam Road, Hong Kong
{cpng+clwang}@csis.hku.hk

Abstract

Access latency and load balancing are the two main issues in the design of clustered Web server architecture for achieving high performance. In this paper, we propose a new document distribution algorithm for load balancing on a cluster of distributed Web servers. We group Web pages that are likely to be accessed during a request session into a migrating unit, which is used as the basic unit of document placement. A modified binning algorithm is developed to distribute the migrating units among the Web servers to fulfill the load balancing. We also present a redirection mechanism, which make use of migrating unit's property, to reduce the cost of request redirections. The distribution of Web documents would be recomputed periodically to adapt to the changes in client request patterns and system configuration. Simulation results show that our solution can reduce the amount of request redirection and document migration, and it can distribute workload properly among Web servers.

1 Introduction

The Internet's ever-increasing popularity demands the Web sites to handle large amount of requests. This has created an urgent need for a more powerful Web server architecture to handle this problem. Distributed Web server (DWS) consists of multiple Web servers, which are connected by LAN or WAN, and work together to handle the Web requests. It is one of the solutions to improve response time and reduce the resulting traffic congestion.

Distributed Web server systems can be classified by the degree of document duplication among Web server nodes. One type of *DWS* duplicates all files in every Web server nodes. This approach would waste large amount of storage for replicating files that are not frequently requested by Web clients. In addition, full duplication may generate heavy traffic in the network and load in the servers to maintain Web content consistency.

On the other hand, for Web server system without document duplication, it is impossible to achieve load balance if there exists some hot spots (popular Web pages with very high request rates) in the Web site [4]. For other type of *DWS*, only subset of files is duplicated in each Web server nodes. Some content-aware distributor [8,9] is adapted to dispatch HTTP requests to the Web server that consists of the requested document. Duplication of frequently requested documents can balance the workload of the servers and improve the storage utilization. However, this approach may lead to a performance bottleneck at the distributor since redirection overhead is charged for every HTTP request, which results in longer access latency.

In this paper, a distributed Web server architecture, called *Extensible Web Server* (EWS), is proposed. The EWS allows server nodes to be added or removed at any time. The dynamic nature of system configuration has added the challenges in the design of efficient document distribution algorithms to achieve high-performance Web services with minimum storage consumption and redirection overheads. Main features of EWS discussed in this paper are as follows:

(1) *Avoid storing unnecessary copy of document* – It has been observed that 10% of files in a Web site accounts for 90% of server requests and 90% of the bytes transferred [7]. To improve the storage utilization without scarifying performance, we propose new policies for document replication, which reduce the number of copies for documents with low request rate or those with large file size.

(2) *Reduce the number of HTTP redirections* – As each

* The research was supported by HKU University Research Grant 10203009.

server only hosts a portion of the Web contents, redirection is always needed to route the client request to a server node that stores the requested Web page [8]. We proposed a new redirection mechanism and a Web document distribution algorithm, which can reduce the average number of request redirections encountered by a client. Thus the average access latency can be shortened.

(3) *Load Balancing* – Each document in the Web server is associated with a *weight* based on some criteria, such as access frequency and document size. It is desirable that the Web document distribution results in balanced workload among the Web servers for future client requests. This is important when there are a large number of concurrent client requests towards the servers.

The rest of the paper is organized as follows. In section 2, the architecture of EWS would be shown in detail. In section 3, the Web content partitioning and distribution algorithms would be presented. Section 4 mentions the experiment and the simulation results. Related works are discussed in Section 5. Finally, the conclusions are given in Section 6.

2. Architecture of EWS

The Extensible Web Server (EWS) [15] consists of multiple distributed Web servers that function cohesively as a single logical Web server. The main feature of EWS is that it allows server nodes to be added or removed at any time.

In EWS, there is one *redirection server* installed in the central site. It would handle the first request of a visitor's request session. It is responsible to select a suitable web server for the client and redirects the client to the selected server using HTTP redirection function. The redirection server serves as a single point of entry in the view of clients.

In EWS, documents of a Web site are stored in the *master server* (which can be hosted by the same machine as the redirection server) in the beginning stage. Periodically, the documents of the Web site would be partitioned and copied to the rest of server nodes based on Web request logs to adapt to the changes of client access patterns. Replication of popular documents among different server nodes is permitted for load sharing.

