Document Replication and Distribution in Extensible Geographically Distributed Web Servers ⋆

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.edu

Preprint submitted to Elsevier Science *10 May 2003*

Abstract

A geographically distributed web server (GDWS) system consists of multiple server nodes interconnected by a metropolitan area network (MAN) or a wide area network (WAN). It can achieve better efficiency in handling ever-increasing web requests than centralized web servers because its 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 architecture. We adopted a partial duplication scheme where document replication targets only at hot objects in a website. 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 the Greedy-penalty algorithm yields most stable load balancing performance, and the Greedy-cost algorithm causes least internal traffic. Our scheme can achieve 80% of the performance of full-replication, with half the disk space.

Key words: Document replication and distribution, Distributed web servers

⋆ This work was partly supported by HKU Seed Funding for Basic Research under code number 10204215.

1 Introduction

The rapid growth of the World Wide Web has brought huge increases in traffic to popular websites. One prominent example is the website for the 2000 Olympic Games (www.olympics.com), which had to handle an average of 7900 requests per second [22]. Web users of such popular websites would encounter slow responses if the servers behind them are not powerful enough. Techniques such as Web caching and prefetching have been used to mitigate this problem by placing the web documents closer to the users. However, previous research has shown that the maximum cache hit rate achievable by any caching algorithm is bounded [1]. The demand for more powerful web servers that can efficiently handle large amount of HTTP requests persists.

To construct a powerful web server, one could either use a single, powerful machine, or a collection of machines working together as a distributed system. The single-machine approach is not attractive due to the ever-increasing user demand and the fast obsolescence of today's hardware. The cost to maintain a powerful server machine could also be a problem. The other approach is more flexible and sustainable. It uses a *Distributed Web Server* (DWS) system, in which multiple server nodes are interconnected to act as one single server. We classify a DWS system as either "locally" or "geographically" distributed, based on how its server nodes are interconnected. A locally DWS (LDWS) system takes the form of a computing cluster which is normally based on a LAN. Its performance is thus limited by the bandwidth of the system's connection to outside network. A geographically DWS (GDWS) system uses public networks (MAN or WAN) for its interconnections, and may span over multiple networks. Previous research has shown that GDWS systems is able to handle large amounts of user requests better than traditional web servers and LDWS systems [8].

As the network speed of a MAN or WAN is limited when comparing to a LAN-based network, in GDWS systems, movement of documents between server nodes is an expensive operation. Therefore, a strategy that can optimally and efficiently distribute documents and replicate selected documents is needed in the design of a GDWS system. Such a strategy is the subject 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 maintains copies of all the documents of the site [14]. Incoming requests are then passed on to the server nodes using content-blind mechanisms (e.g. via a DNS server). Full-replication is most fault-tolerant, which however could waste much disk space because 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 direct a client request to the server node that has the requested document. 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 website has too many popular web pages with extremely high request rates, to equalize the load is absolutely non-trivial.

In a partial-replication GDWS system, only a subset of the web documents are replicated [16,30]. Partial-replication aims at using a reasonable amount of disk space to achieve a good replication which in turn helps remove possible hot spots. And fewer hot spots automatically translates to a more even load across the system. We believe that partial-replication GDWS system is the most promising solution to providing a powerful web server.

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—collectively known as the *document distribution scheme*.

Existing document distribution schemes for partial-replication GDWS systems fall into two main categories. The first is based on a dynamic approach, where *dynamically*, each server node checks the current global load situation; the server then replicates one of its documents to the underloaded servers and revokes one of its replicas from the overloaded servers. This approach is used in DC-Apache [16]. As we will see in later sections, the performance of this approach is not always satisfactory. The other approach is to replicate and distribute the documents based on past access patterns [20]. However, how to update the document distribution based on up-to-the-minute access patterns as well as current document location information is rarely discussed in existing literature.

In this research, we propose an architecture for GDWS systems, which is called the Extensible Geographically Distributed Web Server (EGDWS). With this architecture, documents are periodically replicated according to their popularities during the past period. The replicated documents are then distributed among the server nodes taking into account the current locations of the documents. Thus internal traffic generated can be reduced. The system is extensible in that after adding a new server node, the system can resume its load-balance promptly, and the existing server nodes minimally affected by the addition. We propose several algorithms for the operation of an EGDWS system. The replication algorithm makes use of a "density" attribute (the workload each unit of a document brings to a server) of a document to calculate the desired number of replica. We try and compare several different distribution algorithms: the Greedy-cost algorithm minimizes 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 a certain sequence in order to reduce the traffic; the Proximity-aware algorithm distributes the documents according to their popularities in different Internet regions. Results we obtained show that our scheme can achieve 80% of the performance of a full-replication scheme, but with half the disk space.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 presents details of the architecture of our EGDWS, and the formulation of the data replication problem. Section 4 proposes and discusses the document 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 a substantial amount of work done on the subject of data replication. The general problem refers to how to decide on the number and placement of replicas across multiple servers, with the aim to improve the overall system performance and minimize the costs involved. Previous work on data replication has focused on networked systems, distributed database systems, and most recently, content delivery networks (CDNs).

The File Allocation Problem (FAP) [10] studies how to assign files to different nodes in a distributed environment to optimize a certain performance metric (e.g., total network traffic). The files and their copies are allocated with respect to the storage capacity of the nodes. 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. This problem has been proved to be NP-complete. An old but useful survey on models for FAP in computer network systems can be found in [10]. Most previous results were based on the assumption that access patterns are known and remain unchanged. Some solutions for dynamic environments were proposed in [17,4,29].

