GBDU: An Effective Method to Achieve Load Balancing for Web Server System Design

*Shang-Wei Huang, Yi-Hsuan Lee, Tzu-Han Tsao, and Cheng Chen**

Department of Computer Science and Information Engineering

National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

Tel: (8863) 5712121 ext: 54734, Fax: (8863) 5724176

E-mail: {huangsw, yslee, thtsao, cchen}@csie.nctu.edu.tw

Abstract

Recently, with the growth of network, many applications on it have been developed to improve the human life. One of the powerful applications is the World Wide Web system. Nowadays, many enterprises develop web applications to service clients, such as stock trading and E-commerce. Therefore, to have a high performance web server system is important. In this paper, we design a web system based on the network infrastructure and propose an effective method called Grouping Based Dynamic Updating method to improve loading balance on web servers. In our method, we will use it to group the strongly related documents together. Then we distribute the groups among web servers and keep them to handle the same number of client requests. Finally, we dynamically update dispatch table, and sustain the loading balance on web servers constantly. According to the performance evaluations, our method is superior to other schemes in response time and loading balance status.

Keywords: *Web server, Load balance, Grouping, Dispatching, LBM*

1. Introduction

Recently, with the growth of the World Wide Web, many services have been developed on them [6-8]. Hence, the rapid expansion of the web service has led to exponential growth in the request rate to some popular sites [5]. In order to prevent web system from crashing down caused by mass requests, many web systems try to use clustered web servers to improve their performance and then upgrade their QoS [1, 4, 9]. One of the most important issues to design these systems is how to keep loading balance on web servers.

In this paper, we design a web system based on the network infrastructure today. This system includes a Dispatcher Server, a Monitor Server, and Clustered Web Servers. The Dispatcher Server focuses on receiving requests from clients and dispatches them to appropriate web servers. The Monitor Server manages the documents, and monitor the loading status. The Clustered Web Servers are responsible for handling the requests of clients, and send appointed data back to clients. In order to achieve loading balance, we also propose an effective loading balance method named *Grouping Based Dynamic Updating* (*GBDU*). This method includes grouping algorithm, group distribution algorithm, and dynamically update dispatch table techniques to achieve load balancing on our web system. We also construct a simulation and evaluation environment to evaluate it. According to the simulation results, we can show that our algorithm can achieve load balancing effectively.

The organization of this paper is as follows. In section 2, some fundamental background knowledge will be introduced. Our system architecture and design issues will be described in section 3. In section 4, we focus on expressing our proposed algorithm. In section 5, we introduce our simulation environment and compare our method with other schemes. Finally, we will give a conclusion and future work in section 6.

2. Fundamental Background and Related Work

2.1 Survey of Load Balancing Methods

With the rapid development of the WWW, how to achieve load balancing on web servers becomes more important when designing the web server system. Generally, we can classify the load balancing approaches based on different architectures or content availabilities [5, 7]. In the following, we give a brief survey on them.

There are four kinds of approach about the architecture. Client-based approach uses the software modification on the client side to decide how to achieve load balancing. DNS-based approach controls the URL-to-IP mapping to play an important role in load balancing [5]. Dispatcher-based approach uses a centralized request scheduling and completely controls the request routing [4]. Server-based approach lets all servers to participate in load balancing [1].

As for content availability, two approaches were proposed. Mirror-based approach copies contents to every node in the clustered web servers [3, 7]. It can increase the content availability, but suffers from the worse disk space utilization if the number of data sets increases. Content-based approach places different data sets on different web servers, and the client requests will be dispatched to a web server only if it contains that data set [7].

2.2 Related Work

2.2.1 National Center for Supercomputing Applications (NCSA) [3]

NCSA belongs to DNS-based and mirror-based approaches described in section 2.1. It comprises a couple of web servers, distributed file systems, and a primary DNS. The commonly used technique in NCSA to achieve load balancing is *Round-Robin Domain Name Server* (RR-DNS) technique. In order to reduce the message exchanging traffic in name servers, during a predefined time period, *Time-To-Live* (*TTL*), requests from the same client will be served by the same web server. Recent studies about NCSA RR-DNS have shown that it works well only when request rates of all clients are the same.

2.2.2 Extensible Web Server Architecture (EWS) [1]

EWS consists of multiple distributed web servers and a redirection server installed in the central site. When the first request in a new session comes to this system, the central site will choose a suitable web server to handle all requests in this session and redirect requests to the web server.

In EWS, a new document distribution algorithm has been adopted, which can distribute migrating units among web servers to fulfill the load balancing. The redirection mechanism used in EWS also can reduce the redirection cost and decrease the access latency. Besides, using migrating unit can save a lot of message exchanging overhead, and related files in the migrating unit can increase the hit ratio on web servers. However, the drawback of EWS system is that it cannot adjust the number of replicas according to server loading in real time.