To enable the transparent redirection, all the hyperlinks of HTTP documents, which reference to the documents within the same Web site, are shown in a relative URL form. For example a hyperlink <A HREF="http://ews/a.html"> would be modified to <A HREF="a.html">. Through the modification of the hyperlink, we can force the visitor to send the subsequent HTTP requests to the Web server that serves its previous requests. As each Web server only keeps a subset of documents, the server has to deal with three cases: (1) If the requested document is locally available, the visitor's request is served. (2) If the requested document is not available locally but the server is aware of the location of the requested document, it will redirect the visitor's request to the suitable Web server node. (3) If the Web server cannot serve the document request and has no information about which Web server(s) can serve the request, it would redirect the visitor's request to the master server since the master server keeps all Web documents.

For example, there are 5 html files A, B, C, D and E in the Web site. Web server 1 stores file A, B and Web server 2 stores C, D and E. Firstly, the visitor requests A, it would redirect the visitor to Web server 1. After that, the visitor requests file B by following a hyperlink in A. The visitor would send the request to Web server 1 directly. If the visitor then requests file C. The request would be sent to Web server 1. However, as Web server 1 does not have file C. It would redirect the visitor to Web server 2.

Ideally, if the files are partitioned and distributed properly among the Web servers, no further redirection is required after the visitor's first HTTP request is redirected to one of the server node. For example, visitors always request file A, B and C in the same session. If we put the 3 files in the same Web server, visitor would encounter only one HTTP request redirection. This is better than redirection-based hierarchical server architecture [4], which requires one HTTP redirection for every HTTP request, which results in longer access latency. Using our approach, both the redirection server and Web servers share the workload of redirecting HTTP requests. This avoids the redirection server become the bottleneck of the system. Benefits of using this redirection mechanism are as follows:

(1) Avoid redirection server becoming a bottleneck. Redirection can be taken place at the redirection server and all server nodes. Thus, the redirection cost can be shared by all servers.
(2) Load balancing on document retrieval. With duplicated Web documents, load balancing can be embedded in the selection of nodes for handling HTTP requests. The redirector can choose a less loaded server to process the request.
(3) The redirecting structure can be changed easily by updating the document location information in the master server and other server nodes. There is no need to modify any Web documents during runtime.
(4) This approach can be implemented on system with Web server nodes connected by LAN or WAN.

3. Web Documents Distribution

In some large Web site, the Web site administrator would partition the whole Web site into multiple subsets based on the hierarchical structure of the local file system or topics of special interests. They would assign a distinct Web server to host each subset in order to distribute the workload. This approach can avoid full replication of all documents. However, it is not easy to achieve load balancing since some popular Web pages will be requested intensively during certain period while others are not. In addition, the "topics of interests" are always changed by time.

Our document distribution policy intends to distribute the workload generated by the future client requests evenly among the server nodes. This is carried out by two steps: *migrating unit grouping* and *migrating unit distribution*.

3.1 Migrating Unit Grouping

We partition the documents of a Web site that are closely related to each other to form a *migrating unit*. Documents that are closely related imply that visitors are likely to request them during the same session. *Migrating unit* is the basic unit of document distribution. The effect of migration unit grouping is that the Web server can process the subsequent requests generated by the same visitor without further redirection. Ideally, the visitor would only encounter request redirection for the first HTTP request. This can minimize the number of HTTP redirections and shorten the access latency.

Let $W = \{o_1, o_2 \ldots o_n\}$ be the set of all documents in the Web site and $S = \{S_{master}, S_1, S_2 \ldots S_n\}$ be the set of all Web server nodes in the system. For simplicity, we assume there is only one master server denoted as s_{master}. This Web server contains all the documents in the Web site. For large Web sites, multiple master servers can be employed. A Round Robin DNS server can be used to select one of the master servers for redirecting the client's first request.

In the m*igrating unit grouping* step, firstly, we need to measure the closeness between documents in W. The closeness between any 2 documents (o_a and o_b) is measured using $P_{b|a}$. $P_{b|a}$ is the probability of requests for file a that are followed by a request for file b within [0, T) seconds (i.e., during the past T seconds). If $P_{b|a}$ between two documents is high, the two documents should be placed within the same migrating unit. The Web server log is scanned to calculate $P_{b|a}$ for all documents in W. $P_{b|a}$ can be calculated using the simple formula. Similar grouping algorithms have been used in proxy server systems [10,11,12], which are used as hints for pre-fetching documents in proxy server to improve the cache hit rates.

