Load Balancing in Distributed Web Server Systems with Partial Document Replication*

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

How documents of a Web site are replicated and where they are placed among the server nodes have an important bearing on balance of load in a geographically Distributed Web Server (DWS) system. The traffic generated due to movements of documents at runtime could also affect the performance of the DWS system. In this paper, we prove that minimizing such traffic is NP-hard. We propose a new document distribution scheme that periodically performs partial replication of a site's documents at selected server locations to maintain load balancing. Several approximation algorithms are used in it to minimize traffic generated. The simulation results show that this scheme can achieve better load balancing than a dynamic scheme, while the internal traffic it causes has a negligible effect on the system's performance.

1. Introduction

The increasing popularity of the World Wide Web has resulted in large bandwidth demands which translate into high latencies perceived by Web users. To tackle this latency problem, multiple copies of documents are distributed geographically and placed in caches at optimal locations.

Web caching attempts to reduce network latency by storing the commonly requested documents as close to clients as possible. Simple caches have no information on users' access pattern, and so they would habitually try to keep a copy of any document just requested. This may limit the performance of caches. For example, research in [1] shows that the maximum cache hit rate achievable by any caching algorithm is bounded under 40% to 50%

A proactive Web server on the other hand can decide where to place copies of a document in a distributed Web server (DWS) system where the server nodes are distributed geographically. In most existing geographically DWS systems, each server node keep the entire set of Web documents managed by the system. Incoming requests are distributed to the server nodes via DNS servers [6,9,11]. Although such systems are simple to implement, the caching of IP addresses on the client side or in intermediate DNS servers could easily result in uneven load among the server nodes. Moreover, the full replication leads to much waste of disk space due to those documents that are not frequently requested.

To achieve better load balancing as well as to avoid disk wastage, one can replicate part of the documents on multiple server nodes [5,12,14,15,18,19] and use content-aware distributor software to redirect a client request to a server node that has the requested document [16]. Some rules are then needed in such a geographically DWS system to determine each document's number of replicas and the distribution of these replicas. These rules constitute what we call the *document distribution scheme*, and they should achieve the following goals.

- *Load Balancing*: Since requests tend to target at a small part of the entire collection of documents [3], frequently requested documents should be replicated to avoid bottlenecks. Documents and their replicas should be placed in such a manner that most of the time the load of the participating server nodes is equalized.
- *Reduced Traffic*: To adapt to users' access patterns, documents need to be re-duplicated and re-distributed among the server nodes dynamically or periodically. Communications caused by such actions should be kept to the minimum so that the performance of the geographically DWS system would not be adversely affected.

Existing schemes mainly focus on balancing the load, but not the traffic issues. In this paper, we propose a new document distribution scheme that can improve load balancing performance of geographically DWS systems,

* This research was supported by CRCG Grant 10203944 and HKU 2001/02 Large Items of Equipment Grants 10003.01/02/001

while minimizing the communication cost needed. We assume that each document has approximately the same popularity in all the server locations. Therefore, we will not consider network proximity to clients in replicating and distributing the documents.

The performance of our scheme is evaluated through simulation using real access log data. The results show that this scheme can balance the load in the DWS system during run-time efficiently, and the internal traffic generated due to these algorithms is reasonably minimal.

The rest of the paper is organized as follows. Section 2 formulates the document distribution problem in the DWS systems and gives a proof of its NP-hardness. Section 3 presents our document distribution algorithms. In Section 4, we describe our simulation methodology and present the performance results. Section 5 surveys related work, and Section 6 concludes the paper and discusses future work.

2. Problem Formulation

In this section, we formulate the document distribution problem in DWS systems. Chen [7] proved that minimizing the maximum load over all server nodes is NP-complete. We will prove that even when the load balancing constraint is removed, the problem of minimizing the communication cost of moving the documents is NP-hard.

Suppose there are N documents and M server nodes in the system. Each server node has storage capacity C. Each document has size of s_i and number of replicas c_i (In this paper, if we don't state otherwise, we assume $i = 1,\ldots N$ and $j = 1,\ldots M$).

A "cost link" is constructed between each document and each server: p_{ij}, associated with the number of bytes to be transferred if document i is assigned to server j. We also have variables $t_{ij}^{\ l}$ ($l = 1,\ldots c_i$), which is 1 if lth replica of ith document is placed on jth server; otherwise, it is 0.

The determination of c_i is under the limitation of total storage, i.e., $\sum_{i=1}^{N}(s_i * c_i) \le M * C$.

After c_i is determined, all the documents and their replicas are placed on the server nodes under these constraints: (1) each server can only hold replicas whose total size does not exceed its disk space; (2) each server can hold at most one replica of a document; (3) no document is left unassigned to any server node; (4) load is equalized among the server nodes.