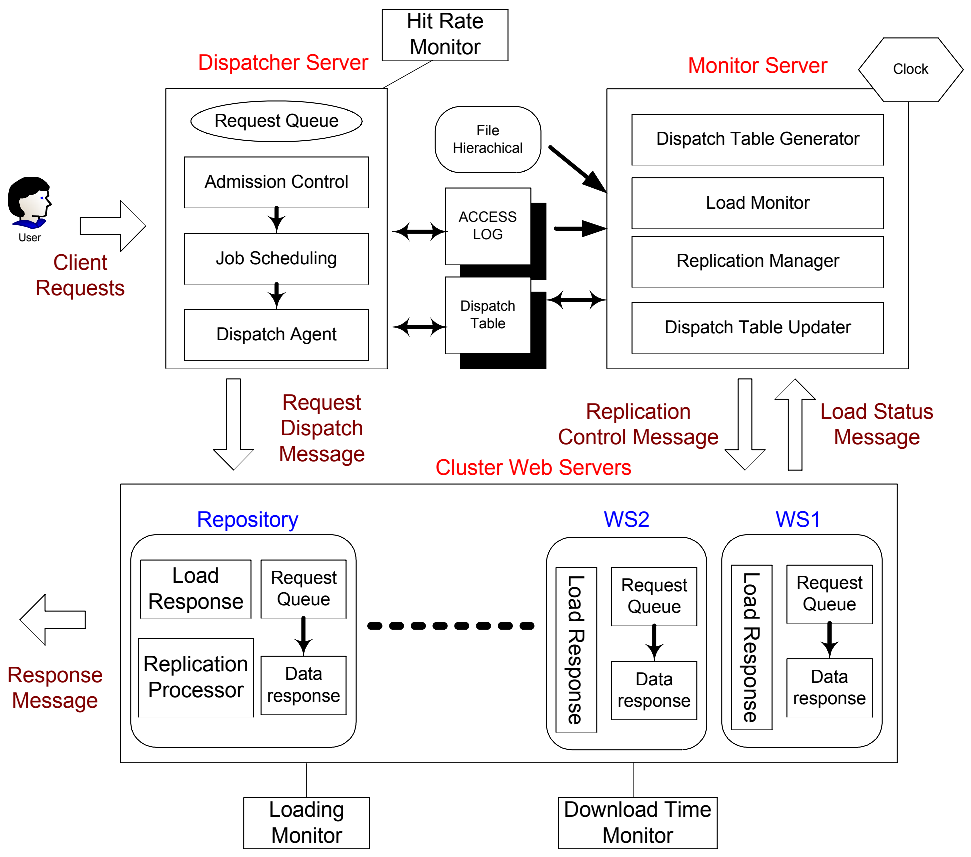


Figure 1. Proposed architecture of web server system.

3. System Architecture Design

3.1 Overview of System Architecture

According to our previous survey, we find that the dispatcher-based approach has higher ability to control the clustered web server system in load balancing status. Besides, in order to increase the space utilization, a content-based approach is a good method to be adopted. Content-based approaches are also appropriate for large-scale web server system that can efficiently increase the throughput of web servers by using well-designed content placement algorithm. Thus, in order to increase the performance, we will propose a system based on both dispatcher-based and content-based approaches. As shown in Figure 1, our system is composed of a dispatcher server, a monitor server, and clustered web servers. We will further introduce each server on our system in the following subsections.

3.1.1 Dispatcher Server

Dispatcher Server in our system can be regarded as a doorkeeper of the whole system and is usually the bottleneck of a web server system. In our system, we implement the scheduling algorithm in monitor server, which can upgrade the dispatch ability of dispatcher server. The dispatch ability here indicates the dispatch speed directly. If the dispatcher server has faster dispatch speed, the dispatch delay will be decreased.

3.1.2 Monitor Server

In our system design, the monitor sever plays an important role. It can communicate with the web servers to avoid them falling into overloaded status. The monitor server consists of *Dispatch Table Generator*, *Load Monitor*, *Replication Manager*, and *Dispatch Table Updater*. The dispatch table generator refers to the file hierarchy and access log and generates an appropriate dispatch table initially. The load monitor manages the loading status collection. If one of the web servers is near overloading, the load monitor will trigger replication manager to replicate contents on other servers to share the requests loading efficiently. Replication manager is responsible for replication action. We will propose an algorithm to decide how many replicas should be placed on web servers and how to adjust their counts in the next section.