Let $P_{b|a} = c_{b|a} / c_a$ where c_a is the total number of client requests for file a and $c_{b|a}$ is the number of times a request to file a followed by a request to file b within [0, T). The request logs would be collected from the master server in the beginning stage or all participating server nodes after the initial stage. The collected server logs are reorganized into a number of sorted request lists such that each request list contains request records from the same client[1]. The request records within a list are sorted in ascending order based on their request time.

We then compute the values of the counters c_a and $c_{b|a}$ by scanning through each request list. For each request record for file a in the request list, we would increase the value of c_a by 1. Besides, we also need to find the value of each counter $c_{b|a}$. Special treatment is taken when there are multiple requests to the same file within the same period [0, T). When file b is requested consecutively multiple times after an access to file a (e.g. abbb), $c_{b|a}$ would be increased by one only (avoid $P_{b|a} > 1$) since it is not reasonable to update the $c_{b|a}$ multiple times as there is at most one prediction in this case. For aaab, $c_{b|a}$ would be updated (i.e. add 1 to $c_{b|a}$) for each occurrence of a. If we need to calculate $P_{b|a}$ for all pairs of documents, it would need O(n^2) $c_{b|a}$ counter. In order to reduce the space requirement, we only calculate $P_{b|a}$ where file b is an HTTP file.

After all request lists are processed, we calculate the closeness between two documents using the equation $P_{b|a} = c_{b|a} / c_a$. Then we group the documents into migrating units based on the input parameter *min_prob*. The parameter *min_prob* is a threshold value within the range of [0,1]. It is used to separate documents to different grouping units. We would add a file b to a migrating unit *mu*, if there exists a file a in *mu* and $P_{b|a}$ greater than the threshold value *min_prob*. The value of *min_prob* may affect the granularity of document grouping and the amount of request redirections. For example, assigning 0.2 to *min_prob* implies that there may be a hyperlink between documents (e.g. A, B) in two migrating units with a measured closeness of 0.2. Thus if user requests A after B, a request redirection may be required. When *min_prob* is 0, there would be no hyperlink with measured closeness greater than zero between any two migrating units. In this case, one or more migrating units would be formed but they are independent from one another.

The accuracy of prediction for the migrating unit grouping would be higher than that in cache proxy server. The EWS computes smaller problem space as compared with proxy server since the request logs of *EWS* contain information for requests in a single Web site only. On the other hand, cache proxy server computes request logs generated by client requests for all documents from various Web sites that are cached by it. Hence in EWS, more client access patterns over a smaller set of documents are collected and analyzed. This would make more accurate prediction. Interested readers can refer to [10,11,12] for various grouping algorithms.

3.2 Migrating Unit Distribution

The *migrating unit distribution* regards the placement of migrating units among the Web servers in EWS to achieve load balancing. Each migrating unit would be

[1] We use the IP address to distinguish different clients.

associated with a *weight* based on number of requests on its member documents and also the file size. The total *weight* contributed by the migrating units assigned to each Web server should be proportional to the performance of the server, in order to balance the workload.

Let $MU = \{mu_1, mu_2, \ldots, mu_m\}$ be the set of migrating units generated after the migrating unit grouping step. Initially, all documents appear once in one of the migrating units. The *weight* of each file is the estimate of the expected workload for serving the file. It is defined as a product of total number of requests on the file and the file size, i.e., the total bytes transferred. For all migrating units appear in *MU*, their *weight* is equal to the total *weight* of all documents in the migrating unit. During the migrating unit distribution step, a migrating unit mu_i can be replicated into two migrating unit mu_{i1} and mu_{i2} which

(1) $mu_i, = mu_{i1} = mu_{i2},$ both mu_{i1} and mu_{i2} contain the same documents as mu_i, and
(2) $weight(mu_i) = weight(mu_{i1}) + weight(mu_{i2})$; where $weight(x)$ denotes the weight of migrating unit x.