The problem of data replication also exists in distributed database systems. It can be modeled by an FAP-like formulation [2]. The author of [2] investigated the complexity of the data replication problem in database systems, and proposed several methods for obtaining optimal as well as heuristic solutions taking into account the data allocation cost. The work of [15] and [6] studied the data allocation problem in multimedia database systems and video server systems respectively. Many proposed algorithms in this area try to reduce the volume of data transferred in processing a given set of queries [2,15].

Another important data replication problem exists in Content Delivery Networks (CDNs). In a CDN, the set of documents in a website is replicated to servers located at widely separated locations around world or in some large geographic region. In [25], the replica placement problem in CDNs is formulated as the uncapacitated minimum K-median problem [19]: 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 [26] and [25], different heuristics were proposed based on this K-median formulation to reduce network bandwidth consumption. The authors of [11] take storage constraints into consideration, and reduce the knapsack problem to the replica placement problem in CDNs.

The document distribution problem of GDWS systems can be seen as an instance of the FAP, in which each document has multiple copies and storage capacities are considered. It is also similar to the replica placement problem in CDNs. However, the existing solutions cannot be directly applied to GDWS systems, for the following reasons.

(1) **Load constraints and load balance**

Each server node of a GDWS system has a computational load capacity in order to maintain certain service quality. This constraint needs to be taken into consideration when allocating the documents. Not many of the existing solutions to the FAP consider load capacities [12]. On the other hand, current research on CDNs often assumes that the load capacity of the server is unlimited. Thus, load balancing among the server nodes is not considered an issue [12]. This assumption may be reasonable for CDNs for they can afford to deploy powerful servers. But for GDWS systems that are made of ordinary server machines, an unbalanced workload could be a hindrance to satisfactory performance.

(2) **Internal traffic**

In existing solutions for data replication problems, how to update the data allocation among the nodes based on changes of user access patterns is seldom investigated. Furthermore, internal traffic caused by such updates was rarely discussed in solutions to the FAP. As most CDNs employ their own private high-speed networks [25], such an issue of traffic caused by data replication is not of concern. But since GDWS systems are assumed to be constructed over public networks, the induced traffic may have a significant influence on the performance.

3 EGDWS

In this section, we will discuss our proposed GDWS model, the Extensible Geographically Distributed Web Server (EGDWS). We also present the formulation of the document distribution problem, and our proposed document replication algorithm.

The EGDWS has the following characteristics.

(1) *Adaptive document replication*: The replication and distribution of documents are periodically updated in adapting to the new access patterns.
(2) *Hot object replication*: We identify popular documents and replicate them to different server nodes to achieve balance of load. These hot objects are identified based on the access log collected during the immediately 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.

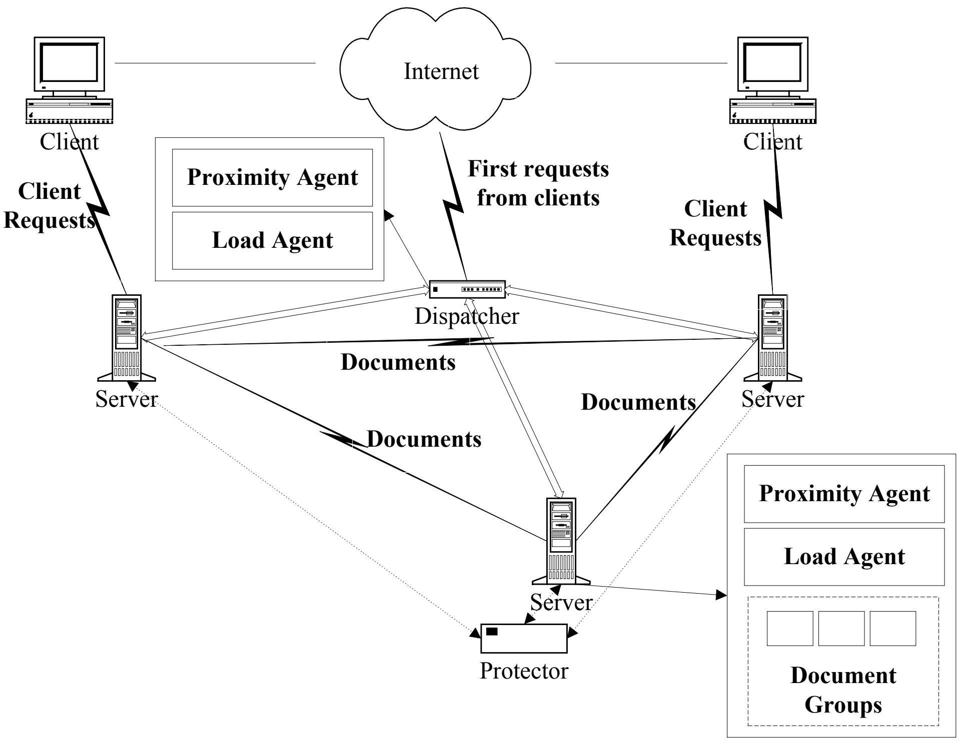


Fig. 1. Overview of the EGDWS architecture

(4) *Extensibility*: Adding a new server node or new web documents to the system is simple, and will not generate too much traffic or cause unduly disturbances to existing nodes.

3.1 Architecture Overview

Figure 1 gives an overview of the EGDWS architecture. The EGDWS consists of several main components, including the request dispatcher, the document distributor, and the server nodes.

The request dispatcher represents the advertised URL of the website and is the entry into the system. The request dispatcher keeps an index to each server node's documents. When a request arrives, it redirects the request to a server node holding a copy of the requested document. In our EGDWS prototype, we use HTTP redirection to route the incoming requests to the server nodes, because it is a standard web protocol and has been deployed in many systems. In addition, HTTP redirection has better control of incoming requests than other methods, and compared with DNS-based dispatching, it can achieve finer load balancing.