3.1.3 Clustered Web Servers

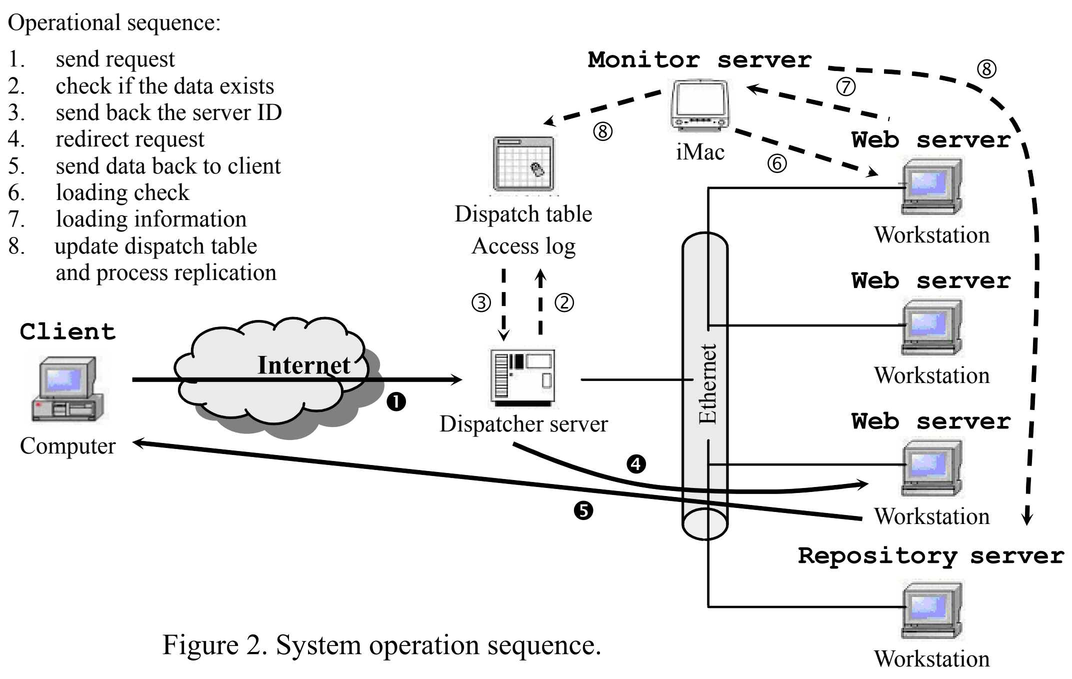


Figure 2. System operation sequence.

Recently, many web sites use clustered web servers instead of one powerful web server. The most important advantage by using clustered web servers is that when one web server crashes, the others can still service normally. In our system design, web servers store different data sets in their local disks, and the more popular data have more replicas stored on different web servers. We also use a repository server to store all documents as a backup server and to service light requests from clients. When a document is not popular enough to be stored on any web server, the repository will service the requests.

3.2 Operational Scenario

Figure 2 shows the overall procedure when a client wants to query a web page. The sequence ❶②③❹❺⑥⑦⑧ shows the processing steps in sequence when the clients start to send a request. Steps ❶❹❺ represent the request message flow, and ②③⑥⑦⑧ indicate the internal message control.

4. Grouping Based Dynamic Updating (GBDU) Method

In this section, we describe our *Grouping Based Dynamic Updating* (*GBDU*) method completely. GBDU is designed to achieve the following goals: (1) to increase the disk space utilization, (2) to minimize the download time of the client, and (3) to keep the loading balance among web servers.

4.1 Overview of GBDU Method

The main principle of our GBDU consists of three processing stages to achieve above goals. First, we propose a documents grouping method used to determine the migrating unit. In this stage, we define the relation of documents and find suitable threshold values to group documents. In the second stage, a Grouping Distribution technique and dispatch table is used to distribute requests among web servers in balance. Finally, we apply the Dynamic and Static Updating techniques to keep the load of each web server in balance. We explain each stage in the following sections clearly.

4.2 Documents Grouping Stage

Our documents grouping algorithm is similar to that used in EWS architecture [1]. In our algorithm, we partition documents of the system into many migrating units, and all files in a migrating unit may have some relationship among each other. When a client requests a web page belonging to one group, it has high probability to request other files in the same group later. This feature will increase the hit ratio on a web server and the utilization of disk space.