As we stated at the beginning of the section, we won't include constraint (4) in the formulation. The replica placement problem formulation is therefore as below:

$$\text{minimize } z = \sum_{j=1}^{M}\sum_{i=1}^{N}\sum_{l=1}^{c_i} t_{ij}^{l} p_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{N}\sum_{l=1}^{c_i} t_{ij}^{l} s_i \le C \qquad (1)$$

$$\sum_{l=1}^{c_i} t_{ij}^{l} \le 1, \qquad (2)$$

$$\sum_{j=1}^{M}\sum_{l=1}^{c_i} t_{ij}^{l} = c_i \qquad (3)$$

$$t_{ij}^{l} = 0 \text{ or } 1, \ l = 1,\ldots c_i$$

A replica placement that fulfills all the above constraints is a "feasible placement." Our discussion is under the assumption that a feasible placement always exists. We call this optimization problem the *Replica Placement Problem* (RPP). When $c_i = 1$, the problem is 0-1 RPP.

Lemma *0-1 RPP is NP-hard*

Proof: We reduce the bin-packing problem, which is NP-hard [13], to the 0-1 RPP. For the bin-packing problem, s_i denotes the size of object i and the bin's size is C. We assume that, in any feasible solution, the lowest indexed bins are used. This means that if there are two bins with the same available storage, the object will be placed in the one with the lower index.

Given the bin-packing problem, we can construct a 0-1 RPP with costs p_{ij} as follows.

$$p_{ij} = \begin{cases} 1, & i \in \{1,\ldots N\}, j = 1 \\ (p_{i,j-1}) * N + 1, & i \in \{1,\ldots N\}, j \in \{2,\ldots M\} \end{cases}$$

With such costs, the total cost of any set of replicas assigned to $\{s_1,\ldots s_j\}$ is lower than the total cost of any set of replicas assigned to $\{s_1,\ldots s_{j+1}\}$. It is then obvious that the bin-packing problem gets the minimal number of bins used if and only if the 0-1 RPP gets the minimal total communication cost.

Since the 0-1 RPP is a special case of the RPP, our document placement problem is NP-hard.

3. Document Distribution Scheme

In this section, we propose our document distribution scheme, which periodically re-replicates and re-distributes the documents based on the access pattern in the past period. We first describe an algorithm for determining the number of replicas for each document. Next, we present several heuristics that use available information in different ways in order to achieve minimal communication cost.

3.1. Density Algorithm

Intuitively, we should prefer to duplicate documents that require more work on the part of the DWS system as

well as the small-size ones. We use the concept of "density" to represent the workload per unit storage of the document. The larger a document's density is, the more replicas it will have.

Input: d_i, s_i, C, M, N,
Variables: S, total size of document
S_disk, available disk space
d_{min}, minimal density
$temp\_S$, total size of temporary replicas
$temp\_c_i$, temporary number of replicas
Output: c_i (i =1, …N)
1.compute S, $S\_disk = M * C - S$
2.sort documents by decreasing density d_i,
find d_{min}
3.**for** i = 1 to N {
$temp\_c_i = d_i / d_{min}$ }
compute $temp\_S$
4.**for** i = 1 to N {
$c_i = temp\_c_i * S\_disk / temp\_S$
if (c_i >= M-1){
$c_i = M$-1
$temp\_S = temp\_S – temp\_c_i * s_i$
$S\_disk = S\_disk – c_i * s_i$ }}
5.finally decide c_i (i = 1,…N)

Figure 1. Density algorithm

To compute a document's density, we associate document i with weight w_i, which represents the workload it brings to the server node holding it. In our algorithm, w_i is computed as $s_i * r_i$, where s_i is the document's size and r_i is its access rate in the past period. The density of a document d_i, therefore, equals to w_i / s_i. If a document is duplicated, we assume that the workload is divided evenly among its replicas (true if we assign requests to the replicas in a round-robin manner). Therefore, a replica of document i has weight of w_i / c_i and density of d_i / c_i.

The Density Algorithm is shown in Figure 1. First, the space equal to the total size of documents is reserved to guarantee that each document has at least one copy in the system. Step 2 sorts the documents by their densities decreasingly. In Step 3, each document gets a temporary replica number. The densities of the temporary replicas are nearly equal to the minimal density. Step 4 adjusts the temporary replica numbers under the storage limitation. Replica numbers are computed according to the ratio between available disk space and the total size of the temporary replicas, thus the resulting replicas still maintain similar densities. In Step 5, each replica number is finally decided as an integer not larger than M.

This algorithm replicates the documents according to their densities under the storage limitation. The time complexity of it is $\Theta(N \log N + N)$. From Step 4, we know that for any two documents u and v, if $1 < c_u, c_v \le M$ and $d_u > d_v$, $d_u / (c_u - 1) \approx d_v / (c_v - 1)$. Thus, we can assume if $d_u > d_v$, $d_u / c_u > d_v / c_v$, for any two documents u and v.