If multiple server nodes have the requested document, the request dispatcher chooses one as directed by a certain dispatching algorithm. As only a subset of documents are duplicated, a content-aware request dispatching algorithm is used in the EGDWS. A table is kept in the request dispatcher, which records where each document and its copies are located. The following is an elaboration of the dispatching process.

- *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 have the location information of the document copies.
- *Proximity-aware*: A request is assigned to the server node that is "closest" to the client. We use RTT (round trip time) to measure the proximity of a client to a server. Using RTT can lead to reduced client-perceived latency [21]. RTT takes also the least effort to obtain via existing simple utilities such as *traceroute* and *ping*.

To prevent the dispatcher from becoming a performance bottleneck, we can deploy multiple dispatchers in the system to share the load. Round-Robin DNS can be used to distribute the requests among the dispatchers.

A document distributor is also located at the central site. It collects the access records periodically from the server nodes, analyzes them to find new access patterns and to decide how documents should be replicated and redistributed. It will then send instructions to all concerned server nodes to initiate the actual document movements. The goal is to achieve better load balancing and to minimize the access latency.

In the EGDWS, server nodes are connected by a MAN or WAN. Each of them holds a part of the website's documents. When a request comes to a server node, if the requested document is stored locally at the node, the node serves the request directly. Otherwise, the node redirects the client to another server node based on a partial-URL-to-server mapping table. If the mapping table is not available, the node returns the request to the request dispatcher at the central site.

3.2 Problem Formulation

There are three main steps in our document distribution scheme:

(1) *Access pattern analysis*: To analyze the log files collected from all participating web servers to determine the popularity of each document.

(2) *Document replication*: To determine the number of replicas for each document according to its popularity and the available storage space.

(3) *Document distribution*: To determine the distribution of the documents and their replicas. As this decision is made based on the result of the document replication step as well as the current location of documents, this step is also called "document redistribution".

We introduce the variables to be used in the model. Suppose there are M server nodes and N documents in the system. S_j $(j = 1, \ldots, M)$ refers to the

jth server node. D_i ($i = 1, \ldots, N$) refers to the ith document. We have the following variables:

- s_i denotes the **size** of D_i.
- r_i denotes the **access rate** of D_i during the past period.
- w_i refers to the **weight** of D_i. It 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, $w_i = w_{disk} = s_i \times r_i$.
- 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 the **replica set** i. It contains the replicas of the set of document D_i.
- 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 a *homogeneous* system. For a homogeneous system, C and L denote the storage capacity and the load capacity respectively.
- W_j denotes the **weight** or **load** of S_j It is equal 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 is equal to the number of bytes S_j needs to fetch from other server nodes if it is assigned a replica of D_i. We can see that p_{ij} is equal to either zero or s_i, depending on whether S_j already holds

a replica of D_i.

- We also have the following variables to show whether a replica of D_i is assigned 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, $\sum_{i=1}^{N} s_i \leq \sum_{j=1}^{M} C_j$. The replication and distribution are supposed to be done under the following constraints: (1) a server can only hold replicas whose total size does not exceed the server's storage capacity; (2) a server can hold at most one replica of any document; (3) all replicas are placed on the server nodes; (4) the weight of a server is proportional to its load capacity.

Based on these constraints, we can construct an optimization problem. Its objective function is to minimize the total communication costs needed to update the document distribution. This problem was proved to be NP-complete [31]. We call a replica placement that fulfills all the constraints "feasible placement". However, because of constraint (3), this optimization problem does not necessarily have a feasible solution. Therefore, for 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 to 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 more general heterogeneous systems can be much

```
Step1.  R = \sum_{j=1}^{M} C_j - \sum_{i=1}^{N} s_i , R'= 0, c_i'= 0 (i = 1,...N)
Step2.  Sort documents by decreasing density d_i (i = 1,...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 }
```

Fig. 2. Pseudo code of the proposed Density algorithm

more complicated.

3.3 Document Replication Algorithm

Intuitively, we prefer to duplicate documents that bring more workload to the system while having a small size. Therefore, in the replication algorithm, we use the density of a document as the measure of its popularity. The larger a document's density, 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 the "Density algorithm".

The Density Algorithm is shown in Figure 2. We first reserve the disk space that is equal 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 decreasingly, and finds 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 the available disk space and the total size of the temporary replicas. Thus, the resulting replicas will still have similar densities. One special case is when 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 have at most M replicas and the space for one copy of it has been reserved in Step 1. We then subtract the appropriate space from the 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 showing how the Density algorithm works. Suppose there are 8 documents and 4 server nodes. For simplicity, we let these documents be of the same size 1 and the size of the server node be 4. The algorithm executes as follows.

In Step 1, we get R = 8.

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

In Step 4:

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

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

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

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

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

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

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

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

In Step 5, the replica numbers are determined as $\{4, 3, 2, 2, 2, 1, 1, 1\}$.

From this example, we can see that the larger 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 will not 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 target at only a very small portion of the popular documents [3,23].

The Density algorithm's time complexity is $\Theta(N \log N + N)$. 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 Document Distribution Algorithms

After determining the replication of the documents, the replicas (in the following discussion, this refers to all the documents and their replicas) need to be distributed among the server nodes. We first present several algorithms that can reduce the internal traffic of the EGDWS system. Afterwards, an algorithm that is aware of network proximity is proposed.

```
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 }}
```

Fig. 3. Pseudo code of the Greedy-cost Algorithm

4.1 Greedy-cost Algorithm

This very first algorithm aims to minimize the traffic by keeping as many as documents untouched as possible, and does not care if the load of the server nodes is balanced or not. In each round of distributing the replicas, it chooses a pair of $<$ document, server node $>$ which is has the smallest communication cost among the remaining pairs.