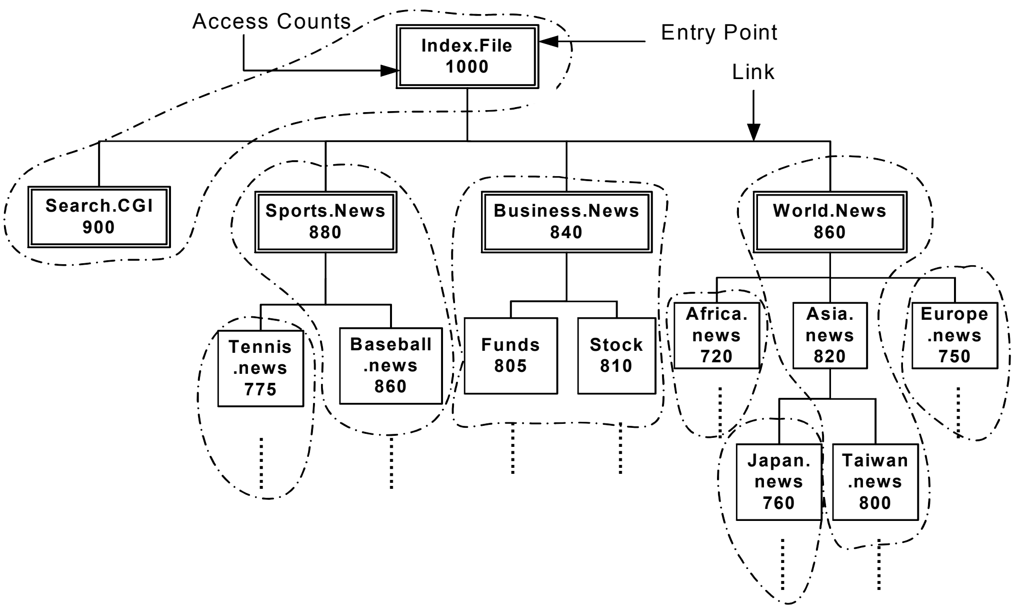


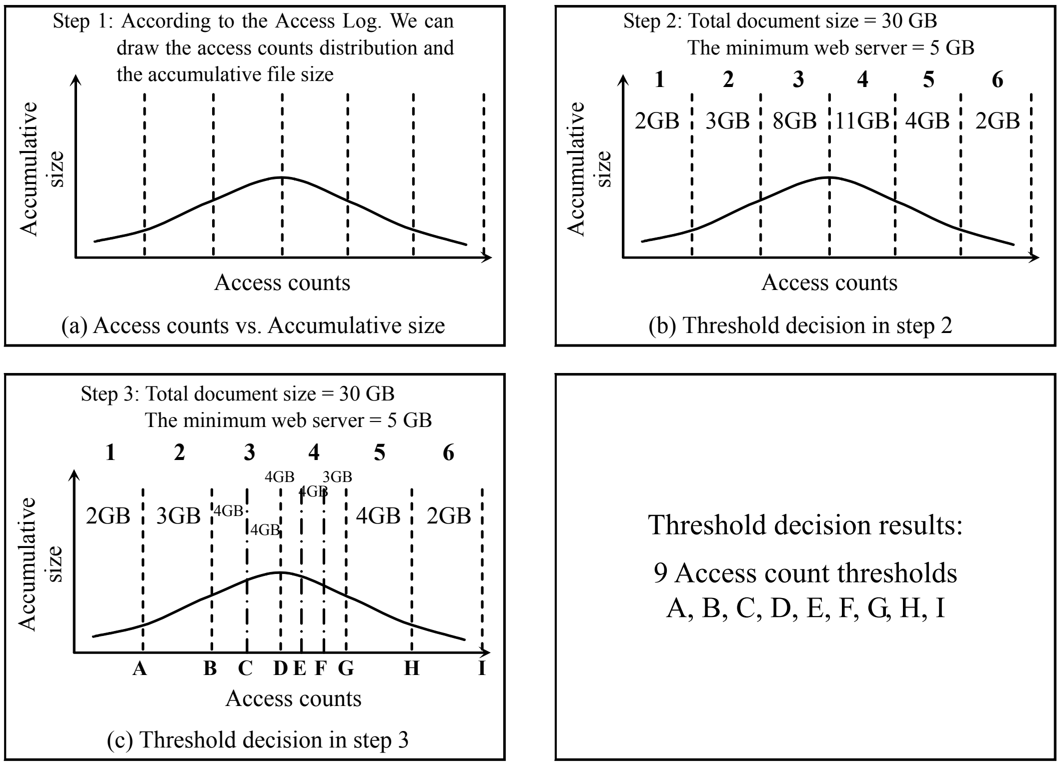
Figure 3. Example of grouping relation tree.

4.2.1 Definition of Documents Relation

The relation of documents plays an important role in designing a partition algorithm. As shown in Figure 3, we use a relation tree to define the relation of documents. The root of the relation tree represents the main web page of this system, and its child nodes represent the link files of the root page. Only files stored in the web system can be contained in the relation tree. Without loss of generality, we assume that most of the clients request from the root file first, and parent files always have more request counts than that their child nodes.

4.2.2 Threshold Values Selection

After defining the relation of documents, we have to decide some *threshold values*. *Threshold values* are a list of numbers recorded in a sorted array and used to determine which documents should be placed in the same group. Smaller group may lead to higher miss ratio when clients request many related documents, and larger group may waste the disk usage. In


Step 1: According to the Access Log. We can draw the access counts distribution and the accumulative file size
Accumulative size
Access counts
(a) Access counts vs. Accumulative size
Step 2: Total document size = 30 GB
The minimum web server = 5 GB
1 2 3 4 5 6
2GB 3GB 8GB 11GB 4GB 2GB
Accumulative size
Access counts
(b) Threshold decision in step 2
Step 3: Total document size = 30 GB
The minimum web server = 5 GB
1 2 3 4 5 6
2GB 3GB 4GB 4GB 4GB 4GB 3GB 4GB 2GB
A B C D E F G H I
Accumulative size
Access counts
(c) Threshold decision in step 3
Threshold decision results:
9 Access count thresholds
A, B, C, D, E, F, G, H, I
Figure 4. Threshold selection processing.

our system, we select threshold values based on the access distribution and the limitation of server disk space. First, we generate the relation graph of access count and accumulative document size. Then, we partition the request frequency into several areas according to the ratio of total size of documents and the minimal disk space of web servers. Finally, we check all areas and divide them if their documents sizes are larger than the minimal disk space of web servers. Figure 4 shows the threshold selection steps.

4.2.3 Grouping Method

After defining the relation of documents and selecting threshold values, we can begin to group documents. Our main idea is to traverse the relation tree by Depth First Search method. As shown in Figure 3, for example, we traverse the relation tree from **index.file**. Assume that our threshold values are {Threshold #1=900, Threshold #2=800, Threshold #3=700}. Using

DFS method, we can obtain 8 groups as shown in Figure 3. Our method can group documents into appropriate migrating units with the following features. (1) The size of each migrating unit cannot exceed the disk space of any web servers. (2) Documents in a migrating unit are strongly related. (3) According to our grouping method, migrating unit containing lower level nodes usually has smaller size. This feature can lead the disk space to not being wasted when we place the less popular migrating unit on servers.

