Document Replication and Distribution in Extensible Geographically Distributed Web Server

Ling Zhuo Cho-Li Wang Francis C.M.Lau
Department of Computer Science and Information Systems
The University of Hong Kong
{*lzhuo, clwang, fcmlau*}*@csis.hku.hk*

Abstract

A geographically distributed web server (GDWS) system, consisting of multiple server nodes interconnected by a MAN or a WAN, can achieve better efficiency in handling the ever-increasing web requests than centralized web servers because of the proximity of server nodes to clients. It is also more scalable since the throughput will not be limited by available bandwidth connecting to a central server. The key research issue in the design of GDWS is how to replicate and distribute the documents of a website among the server nodes. This paper proposes a density-based replication scheme and applies it to our proposed Extensible GDWS (EGDWS) architecture. Its document distribution scheme supports partial replication targeting only at hot objects among the documents. To distribute the replicas generated via the density-based replication scheme, we propose four different document distribution algorithms: Greedy-cost, Maximal-density, Greedy-penalty, and Proximity-aware. A proximity-based routing mechanism is designed to incorporate these algorithms for achieving better web server performance in a WAN environment. Simulation results show that our document replication and distribution algorithms achieve better response times and load balancing than existing dynamic schemes. To further reduce user's response time, we propose two document grouping algorithms that can cut down on the request redirection overheads.

1 Introduction

The ever-increasing popularity of the World Wide Web brings tremendous increase in traffic to popular websites. One prominent example is the web server for the 2000 Olympic Games' website (www.olympics.com), which needed to handle an average of 7900 requests per second [28]. Web users of such popular websites often need to put up with slow response time if the supporting webservers are not powerful enough. Various techniques have been introduced to empower web servers to meet the increasing user demand, such as web caching and web prefetching which try to mitigate the problem by placing the web documents closer to the users. Previous research however has shown that the maximum cache hit rate achievable by any caching algorithm is bounded [1]. Therefore, the problem is still not completely solved, and the need for building more powerful web servers that can handle large amount of HTTP requests efficiently persists.

There are generally two approaches to constructing a powerful web server. One of them is to use a powerful, expensive server machine with advanced hardware support and optimized server software. However, with the rapid advances in hardware and software technologies nowadays, a powerful server today can easily become inadequate or even obsolete the next day, and upgrades of the server hardware could be costly and

require substantial effort.

The other approach which is more flexible and sustainable is to use a *distributed web server* (DWS) system. In a DWS system, multiple server nodes are interconnected together to act as a single logical server. We classify a DWS system as either "locally" and "geographically", distributed based on how its server nodes are interconnected. A locally DWS (LDWS) system takes the form of a cluster which is built using either dedicated connections or via a private high-speed LAN. A geographically DWS (GDWS) system uses a public network (MAN or WAN) for its interconnections. The performance of an LDWS system is limited by the bandwidth of the system's connection to the outside network [9]; whereas a GDWS system, which distributes network traffic over multiple connections or networks, has stronger ability in handling large amounts of user requests. On the other hand, however, as the network speed of a MAN or WAN is poorer than that of a LAN-based network, document retrieval from other server nodes is an expensive operation. Therefore, an important strategy which can lead to efficient and effective document replication and distribution is important in GDWS design. Such a strategy for GDWS systems is the focus of this paper.

According to the degree of document replication among the server nodes, GDWS systems can be classified into three categories: *full-replication*, *non-replication*, and *partial-replication*. In a full-replication GDWS system, each server node receives and maintains copies of the entire collection of documents of the site [18]. Incoming requests are then distributed to the server nodes using content-blind mechanisms (e.g. DNS servers). Full-replication is most tolerant to system failures. Its obvious disadvantage is the wasting of disk space due to copies of documents that are not frequently requested. In non-replication GDWS systems, no documents are replicated [5]. They depend on content-aware distribution software to redirect a client request to the server node that has the requested document. Some advanced features can be added to such a GDWS system, such as when a server node is overloaded, some of its documents are migrated to other nodes. But if the system happens to contain many hot spots (i.e., popular web pages with extremely high request rates), to equalize the load is absolutely non-trivial.

In a partial-replication GDWS system, only a selected subset of the web documents are replicated [20, 40]. Partial-replication helps diminish possible disk wastage and remove possible hot spots. Since websites today are increasing in both popularity and size, it is believed that partial-replication GDWS system is the most promising solution.

In this research, we focus on document replication and distribution algorithms for partial-replication GDWS systems. These algorithms implement the mechanism to decide which documents to replicate, how many replicas and where they should be placed—i.e., the *document distribution scheme*.

A document distribution scheme in a GDWS system should meet the following requirements:

- **Load balancing**: Since most requests are for a small part of the entire collection of documents, it makes sense to duplicate frequently requested documents and distribute the copies to many server nodes so that these nodes can all share the load.
- **Reduced document redistribution traffic**: To adapt to users' changing access patterns, replication and distribution of document copies should be evaluated and re-executed periodically. When this is being carried out, some existing copies may need to be transferred from one node to another in order to realize the new document distribution. The communication amounts due to such transfers should be kept to the minimum since the bandwidth available in a WAN/MAN could be limited.
- **Extensibility**: Adding a new server node or new documents to the system should be easy and transparent, and the newly extended system should maintain the balanced workload. The redistribution of

document copies triggered by these additions should be done efficiently and with minimal disturbance to existing nodes and activities.

- **Proximity**: As the server nodes could be placed in different geographic regions, a web document may experience different popularities in different regions. In this case, a document's replicas should be distributed to where they are most wanted.

Existing document distribution schemes for GDWS systems employing partial-replication mainly use two different approaches. The first is to partition the documents among the server nodes and *dynamically* replicate a document or revoke a replica based on current global load situation and statistics [20]. As we will see in later sections, the performance of this approach is not most satisfactory, and there is room for improvement. The frequent replication and revocation of this approach causes much traffic inside the system. The other approach is to replicate and distribute the documents based on past access patterns [26]. However, how to update the document distribution based on new access patterns and current locations of documents is rarely discussed in existing literature.

In this research, we propose a new architecture for GDWS systems, which is called Extensible Geographically Distributed Web Server (EGDWS). With this new architecture, documents are periodically replicated according to their popularities during the past period. The replicated documents are then distributed among the server nodes based on current document locations so that internal traffic thus generated can be reduced. The system is extensible in the sense that when adding a new server node, the system can resume load balancing quickly and the existing server nodes' operations are minimally affected. We propose several different algorithms that would be needed during the operation of an EGDWS system. The replication algorithm makes use of the density attribute (the workload each unit of a document brings to a server) of each document to calculate the desired number of replica. We try and compare several different distribution algorithms. The Greedy-cost algorithm tries to minimize the traffic by keeping as many documents stationary as possible. The Maximal-density algorithm tries to balance the load among the server nodes. The Greedy-penalty algorithm tries to improve Maximal-density by allocating the documents in certain sequences in order to reduce traffic. The last algorithm, Proximity-aware algorithm, distributes the documents according to their popularities in different Internet regions. Results we obtained show that using these algorithms leads to shorter response times as perceived by the users, and more balanced load than using other existing dynamic schemes. The internal traffic is also reduced substantially.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 presents details of the architecture of our EGDWS. Section 4 studies our document replication and distribution algorithms. Section 5 presents the simulation results. Section 6 concludes the paper and suggests future directions in this research area.

2 Related Work

There has been extensive research work done on data replication. This general problem refers to how to decide on the number and placement of replicas across multiple servers in order to increase the overall system performance. Previous work on data replication problems mainly focused on networked systems, distributed database systems, and most recently, content delivery networks (CDNs).

The File Allocation Problem (FAP) [13] studies how to assign files to different nodes in a distributed environment in order to optimize certain performance metric (e.g., total network traffic). The files can have

multiple copies and they are allocated to different nodes with respect to the available storage capacity of these nodes. The FAP can be defined as follows: Given a network of M sites with different storage capacities and N files exhibiting various read frequencies from each site, allocate the objects to the sites in order to optimize the given metric. FAP has been proved to be NP-complete. Many previous research projects assumed that access patterns are known and remain unchanged. Some solutions for dynamic environments were proposed in [21, 4, 39]. A complete survey on models for FAP in computer network systems can be found in [13].

The document distribution problem of GDWS systems can be seen as an instance of the FAP, in which each document can have multiple copies and storage space limitation is considered. However, the solutions for the FAP cannot be used directly in a GDWS system for the following reason. Firstly, files in computer systems have little correlations and are usually allocated independently. On the contrary, web documents tend to have correlations, that is, some web documents are always accessed together in the same session. Allocating these documents to the same nodes can help reduce response time. Secondly, each server node of a GDWS system has a computational load capacity in order to attain certain service quality, which needs to be taken into consideration when allocating the documents. However, not many of the solutions for the FAP consider load constraints [16].

The problem of document/data distribution also exists in distributed database systems, which can be modeled in FAP-like formulations [2]. The author of [2] investigated the complexity of the data replication problem in database systems, and proposed several methods for obtaining optimal and heuristic solutions with the consideration of data allocation cost. The work of [19] and [7] studied the data allocation problem in multimedia database systems and video server systems respectively.

Distributed database systems consider the access dependency among the data. Analyses of user queries are performed in order to assist the decision of data placement among the distributed nodes. Most proposed algorithms try to reduce the volume of data transferred in processing a given set of queries [2, 19]. As a query always involves data that are spread among different nodes due to the design of distributed database system, these solutions are inapplicable to GDWS systems. In an ideal solution for serving web requests, all the requested documents belonging to one user session should be located at a single server node that is closest to the client.

Unlike the above systems, in a Content Delivery Network (CDN), a unit of replication/allocation is the set of documents in a website that has registered for some global web hosting service. There has been much work done on the replica placement problem in CDNs during the last several years. In [31], the replica placement problem in CDN is formulated as the uncapacitated minimum K-median problem [25]: Given M sites, we must select K sites to be centers, and then assign each input point j to the selected center that is closest to it; if location j is assigned to center i, a cost is incurred; the goal is to select K centers so as to minimize the sum of such costs.

In [32] and [31], different heuristics were proposed based on this K-median formulation to reduce network bandwidth consumption. The authors of [15] take storage constraints into consideration, and reduce the knapsack problem to the replica placement problem in CDNs. They propose several approximation algorithms that can minimize with a probability the average number of autonomous systems a client needs to traverse in fetching an object.

The replica placement problem in CDNs is quite similar to the document distribution problem in GDWS systems. However, there are some important differences between the two. Firstly, the objects in CDNs are sets of documents of different websites. Therefore, they are not correlated and can be allocated independently. Secondly, as most CDNs employ their own private high-speed networks [31], replica placement al-

gorithms for CDNs need not be too concerned with internal traffic caused by updating document distribution. In contrast, GDWS systems are constructed over public networks so that traffic among the server nodes may have significant influence on the performance and thus needs to be reduced. Thirdly, current research on CDNs assumes that the load capacity of each server is unlimited and therefore does not consider load balancing among the server nodes an issue [16]. This may not be a problem for CDNs because they can afford to deploy very powerful servers. However, for GDWS systems that are made of ordinary server machines, an unbalanced workload could be a hindrance to satisfactory performance.

In designs for web proxy caching, documents replication and distribution is an important element. Documents are placed in different proxy servers in order to reduce network latency. Two caching architectures, hierarchical architecture and distributed architecture, have been proposed to enable proxy caches to collaborate [37, 33, 6]. With hierarchical caching, caches are placed at multiple levels of the network. When caches of one level cannot satisfy a request, the request is directed to the caches at an upper level. Examples are the Harvest project [10], and adaptive web caching in [24]. In a distributed caching architecture, there are only two levels of caches: browser caches and proxy caches. When a request encounters a browser cache miss, it goes to a proxy cache, and proxy caches serve one another's misses. In order to decide which proxy cache to go to retrieve a missed document, all proxy caches keep meta-information about the content of the other cooperating servers. Examples include the Internet Cache Protocol (ICP) [38], Cache Array Routing Protocol (CARP) [36], and CRISP [14].

The main difference between document replication in cooperative proxy server architectures and that in GDWS systems lies in when to replicate a document. Proxy servers will not do it until the document is requested by some clients, assuming that it will be re-accessed again. GDWS systems, however, replicate documents based on speculatative information on their popularities, expecting that the replication can reduce the user response time as well as balance the overall system load.

3 Extensible Geographically Distributed Web Server

Our proposed Extensible Geographically Distributed Web Server (EGDWS) architecture has the following characteristics.