This algorithm is shown in Figure 3. It first sorts the $< i, j >$ pairs by the communication cost between D_i $(i = 1, \ldots, N)$ and S_j $(j = 1, \ldots, M)$ increasingly. 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 $\Theta(MN \log MN + MN)$. From the definition of p_{ij}, we know that this algorithm only duplicates and moves those documents whose replica numbers in this period are larger than the numbers of their copies in the system. It keeps most of the other documents untouched in order to reduce the communication costs.

Although this algorithm does not consider the server load during document distribution, we still expect that it would lead to some degree of 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 the user access pattern quickly, because it tends to keep the documents where they are.

4.2 Maximal-density Algorithm

```
Step1.  Sort Ri (i = 1,...N) by decreasing density, di / ci
Step2   for j = 1 to M {Do Vj = Cj }
Step3   for each Ri in the sorted list{
            3.1 Sort Sj (j = 1,...M) by increasing communication cost, pij.
                    Servers having the same pij are sorted by decreasing density, Gj
            3.2 for j = 1 to M in the sorted list{
                    if (ci > 0) && (Vj ≥ si) && (tij^l = 0, l = 1,...ci) {
                    // allocate a replica of Di to Sj
                    Vj = Vj – si, ci = ci –1
                    l = ci, tij^l = 1
                    Wj = Wj + wi / ci
                    Gj = Wj / (Cj – Vj) }}}
```

Fig. 4. Pseudo code of the Maximal-density algorithm

Unlike the Greedy-cost algorithm, the 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 see a balanced load after distributing the replicas. We expect this algorithm to achieve better load balancing performance than the Greedy-cost algorithm.

If the load is balanced after the document distribution, a server node's weight should be approximately proportional to its load capacity. In the EGDWS, a server node's storage capacity is proportional to its load capacity; and the Density algorithm makes best use of the total 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 tries to equalize the densities of the server nodes in order to balance the load among the server nodes.

The details of the Maximal-density algorithm are shown in Figure 4. 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; 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 reduces 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 $\Theta(N \log N + NM \log M)$. 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 $\Theta(NM \log M)$ time.

4.3 Greedy-penalty Algorithm

The Greedy-penalty algorithm is proposed to as a method to further decrease the communication costs. It follows a more careful allocation sequence of the replica sets. This is motivated by the observation that 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}. We call the extra traffic caused by an improper allocation sequence "penalty". We use

```
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 1^st 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 )}}
```

Fig. 5. Pseudo code of the Greedy-penalty algorithm

f_i $(i = 1, \ldots, N)$ to denote the value of penalty for D_i and try to minimize the sum of penalties. Similar algorithms have been used to solve the General Assignment Problem [18].

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 "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 c_ith 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 is placed with its best placement. If there are multiple replica 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 in the wrong order. However, using this algorithm, documents that have many replicas may have small penalties and thus be placed after unpopular documents, which may lead to large load imbalance. To avoid such problems, when there are only or fewer 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 that has enough space, like what is shown in Step 3.1 of Figure 5. In this way, most popular documents are always placed earlier than other documents and the load will be balanced to some extent.

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

4.4 Proximity-Aware Algorithm

In the above algorithms, we assume that the popularity distribution among the different geographic regions is uniform, that is, most documents experience the same popularity at different server locations. For the next algorithm, we assume that the server nodes are distributed in U different Internet regions, and documents may have different popularities in different Internet regions. We introduce a proximity-aware document distribution algorithm which dis-

```
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 }}}}
```

Fig. 6. Pseudo code of the Proximity-aware algorithm

tributes 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 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 6. 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 is 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 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.

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

4.5 Proofs and Discussion

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

4.5.1 Correctness

To satisfy constraint (3), after executing any of the above document distribution algorithms, an adjustment algorithm is needed. It checks if there exists a document without a copy in the system, and if there is, it removes copies of some other documents to make room for this document.

We confine the correctness discussion 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 algorithm can place at least one copy of every document 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 copies of documents with more than one copy in the system, and places D_a in 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. 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 contradictory to our assumption.

4.5.2 Communication Cost

We use MAX_TRAFFIC to denote the maximal traffic an algorithm may cause. In the 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 the Maximal-density algorithm, some replicas may not be placed on server nodes that have smaller communication costs. 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 the 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 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, and 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, the Maximal-density algorithm generates more traffic than the 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 the request distribution, the smaller is the cost.

4.5.3 Load Balancing

It is difficult to analyze the load balancing performance of the algorithms; therefore we only discuss the main factors influencing load balancing. To do this, we use y_i to denote the number of copies of D_i in the system after document distribution. We know that the more similar the densities of replicas, i.e., d_i/y_i $(i = 1, \ldots, N)$, the smaller is the load imbalance. In the 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 is not equal to c_i. As Maximal-density and Greedy-cost place replicas in order of increasing sizes, their load balancing may be affected more than Greedy-penalty by this process. 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 the request distribution, the smaller is the load imbalance in the system.

5 Performance Results and Analysis

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

5.1 Simulation Setup

We use the CSIM 18 package [9] to simulate a homogeneous EGDWS system of 16 server nodes. Each node has storage capacity C and load capacity $L = 3000$ [13]. HTTP 1.1 is used in the simulation. The first request of a connection always goes to the request dispatcher. The dispatcher then redirects the client to a server node according to its request dispatching algorithm. If the chosen server node is already holding L existing HTTP connections, the request is dropped. Otherwise, the client sends its subsequent requests directly to the chosen server node. When the client requests a document that is not stored locally in this server node, the request is returned to the dispatcher, which then selects another server node for the client.

Initially, the website's documents are randomly assigned to the server nodes without replication. Afterwards, the document distribution scheme is executed once every 3 hours. The simulation lasts for one day. The algorithms are simulated in the following patterns.