4.3 Group Distribution Stage

We directly let a group as a migrating unit. We have to well distribute migrating units to appropriate web servers and dispatch requests from clients among web servers in balance.

4.3.1 Group Distribution Method

In our group distribution method, we focus on achieving the following goals. (1) The total size of documents in a web server satisfies the disk space constraint. (2) Documents are distributed to appropriate web servers such that all servers are loading balance. (3) More popular documents have more replicas placed on different web servers. This can reduce the loading of a single web server. Figure 5 shows the algorithm used to determine the number of replicas of a document during distributing.

4.3.2 Construction of Dispatch Table

After distributing groups to web servers, a hash table is constructed to store the mapping information between documents and servers. In our system, we use the basic collision resolution by chaining and division methods to construct hash table and hash function respectively. Formula 4-1 is used to calculate the hash table size. In order to reduce the table size, we don’t need to store all files and their mapping. This is because that if a request misses

Input:

A group array: **Group[0], Group[1], Group[2], …, Group[m]**

Web Server array: **WS[0], WS[1], WS[2], …, WS[N-1]**

Output:

New dispatching table T such that every access count on web servers is equivalent

Program:

```
Group_Distribute (*Group, *AC, N){
  do{
    GID = Highest_average_AccessCount(Group);
    WS_ID = min_ac_webserver(N);
    do{
        WS_ID=next_min_ac_web(N);   // find the 2nd, 3rd,…. Nth mini access count web
        If(WS_ID==Null && EVERY_WS_SAPCE>0) GID=next_high_avg_AccessCount();
        If(WS_ID==Null) goto final_step; // if can't find enough space to place, then to final step
    }while(WS[WS_ID].freespace()<group[GID].size || WS[WS_ID].contain(GID)==TRUE)
    WS[WS_ID].add(group[GID].files); Group[GID].replica++;
    Group[GID].now_avg_ac = Group[GID].original_avg_ac / Group[GID].replica;
  }while(min_free_space(N) >0)
final_step:
  WS_ID=min_free_space_webserver(N);    // find out the web server owns the min free space
  Expect=WS[WS_ID].total_access_count;
  for( i=0; i<N; i++){
    if(WS[i].total_access_count>Expect) WS[i].remove_replica();
    if(WS[i].total_access_count<Expect) WS[i].add_group();
  }}
```

Figure 5. Group distribution algorithm.

in the hash table, it can be redirected to and service the repository server. Hence, information of file placed on repository server is unnecessary to be stored in dispatch table.

$$X = \sum_{i=1}^{n} \sum_{j=0}^{Ci} FCij \quad \text{where} \begin{cases} FCij \text{ : file count of group } j \text{ on WS } i \\ Ci \text{ : group count of WS } i \\ n \text{ : web server count} \end{cases} \qquad (4.1)$$

4.4 Dynamic and Static Update Dispatch Table Stage

After grouping and distributing documents to web servers, we still can't guarantee that the balance status will sustain for a long time. This is because popularities of documents will be changed according to the access behavior. Hence, in our system, we propose both static and dynamic methods to keep the load balancing for a long time.

The idea of statically updating dispatch table is based on the message exchanging in a predefined time period. By communicating with web servers periodically, we can obtain loading information of every web server and adjust the dispatch table statically. According to the access log file, if any web server is under imbalance status, we must adjust the replicas placed on all web servers. A suitable time period can be chosen based on the access behavior.

In addition, we also must prevent single server from receiving mass requests. This situation occurs when a document becomes popular suddenly but it has only one replica in the system. In our dynamically updating method, web servers are asked to alarm when they are near overloading. We can use the double threshold (Thr2) mechanism to avoid this crashing situation. When a server is near overloading, it should be suspended to allow us to adjust contents of web servers and prevent another server to overload again.

Finally, in order to response current popularity of documents and avoid the repository server overloading, we should dynamically update the web server contents and group contents. If a group stored in the repository server becomes popular, it should replace the less popular group in web servers. Besides, if only one document in a group stored in the repository server becomes popular, it should be moved to its parent’s group that can keep documents in a group