1. *Adaptive Document Replication*: The replication and distribution of documents are periodically updated in order to adapt to the new access patterns.
2. *Hot Object Replication*: We adopt a partial replication scheme, whereby popular documents are identified and replicated to different server nodes to achieve balance of load. Hot objects are identified based on analysis of the access log collected in the past period.
3. *Proximity-Aware Request Dispatching*: As the server nodes are globally distributed, it is desirable to find the server node that is "nearest" to a client. The dispatching takes into account proximity when performing request dispatching.
4. *Document Grouping*: Documents that are likely to be accessed in one user session are grouped together and are always placed in the same server node. This can reduce unnecessary request redirections.
5. *Extensibility*: Adding new server nodes or new web documents to the system is simple, without causing too much traffic or disturbance to existing nodes.

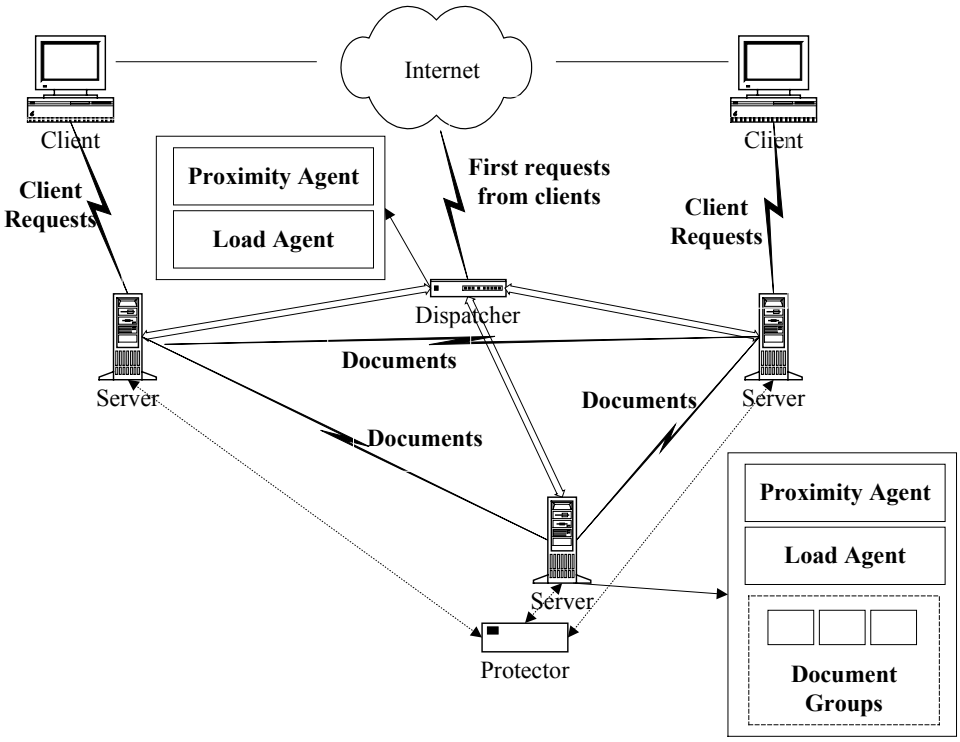


Figure 1: An Overview of EGDWS Architecture

Figure 1 gives an overview of the EGDWS architecture which consists of the following main components.

- *Request Dispatcher*: It represents the advertised URL of the website and is the entry into the system. It keeps track of each server node's documents, and redirects clients to a server node having a copy of the requested document, using HTTP redirection. If there are multiple server nodes having the copy, it chooses one using certain policy. The dispatcher has two agents: the load agent, which is in charge of collecting the server nodes' load information; and the proximity agent, which works together with the proximity agents on the server nodes to collect the network proximity information.
- *Document Distributor*: It periodically make decisions on replication and distribution of documents among the server nodes. There is a document agent in each server node. The document distributor collects the access records of the last period from these agents, and performs document redistribution according to decisions made.
- *Server nodes*: Server nodes are connected by a MAN or WAN and are in charge of handling client requests. Each of them holds part of the website's documents. If the requested document is in a node's storage, the node serves the request directly. The server node can also redirect the client to another server node based on a partial-URL-to-server mapping table. If the mapping table is not available, the node forwards the request to the request dispatcher at the central site.

In the rest of this section, we discuss several design issues in EGDWS.

3.1 Request Redirection

In our EGDWS system, we use HTTP redirection to route the incoming requests to the server nodes because HTTP redirection is a standard web protocol and has been deployed in many systems. In addition, HTTP redirection has better control of incoming requests and can achieve finer load balancing as compared with DNS-based routing. The drawback however is that each redirection would introduce additional round-trip delay and overhead for establishing a new TCP connection. In section 3.3, we discuss how to cut down the number of redirections through document grouping.

3.2 Request Dispatching

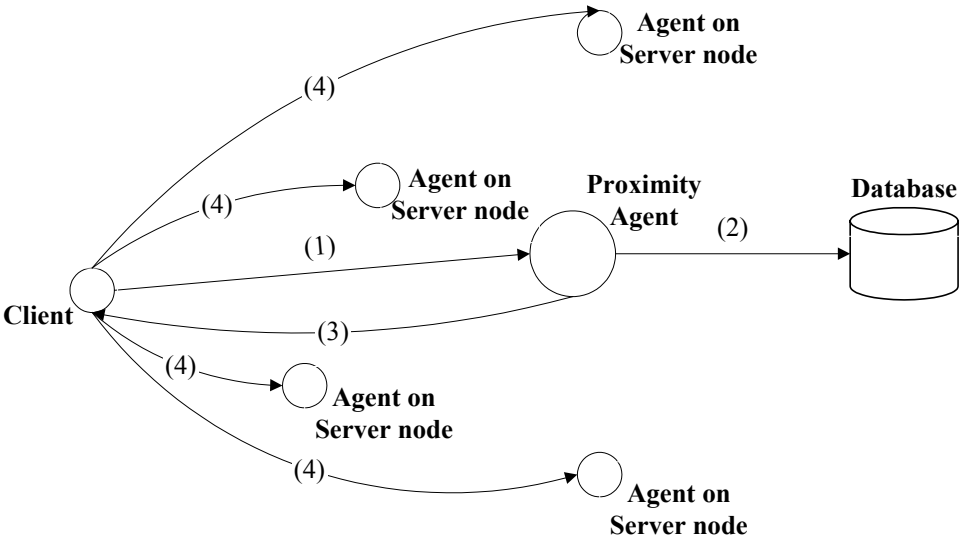
In selecting a server node to redirect the request, EGDWS needs a dispatching algorithm. As documents are partially duplicated, a content-aware request dispatching algorithm is used. A table is stored in the request dispatcher, which records where each document and its copies are located. In the case that there are more than one server nodes holding the requested document, the following methods can be used to arrive at a candidate.

- *Round Robin*: The requests for a document are assigned in a round-robin fashion among the servers having a copy. With this approach, the request dispatcher only needs to know where the documents are located.
- *Proximity-aware*: A request is assigned to the server node that is "closest" to the client. To estimate the proximity of a client to a server, we use RTT (round trip time) as an indicator. It is believed that using this metric can reduce client-perceived latency most efficiently [27]; also, it is the least expensive metric to measure, since existing utilities in UNIX for example (*traceroute* or *ping*) can be used and there need not be any any modification on the current network infrastructure, web browsers, or web server software.

  The proximity agent of the request dispatcher is in charge of collecting network proximity information. Each server node also has its own proximity agent, which reports network distances between the node and its clients to the proximity agent. These proximity agents together set up and maintain a central database of network distances from each server node to known client networks. In assigning a request, the request dispatcher looks up its database and find the server node closest to the client. If it cannot find the client network in the database, its proximity agent would broadcast to the agents on all server nodes and ask them to measure the RTT between the client and the server nodes using the ping utility. It would then collect the results and update the database. This procedure is shown in Figure 2.

3.3 Document Grouping

To reduce the user response time and the workload of the request dispatcher, we can try to cut down on the HTTP redirections in the system. We can capitalize on the fact clients tend to request related documents in a single session. Therefore, once a server node is assigned a client request, it should be relied upon to handle all the subsequent requests in the user's session, without having to redirect these latter requests to the request dispatcher. Thus, the user would only experience one HTTP redirection throughout the HTTP session. To achieve this, we place documents that have high "correlation"—that is, documents that are very likely to be



Step 1: Request reaches the proximity agent on the request dispatcher
Step 2: Proximity agent looks up the client network in the database
Step 3: Redirect the client
Step 4: If the client network is not found, the agents on the server nodes measure the RTT between them and the client

Figure 2: Proximity-aware Request Distribution

accessed in one single user session—on the same server node. Therefore, in EGDWS, we partition a website's collection of documents into groups, and use these groups as basic units in the document distribution scheme. There are two ways of grouping the document: directory based and access pattern based.

3.3.1 Directory-based

In this approach, the documents are grouped based on the directory tree of the website. Thus, the request dispatcher only needs to look at the pathname of a document to determine which group the document belongs to, and does need any mapping table. Such a directory-based approach is based on the assumption that the documents under the same directory tend to be accessed together. It works fine for the websites where contents of different directories have little correlation, such as websites that mainly host personal homepages. However, as this approach has no idea of the users' access pattern, it may not be able to identify the correlation between documents accurately.

3.3.2 Access pattern based

In this approach, documents are grouped based on users' access pattern extracted from the system's access log files. The grouping algorithm we adopt is similar to those used in web prefetching and cluster mining in data mining [11, 22]. The difference is that those other algorithms compute the probability that another document is accessed after one document is accessed. In our algorithm, however, the correlation between

two documents is computed as the probability that they are accessed in the same session, regardless of the access order. Also, our algorithm controls the sizes of the groups so that a group only contains documents that will be accessed in one user session, whereas other existing algorithms only have indirect control on group sizes.

The grouping algorithm goes through the following steps.

1. **Process the access log into sessions, and record the length of the longest session each document belongs to**. A user session is a series of interactions between a user and the web server during the span of a single HTTP connection. In our research, we assume that the requests from the same IP address are generated by the same user and the period between two consecutive requests in one session should not exceed one hour.
2. **Compute the correlations between documents and create a correlation matrix**. As already mentioned, the correlation between two documents is the probability that they are accessed in the same user session. Therefore, for each pair of documents, $P1$ and $P2$, we record the number of sessions in which they are accessed, in $s(P1)$ and $s(P2)$ respectively. We also maintain a counter $c(P1, P2)$ to count the number times that they are accessed in the same session. Then we compute $p(P1, P2) = c(P1, P2)/s(P1)$, the probability $P1$ is accessed in the same session with $P2$, and $p(P2, P1) = c(P1, P2)/s(P2)$, the probability $P2$ is accessed in the same session with $P1$. The correlation between $P1$ and $P2$ is the minimum of these values, and is added into the correlation matrix. We use the minimum of these two probabilities to avoid mistaking an asymmetrical relationship for a true case of high correlation. For example, a popular document $P1$ may be on the most common path to a more obscure document $P2$. In such a case, $p(P2, P1)$ will be high, perhaps leading us to think the documents are highly correlated. However, $p(P1, P2)$ could be quite low, as $P1$ is on the path to many pages.
3. **Create a graph corresponding to the matrix, and find cliques in the graph**. A graph is created based on the correlation matrix, in which each document is a vertex and each non-zero cell of the matrix is mapped to an edge with a length value. The length of an edge is equal to the correlation value between its two vertices. To reduce noise, we remove edges that are shorter than a given value (in our experiments, it is 0.1). We then group documents by identifying cliques in the graph. A *clique* is a subgraph in which every pair of documents has an edge between them; therefore, a clique contains documents that are likely to be accessed in one session. The detailed grouping algorithm is shown in Figure 3. In Step 1, we exclude the documents that are not related to any other documents from the grouping process. Grouping work is mainly done in Step 2. In constructing each clique, we always start from the two vertices that have the longest edge in between, that is, the two documents having the highest correlation. Then, all ungrouped vertices are examined to see if they are connected to all the vertices in the clique. If there are multiple such vertices, we choose the one whose shortest edge with the members of the clique is longest. In this way, we choose the document that has the highest correlation with the current documents in the group.