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

For the purpose of comparison, we include another document distribution scheme, **Dynamic Scheme (DS)**, in the simulation. In this scheme, each server node owns a part of the collection of documents. Periodically each server examines the number of bytes a server has served in the last period, and determines whether it is under-loaded or overloaded. A server is overloaded if the number of bytes it served is 50% more than the average number of bytes served by a server [1]. A server is under-loaded if the number of bytes it served is less than the average value. Each server then replicates one of its documents to the under-loaded nodes or revokes a replica of its documents from the overloaded nodes. The load of the servers is updated correspondingly. This scheme is used in DC-Apache [16]. In the simulation, the servers check the load status every 30 minutes.

[1] We also did simulation with "overloaded" being 10%, 20%, 30% and 40%. The results were similar and thus are not shown here.

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
Document Size Variance	2840	1770	135676
Total size of Documents (KB)	250,868	437,544.9	7,464,779.2
Duration	Aug 3-31, 1995	Sep 4-10, 1995	Jan 24-30, 2001

We used three real traces of web accesses [2] . They are summarized in Table 1. In the simulation, for simplicity, documents that are under the same second-level directory are grouped together. The resulting groups are used as units in document replication and distribution. Figure 7 reflects the reference locality of the three 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 3, a smaller part of requests are responsible for most requests than in Data Set 1 and Data Set 2.

The following metrics are used in the presentation of the simulation results.

(1) *Load Balance Metric (LBM)*

The Load Balance Metric (LBM) was proposed in [7]. It is used to measure load-balancing performance. During the simulation, the *utilization*, i.e., the percentage of time a server node is busy, is measured once every

[2] Downloaded from http://ita.ee.lbl.gov/html/traces.html

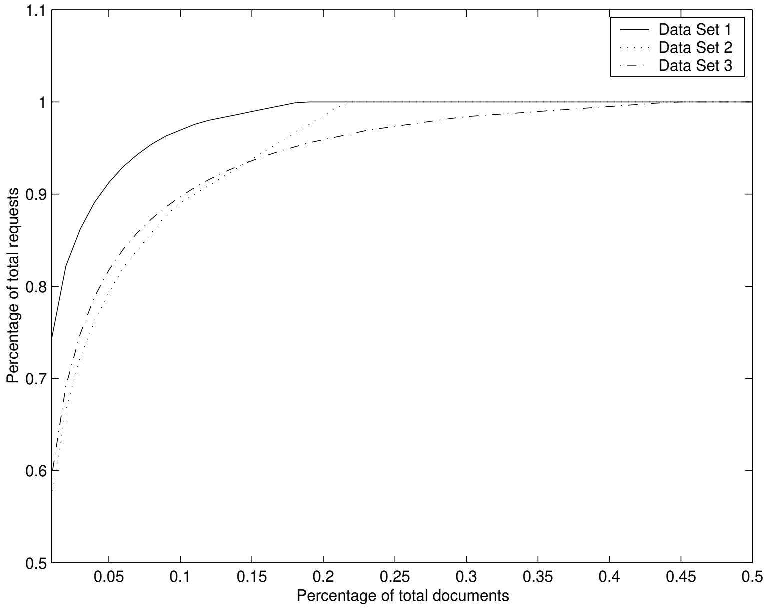


Fig. 7. Reference locality of the tested log files

10 minutes. The LBM value is the weighted average of the peak-to-mean ratios captured at the sampling points. The sampling point with a heavy load has a higher weight. The value of the LBM ranges from 1 to M, where M is the number of server nodes in the system. A smaller LBM value indicates better load balancing performance.

(2) *Average Traffic (AT)*

This metric is used to compare the traffic caused by different algorithms in redistributing documents. We record the total traffic caused by all servers each time the document distribution scheme is executed. At the end of the simulation, the average traffic is computed. The ratio between the average traffic and the total size of web documents without duplication is computed. This ratio is denoted as AT in the following discussion.

We performed two sets of experiment. In Experiment 1, we simulated a MAN-based environment, in which all the server nodes are in one Internet region. In

Experiment 2, the server nodes are dispersed in different Internet regions. The proximity-based document distribution algorithm and the request dispatching algorithm are deployed in Experiment 2. The improvement on performance is then analyzed.

5.2 Experiment 1—MAN-based Experiment

In this experiment, we assume all 16 server nodes are in the same Internet region. Therefore, the distances between a client and different server nodes are treated as equal. The reported processing time of 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. For simplicity, we assume the redirection time is fixed at 100 ms [24]. If the redirection is initiated by a server node and not by the request dispatcher, the client will be redirected twice. Thus, the redirection time is equal to $2 \times 100 = 200$ ms. The dispatcher deploys a round-robin request dispatching algorithm. The disk latency parameters are derived from Seagate ST360020A technical specification [27]. Disk access time is of 8.9 ms and disk transfer time is of 21 MB/s.

We first compare the average user-perceived response time achieved by the proposed algorithms. The results are shown in Figures 8, 10 and 12. The x-axis shows the degree of document replication, that is, the ratio between the servers' storage capacity and the total size of the documents, $C/\sum_{i=1}^{N} s_i$. For example, $C/\sum_{i=1}^{N} s_i = 1/16$ implies no replication and $C/\sum_{i=1}^{N} s_i = 16/16$ implies a full-replication scheme. The y-axis is the speedup achieved by various document distribution algorithms, as compared to the average response time measured under the same configuration but without document replication. The figures show that our document distribution scheme can improve the speedup up to three times. However, the improvement in Data Set 3 is not so apparent. This is because the reference locality in Data Set 3 is not as obvious as in the other data sets. In this case, the scheme of replicating popular documents is not efficient in distributing the load or cutting down redirections.