Let W_{Sj} be the set of migrating units assigned to server S_j and R_j be the total weight of all migrating units in server $S_{j.}$ A modified version of *binning algorithm* (see Algorithm 1) was designed to distribute the migrating units. Based on the algorithm, we assign migrating units, $R_{exp}(S_j)$ to server S_j which has $resource(S_j)$ processing power. $R_{exp}(S_j)$ is defined as,

$R_{exp}(S_j)$ = (total weight of all migrating units) * ($resource(S_j)$) / (sum of $resource(S_i)$ for all server nodes in the system)

The modified binning algorithm performs as follows. Firstly, we calculate the workload that should be allocated to each server node according to the server's computing power. In general, the computing power is a measurement of the server's maximum throughput that it can handle. Each time we pick one under-loaded Web server S_j and allocate one migrating unit to it, then continue the allocation on the next server until all servers (except the master server) receive the expected amount of workload that is equal to R_{exp}. If adding the weight of the selected migrating unit to a server S_j will exceed its expected weight $R_{exp}(S_j)$ (i.e., case 3), the migrating unit would be replicated to form two new migrating units. The 1st migrating unit (mu_A) would be assigned to S_j, while the 2nd migrating unit (mu_B) would be returned to MU set and will be allocated to other server in the next run. After all non-master Web servers are assigned with migrating units, all the remaining migrating units in MU would be assigned to the master Web server (Step 3).

Our modified binning algorithm allows a migrating unit to be replicated to form multiple migrating units. For example, a migrating unit mu_i can be possibly replicated to form several migrating units $mu_{i1}, mu_{i2} \ldots mu_{ik}$. such that $weight(mu_i) = weight(mu_{i1}) + weight(mu_{i2}) + \ldots + weight(mu_{ik})$. These replicated migrating units keep the same set of documents but assigned with different *weight*. Thus, if a client is requesting a file in mu_i, it can be found in k different servers. A weighted redirection mechanism can be employed to select one among the k servers to handle the request. We would select one server among the k servers randomly with a probability proportional to the assigned weight. For example, *mu*={file A, file B} with weight 20 is assigned to server S_1 and *mu'* = {file A, file B} with weight 80 is assigned to server S_2. The probability to redirect request of file A to S_1 and S_2 should be 0.2 and 0.8 respectively. The weight assignment scheme enhances the load balancing when the migrating unit contains hot Web objects.

Algorithm 1: Modified Binning Algorithm

Step 1: Calculate $R_{exp}(S_j)$ for all S_j in S
Step 2: Determine W_{Sj} for all S_j (except S_{master}) in S
while ($R_{exp}(S_j) > R_j$ for some j in $1,2 \ldots n$) {
/* allocate migrating units among n server nodes in a round robin manner */
for (j = 1 to n) {
if ($R_{exp}(S_j) > R_j$) { // not fully allocated yet
Step 2.1: Get the migrating unit mu_i from MU with maximum value of weight(mu_i)/ (total file size of all documents in mu_i)
Step 2.2: if ($R_{exp}(S_j) - R_j >=$ weight(mu_i))
{ $W_{Sj} = W_{Sj} + \{mu_i\}$;
MU = MU – $\{mu_i\}$; } // case 1
else if ($R_{exp}(S_j) - R_j <$ $weight(mu_i)$) { // case 2
1. Duplicate mu_1 to form two migrating units mu_A and mu_B
weight (mu_A) = $R_{exp}(S_j) - R_j$,
weight (mu_B) = weight(mu_i) – weight(mu_A);
2. $W_{Sj} = W_{Sj} + \{mu_A\}$;
3. MU = MU – $\{mi_i\}$+ $\{mu_B\}$;
} } } }
Step 3: Ws_{master} = MU;

While allocating the migrating units to each server S_j, the migrating unit with the largest weight-to-size ratio will be selected first (Step 2.1). Migrating unit with larger weight-to-size ratio implies that the migrating unit consists of documents that are more frequently requested and their file sizes are smaller. Migrating units with low weight-to-size ratio is not a good choice for duplication because either the documents in the migrating units are not requested frequently, or the file size is relatively large. With this allocation policy, we can reduce the cost of migrating unit duplication and data movement in the network. Migrating units with low weight-to-size ratio will be assigned to the master Web server. As master Web server stores all the documents locally, thus there is no duplication and distribution cost in allocating those migrating units. Although master Web server stores all the documents, it would only serve requests for documents belong to the migrating units that are assigned to it. Requests for other documents would be redirected to server that contains the requested documents.