   The size of a clique is bounded by the longest length of the sessions of its members, which is computed in the first step. We set this bound because there is no need to include too many documents in a group, if the user will not access them in one session. Therefore, each time we add a document to a clique, we check if its size needs to be updated. At the end of the algorithm, in Step 3, all the remaining documents are grouped. As these documents have no correlation with any other documents, and each of them forms an individual group.

```
Step1 R = {vertices connected to at least one edge}
Step2 While R is not empty {
      2.1 Find the longest edge in R; its vertices are p1, p2
      2.2 V = {p1, p2}, G = R \ V, C = Φ
      2.3.l = maximal size of V
      2.4 While |V| ≤ l{
            For each vertex p in G {
                  If p is connected to all vertices in V {
                        Record the shortest edge between p and vertices
                         in V
                        Add p to C }}
            If C = Φ {
              delete p1, p2 from G and R.
              goto step 2 }
            Else {
                Choose the vertex v whose shortest edge to V is longest
                Add v to V, delete v from G and R, C = Φ
                Update l }}
Step3 Construct a group for each remaining vertex
```

Figure 3: Access Pattern based Grouping Algorithm

As the access pattern based approach is based on the analysis of the log files, its results reflect the user access pattern better than the directory-based approach. However, using it, the request dispatcher needs a mapping table to map the document to the group it belongs to, and therefore would require more computation.

4 Document Replication and Distribution Algorithms

In EGDWS, each server node holds a subset of the website's documents. A document distribution scheme is needed to decide how the documents should be replicated and distributed. As we discussed previously, previous research on data replication cannot be straightforwardly applied to partial-replication GDWS systems. Moreover, existing document distribution schemes are unable to achieve satisfactory performance as well as to reduce communication costs. Thus we propose a new scheme.

There are three main steps in our document distribution scheme. They are described as follows.

1. *Access pattern analysis*: analyzing log files collected from all participating web servers and identifying the popularity of each document.
2. *Document replication*: determining the number of replicas for each document according to its popularity and the available storage.
3. *Document distribution*: determining the distribution of the documents and their replicas. As this decision is made based on the document replication result as well as the current location of documents, this step can also be called "document redistribution".

In this section, we first formulate the document distribution problem in EGDWS. Then we discuss the document replication and distribution algorithms in our scheme.

4.1 Problem Formulation

Before the discussion, we first introduce the variables we will use. Suppose there are M server nodes and N documents in the system. S_j refers to jth server node. D_i $(i = 1, \ldots, N)$ refers to ith document or document group.

We have the following variables:

- s_i denotes the **size** of D_i. If D_i is a group, s_i equals to the total size of the group members.
- r_i denotes the **access rate** of D_i during the past period. If D_i is a group, r_i equals to the total access rate of the group members.
- w_i refers to the **weight** of D_i, which represents the workload D_i brings to the server node holding it. $w_i = \alpha \times w_{cpu} + (1 - \alpha) \times w_{disk}$. For dynamic documents, $\alpha = 0.5$. For static documents, $\alpha = 1$. In this research, we only consider static documents. Therefore, $w_i = w_{disk}$. If the document sizes are varied, $w_i = s_i \times r_i$; otherwise, $w_i = r_i$. If D_i is a group, w_i equals to the total weight of the group members.
- d_i refers to the **density** of D_i, which is computed as $d_i = w_i/s_i$. d_i represents the workload per unit storage of D_i brings to a server node.
- c_i denotes the **replica number** of D_i. If D_i is replicated, each of its replicas has size of s_i, weight of w_i/c_i, and thus density of d_i/c_i.
- R_i is **replica set** i. It is the set of document D_i's replicas.
- C_j denotes server node S_j's **storage capacity** and L_j denotes S_j's **load capacity**, which is the maximum number of simultaneous HTTP connections S_j can support. If $C_1 = C_2 = \cdots = C_M$ and $L_1 = L_2 = \cdots = L_M$, we call such a system *homogeneous* opposed to *heterogeneous* system. For a homogeneous system, C and L denote the storage capacity and load capacity respectively.
- W_j denotes the **weight** or **load** of S_j, and it equals to the sum of the weights of all replicas on S_j.
- G_j refers to the **density** of S_j, computed as W_j / (amount of used disk space in S_j).
- p_{ij} $(i = 1, \ldots, N, j = 1, \ldots, M)$ denotes the **cost link** between document set R_i and server node S_j. It equals to the number of bytes S_j needs to fetch from other server nodes if a replica of D_i is distributed to it. There are two cases in computing p_{ij}: (a) D_i is either a single document, or a document group and the grouping is not changed. We can see that p_{ij} equals to either zero or s_i, depending on whether S_j already holds a replica of D_i (b)D_i is a document group and the grouping of this period is different from that the past period. Therefore, p_{ij} refers to the total size of the documents that are in D_i and are not on S_j. For example, suppose there are 5 documents and 2 servers in the system. During the last grouping of documents, $D'_1 = \{a, b, c\}$ and $D'_2 = \{d, e\}$. D'_1 is placed on server 1 and D'_2 is placed on server 2. During this period, however, $D_1 = \{a, d\}$ and $D_2 = \{b, c, e\}$. Therefore, $p_{11} = s_d$, $p_{12} = s_a$, $p_{21} = s_e$, $p_{22} = s_b + s_c$.
- We also have the following variables to show whether a replica of D_i is placed to S_j after document distribution:

$$t_{ij}^l = \begin{cases} 1, & \text{if } D_i^l \text{ is allocated to } S_j \\ 0, & \text{otherwise} \end{cases} \quad j \in \{1, \ldots, M\}, i \in \{1, \ldots, N\}, l \in \{1, \ldots, c_i\}$$

In the document distribution scheme, we assume that the total size of the documents does not exceed the total storage capacity, that is, $\sum_{i=1}^{N} s_i \leq \sum_{j=1}^{M} C_j$. We hope the replication and distribution can be done

Step1. $R = \sum_{j=1}^{M} C_j - \sum_{i=1}^{N} s_i$, $R' = 0$, $c_i' = 0$ ($i = 1,\ldots N$)
Step2. Sort documents by decreasing density d_i ($i = 1,\ldots N$).
Find minimal density d_{min}.
Step3 **for** i = 1 to N {
3.1 $c_i' = d_i / d_{min}$
3.2 $R' = R' + s_i * c_i'$ }
Step4. **for** i = 1 to N {
4.1 $c_i = c_i' * R / R'$
4.2 if ($c_i >= M$-1){
$c_i = M$-1
$R' = R' - c_i' * s_i$
$R = R - c_i * s_i$ }}
Step5. **for** i = 1 to N { $c_i = c_i + 1$ }

Figure 4: The pseudo code of proposed Density Algorithm

under the following constraints: (1) Each server can only hold replicas whose total size does not exceed the server's storage capacity. (2) Each server can hold at most one replica of one document. (3) All replicas are placed on the server nodes. (4) The weight of each server is proportional to its load capacity.

Based on these constraints, we constructed an optimization problem whose objective function is to minimize the total communication costs needed to update the document distribution. This problem was proved to be NP-complete [41].

A replica placement that fulfills all the constraints is a "feasible placement." However, because of constraint (3), an instance of this optimization problem does not necessarily have a feasible solution. Therefore, in practical document distribution, we relax the constraint (3) to : each document has at least one copy in the system.

In this research, we confine our discussion on homogeneous systems and heterogeneous systems in which $L_1/C_1 = L_2/C_2 = \cdots L_M/C_M$, that is, the load capacity is proportional to the storage capacity. The document replication and distribution problem in other heterogeneous systems are much more complicated and is not covered in this research. It needs to be noted that although in the following discussion, we call D_i a "document" for simplicity, D_i can be either one document or one group of documents.

In following discussion, we present several approximation algorithms that use the available information in different ways to achieve load balancing as well as reduce the communication costs in relocating the documents. We first describe the document replication algorithm, and then the document distribution algorithms.

4.2 Document Replication Algorithm

Intuitively, we prefer to duplicate documents that bring more workload to the EGDWS system as well as have small sizes. Therefore, in the replication algorithm, we use the density of a document as the metric of its popularity. The larger a document's density is, the more replicas it should have, that is, D_i's number of replicas is roughly proportional to its d_i. The algorithm is thus named as "Density algorithm".

The Density Algorithm is shown in Figure 4. We first reserve the disk space equals to the total size of the documents, so that c_i for each document is at least 1. Step 2 sorts the documents by their densities de-

creasingly, and find the minimal density. In Step 3, each document gets a temporary replica number, which is computed in such a way that the densities of the temporary replicas are nearly equal to the minimal density. Normally, in Step 4, the replica numbers are computed according to the ratio between available disk space and the total size of the temporary replicas. Thus, the resulting replicas will still have similar densities. One special case is that the replica number c_i of D_i is larger than $M-1$. In this case, c_i is reduced to $M-1$, because a document can at most have M replicas and the space for one copy of it has been reserved in Step 1. We then subtract appropriate space from available disk space and the total size of the temporary replicas to adjust the ratio between these two values. After computing the replica number for each D_i, in Step 5, c_i is finally decided as an integer not exceeding M.

We give an example here showing how the Density algorithm works. Suppose there are 8 documents and 4 server nodes. For simplicity, we suppose these documents are of the same size 1 and the size of the server node is 4. The algorithm is running in the following order:

In Step1, we get R = 8.

After Step2 and 3, we get the set of densities: $\{30, 10, 6, 5, 4, 3, 2, 1\}$, $R' = 63$.

In Step4:

For D_1, $c_1 = 30 \times 8/63 \approx 4$. Since $c_i > M-1$, $c_i = 3$. $R = 8-3 = 5$, $R' = 63-30 = 33$.

For D_2, $c_2 = 10 \times 5/33 \approx 2$.

For D_3, $c_3 = 6 \times 5/33 \approx 1$.

For D_4, $c_4 = 5 \times 5/33 \approx 1$.

For D_5, $c_5 = 4 \times 5/33 \approx 1$.

For D_6, $c_6 = 3 \times 5/33 \approx 0$.

For D_7, $c_7 = 2 \times 5/33 \approx 0$.

For D_8, $c_7 = 1 \times 5/33 \approx 0$.

In Step5, the replica numbers are determined as $\{4, 3, 2, 2, 2, 1, 1, 1\}$.

From this example, we can see that the larger is a document's density, the more replicas it has. However, even though D_a is extremely popular, its c_a cannot exceed the number of server nodes. Although this may decrease the gap between the density of D_a's replicas and the densities of other replicas, the load balancing won't be affected because D_a's replicas are placed on all server nodes. On the other hand, when the number of server nodes is very large, replicating the popular documents on each server node may result in the decrease in other documents' replica numbers. This is reasonable because previous research has shown that most accesses are for a very small part of popular documents [3, 29].

The Density algorithm replicates the documents according to their densities under the storage limitation. Its time complexity is $\Theta(NlogN + N)$. From Step 4, we know that for any two documents D_u and D_v, if $1 < c_u, c_v = M$ and $d_u > d_v$, $d_u/(c_u-1) \approx d_v/(c_v-1)$; and if $c_v = 1$, $1 < c_u$ and $d_u > d_v$, $d_u/(c_u-1) \approx d_v$. Thus, we can assume that using the Density algorithm, for any two documents D_u and D_v, if $d_u > d_v$, then $d_u/c_u > d_v/c_v$. This conclusion will be used in the following document distribution algorithms.

4.3 Document Distribution Algorithms

After determining the replication of the documents, the replicas (in the following discussion, "the replicas" refers to all the documents and their replicas) need to be distributed among the server nodes. In this subsec-

```
Step1.  Sort (i, j) pairs by increasing cost, p_ij
Step2.  for j = 1 to M {V_j = C_j }
Step3.  for each (i, j) in the sorted list{
          if (c_i > 0) && (V_j ≥ s_i) && (t_ij^l = 0, l = 1,…c_i){
              // allocate a replica of D_i to S_j
              c_i = c_i –1, V_j = V_j – s_i
              l = c_i, t_ij^l = 1 }}
Step4.  Adjustment algorithm
```

Figure 5: The pseudo code of proposed Greedy-cost Algorithm

tion, we first present several algorithms that try to reduce internal traffic of the EGDWS system. Afterwards, an algorithm that is aware of network proximity is proposed. All of these algorithms need an adjustment algorithm to help them to guarantee each document has at least one copy in the system.

4.3.1 Greedy-cost Algorithm

We first propose an algorithm which aims to minimize the traffic by keeping as many as documents unmoved, without caring if the load of the server nodes is balanced. In distributing the replicas, each time, it chooses a pair of ¡document, server node¿ which is associated with the smallest communication cost among the remaining pairs.

This algorithm is shown in Figure 5. It first sorts the $< i, j >$ pairs by the communication costs between D_i $(i = 1, \ldots, N)$ and S_j $(j = 1, \ldots, M)$ increasingly. Then based on this order, a replica of D_i is allocated to S_j, if S_j has enough empty space and has not been assigned the same replica in this period. The total time complexity of Greedy-cost algorithm is $\Theta(MNlogMN + MN)$. From the definition of p_{ij}, we know that this algorithm tries to reduce the communication cost by only duplicating and transferring the documents whose replica numbers in this period are larger than their copies in the system and keeping most of other documents are unmoved.