We next examine the load balancing performance. Figures 9, 11 and 13 present the LBM values at different document replication degrees. When $C/\sum_{i=1}^{N} s_i$ $= 1/16$, no documents are replicated and all algorithms yield the same LBM value. Using our scheme, the system can be five to six times more balanced than using the Dynamic Scheme (DS). The load balancing performance of our scheme increases as storage capacity increases. This is because the Density Algorithm replicates as many documents as the storage capacity allows. One exception is for Data Set 3, when $C/\sum_{i=1}^{N} s_i$ is larger than $1/2$, LBM value increases instead of decreases. This may be due to some dramatic access pattern change in the log file.

The three proposed algorithms differ little in load balancing performance because they use the same replication algorithm. As expected, GC generally performs worst as it does not care about the system load when it distributes the replicas. However, when the storage capacity is relatively small, MD sometimes yields the largest LBM values. This is because MD places the replicas in order of decreasing densities; when some unpopular but large documents are not placed on any server nodes, the adjustment algorithm will delete other documents' replicas and 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

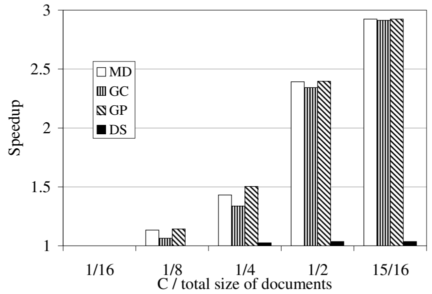


Fig. 8. Response time speedup for Data Set 1

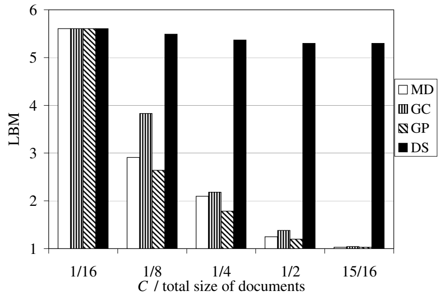


Fig. 9. Load balancing effect for Data Set 1

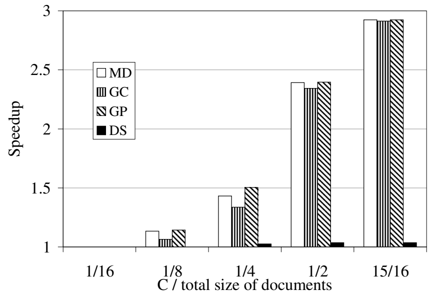


Fig. 10. Response time speedup for Data Set 2

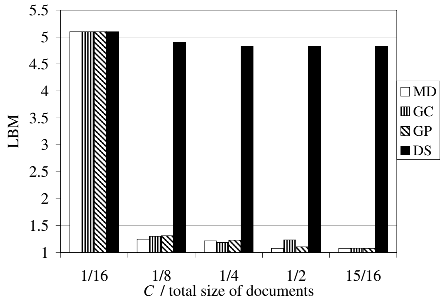


Fig. 11. Load balancing effect for Data Set 2

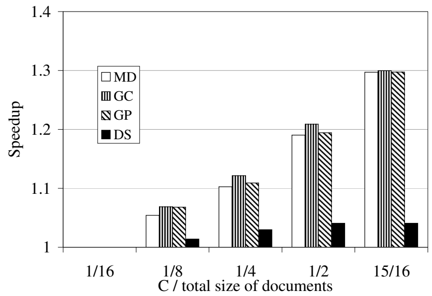


Fig. 12. Response time speedup for Data Set 3

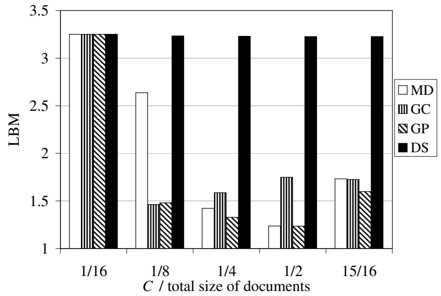


Fig. 13. Load balancing effect for Data Set 3

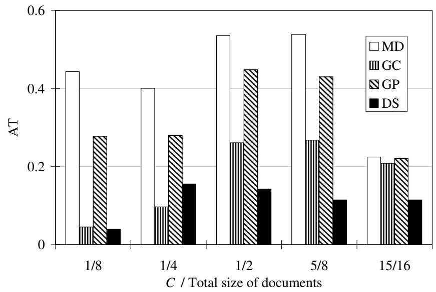


Fig. 14. Average traffic for Data Set 1

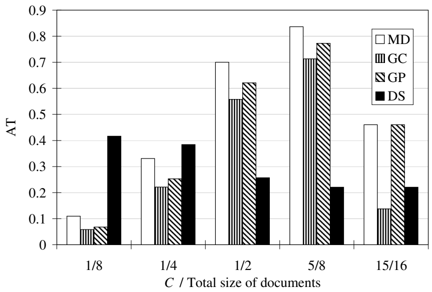


Fig. 15. Average traffic for Data Set 2

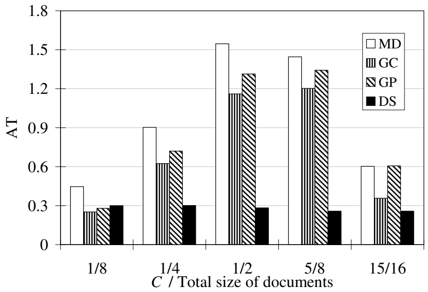


Fig. 16. Average traffic for Data Set 3

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. Average Traffic (AT) is represented in y-axis in Figures 14, 15 and 16. We can see that when the number of server nodes is fixed, the traffic caused by our scheme 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; meanwhile, the numbers of replicas of popular documents are larger. Therefore, once the access pattern changes, more replicas of the past period are revoked and more new replicas need to be distributed. For example, Data Set 3 has the least reference locality and thus causes most traffic. Our scheme at most causes two times more average traffic than the DS. Among the algorithms, GC incurs the least traffic, 50% less than MD. GP generates 20-30% less traffic than MD. When the storage capacity is large, more documents have replica numbers equal to or close to M. Thus, the difference between traffic caused by MD and GP decreases.