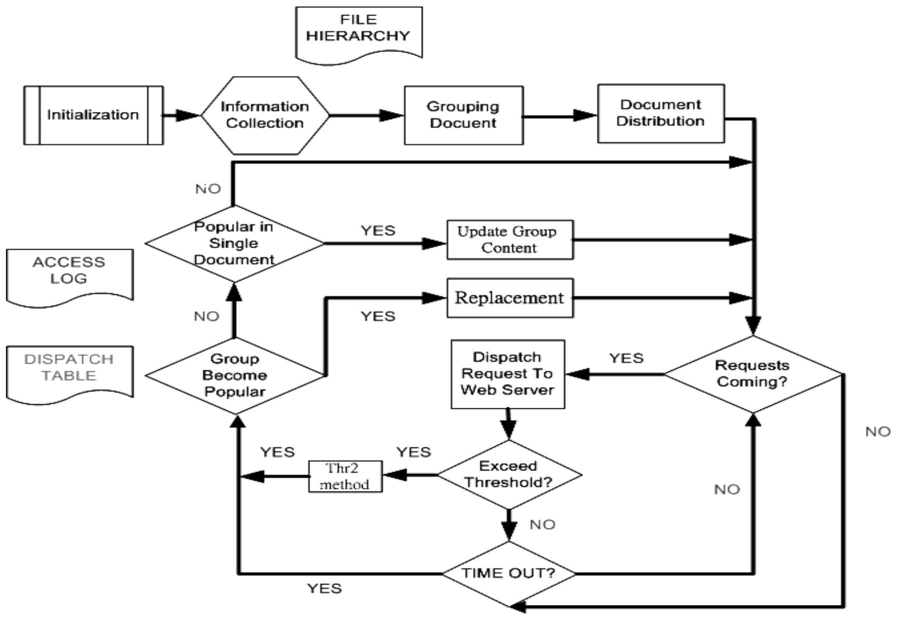


Figure 6. Overall flowchart of our system design.

still strongly related. This updating technique can increase the hit ratio on web servers. The overall flowchart of our system design is shown in Figure 6.

5. Simulation Environment and Performance Evaluations

5.1 Simulation Parameters

We construct a simulation environment based on system architecture proposed in section 3. Figure 7 contains the overall architecture of our simulation environment. In order to evaluate the system performance, we define some related parameters in Table 1 and list some important factors as follows. (1) Assume that the total document size is always larger than the web server space. So we must control the relation among parameters to prevent all documents from being put on the same web server like mirror-based system. (2) According to the web server access log of NCTU CSIE, we found the average access counts are 200 and 300 per minute and a large-scale web system may exceed 400 access counts per second. Without loss

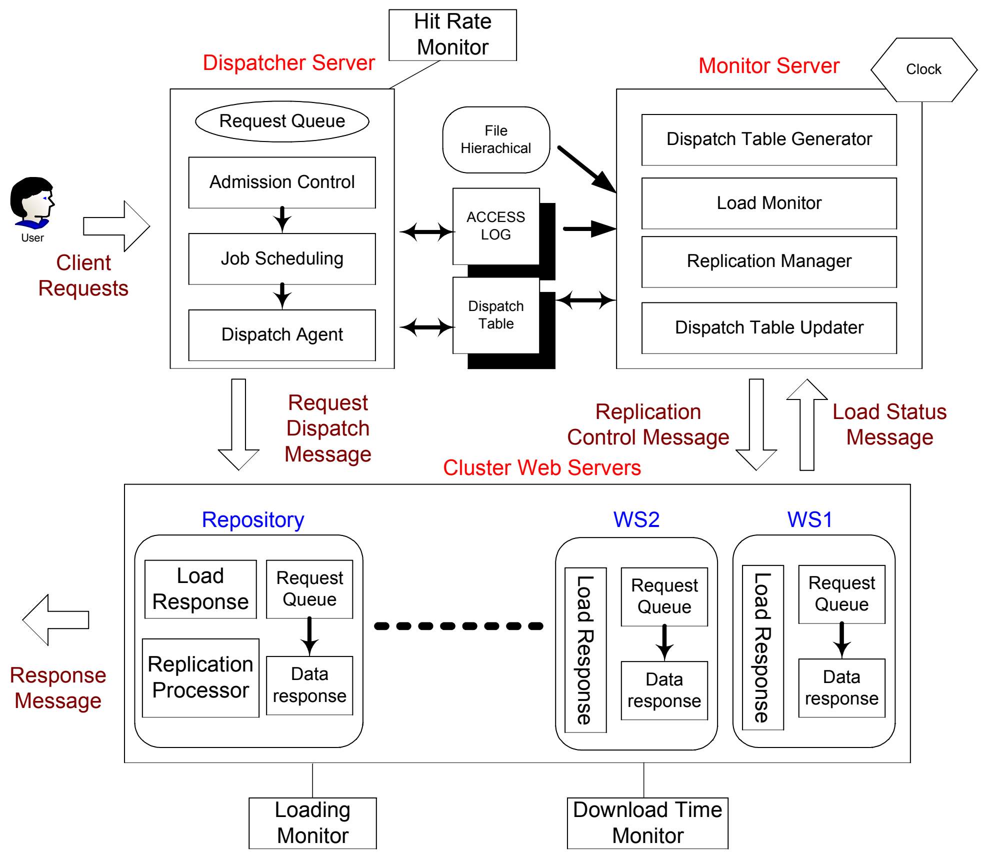


Figure 7. Overall architecture of our simulation environment.

Table 1. Parameters used in our simulation environment.

Parameter	Notation	Value
Web Server Counts	Wn	2 ~ 32
Number of Documents	Dn	100000
Web Server Storage	Swi	800MB ~ 1024MB
Document Size	Dsi	100KB ~ 1024KB
Request Arrival Rate	R	100, 200, 400, 800
Probability of Request Document [10]	P	PureZipf, Zipf 90/10, Zipf 80/20
Processing Power	PW	1 ms/KB
Redirection Delay	RD	0.1 ms/request

of generality, we set the parameter as 100 to 800 per minute.