Although this algorithm does not consider the servers load during the document distribution, we still expect it to succeed in spreading the load among the server nodes because the Density algorithm has replicated the documents as needed. The main problem with the Greedy-cost algorithm is that it may not be able to adapt to the change of users' access pattern quickly, for it tends to keep the documents where they are.

4.3.2 Maximal-density Algorithm

Unlike Greedy-cost algorithm, Maximal-density algorithm hopes to adapt to the changing user access pattern as well as reduce the traffic caused in updating the document distribution. To achieve this, Maximal-density algorithm aims to balance the load after distributing the replicas.

If the load is balanced after the document distribution, we expect to see that the weight of a server node is approximately proportional to its load capacity. In EGDWS, a server node's storage capacity is proportional to its load capacity; and the Density algorithm makes best use of the total size of available storage. Therefore, we expect the densities of the server nodes to be approximately the same if the load is balanced. Inspired by this, Maximal-density algorithm aims to equalize the densities of the server nodes to balance the load among the server nodes.

```
Step1.  Sort R_i (i = 1,…N) by decreasing density, d_i / c_i
Step2   for j = 1 to M {Do V_j = C_j }
Step3   for each R_i in the sorted list{
            3.1 Sort S_j (j = 1,…M) by increasing communication cost, p_ij.
                Servers having the same p_ij are sorted by decreasing density, G_j
            3.2 for j = 1 to M in the sorted list{
                 if (c_i > 0) && (V_j ≥ s_i) && (t_ij^l = 0, l = 1,…c_i) {
                // allocate a replica of D_i to S_j
                V_j = V_j – s_i, c_i = c_i –1
                l = c_i, t_ij^l = 1
                W_j = W_j + w_i / c_i
                G_j = W_j / (C_j – V_j) }}}
Step4.  Adjustment algorithm
```

Figure 6: The pseudo code of proposed Maximal-density Algorithm

The details of Maximal-density algorithm are shown in Figure 6. To equalize the server nodes' densities, it makes use of the densities of the replicas. First, the replica sets R_i $(i = 1, \ldots, N)$ are sorted by decreasing density and they are then allocated in this order. To reduce communication costs, when choosing server nodes for R_i, the server nodes are sorted by increasing communication cost p_{ij}. If two server nodes have the same cost, the one with the larger density G_j is chosen. As the densities of the replicas waiting to be assigned must be smaller than G_j $(j = 1, \ldots, M)$, such a choice decreases the difference between the densities of the two server nodes. Each time a replica is distributed to S_j, W_j and G_j are updated. The time complexity of Maximal-density algorithm is $\Theta(NlogN + NMlogM)$. For simplicity, we can use the sorting result of the Density algorithm, based on the assumption that for any two documents D_u and D_v, if $d_u > d_v$, $d_u/c_u > d_v/c_v$. Thus the algorithm only takes $\Theta(NMlogM)$ time.

We expect this algorithm to achieve better load balancing performance than the Greedy-cost algorithm.

4.3.3 Greedy-penalty Algorithm

The allocation sequence of the replica sets may affect the total traffic generated. For example, if we allocate D_i at time X, we can assign it to server x with smaller p_{ix}; if we delay allocating it for a while to time Y $(X < Y)$, however, server x may have become full and D_i has to be placed on server y with larger p_{iy}.

Greedy-penalty algorithm is proposed to further decrease communication costs by following a more careful allocation sequence of the replica sets. We call the resulting extra traffic caused by an improper allocation sequence "penalty". We use f_i $(i = 1, \ldots, N)$ to denote the value of penalty for D_i and try to minimize the sum of them. Similar algorithms have been used to solve the General Assignment Problem [23].

In Greedy-penalty algorithm, f_i is computed as the difference between the costs of R_i's best and second best allocations, according to the current status of the server nodes. A possible allocation for R_i is called "better" if it incurs less communication cost. When the server nodes with enough empty space are sorted in increasing p_{ij}, the first to the cith server nodes form the best allocation, while the second to the (c_i+1)th form the second-best allocation. Therefore, f_j is the difference between the communication costs of the first server node and the $(c_i + 1)$th server node in the sorted list. The algorithm iteratively places the replica sets until they are all allocated to some server nodes. Each time it computes the penalties for all unassigned replica sets, and the set yielding the largest penalty are placed with its best placement. If there are multiple replica

```
Step1.  Sort R_i (i = 1,…N) by decreasing density, d_i / c_i
Step2.  for j = 1 to M {V_j = C_j }
Step3.  While there are unassigned replica sets {
        3.1 for each unassigned replica set R_i{
              u = 0, X = Φ
              for j = 1 to M {
                        if V_j ≥ s_i, u = u + 1, add S_j to X }
              if (u ≤ c_i) {
                    for each S_j in X {
                          allocate a replica of D_i to S_j
                          V_j = V_j – s_i }
                    goto step 3 }
              else {
                    sort servers in X by increasing cost, p_ij. servers having the
                    same p_ij are sorted by decreasing density, G_j
                    f_i = cost of (c_i+1)th server node - cost of 1st server node }}
        3.2 Sort unassigned replica sets in decreasing penalty, f_i
              find the replica set R_max with largest f_i
        3.3 for each S_j in best placement for R_max {
                    if (c_i > 0) && (V_j ≥ s_i) && (t_ij^l = 0, l = 1,…c_i) {
                          //allocate a replica of D_i to S_j
                          V_j = V_j – s_i, c_i = c_i –1
                          l = c_i, t_ij^l = 1
                          W_j = W_j + w_i / c_i
                          G_j = W_j / (C_j – V_j )))
Step4.  Adjustment algorithm
```

Figure 7: The pseudo code of proposed Greedy-penalty Algorithm

sets with the same penalty value, they are placed in the order of decreasing densities.

The Greedy-penalty algorithm reduces the traffic by reducing penalties caused by distributing the replicas at wrong orders. However, using this algorithm, documents that have many replicas may have small penalties and thus be placed after unpopular documents and cause large load imbalance. To avoid such problems, when there are only or less than c_i server nodes having enough empty space for R_i, f_i is set to MAX_VALUE. Then the replica of D_i will be placed on every server with enough space, like shown in Step 3.1 of Figure 7. In this way, most popular documents are always placed earlier than other documents and the load is balanced to some extent.

The time complexity of this algorithm is $\Theta(NlogN + N^2logN + NMlogM)$. If we use the sorting results of the Density algorithm, the time complexity is reduced to $\Theta(N^2logN + NMlogM)$.

4.3.4 Proximity-Aware Algorithm

In the above algorithms, we assume that each document experiences the same popularity in all server locations. Now we assume that the server nodes are distributed in U different Internet regions, and documents may have different popularities in different Internet regions. We deploy a proximity-aware document distribution algorithm which distributes the replicas of a document according to its popularities at different Internet regions.

The set of Internet regions is represented by $E = \{E_1, \ldots, E_U\}$. We use r_{iu} to denote the access rate D_i

```
Step1.  Sort R_i (i = 1,…N) by decreasing density, d_i / c_i
Step2.  for j = 1 to M {V_j = C_j }
Step3.  for each R_i in the sorted list{
          3.1 sort E_u (u = 1,…U) by decreasing weight, w_iu
          3.2 for u = 1 to U in the sorted list{
                        copy_u = (w_iu / Σw_iu) * c_i
                        if (copy_u > M / U)
                              copy_u = M / U }
          3.3 u = 0 in the sorted list
              while (c_i – Σcopy_u > 0) {
                              if (copy_u < M / U)
                                    copy_u ++;
                              u = u + 1
                              if u = U, u = 1; }
          3.4 for u = 1 to U in the sorted list {
                sort servers in E_u by increasing communication cost, p_ij
                  for each server S_j in the sorted list {
                      if (c_i > 0) && (V_j ≥ s_i) && (t_ij^l = 0, l = 1,…c_i)
                          && (copy_u > 0){
                          // allocate a replica of D_i to S_j
                          V_j = V_j – s_i
                          c_i = c_i –1
                          l = c_i, t_ij^l = 1
                          copy_u =copy_u – 1 }}}}
Step4.  Adjustment algorithm
```

Figure 8: The pseudo code of proposed Proximity-aware Algorithm

experiences in region E_u, and use w_{iu} to denote the weight of D_i in region E_u.

The details of the algorithm are shown in Figure 8. We first sort the replica sets by their densities decreasingly. In Step 3, for each replica set in the sorted list, we compute its number of replicas $copy_u$ in each Internet region according to w_{iu} $(u = 1, \ldots, U)$. As there are only M/U server nodes in each region, $copy_u$ cannot exceed M/U. Thus, it's possible that the sum of $copy_u$ $(u = 1, \ldots, U)$ is less than c_i. If this is the case, in Step 3.3, the remaining replicas of D_i are distributed to the Internet regions, in the order of decreasing w_{iu}. In Step 3.4, the server nodes in the same region are sorted according to their communication cost with D_i, increasingly, and the replicas of D_i are distributed to them in this order. At the end of the algorithm, document adjust algorithm is invoked.

We can see that this algorithm places more replicas of D_i in E_u with large w_{iu}. However, it does not consider much about load balancing or communication costs. Its load balancing performance and traffic generated depend largely on the distribution of user accesses among the Internet regions.

4.3.5 Adjustment Algorithm

To fulfill the constraint (3), after each of the above document distribution algorithms, an adjustment algorithm is needed. It checks if any documents don't have a copy in the system, and deletes some other documents' copies to make space for these documents.

The adjustment algorithm first counts the number of copies in the system for each document and sorts the documents according to these numbers increasingly. Next, for each document that does not have any copies

in the system, a replica of another document is deleted to make space for it. We call the latter document "helping document", whose replica number must be larger than 1. In the potential helping documents, we start from the one whose replicas' density is lowest, aiming to reduce the affect on the load balancing in the system. Among the multiple server nodes that holding the helping documents, we choose the one with largest available space. This adjustment process iterates until the document has at least one copy in the system or there are no documents with more than one copy in the system.

4.4 Discussion

In this subsection, we first see if these document distribution algorithms can keep at least one copy for each document in the system. Next, we compare their load balancing performance and communication costs caused.

4.4.1 Correctness

We confine the correctness discussion of the algorithms to systems with $\sum_{i=1}^{N} s_i \leq \sum_{j=1}^{M} C_j/2$, $\sum_{i=1}^{N}(c_i \times s_i) \leq \sum_{j=1}^{M} C_j$, $s_i < C_{min}/2$ $(i = 1, \ldots, N)$, where C_{min} is the smallest capacity among C_j $(j = 1, \ldots, M)$. We prove that in such situations, the adjustment algorithms can place at least one copy of the documents in the system.

Suppose before the adjustment algorithm is invoked, D_a does not have a copy on any of the server nodes. The adjustment algorithm then deletes the copies of other documents with more than one copy in the system, and places D_a to the first server node with enough available space. The worst case is that each of the documents already in the system has only one copy left before we can place D_a. We know that at this time, D_a can definitely be assigned to one server node, because there must be one server node S_j whose available space is equal to or bigger than $C_j/2$. Otherwise, the total used storage in the system will be larger than $\sum_{i=1}^{N} s_i$ and this is contradictive to our assumption.

4.4.2 Communication Cost

We use MAX_TRAFFIC to denote the maximial traffic an algorithm may cause. In Greedy-cost algorithm, replicas are distributed in the order of increasing communication cost with server nodes. Thus,

$$MAX\_TRAFFIC_{greedy-cost} = \sum_{i=1}^{N}(c_i \times s_i - \sum_{j=1}^{M} p_{ij})$$

On the other hand, using Maximal-density algorithm, some replicas may not be placed on server nodes that have smaller communication costs with them. This is because the nodes may not have enough empty space to hold them. Such documents form the set Q. Thus,

$$MAX\_TRAFFIC_{maximal-density} = \sum_{i \in D/Q}(c_i \times s_i - \sum_{j=1}^{M} p_{ij}) + \sum_{i \in Q} c_i \times s_i \,,$$

$$Q = \{D_i \text{ that cannot be distributed to server nodes with minimal } p_{ij}\}$$

Similarly, the upper bound of communication cost of Greedy-penalty algorithm is:

$$MAX\_TRAFFIC_{greedy-penalty} = \sum_{i \in D/P} (c_i \times s_i - \sum_{j=1}^{M} p_{ij}) + \sum_{i \in P} c_i \times s_i \ ,$$

$$P = \{D_i \text{ that cannot be distributed to server nodes with minimal } p_{ij}\}$$

We can see that generally the communication costs of the Maximal-density algorithm and the Greedy-penalty algorithm are larger than that of the Greedy-cost algorithm. Also, as the Maximal-density algorithms distributes the replica sets in the order of decreasing densities, documents in set Q tend to have larger sizes than documents in set D/Q. On the other hand, as in most cases, penalty f_i is equal to s_i, we know that the Greedy-penalty algorithm tends to distribute the replicas with larger sizes before the replicas with smaller sizes. This means that documents in set P tend to have smaller sizes than documents in set D/P. Therefore, generally, Maximal-density algorithm generates more traffic than Greedy-penalty algorithm. The more varied are s_i $(i = 1, \ldots, N)$, the larger is the difference in communication costs of these algorithms.