4. Simulation Details and Results

4.1 Experimental Setting

Simulation was performed to study the performance of the proposed algorithms. The HOWTO section of the Linux Document Project (http://www.linux.org) and the Web server logs collected from the Department of Computer Science at University of Hong Kong (http://www.csis.hku.hk) are used for the tests.

Experiment 1 - Using randomly generated data

A Web document graph is built based on the HOWTO section of the Linux Document Project. Each node in the graph represents one file. There is a link from node *a* to node *b* if there is a hyperlink in file *a* to file *b*. The weight of the link from node *a* to node *b* is defined as the probability that a visitor would request node *b* after node *a*. The weight of each node (file) is the product of the total number of requests on the file and size of the file.

Entry point is the 1st file requested by a visitor when they visit the Web site. It is a reasonable assumption that a Web site only has a few well-known entry points. In this simulation, we assume that there are two *entry points* in the Web document graph.

The HOWTO section consists of groups of html files. Files belonged to the same group are html files included in the same tutorial topic. We assumed that the weight of link between any two files is 0.8. For hyperlinks from the entry points to each group of html files, we assume that 2% of them have a weight of 0.5, 13% of them have a weight of 0.3 and 85% of them have a weight of 0.05.

The Web document graph would be used to generate random sequence of requests using Algorithm 2. The data generated would be used as input for our simulation experiment. We assumed that client-side caching is enabled. Thus the files ever requested by the client would not be requested again within the same session. Algorithm 2 used to generate the requests

We assume that the inter-arrival time of requests within the same session is an exponentially distribution with a mean of 25 seconds. 131457 randomly generated requests from 5000 clients are used as input to perform the migrating unit grouping. After that, another 264275 randomly generated requests from 10000 clients are used to evaluate the effectiveness of the document distribution algorithm. Simulation is carried out with different combination of *min_prob* (used in grouping) and number of Web servers.

Algorithm 2: Random Request Generator
Let *link_set* is a FIFO that stores the set of links not used yet for generating the next request.
Let *doc_set* is a link list that stores the documents to be requested in this visit session.
Step 1: Select an entry point randomly.
Step 2: Add the document selected in step 1 to *doc_set*. Add all the links associated with the document to *link_set*
Step 3: while (*link_set* not empty)
  Step 3.1: Get a link from the head of *link_set*. Remove the link from the queue.
  Step 3.2: If the document referenced by the link already in *doc_set*, then goto Step 3
  Step 3.3: Generate a random number *rand* from a uniform distribution between 0 and 1.
   If *rand* > weight of the link, then
    Step 3.3.1: Add the document referenced by the link to the doc_set.
    Step3.3.2: Add all the links associated to the newly added document to the head of the *link_set*.

Experiment 2 –Using real Web server logs

In this experiment, Web server logs collected from Department of Computer Science at the University of Hong Kong's Web servers are used.

The first half of the Web server logs is used as input to perform the migrating unit grouping. The second half of the Web server logs is used as the input to test the performance.

4.2 Redirection Overhead Analyses

In EWS, redirection would occur only when: (1) 1st requests to the system, redirected by the redirection server (2) When the request cannot be handled by the current server, as Web document is not stored in that server. Figure 1 and figure 2 show the average number of redirections encountered per client. In Experiment 1, the average number of Web requests generated by each client is about 26.4. Using our approach, the average number of redirection encountered by each client is 1.07 for two servers and 1.75 for 32 servers. For most redirection-based server architectures, such as the RobustWeb [8], the HTTP redirection is required for every HTTP request. The number of request redirection is equal to the number of requests. This would introduce overheads in serving HTTP requests and increase the workload for handling request redirection in the system. We can see that our approach provides a better request redirection mechanism.