5.2 Performance Evaluations

5.2.1 Load Balance Metrics (LBM) [2]

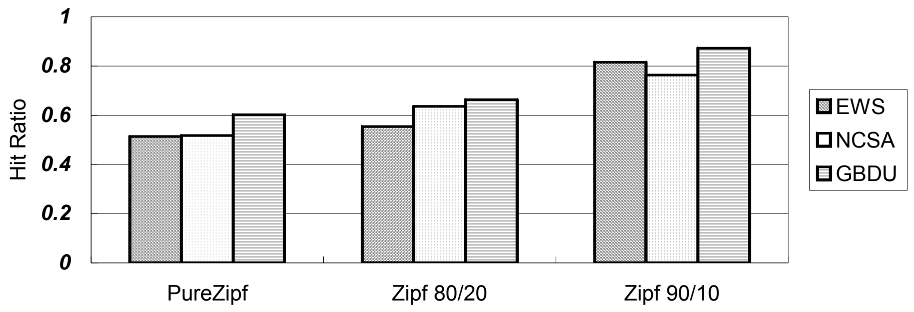


Figure 8. Hit ratio vs. Access distribution.

In order to evaluate the load balance status, we adopt the LBM (Load Balance Metric) to measure our system performance. The value of the LBM ranges from 1 to *n*, where *n* is the number of web server. Smaller values of the LBM indicates the better loading balance performance. The measure function is shown in the following.

$$\frac{\sum_{j=1}^{m} peak\ \_loadj}{(\sum_{j=1}^{m}\sum_{i=1}^{n} loadi\ , j)/n} \qquad (5\text{-}1)$$

5.2.2 Comparison of GBDU with Other Schemes

Before comparing with other schemes, two scheduling methods are cooperated with our GBDU method. One is called GBDU with Round Robin (GBDU_RR) method and the other is GBDU with Least Loading (GBDU_LL) method. GBDU_RR method services the clients by using round robin method. GBDU_LL method chooses a server with the least loading to service. In the following, we compare and evaluate those of GBDU_LL, GBDU_RR, EWS, and NCSA methods.

As shown in Figure 8, we can find that our GBDU method is superior to EWS and NCSA methods in hit ratio. It is because we keep the document relation and group them

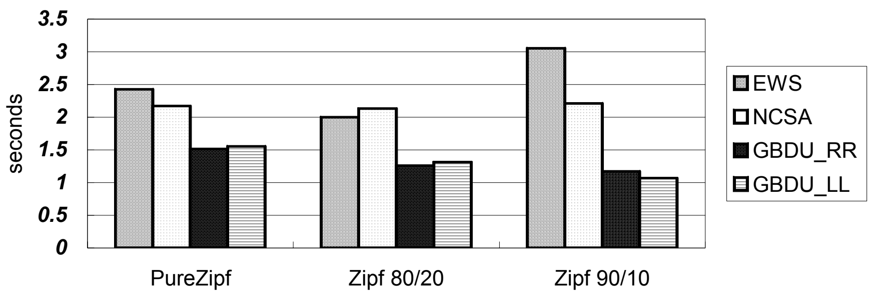


Figure 9. Download time vs. Access distribution.

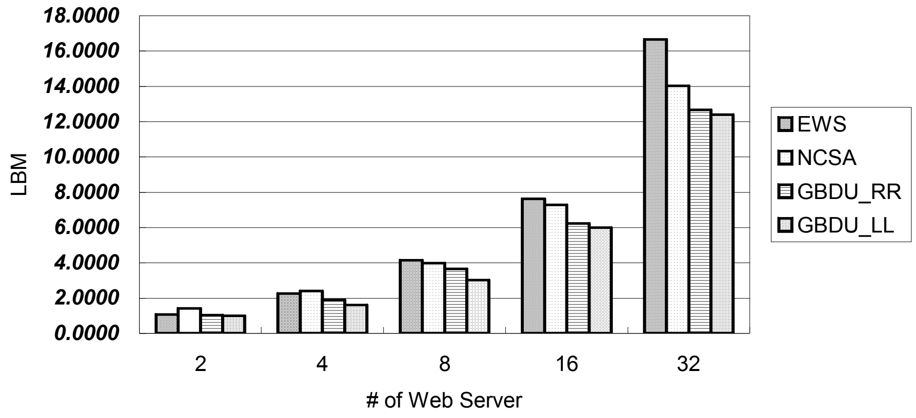


Figure 10. LBM vs. Number of web server.

together. The hit ratio of GBDU_RR and GBDU_LL are the same because they are based on the same grouping method. Figure 9 shows results of different schemes under different request distribution. According to the observation from the status of web servers, we can find that requests are queued in repository server under Pure Zipf distribution. In this case, our GBDU_LL method can save about 65% and 52% download time than EWS and NCSA methods respectively. In Figure 10, we show the LBM under different schemes and change the number of web servers. It is obvious that GBDU_LL still performs well no matter in small-scale or large-scale web system.