The communication cost of the proximity-aware algorithm depends largely on the stability of the request distribution of the documents among the Internet regions. The more stable is the request distribution, the smaller is the cost.

4.4.3 Load Balancing

It's difficult to analyze the load balancing performance of the algorithms, therefore we only discuss the main factors influencing it. To do this, we use y_i to denote the copies of D_i in the system after document distribution. We know that the more similar are the densities of replicas, i.e., d_i/y_i $(i = 1, \ldots, N)$, the smaller is the load imbalance. In ideal case, all replicas are placed on the server nodes and $y_i = c_i$ $(i = 1, \ldots, N)$. c_i $(i = 1, \ldots, N)$ are decided by the Density algorithm and are the same for all document distribution algorithms. However, during placing the replicas, the adjustment algorithm may delete some replicas so that y_i does not equal to c_i. As Maximal-Density and Greedy-Cost place replicas in the order of increasing sizes, their load balancing may be affected more than Greedy-penalty by this process. Next, the distribution of replicas also affects the load balance. As Greedy-Cost places the replicas according to their communication cost with the server nodes, generally, its load balancing performance is not as good as Maximal-Density and Greedy-Cost, which monitors G_j $(j = 1, \ldots, M)$ in placing the replicas. Finally, the load balancing performance of the proximity-aware algorithm also depends on the request distribution of the replicas among the Internet regions. The more homogeneous is the request distribution, the smaller is the load imbalance in the system.

5 Experiment Results

In this section, we present our simulation results and the analysis of the performance.

5.1 Simulation Setup

We use the CSIM 18 package [12] to simulate a homogeneous EGDWS system, in which each server node has storage capacity C and load capacity L. The first request of a client always goes to the request dispatcher,

Table 1: Data Sets Statistics

	Data Set 1	Data Set 2	Data Set 3
Total Documents	9,354	26,915	131,142
Minimal Document Size (KB)	0.028	0.015	0.001
Maximal Document Size (KB)	2,854.8	5,597.6	179,803.2
Total size of Documents (KB)	250,868	437,544.9	7,464,779.2
Total Document Groups	380	774	206
Minimal Group Size (KB)	0.028	0.015	0.052
Maximal Group Size (KB)	15646.6	15497.3	671834.8
Duration	August 3-31, 1995	September 4-10, 1995	January 24-30, 2001
Extracted Number of Requests	531,012	707,058	285,171

which redirects the client to a server node according to its request dispatching algorithm. If the number of existing HTTP connections of the chosen server node has reached L, the request is dropped. Otherwise, the client sends its subsequent requests directly to the server node it is redirected. If the server node cannot find a requested document locally, it redirects the client to the dispatcher, which has all the information for request dispatching. The dispatcher will then choose another server node for the client. In our simulation, L = 3000 [17]. Initially, the website's documents are placed on the system randomly. Afterwards, every 3 hours the document distribution scheme is executed. The simulation lasts for one day. We simulate our document distribution scheme, using the algorithms presented in last section. The Density algorithm is combined with the four distribution algorithms respectively:

- **GC**: Density algorithm used together with Greedy-cost algorithm
- **MD**: Density algorithm used together with Maximal-density algorithm
- **GP**: Density algorithm used together with Greedy-penalty algorithm
- **Prox**: Density algorithm used together with Proximity-aware algorithm

For the purpose of comparison, we include another document distribution scheme into the simulation:

- **Dynamic scheme (DS)**. In this scheme, each server node owns a part of documents. Dynamically each server examines the bytes a server served in the past period, and determines if they are under-loaded or overloaded. A server is overloaded if the bytes it served are 50% more than the average bytes served, [1] while a server is under-loaded if the bytes it served are lower than the average bytes served. It then replicates one of its documents to the under-loaded node or revokes one replica of its documents from the overloaded node. The load of the servers is then recomputed correspondingly. This scheme is actually the one used in DC-Apache [20]. In the simulation, the servers check load status every 30 minutes.

In our simulation, if not stated, we are using the directory-based grouping method. During the simulation, the grouping of documents is not changed. The number of server nodes is 16.

[1] we also did simulation when "overloaded" is 10%, 20%, 30% and 40%, but the results were similar and thus were not shown here

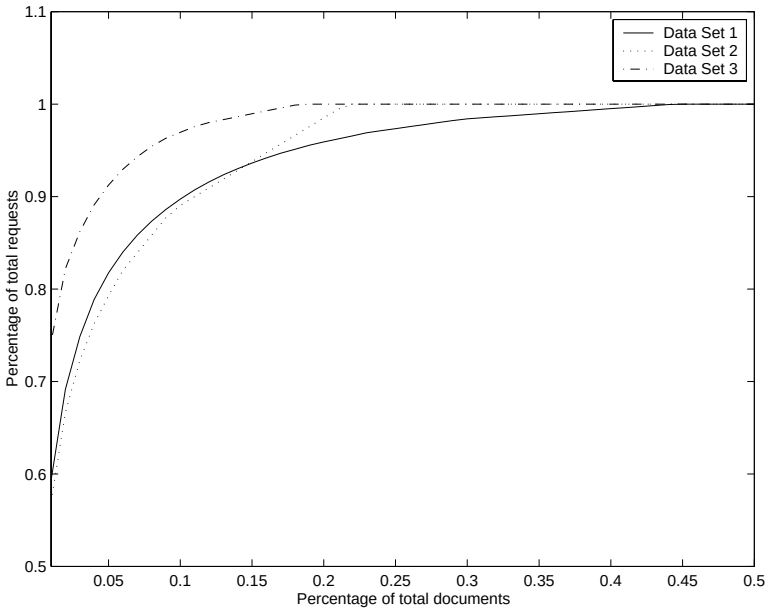


Figure 9: Reference Locality of Log Files

In the simulation, we use three real traces of web access, and they are called Data Set 1, Data Set 2 and Data Set 3 [2] respectively. The data sets are summarized in Table 1. We can see that the document sizes of them varied. Therefore, in the simulation, $w_i = s_i * r_i \ (i = 1, \ldots, N)$. Figure 9 reflects the reference locality in the web traces. The x-axis represents the percentage of total documents, and the y-axis represents the percentage of total requests. We can see that in Data set 1, a smaller part requests are responsible for most requests than in Data Set 2 and Data Set 3.

Before we present the simulation results, we need to explain some of the metrics we used.

1. *Load Balance Metric (LBM)*
   The Load Balance Metric (LBM) was proposed in [8]. It is used as a metric for measuring load-balancing performance. We use the utilization, that is, the percentage of busy time, of the server nodes to compute LBM. During the simulation, the utilization of the server nodes is measured every 10 minutes. Assume we have measured the utilization of e ach server node at X sampling points, the LBM is the weighted average of the peak-to-mean ratios captured at the sampling points. The sampling point with heavy load has a higher weight. The value of the LBM ranges from 1 to at most M, the number of server nodes in the system. A smaller LBM value indicates better load balancing performance.
2. *Average Traffic (AT)*
   Documentation redistribution affects the performance of EGDWS as it also consumes the web server resources. To compare the traffic caused by different algorithms in updating document distribution, we record the total number of bytes transferred inside the system each time the document distribution scheme is executed. We don't take the traffic in the first time into account, as the documents are distributed without duplication initially and it can be (and always is) done before the system begins working. We record the total traffic caused at each period and at the end of the simulation an aver-

[2] These traces were downloaded from http://ita.ee.lbl.gov/html/traces.html

age traffic is taken. The ratio between the average traffic and total size of web documents (without duplication) is computed. We use AT to denote this ratio in the following discussion.

5.2 Experiment 1 - MAN-based Experiment

In this experiment, we assume that all the server nodes are in one Internet region, that is, the distances between the clients all the server nodes are equal and the round-trip time of redirection is identical, which is set to 100 ms [30]. If the redirection is initiated by a server node instead of by the request dispatcher, the time of redirection equals to $2 \times 100 = 200$ ms, for the request will be redirected twice.

As the server nodes are of the same distance to the clients, we don't need to count the network latency outside the EGDWS into consideration in computing user response time. Therefore, in this experiment, processing a web request comprises (1) redirection (if necessary), (2) waiting in the queue of the serving server node, and (3) reading the document from disk. The disk reading parameters are derived from Seagate ST360020A technical specification [34], with the disk access time of about 8.9 ms and the disk transfer time about 21 MB/s. In this experiment, the requests are distributed to the server nodes holding the requested documents using round-robin strategy.

We first compare the speedup of average user-perceived response time achieved by the proposed algorithms. We fix the number of server nodes M = 16, and increases the storage capacity C of the servers. The results are shown in Figure 10, 12 and 14. The x-axis shows the degree of documentation replication, which is the ratio between C and the total size of the documents, $C/\sum_{i=1}^{N} s_i$. The y-axis is the speedup as compared with the response time measured when no documents are replicated. From the figures, we see that using our document distribution scheme, as the storage capacity increases, response time speedup is much larger than that of the dynamic scheme. Specifically, under our scheme, the response time performance of Data Set 1 improves faster than that of the other two data sets. As in Data Set 1 the smaller parts of documents are responsible for most of the requests, this is because our scheme can replicate popular documents efficiently and cut down redirections. For the same reason, the response time performance of Data Set 2 improves faster than that of Data Set 3. On the other hand, using the dynamic scheme, the response time speedup is similar for the three data sets.

We continue to examine the load balancing performance of our scheme. The Load Balance Metric (LBM) is used as the performance metric. Figure 11, 13 and 15 present the load balancing performance of our scheme when the number of servers is fixed as 16. The y-axis is LBM value, and the x-axis remains $C/\sum_{i=1}^{N} s_i$. When $C/\sum_{i=1}^{N} s_i$ = 1/16, no documents are replicated. Therefore, all four algorithms yield the same LBM value because no load balancing effect can be observed.

We see that DS doesn't improve load balancing much as the storage capacity increases. This is because in DS, each server node can only replicate one document once a time so that the available disk space is not utilized efficiently to remove hot spots.

On the contrary, the load balancing performance of our scheme increases as storage capacity increases because the Density Algorithm replicate as many as documents as the storage limit allows. One exceptive case is for Data Set 3, when $C/\sum_{i=1}^{N} s_i$ is larger than 1/2, LBM value increases instead of decreases. The reason is that in Data Set 3, some unpopular documents become popular between two distributions. When the storage capacity is relative small, these documents are scattered among the server nodes so that the influence on load balancing is unnoticeable. However, when the storage capacity increases to the extent that