The above simulation results indicate that our document distribution scheme can lead to shorter response time and better load balancing. Another observation is that when the document replication degree $C/\sum_{i=1}^{N} s_i = M/2$ (i.e., half of the server nodes), our scheme can achieve 80% of the performance of full-replication. On the other hand, when the replication degree continues to increase, the load balancing performance (LBM) improves little while the internal traffic (AT) increases much. Therefore, we conclude that our document distribution scheme is most suitable for GDWS systems with small to moderate storage capacity.

The simulation results also enable comparison of the different document distribution algorithms. This comparison will help select an appropriate algorithm for a given system. Among the three algorithms discussed in Experiment 1, GC's load balancing performance is not as good as that of GP and MD, and is the easiest to be affected by initial placement of the documents. However, it 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, MD's load balancing performance may deteriorate when the storage capacity is small. Comparatively, GP yields the most stable load balancing performance, but it requires more computation than the other algorithms.

Table 2

Network types in WAN-based connection

Bandwidth Type	Bandwidth (B)	Redirection Time (rdt)
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

Table 3

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

5.3 Experiment 2 - WAN-based Experiment

In this experiment, we assume that the Internet is divided into four regions located in different geographic areas. Each region contains $M/4$ server nodes. The network in a WAN-based connection can be classified into four types: large, medium-large, medium-narrow, and narrow, as shown in Table 2. The bandwidth values reported in the table are taken from [28]. We assume that the network bandwidth does not change during the experiment.

In this experiment, the client requests come from different Internet regions. If a request originates from E_i and is redirected to E_j by the request dispatcher, the redirection time equals to rdt_{ij}. If the redirection is initiated by a server node in E_l, the redirection time is $rdt_{li} + rdt_{ij}$. In addition to the redirection time, waiting time and disk reading time, the response time also includes the time needed to transfer the documents over 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}, where B_{ij} is the bandwidth between E_i and E_j.

In this experiment, we simulate three scenarios:

- **MD-RR**: Maximal-density algorithm with round-robin request dispatching strategy.
- **MD-Prox**: Maximal-density algorithm with proximity-aware request dispatching strategy.
- **Prox-Prox**: Proximity-aware algorithm with proximity-aware request dispatching strategy.

For better comparison, in the following discussion, the results of MD-RR are used as the baseline. The results of MD-Prox and Prox-Prox are presented in the form of relative values. Figures 17, 18 and 19 present the response time results. The x-axis is document replication degree, $C/\sum_{i=1}^{N} s_i$. The y-axis is relative response time which is equal to response time / MD-RR's response time. We can see that MD-Prox and Prox-Prox shortened the user response time by up to 50%. This proves that the proximity-aware document dispatching algorithm can reduce network latency. Prox-Prox further improves the response time because it tries to place replicas near where they are most needed. The gap between MD-Prox and Prox-Prox decreases as C increases. This is because there are more replicas in the system and more documents are placed on all the server nodes.

Next we compare the load balancing performance of the three scenarios. In Figures 20, 22 and 24, the y-axis is the *relative LBM value*. We see that MD-Prox and Prox-Prox yield worse load balancing than MD-RR. Their LBM values can be up to 2.5 times larger. With the proximity-aware request dispatching algorithm, the requests are not assigned equally to server nodes holding the requested documents. Instead, requests are assigned to server nodes in a certain Internet region, based on where the requests originate. The worse load balancing performance of MD-Prox thus implies that the documents experience different popularities in different Internet regions. However, by placing more replicas of a document where it is more popular, Prox-Prox did not improve load balancing. Using both MD-Prox and Prox-Prox, the relative LBM values increase as the storage capacity increases. 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 every server node, redirecting requests to the nearest server node causes more obvious load imbalance.

Figures 21, 23 and 25 present the average traffic generated in the three scenarios. The y-axis is the **relative AT**. We see that MD-Prox generates the same amount of traffic as MD-RR, because they use the same document distribution scheme. Actually, MD-Prox's traffic is mainly caused by the change of the documents' replica numbers. For example, when unpopular documents in the 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 the documents' request distributions. Even if a document's overall popularity remains the same, when its access popularity in different Internet regions changes, its replicas still need to be moved among the server nodes in different regions. Therefore, generally Prox-Prox generates more traffic than MD-Prox. However, the actual delivery time of Prox-Prox could be shorter than that of MD-Prox because the replicas may be transferred over networks with higher bandwidths.

From the results of Experiment 2, we see that in a WAN-based EGDWS, proximity-aware request dispatching algorithm and document distribution algorithm can effectively reduce the response time. However, they also disturb the load balancing in the system, especially when the storage capacity is large. The performance improvement of the proximity-aware document distribution algorithm is further offset by the additional internal traffic. Therefore, in a WAN-based system, the proximity-aware request dispatching algorithm and document distribution algorithm should be used with carefully so as not to cause overloading in certain server nodes.