Similarly, in experiment 2, there are a total of 236016 requests from 20970 clients, on average of 11.25 requests from each client. Based on our approach, the average number of redirections encountered by each client is 1.14

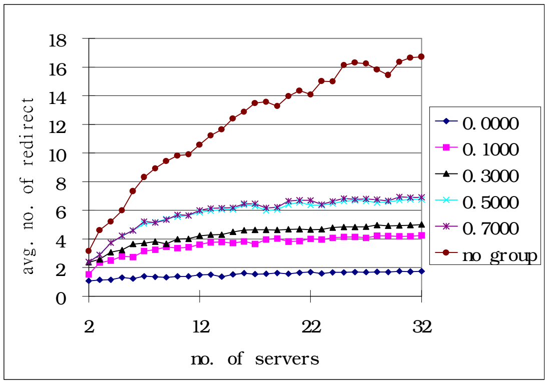

Figure 1: Average number of redirections encountered per visitor with different *min_prob* values in experiment 1

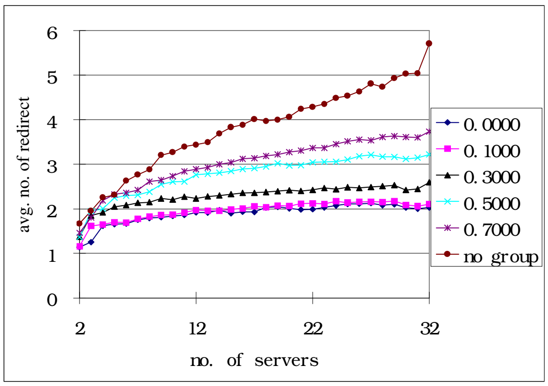

Figure 2: Average number of redirections encountered per visitor with different *min_prob* values in experiment 2

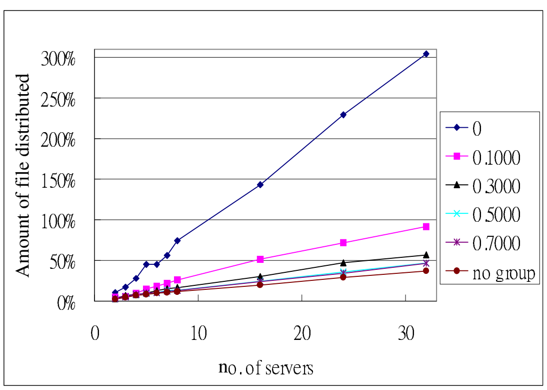

Figure 3: Amount of file distributed (*file distributed/total amount of files in the web site*) in experiment 1

for 2 servers and 2.03 for 32 servers. From the simulation results, we found that EWS helps to reduce the number of HTTP redirections effectively. It is likely that our grouping mechanism can provide good prediction of the future request patterns.

4.3 Storage Consumption Analyses

Figure 3 shows the total amount of documents stored in the web server nodes except master server for different combinations of *min_prob* and total number of Web servers. In our Web document distribution algorithm, we would always select migrating units with the largest weight-to-size ratio first. Thus web server nodes, except master server, are likely to be filled with small-size and more popular documents. Migrating units with low weight-to-size ratio, which are usually documents of large size, will be assigned to the master Web server. As master Web server keeps all the documents locally, thus there is no distribution cost in allocating those migrating units. In EWS, we would duplicate the migrating unit, if a single Web server cannot provide enough resource for a migrating unit. Thus we can reduce the number of documents need to be distributed to web server nodes in each Web document distribution process. The simulation results show that our proposed algorithm is effective in reducing the document distribution cost.

4.4 Load Balancing Analyses

We study the load balancing effects of the proposed algorithms based on the document distribution generated in Experiment 2. The *Load Balance Metric* (LMB) [1] is used as a performance metric to provide relative performance studies. To obtain the value of LMB, the peak-to-mean ratio of server load is measured at different sampling points (1 sampling point every hour) in the simulation. The server load is defined as the total size of Web documents transferred by the server. The LBM is obtained by calculating the weighted average of the peak-to-mean ratios measured, using the total server load as the weight at that sampling period. Smaller value indicates a better load balancing performance. For the purpose of comparisons, two other document distribution solutions are included:

1. *Round robin*: There is a front-end redirection server, which would redirect each incoming request to one of the servers in a round-robin manner. Each server keeps full replication of all the Web documents presented in the web site.
2. *Random*: Similar to EWS, except that the migrating units are distributed among the Web server nodes in a round robin manner without considering the weight of the migrating units. None of the documents are duplicated in this case.

In this experiment, two data sets are used as input in the simulation. Data set 1 consists of 1574781 requests from 94575 clients (2187 requests/hour) and Data set 2 consists of 1403075 requests from 85793 clients (1886 requests/hour).