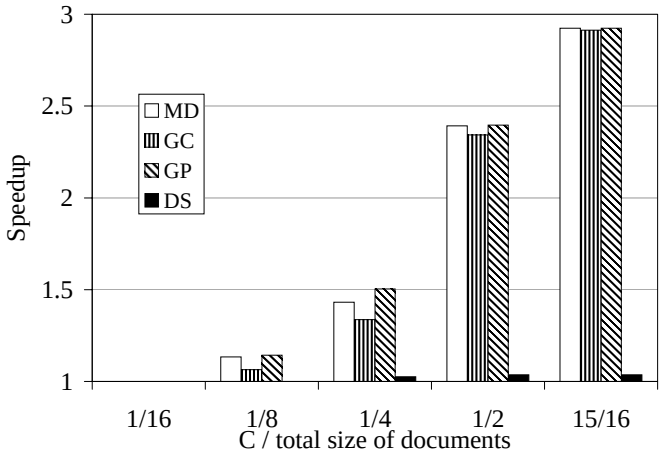


Figure 10: Response Time Speedup for Data Set 1

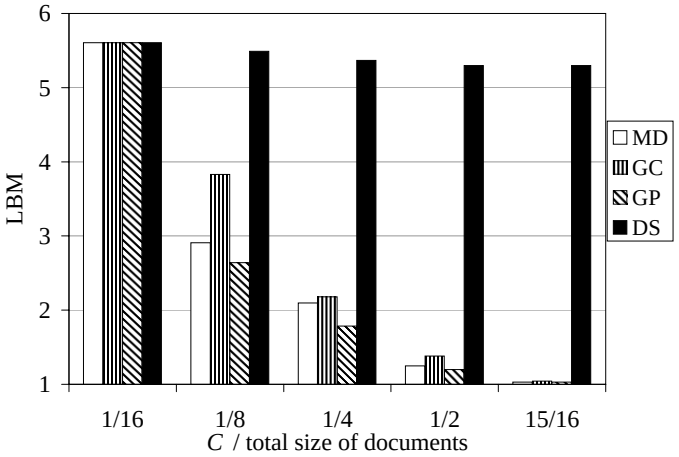


Figure 11: Load Balancing Effect for Data Set 1

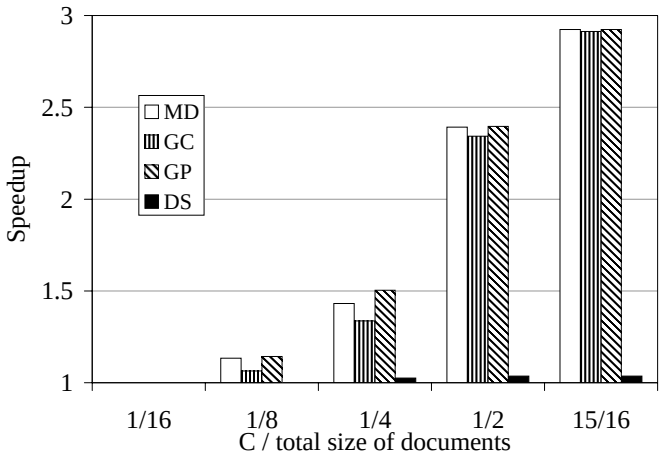


Figure 12: Response Time Speedup for Data Set 2

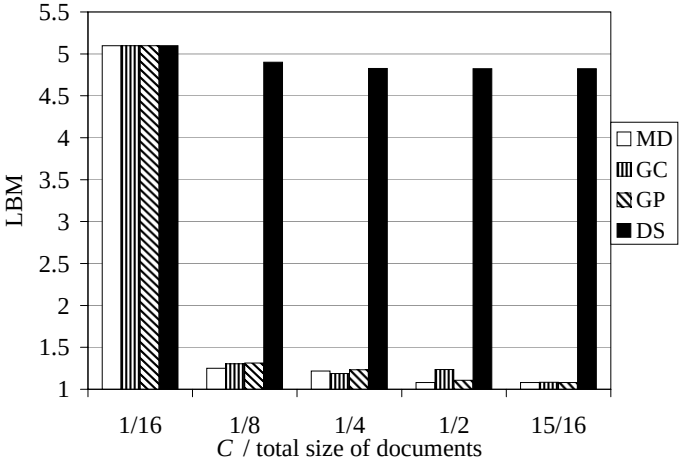


Figure 13: Load Balancing Effect for Data Set 2

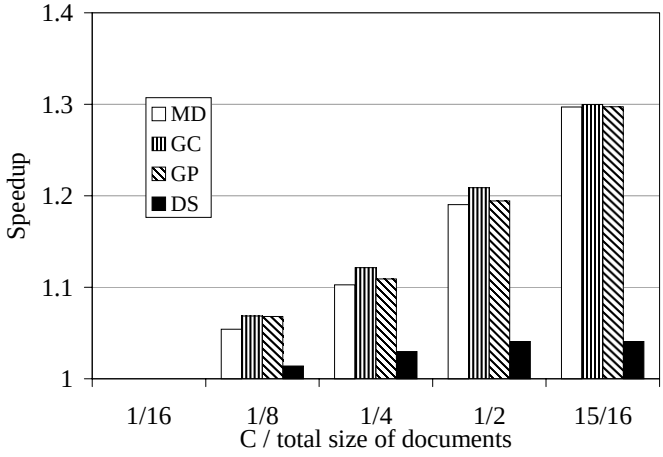


Figure 14: Response Time Speedup for Data Set 3

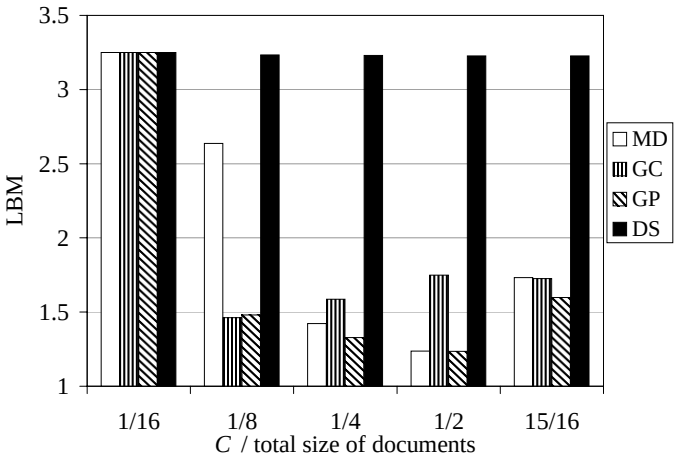


Figure 15: Load Balancing Effect for Data Set 3

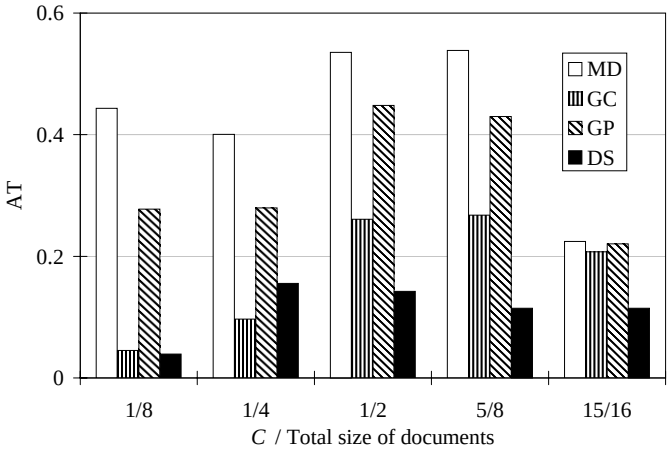


Figure 16: Average Traffic for Data Set 1

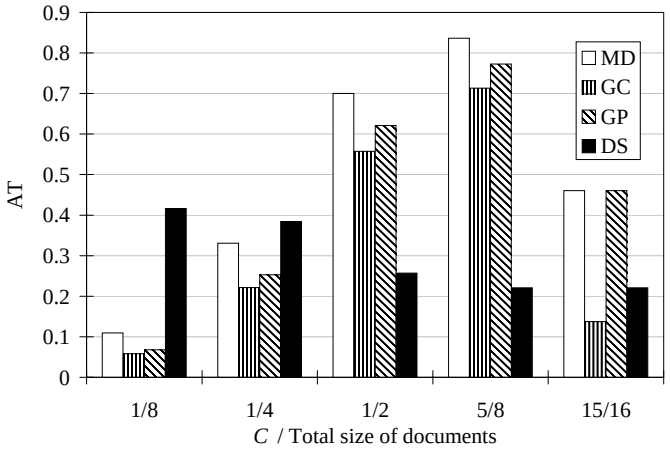


Figure 17: Average Traffic for Data Set 2

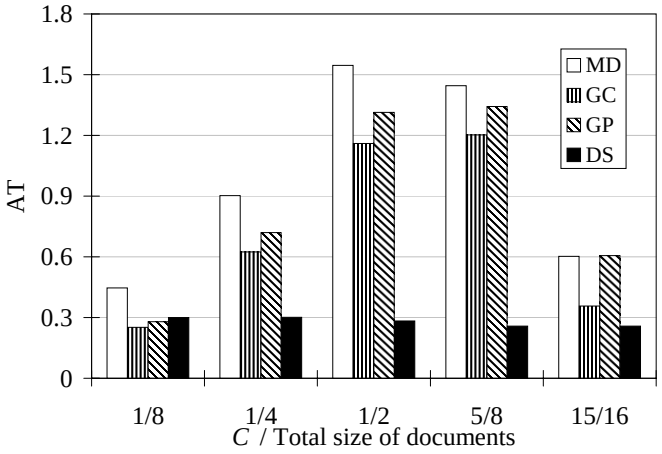


Figure 18: Average Traffic for Data Set 3

most documents, including some of these special documents, are replicated on each server node, the change of popularity of the remained such documents will have relative large influence on the load balance. From another point of view, this case shows that our scheme can work well even when the users' access pattern changes suddenly.

Among the document distribution algorithms, generally, GC performs worst in load balancing than MD and GP, especially when the storage capacity is large. This is because it does not care about the load of the server nodes when distributing the replicas. The difference is not very large because the distribution algorithms use the same replication algorithm. However, when the storage capacity is relative small, MD may perform worse than the other two algorithms. This is because it places the replicas in the order of decreasing densities. If some documents with large sizes are not placed on any of the server nodes, the adjustment algorithm will delete some other documents' replicas and may thus disturb the balance of load. For data sets that contain D_i with very large s_i, e.g. Data Set 3 in this experiment, such case is easier to happen. On the other hand, GP places replicas with larger sizes before replicas with small sizes; therefore, its load balancing results will be less influenced by the variation of s_i $(i = 1, \ldots, N)$.

Finally, we compare the traffic generated by the schemes in updating document distribution. We use Average Traffic (AT) as the metric, which is represented in y-axis in Figure 16, 17 and 18. We can see from the figures that generally when number of server nodes is fixed, the traffic caused by the algorithms first increases as the storage capacity C increases, and then decreases. This is because when there is more available disk space, more documents are replicated and the replica numbers of replicas of popular documents are larger. Once the access pattern changes, therefore, more replicas of the past period are revoked and more new replicas of this period need to be distributed. For example, the popularities of documents in Data Set 3 changes more than that of the documents in the other two data sets and sizes of D_i $(i = 1, \ldots, N)$ are more varied, thus more traffic is generated by our algorithms for this data set.

It is easy to understand that GC, which cares most about communication cost, incurs the least traffic. GP generates more traffic than GC and less traffic than MD. When the storage capacity is large, the difference between traffics caused by MD and GP decreases. This is because in this case, more documents have replica numbers equal to or close to M. In the figures, DS causes less average traffic. However, as its period is much shorter (1 / 6), the total traffic caused by it is actually larger than our scheme, especially when C is small. This is because of the frequent document replication and replica revocation.

From the above simulation results, we see that our document distribution scheme can provide better web service, achieve better load balancing and generate less traffic in a GDWS system than the dynamic scheme. Among the document distribution algorithms, GC's load balancing performance is not as good as that of GP and MD, and is easiest to be affected by initial placement of the documents. However, GC generates the least internal traffic. MD needs shortest computing time. Its load balancing performance is best in most cases and only generates a little more traffic than GP. However, when s_i $(i = 1, \ldots, N)$, are very varied, its load balancing performance may deteriorate when the storage capacity is small. Comparatively, GP yields most stable load balancing performance, but it requires more computation than the other algorithm. A suitable algorithm can be chosen according to the practical situation of a geographically DWS system.

We see that all the three algorithms can achieve performance similar to fully-replication with only half of the total disk space. On the other hand, when the total disk space equals to or exceeds half the total size of documents, our scheme of updating document distribution periodically seems not effective enough. Because in this case, the load balancing performance (LBM) improves little while the internal traffic (AT) increases

Table 2: Number of Redirections under the Two Grouping Approaches

		C / Total size of documents				
		1/16	1/8	1/4	1/2	15/16
Data Set 1	dir-based	494308	426185	325007	169617	127649
	acc-based	141872	158352	35584	25078	19420
Data Set 2	dir-based	243365	135202	115830	108348	104354
	acc-based	184631	151343	131881	105207	88491

much. Therefore, we conclude that our document distribution scheme is most suitable for GDWS systems with small storage capacity.

Comparison of Document Grouping Algorithms

To cut down redirections and reduce the load of the request dispatcher, in our system, documents are grouped. We next compare the two grouping approaches: directory-based grouping (dir-based) and access pattern based grouping (acc-based). In the first approach, documents in the same second-level directory are grouped together. For example, /A/C/1.htm and /A/C/2.htm belong to the same group, while /A/B/3.htm and /A/B/4.htm are of another group. In the second approach, the algorithm described in section 3.3 is used. The previous day's log files is used for grouping. As Data Set 3 is from a website mainly hosting personal homepages and a client requests are always for documents within one single homepage, we don't include it in this comparison. Table 2 lists the number of redirections under the two grouping approaches as the server nodes' storage capacity increases. The MD algorithm is used.