Finally, we trace the loading status of different schemes and evaluate their LBM values

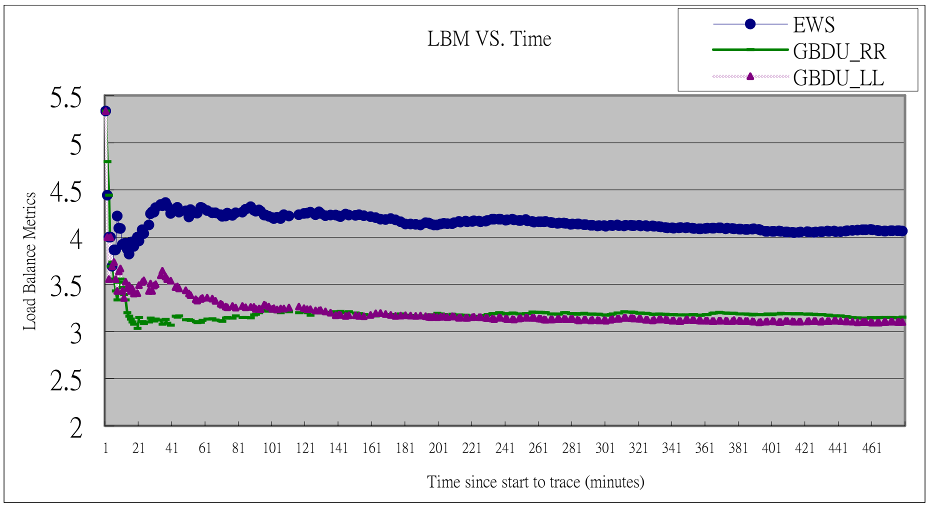


Figure 11. LBM vs. Trace time.

in different time stamp. In Figure 11, we trace the loading status on the grouping based methods for 8 hours. It shows that our method can short about 25% of the time than EWS method to enter stable status.

6. Conclusions and Future Work

In this paper, an effective grouping algorithm named GBDU has been proposed. It groups strongly related files into a group as a migrating unit, which can increase the hit ratio of web server. GBDU also contains a popularity-based distributing algorithm, which let the more popular documents have more replicas on different web servers to share their requests. We also construct a simulation environment to evaluate our algorithm. Based on the simulation results, GBDU can improve the LBM value about 30% and 36% better than NCSA and EWS method respectively. On the other hand, it can save about 52% and 65% download time compared to NCSA and EWS methods, and achieve better QoS no matter in small-scale or large-scale web systems.

In order to enhance the ability of our system, there are still some attractive issues can be exploited in the future. First, an effective and accurate access behavior predicting method is important to the web system nowadays. By accurate predicting the access behavior, we can determine the number of replicas more precisely. Second, with the growth of web system, web can offer many different type of service. How to provide better QoS under this circumstance become one of the most attractive research issues in web system design. Finally, with the different service in web system, relations among documents will become much complex. If these relations can be analyzed and defined more precisely, we can develop a more effective method to achieve loading balance in web system.

References

[1]. Ben Chung-Pun Ng and Ch-Li Wang, "Document Distribution Algorithm for Load Balancing on an Extensible Web Server Architecture", *Proc. of the First IEEE/ACM International Symposium on Cluster Computing and the Grid,* pp. 140 –147, 2001.

[2]. Richard B. Bunt et al., "Achieving Load Balance and Effective Caching in Clustered Web Servers". *Proc. of the fourth International Web Caching Workshop*, pp.159-169, 1999.

[3]. T. T. Kwan, R. McGrath, and D. A. Reed, "NCSA's World Wide Web Server: Design and Performance", *Computer*, Vol. 28, No. 11, pp. 68-74, Nov. 1995.

[4]. L. Aversa and A. Bestavros, "Load Balancing a Cluster of Web Servers: Using

Distributed Packet Rewriting”*, Proc. of the IEEE International Conference on Performance, Computing, and Communications,* pp. 24–29, 2000.

[5]. V. Cardellini, M. Colajanni, and P. S. Yu, “Dynamic Load Balancing on Web-server Systems”, *IEEE Internet Computing,* Vol. 3, Issue 3, pp. 28–39, May-June 1999.

[6]. V. Cardellini, E. Casalicchio, and M. Colajanni, “A Performance Study of Distributed Architectures for the Quality of Web Services”, *Proc. of the 34th Annual Awaii International Conference on System Sciences*, pp.3551-3560, 2001.

[7]. Cheng Zen Yang, Yi Shou Lin, and Cheng Chen, “An Effective Request Distribution Mechanism for Improving Load Balance in Web Server System”, *Proc. of the International Computer Symposium, Taiwan, R.O.C., Workshop on Computer Network, Internet and Multimedia,* pp.191-198, 2000.

[8]. E. Casalicchio and M. Colajanni, “Scalable Web clusters with static and dynamic contents”, *Proc. of IEEE International Conference on Cluster Computing,* pp. 170–177, 2000.

[9]. ***Microsoft Visual Studio***, http://msdn.microsoft.com/vstudio/default.asp

[10]. ***Reference on Zipf’s Law***. http://linkage.rockefeller.edu/wli/zipf/