Data set 1	**EWS**	**Random**	**Round Robin**
2 servers	1.130	1.122	1.009
4 servers	1.092	1.590	1.022
8 servers	1.303	2.890	1.060
16 servers	1.547	3.666	1.093
24 servers	1.600	4.200	1.114

Table 1: LMB measurement using Data Set 1

Data set 2	**EWS**	**Random**	**Round Robin**
2 servers	1.038	1.303	1.010
4 servers	1.207	1.965	1.028
8 servers	1.425	2.892	1.035
16 servers	1.691	3.787	1.069
24 servers	1.783	4.970	1,086

Table 2: LMB measurement using Data Set 2

Table 1 and 2 show the LMB measurements using data set 1 and 2 respectively. From the simulation results, we found that our approach outperforms the *Random policy* for all cases and its performance is close to the *Round-Robin policy* that is with full document replication. Our approach can achieve good load balancing since we consider the document size, request rate, and also the affinity of client accesses while grouping the Web documents.

5. Related Works

There are two different types of approaches to improve the performance of HTTP requests in WWW. The *client-side approach* tries to reduce the delay of serving HTTP requests by means of caching copy of Web object in the client side. Thus if the same Web object is requested in the future, the request can be satisfied by the local copy. This approach includes various types of client-side proxy cache solution, e.g. distributed proxy cache approach [14]. The benefits obtained by this approach depend on the cache-hit ratio. However, it is not easy to achieve good cache hit ratio in proxy cache server, because: (1) The proxy server does not have enough information about the structure of a Web site for making cache replacement. It is not easy to predict the client access pattern for making pre-fetch decision accurately. (2) Total cache size of a proxy cache server is limited. It cannot cache all the documents in WWW. (3) It is not easy to ensure document consistency in proxy cache server.

The *server-side approach* tries to reduce the delay of HTTP requests by using multiple machines in the server side to provide enough computing resource to handle the HTTP requests. In the past, several server-side approaches have been proposed. Probability Based Replacement (PBR) server array [2] consists of multiple server nodes. All the server nodes are connected to a central node through a high-speed LAN. The server nodes would get the Web documents from the central server for service. Some popular Web documents would be copied and cached in the server nodes. This architecture makes use of LAN broadcast mechanism, thus PBR cannot be implemented in a WAN environment.

Redirection-based hierarchical server architecture [3] makes up of two types of servers: *redirection servers* and normal HTTP server. Initially, system administrator should partition all the Web contents into groups and distribute them among different Web servers. Load balancing among Web servers is achieved by moving a subset of documents served in congested Web server to another Web server. This approach requires each document to have a unique base URL. All HTTP requests would be sent to the redirection server first. Then the redirection server would map the base URL to target URL of the document, and it returns the target URL to the client in an HTTP redirection message. In this approach, the redirection server may become a bottleneck easily because all HTTP requests must be redirected by the redirection server. The paper does not provide solution for Web document grouping and distribution.

In [4], a graph-based Web document partitioning algorithm is proposed. A collection of Web documents would be viewed as a directed graph. Every Web server can act as a *home server* and *co-op server* at the same time. All documents originally reside on a home server. Documents may be migrated from the home server to co-op servers for load balancing purpose. But document hyperlinks need to be modified to redirect user requests from home server to co-op server. In this approach, there is no duplication of documents in the system. The performance may be suffered if there are some popular Web objects. The system can only balance the workload by migrating document without duplication since one hyperlink can referenced to one Web document only.

In SWEB [5], the user requests are first evenly routed to SWEB processors via the DNS rotation to one of the available logical server, in a round-robin fashion. Each Web server contains a scheduler and would exchange system load information with another process. When a request is routed to a Web server, it can determine whether to process this request or assign it to another Web server. The use of DNS limits the total number of Web servers in the system [6]. In SWED, each Web server nodes store a whole set of Web content in each Web servers. This would cause storing of unnecessary copies of Web documents, which may result in waste of network and storage resource.

RobustWeb [8] consists of one or more front-end redirection servers and a set of back-end servers. The redirection server will then redirect the request to a document server that maintains the requested document. Access rates information is used to partition the documents across the document servers. They try to equalize the sum of access rates of all the documents stored at servers across all document servers. In their system, every request must be redirected by the redirection server. Thus there is heavy redirection overhead for serving large number of requests.