From Table 2, we can see that when no documents are replicated, the number of redirections of acc-based is much less than that of dir-based. This shows that acc-based can group highly correlated documents together and thus reduce redirections. However, for Data Set 2, when the server node storage capacity increases, the number of redirections of dir-based may become less than that of acc-based grouping. This is probably because requests for Data Set 2 are mainly for documents in a few directories. When they are replicated on most of server nodes, the redirections are cut down greatly. On the contrary, the acc-based grouping may group the documents of these directories into different groups, due to the size limitation of the grouping algorithm. If some of these groups have smaller densities, they are only replicated on several server nodes and redirections are needed. Therefore, the acc-based algorithm is most suitable for websites whose popular documents are widely spread among the directories. Otherwise, directory-based grouping combined with replication can also cut down redirections efficiently.

Extensibility Analysis

At the end of this experiment, we examine how our system can extend without suffering great disturbance. We use load balance as our metric to show how much the system is affected. In this extensibility experiment, there are initially 16 server nodes in the system. At the second distribution, we measure the peak-to-mean ratio of server nodes' utilization U_R as a reference, and then add one new server node with the same storage capacity and load capacity Afterwards, we measure the peak-to-mean ratio of server nodes' utilization U every 10 minutes to see how the load is balanced in the system. The result is illustrated in the Figure 19. The

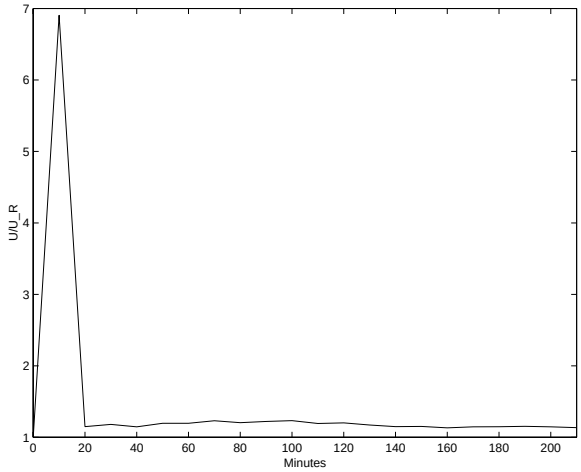


Figure 19: Extensibility Performance

Table 3: Bandwidth Types

Bandwidth Type	Bandwidth (B)	Round Trip Time (rrt)
Large	1.35 Mbps	[40, 70]ms
Medium-large	0.9 Mbps	[120, 150]ms
Medium-narrow	0.7 Mbps	[180, 210]ms
Narrow	0.4 Mbps	[270, 300]ms

x-axis is the minutes passed after the addition of server node, and the y-axis is U/U_R. From the figure, we can see that the load is re-balanced among the server nodes very quickly. Adding a new server node won't disturb the load balance the system is enjoying, and thus won't affect the system's performance greatly.

5.3 Experiment 2 - WAN-based Experiment

In this experiment, we assume that the Internet is divided into U = 4 regions located in different world areas. Each region contains M/U server nodes. The bandwidth within a region and between this region and others is supposed to be large, medium-large, medium-narrow and narrow. The bandwidth values reported in Table 3 are taken in [35], and we assume that the network bandwidth does not change during the experiment.

In this experiment, the client requests come from different Internet regions. If the request originates from E_i and is redirected to E_j by the requests dispatcher, the redirection time equals to rrt_{ij}, the round trip time

Table 4: Bandwidth Types Between Internet Regions

	1	2	3	4
1	Large	Large	Medium-large	Medium-narrow
2	Large	Large	Medium-narrow	Narrow
3	Medium-large	Medium-narrow	Large	Medium-narrow
4	Medium-narrow	Narrow	Medium-narrow	Large

between E_i and E_j. If the redirection is initiated by a server node in E_l, the redirection time is $rrt_{li} + rrt_{ij}$. In addition to the redirection time, waiting time and disk reading time, the response time also contains the time needed to transfer the documents on the Internet. Suppose the user is in E_i, the server node is in E_j, and the size of the document is W, the communication time over the Internet is W/B_{ij}.

In this experiment, we simulate three scenarios:

1. MD-RR: Maximal-density algorithm used with round-robin request dispatching strategy.
2. MD-Prox: Maximal-density algorithm used with proximity-aware request dispatching strategy, in which the request is assigned to the server node that has the requested document, in the Internet region nearest to the client, and currently holds minimal HTTP connections.
3. Prox-Prox: Proximity-aware algorithm used with proximity-aware request dispatching strategy.

Same as in Experiment 1, the number of server nodes M is fixed at 16, and the server nodes have the same storage capacity C. For the purpose of better comparison, in the following discussion the results of MD-RR are used as references, and the results of MD-Prox and Prox-Prox are presented in the form of relative values.

We first present the response time results of the experiment. In Figure 20, 21 and 22, the x-axis is $C/\sum_{i=1}^{N} s_i$, and the y-axis is relative response time = response time / MD-RR's response time. From the figures, we can see that the user response time under MD-Prox and Prox-Prox is shorter than that of MD-RR. This is because the proximity-aware document dispatching algorithm can redirect clients' requests to the server nodes in the nearest Internet region to the clients and thus reduce network latency. As Prox-Prox tries to place replicas near where they are most needed, it further shortens the response time, especially when the storage capacity is small. As C increases, there are more replicas in the system and more documents are placed on all server nodes. Therefore, the gap between MD-RR's response time and those of MD-Prox and Prox-Prox decreases. That's why the curves in the figures become flat and even go up when C becomes large.

Next we compare the load balancing performance of the three scenarios. In Figure 23, 25 and 27, the y-axis is relative LBM value = LBM / MD-RR's LBM. From the figures, we can see that MD-Prox and Prox-Prox yield worse load balancing than MD-RR. For MD-Prox, this is because the requests are not assigned equally to server nodes holding its replicas, but assigned to server nodes in a certain Internet region, based on where the requests originate. The worse load balancing performance of MD-Prox implies the documents experience different popularities in different Internet regions. By placing more replicas of a document in the Internet regions where it is more popular, Prox-Prox cannot improve, but may worsen load balancing. However, as it also uses the Density algorithm to determine replica numbers, its load balancing performance is generally close to that of MD-Prox, which is illustrated in the figures.

In both scenarios, when the storage capacity is large, the relative LBM value reaches the maximal value. There are two reasons for this. The first is the LBM value of MD-RR generally decreases as the storage capacity increases. The other reason is that when most documents have copies on almost all server nodes, redirecting requests to nearest server nodes causes more obvious load imbalance.

We then examine the average traffic generated in the three scenarios. In Figure 24, 26 and 28, the y-axis is relative AT = AT / MD-RR's AT. We see that MD-Prox generates the same amount of traffic as MD-RR. This is because they use the same document distribution scheme and only differ in requests dispatching algorithm. Actually MD-Prox's traffic is mainly caused by the change of documents' replica numbers. For example, when unpopular documents in last period become popular in this period, their replicas need to be copied on more server nodes. On the other hand, Prox-Prox's traffic is also caused by the change of documents' request distributions. Even if a document's overall popularity remain the same, if its popularities in different Internet

regions change, its replicas still need to be moved among the server nodes in different regions. Therefore, generally, Prox-Prox generates more traffic than MD-Prox. When the storage capacity increases, there are more replicas in the system. The change of requests distribution causes less traffic and thus the internal traffic caused by Prox-Prox becomes closer to the traffic caused by MD-Prox. However, for Data Set 1, such traffic first increases as the storage capacity increases. This is because while its documents' popularities remain stable, their request distributions are very unstable. Thus, the more replicas the documents have, the more replicas need to be moved when the request distribution changes. When C continues to increase so that most frequently accessed documents have replicas on all the server nodes, the traffic caused by change of request distribution becomes very small and the traffic decreases in the figure.

From the results of experiment 2, we see that in a WAN-based EGDWS, redirecting requests to the nearest server nodes or distribute the documents according to their popularities in different regions can effectively reduce the response time. However, they also disturb the load balancing in the system especially when the storage capacity is large. Moreover, using the proximity-aware document distribution algorithm, the performance improvement may probably be counteracted by the additional internal traffic it generates. Therefore, we know that in a WAN-based system, a proximity-aware request dispatching algorithm should be used carefully to avoid overloading certain server nodes. On the other hand, a proximity-aware document distribution algorithm is not necessary.

6 Conclusion and Future Work

In this paper, we describe the system architecture of Extensible Geographically Web Server system, and propose a document distribution scheme to support the efficient operation of the system. In EGDWS, documents are replicated and distributed to the server nodes periodically. Our document distribution scheme periodically collects the access records of the server nodes, computes the replica numbers for each document based on its popularity in the last period, and then distributes the replicas to the server nodes. It consists of several constituent algorithms that rely on information regarding the current document locations and the documents' popularities from different perspectives. Their overall aim is to produce a balanced load and to minimize on the traffic generated from document distribution or re-distribution. For EGDWS systems that have a large geographic span, a proximity-aware document distribution algorithm is proposed to distribute replicas of documents to where they are most wanted. Different document grouping algorithms are applied, which can reduce the number of redirections experienced by the user.

Simulation results show that our document distribution scheme could provide better web service, achieve better load balancing, and generate less traffic than the existing dynamic schemes. We also found that our scheme can achieve performance comparable to full-replication schemes but using only half of the total disk space. Therefore, in a MAN environment with modest storage capacity, our scheme is most suitable. For WAN-based systems, our simulation results show that using the proximity-aware document distribution algorithm, the response time can be much reduced, but the load could become less balanced, and there is more internal traffic due to transfers of replicas across long distances. We found that, however, when using the other document distribution algorithms we proposed with a proximity-aware request dispatching algorithm, we can achieve similar response time speedup, but generating much less traffic.

In the future, we hope to design an on-line algorithm that can dynamically replicate the documents in response to the most current user access patterns. We hope such an algorithm can achieve similar or bet-