6 Conclusion and Future Work

In this paper, we describe the system architecture of the Extensible Geographically Web Server system, and propose a document distribution scheme to support the efficient operation of the system. In the 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

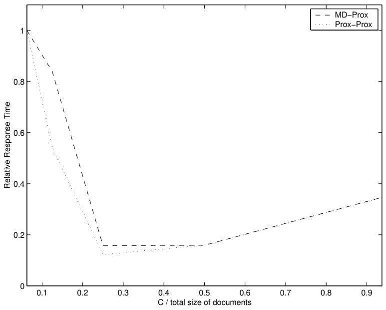


Fig. 17. Relative response time for Data Set 1

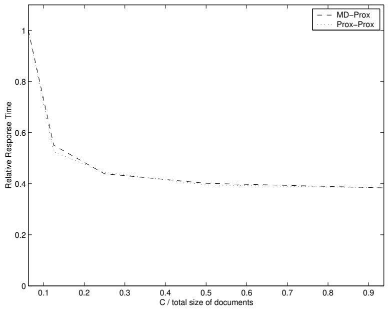


Fig. 18. Relative response time for Data Set 2

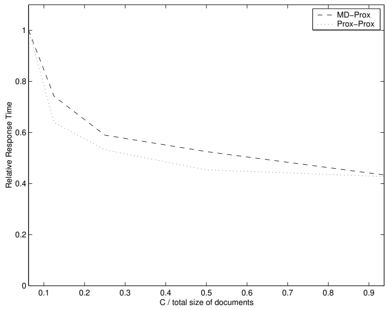
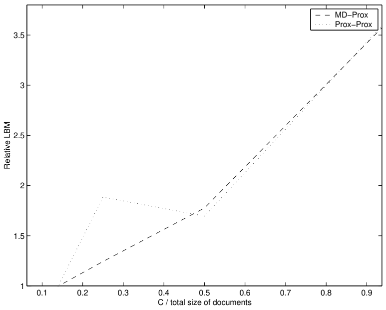


Fig. 19. Relative response time for Data Set 3

the last period, and then distributes the replicas to the server nodes. It consists of several constituent algorithms that rely on information on the current document locations and the documents' popularities from different perspectives. Their overall aim is to produce a balanced load and to minimize the traffic generated from document distribution or redistribution. 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.

Simulation results show that our document distribution scheme could pro-



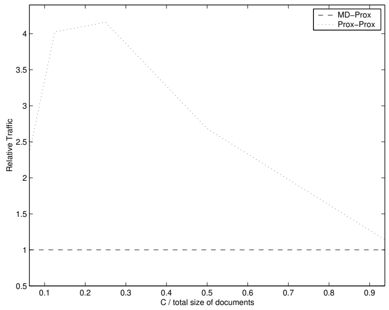
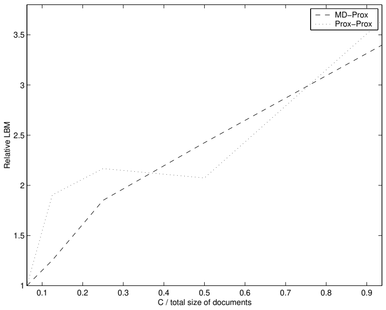


Fig. 20. Relative LBM for Data Set 1

Fig. 21. Relative AT for Data Set 2



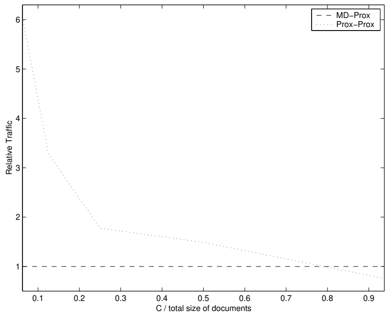
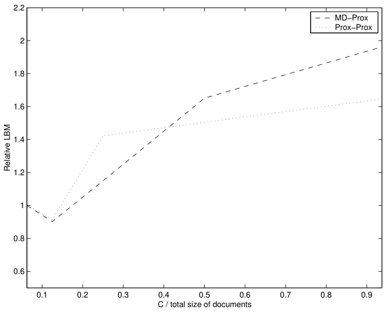


Fig. 22. Relative LBM for Data Set 2

Fig. 23. Relative AT for Data Set 2



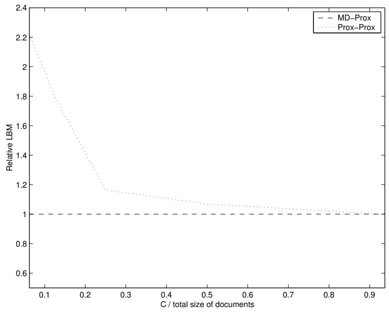


Fig. 24. Relative LBM for Data Set 3

Fig. 25. Relative AT for Data Set 3

vide better web service, and achieve better load balancing 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 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 better 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 situations are good targets for further research.

Acknowledgement

The authors are grateful to the anonymous referees for their comments, and to Savio Tse for many useful suggestions.

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

[7] 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.

[8] 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.

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

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

[11] 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.

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

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

[14] 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.

[15] 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.

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

[17] 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.

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

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

[20] 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.

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

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

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

[30] 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.

[31] 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.

Biographies

Ling Zhuo received her B.Eng. degree from Tsinghua University, China in 2000, and a MPhil degree in CS from the University of Hong Kong in 2002. She is now a Ph.D. student in the University of Southern California.

Cho-Li Wang received his B.S. degree from National Taiwan University, and M.S. and Ph.D. degrees in Computer Engineering from University of Southern California. He joined the Department of Computer Science and Information Systems at The University of Hong Kong in 1995 and is now an associate professor. His research mainly focuses on Cluster and Internet Computing. His current research involves : High-Speed Cluster Networking, Distributed Java Virtual Machine, Parallel and Distributed Web Server System, and Software Architecture for Pervasive/Ubiquitous Computing.

Francis C.M. Lau (S'78-M'87) received his B.Sc. degree from Acadia University, Canada, and a M.Math. and a Ph.D. degree from the University of Waterloo. He joined the Department of Computer Science and Information Systems at The University of Hong Kong in 1987 where he is now the head of the department. His research interests are in parallel and distributed computing, object-oriented programming, operating systems, and Web and Internet computing.