6. Conclusion and Future Work

In this paper, we present a new Web server architecture EWS that can scale in size to increase the computing power for handling large volume of user requests and provide extensible storage space to host large amount of Web documents. EWS has a good potential to achieve large throughput and fast response time for serving Web requests. The proposed grouping algorithm partitions the documents based on the information obtained from Web server logs. This grouping mechanism can improve the access locality and reduce the amount of request redirection overheads. Our document allocation algorithm can balance the load among all server nodes according to the run-time server configuration. In addition, the algorithm tries to avoid storing unnecessary copies of documents in the Web server nodes. This approach can reduce the cost of duplication and maintenance of document consistency. Our request redirection mechanism allows all servers to share the redirection overheads. Thus there is no performance bottleneck incurred by a centralized redirector.

There are still opportunities for further improvement on the EWS. We would improve the accuracy of prediction while grouping Web documents. More aggressive document duplication policies can be exploited to increase the hit rate of Web requests. We will also improve our migrating unit distribution algorithm, such that it can minimize the amount of documents to be relocated when the number of Web servers in the system increased or decreased.

Reference

[1] Bung, R. B.; Eager, D. L.; Oster, G. M.; Williamson, C. L. "Achieving Load Balance and Effective Caching in Clustered Web Servers". *The 4th International Web Caching Workshop,* 1999.

[2] Yeung, K. H. and Suen, K.W. "Probability Based Replacement Algorithm for WWW Server Arrays". *Proceedings of the International Conference on IEEE Parallel and Distributed Systems,* 1998, Page(s): 670 – 677

[3] Mourad, A. and Liu, H. "Scalable Web Server Architectures". *Second IEEE Symposium on Computers and Communications*, 1997, Page(s): 12 –16.

[4] Baker, S.M.; Moon, B. "Scalable Web Server Design for Distributed Data Management". *Proceedings of the 15th International Conference on Data Engineering,* 1999.

[5] Andresen, D.; Yang, T.; Holmedahl, V; Ibarra, O. H., "SWEB: Towards a Scalable World Wide Web Server on Multicomputers". *Proceedings of IPPS,* 1996, Page(s): 850 -856

[6] Cardellini, V.; Colajanni, M.; Yu, P. S. "Dynamic Load Balancing on Web-Server Systems". *IEEE Internet Computing*, 1999

[7] Arlitt, M. F.; Williamson, C. L. "Web Server Workload Characterization: The Search for Invariants", *ACM Proceedings of the ACM SIGMETRICS conference on Measurement & Modeling of Computer Systems,* May 23-26, 1996

[8] Narendran, B.; Rangarajan, S.; Yajnik, S. "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access". *Proceedings of the 16th Symposium on IEEE Reliable Distributed Systems,* 1997 , Page(s): 97 –106.

[9] Yang, C.S. and Luo, M.Y. "A Content Placement and Management System for Distributed Web-Server Systems". *Proceedings of 20th International Conference on Distributed Computing Systems,* 2000, Page(s): 691 – 698

[10] Cohen, E.; Krishnamurthy, B; Rexford, J. "Efficient Algorithms for Predicting Requests to Web Servers". *IEEE INFOCOM '99. Volume: 1,* 1999, Page(s): 284 –293.

[11] Cohen, E.; Krishnamurthy, B.; Rexford, J. "Improving End-to-End Performance of Web Using Server Volumes and Proxy Filters". *ACM SIGCOMM,* September 1998.

[12] Palpanas, T. and Mendelzon, A. "Web Prefectching Using Partial Match Prediction". *The 4th International Web Caching Workshop,* 1999.

[13] Cardellini, V.; Colajanni, M.; Yu, P. S. "Redirection Algorithms for Load Sharing in Distributed Web-server Systems". *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems,* 1999, Page(s): 528 –535.

[14] Rodriguez, P.; Spanner, C.; Biersack, E.W. "Web Caching Architectures: Hierarchical and Distributed Caching". *The 4th International Web Caching Workshop,* 1999.

[15] "Extensible Web Server Project," The Systems Research Group, Department of Computer Science and Information Systems, The University of Hong Kong. URL: http://www.srg.csis.hku.hk/EWS/