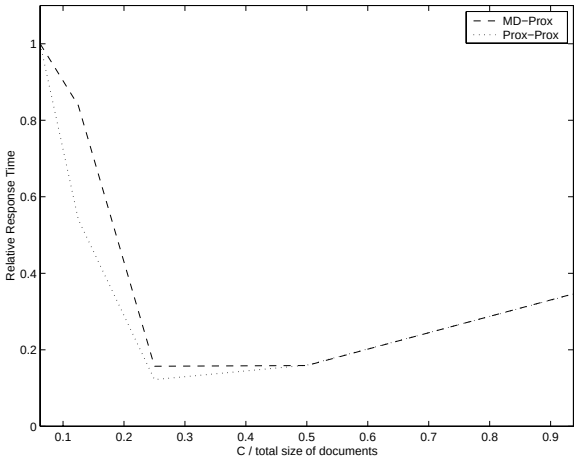


Figure 20: Relative Response Time for Data Set 1

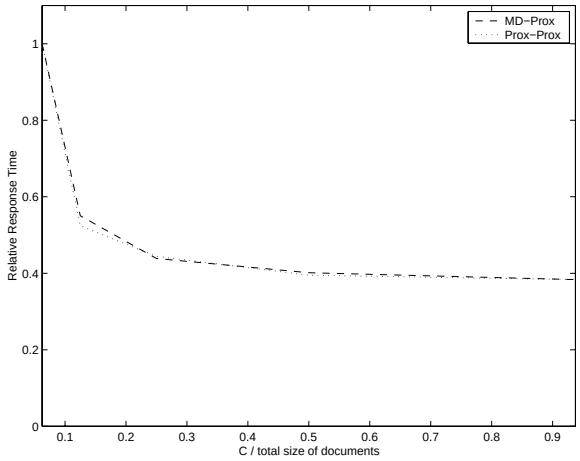


Figure 21: Relative Response Time for Data Set 2

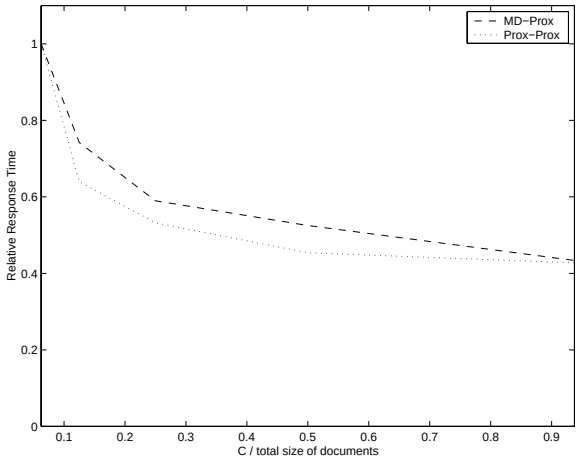


Figure 22: Relative Response Time for Data Set 3

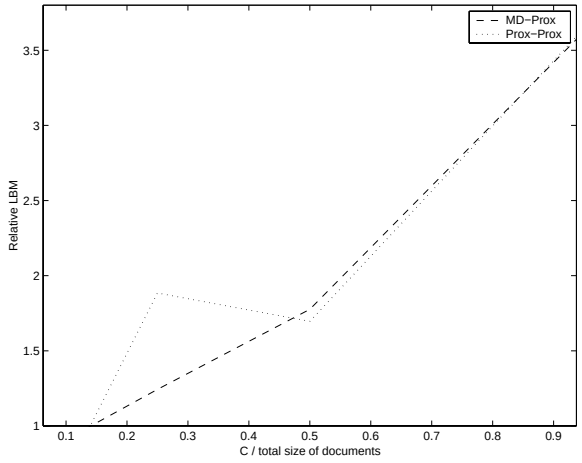


Figure 23: Relative Load Balancing for Data Set 1

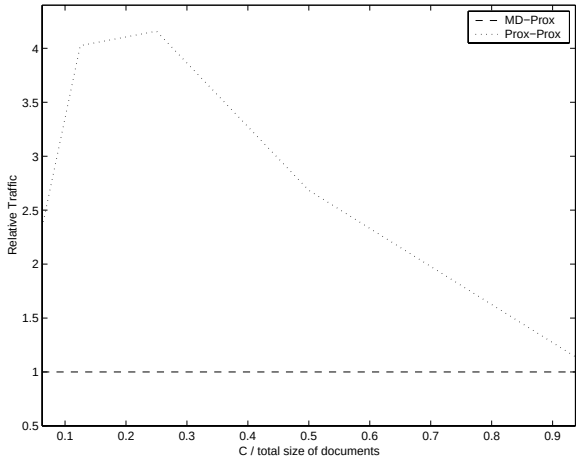


Figure 24: Relative Average Traffic for Data Set 2

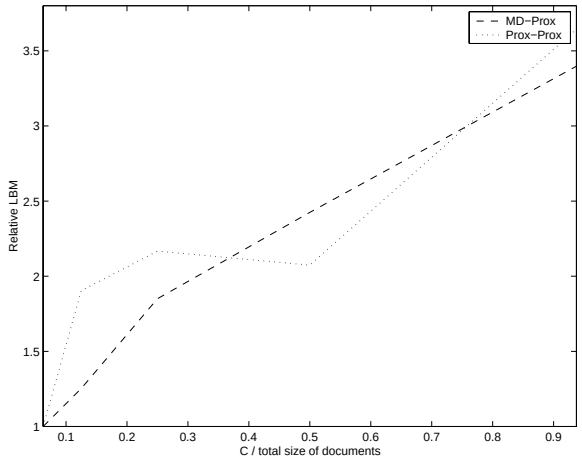


Figure 25: Relative Load Balancing for Data Set 2

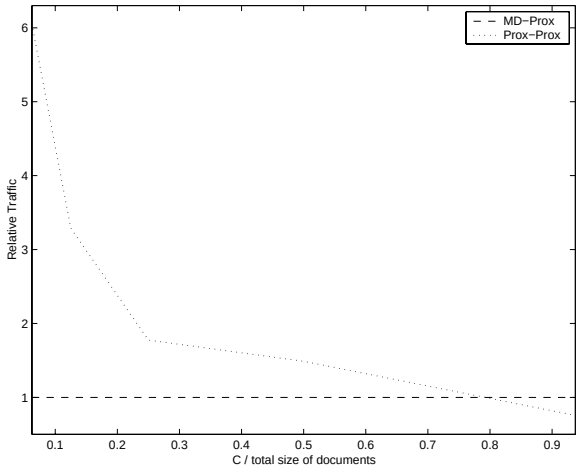


Figure 26: Relative Average Traffic for Data Set 2

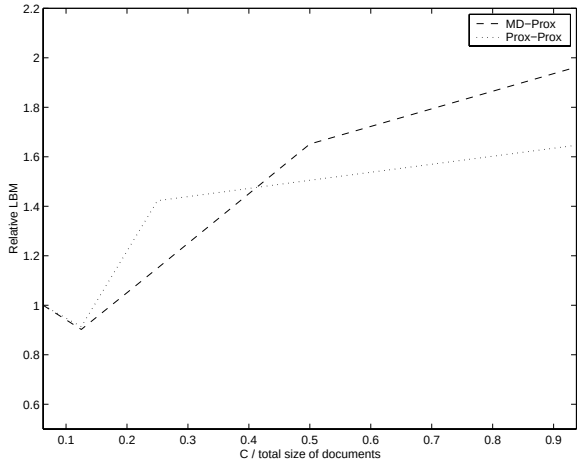


Figure 27: Relative Load Balancing for Data Set 3

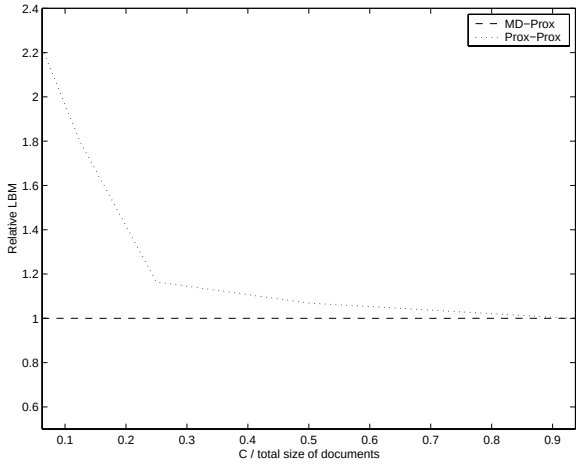


Figure 28: Relative Average Traffic for Data Set 3

ter load balancing performance, and further reduce the internal traffic. This current research is limited to homogeneous systems and heterogeneous systems whose servers' storage capacity is proportional to their load capacity. New document distribution algorithms that can be used in more complicated heterogeneous systems is worth purusing in the future.

References

[1] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox. Caching Proxies: Limitations and Potentials. In *Proc. of 4th International World Wide Web Conference*, pages 119–133, Boston, USA, December 1995.

[2] P.M.G. Apers. Data Allocation in Distributed Database Systems. *ACM Transactions on Database Systems*, 13(3):263–304, September 1998.

[3] M.F. Arlitt and C.L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proc. of the 1996 SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, pages 160–169, Philadelphia, USA, May 1996.

[4] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proc. of 25th Annual ACM Symposium on Theory of Computing*, pages 164–173, Victoria, B.C., Canada, May 1993.

[5] S.M. Baker and B. Moon. Scalable web server design for distributed data management. In *Proc. of 15th International Conference on Data Engineering*, page 96, Sydney, Australia, March 1999.

[6] G. Barish and K. Obraczka. World Wide Web Caching: Trends and Techniques. *IEEE Communications Magazine Internet Technology Series*, 38(5):178–185, May 2000.

[7] C. Bisdikian and B. Patel. Cost-based Program Allocation for Distributed Multimedia-on-demand Systems. *IEEE Multimedia*, 3(3):62–76, Fall 1996.

[8] R.B. Bung, D.L. Eager, G.M. Oster, and C.L. Williamson. Achieving Load Balance and Effective Caching in Clustered Web Servers. In *Proc. of 4th International Web Caching Workshop*, pages 159–169, San Diego, USA, March 1999.

[9] V. Cardellini, M. Colajanni, and P.S. Yu. Geographic Load Balancing for Scalable Distributed Web Systems. In *Proc. of IEEE MASCOTS 2002*, pages 20–27, San Francisco, USA, August 2000.

[10] A. Chankhunthod, M. Schwartz, P. Danzig, K. Worrell, and C. Neerdaels. A Hierarchical Internet Object Cache. In *Proc. of the 1996 USENIX Technical Conference*, pages 153–164, San Diego, USA, January 1996.

[11] A. Cohen, S. Rangarajan, and H. Slye. On the Performance of TCP splicing for URL-aware Redirection. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, pages 117–125, Colorado, USA, October 1999.

[12] CSIM18. http://www.mesquite.com/htmls/csim18.htm.

[13] L.W. Dowdy and D.V. Foster. Comparative Models of the File Assignment Problem. *ACM Computing Surveys*, 14(2):287–313, June 1982.

[14] S. Gadde, J. Chase, and M. Rabinovich. A taste of crispy squid. In *Proc. of Workshop on Internet Server Performance (WISP'98)*, Madison, USA, June 1998.

[15] J. Kangasharju, J. Roberts, and K.W. Ross. Object Replication Strategies in Content Distribution Networks. In *Proc. of Web Caching and Content Distribution Workshop (WCW'01)*, Boston, USA, June 2001.

[16] M. Karlsson, C. Karamanolis, and M. Mahalingam. A Framework for Evaluating Replica Placement Algorithm. Technical report, HP Lab, 2002.

[17] kHTTPd Linux HTTP Accelerator. http://www.fenrus.demon.nl.

[18] T.T. Kwan, R.E. McGrath, and D.A. Reed. NCSA's World Wide Web Server: Design and Performance. *IEEE Computer*, 28(11):68–74, November 1995.

[19] Y.K. Kwok, K. Karlapalem, I. Ahmed, and N.M. Pun. Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems. *IEEE Journal on Selected Areas In Communications*, 14(7):1332–1348, November 1996.

[20] Q.Z. Li and B. Moon. Distributed Cooperative Apache Web Server. In *Proc. of 10th International World Wide Web Conference*, Hong Kong, May 2001.

[21] T. Loukopoulos and I. Ahmad. Static and Adaptive Data Replication Algorithms for Fast Information Access in Large Distributed System. In *Proc. of 20th IEEE International Conference on Distributed Computing Systems*, Taipei, Taiwan, 2000.

[22] O. Etzioni M. Perkowitz. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Proc. of the 15th National Conference on Artificial Intelligence*, pages 727–733, Madison, USA, July 1998.

[23] S. Martello and P. Toth. *Knapsack Problems: algorithms and computer implementation*. John Wiley & Sons Ltd, 1990.

[24] S. Michel. Adaptive Web Caching: Towards a New Caching Architecture. In *Proc. of 3rd International Caching Workshop*, pages 2041–2046, Singapore, June 1998.

[25] P. Mirchandani and R. Francis. *Discrete Location Theory*. John Wiley and Sons, 1990.

[26] B. Narendran, S. Rangarajan, and S. Yajnjk. Data Distribution Algorithms for Load Balanced Fault Tolerant Web Access. In *Proc. of 16th Symposium on IEEE Reliable Distributed Systems*, pages 97–106, Durham, USA, October 1997.

[27] K. Obraczka and F. Silva. Network Latency Metrics for Server Proximity. In *Proc. of IEEE Globecom 2000*, San Francisco, USA, November 2000.

[28] IBM Olympics. http://www.ibm.com/news/2000/09/193.phtml.

[29] V.N. Padmanabhan and L. Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *Proc. of ACM SIGCOMM 2000*, pages 111–123, Stockholm, Sweden, August 2000.

[30] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proc. of 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, USA, October 1998.

[31] L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the Placement of Web Server Replicas. In *Proc. of 20th IEEE INFOCOM*, pages 1587–1596, Anchorage, USA, April 2001.

[32] P. Radoslavov, R. Govindan, and D. Estrin. Topology-Informed Internet Replica Placement. In *Proc. of 6th International Workshop on Web Caching and Content Distribution*, Boston, USA, June 2001.

[33] P. Rodriguez, C. Spanner, and E. W. Biersack. Web Caching Architectures: Hierarchical and Distributed Caching. In *Proc. of 4th International Caching Workshop*, San Diego, USA, April 2001.

[34] Seagate ST360020A. http://www.seagate.com/support/disc/specs/ata/st360020a.html.

[35] K. Thompson, G.J. Miller, and R. Wilder. Wide-areas Internet Traffic Patterns and Characteristics. *IEEE Network*, 6(11):10–23, November 1997.

[36] V. Valloppillil and K.W. Ross. Cache Array Routing Protocol v1.1. Internet Draft, February 1998, http://www.globecom.net/ietf/draft/draft-vinod-carp-v1-03.html.

[37] J. Wang. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.

[38] D. Wessels and K. Claffy. Internet Cache Protocol (ICP), RFC2186. http://icp.ircache.net/rfc2186.txt (July 26, 2002), September 1997.

[39] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithms. *ACM Transactions on Database Systems*, 22(4):255–314, June 1997.

[40] C.S. Yang and M.Y. Luo. An Effective Mechanism for Supporting Content-based Routing in Scalable Web Server Clusters. In *Proc. of the International Workshop on Internet in conjunction with the 28th International Conference on Parallel Processing (ICPP'99)*, Aizu, Japan, September 1999.

[41] L. Zhuo, C.L. Wang, and F.C.M. Lau. Load Balancing in Distributed Web Server Systems with Partial Document Replication. In *Proc. of the 2002 International Conference on Parallel Processing*, pages 305–312, Vancouver, Canada, August 2002.