Since we have proved that the minimization problem of communication cost is NP-hard, in the rest of this section, several approximation algorithms for distributing the replicas to the server nodes are proposed. Before we begin the discussion, we need to introduce a new variable, W_j. It denotes the weight of server j and is computed as the sum of the weights of all replicas allocated to it. Also, in the following discussion, we call document i and its replicas "replica set i".

3.2. Greedy-cost Algorithm

Input: c_i, s_i, p_{ij}, C, M, N
Output: t_{ij}^l (i = 1,…N, j = 1,…M, l = 1,…c_i)
1.sort (i, j) pairs by increasing cost, p_{ij}
2.**for** each (i, j) in the sorted list{
if (c_i > 0) {
allocate a replica to server j if it has enough space and $t_{ij}^l = 0$ (l = 1,…c_i).
$c_i = c_i – 1$ }}

Figure 2. Greedy-cost algorithm

Greedy-cost algorithm aims to minimize traffic by keeping as many as documents as they are in the system, without caring if the load of the servers is balanced.

This algorithm is shown in Figure 2. To minimize the cost, it first sorts the (document, server) pairs by the communication cost between the document and the server increasingly. Then in this order, a replica of document i is allocated to server j if it has enough storage space and has not been assigned the same replica in this period. The total time complexity is $\Theta(MN \log MN + MN)$.

3.3. Greedy-load/cost Algorithm

Unlike Greedy-cost algorithm, Greedy-load/cost algorithm considers balancing the load among the server nodes as well as minimizing the cost caused in distributing the documents.

Input: c_i, s_i, p_{ij}, C, M, N
Output: t_{ij}^l ($i = 1,\ldots N, j = 1,\ldots M, l = 1,\ldots c_i$)
Variable: D_j, density of server j ($j = 1,\ldots M$)
1.sort replica sets by increasing density, d_i / c_i
2.**for** $i = 1$ to N {
sort servers by increasing communication cost, p_{ij}. Servers having the same p_{ij} are sorted by decreasing density, D_j
allocate replica set i
update D_j ($j = 1,\ldots M$) }

Figure 3. Greedy-load / cost algorithm

To achieve load balancing, we expect seeing that after document distribution, the weights of the server nodes are approximately the same. Since the density of a server node D_j equals to W_j / (amount of used disk space in server j), in a homogeneous DWS system, this means that after document distribution, the server nodes have similar densities. Therefore, in Greedy-load/cost algorithm, the replica sets are sorted by decreasing density and are allocated in this order. When choosing server nodes for replica set 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 D_j is chosen.

The time complexity of Greedy-load algorithm is $\Theta(N \log N + NM \log M)$. To simplify it, we can use the sorting result of the Density algorithm, based on the assumption that if $d_u > d_v$, $d_u / c_u > d_v / c_v$. Thus the algorithm only takes $\Theta(NM \log M)$ time.

3.4. Greedy-penalty Algorithm

It is possible that allocating a document at different times generates different traffic. For example, if we allocate document i immediately, we can assign it to server x with $p_{ix} = 0$; if we delay allocating it for a while, however, server x may have become full and the document has to be placed on server y with $p_{iy} = s_i$. In this case, we say we are punished and use f_i to refer to the value of penalty. A penalty-based algorithm hopes to decrease cost by placing documents in certain order. It has been used to solve the General Assignment Problem [13].

We say a placement is "better" if it incurs less communication cost. In Greedy-penalty algorithm, f_i is computed as the difference in the costs of replica set i's best and second best placements, according to the current status of the server nodes. This algorithm iteratively places the replica sets until they are all allocated. Each time it computes f_i for all unassigned replica sets, and the one yielding the largest penalty are placed with its best placement. The time complexity of this algorithm is $\Theta(N^2 \log N + NM \log M)$.

Input: c_i, p_{ij}, s_i, C, M, N
Output: t_{ij}^l ($i = 1,\ldots N, j = 1,\ldots M, l = 1,\ldots c_i$)
Variables: f_j, penalty for document i ($i = 1,\ldots N$)
while there are unassigned replica sets {
for each unassigned replica set i{
if only c_i server nodes have enough storage to hold document i{
allocate replica set i
goto while }
else {
sort servers by increasing cost with document i, p_{ij}.
compute f_i }}
sort replica sets in decreasing penalty, f_i
allocate the replica set with minimal f_i in its best placement}

Figure 4. Greedy-penalty algorithm

If there are only c_i server nodes having enough storage to hold document i, we need to allocate replica set i immediately. Otherwise, we might leave a replica unassigned and violate constraint (3). In this case, we set f_i to ∞. If there are multiple replica sets with infinite penalty, they are placed in the order of decreasing densities. To do this, we can use the sorting results of the Density algorithm.

4. Simulation Results

4.1. Experimental Setup

We use the CSIM 18 package [21] for our simulation. In our simulation model, requests are redirected to the server nodes that have the requested documents using HTTP redirection. When a document has copies in multiple server nodes, the requests for it are assigned in round-robin fashion. Initially, Web documents are randomly placed on the server nodes without replication. Afterwards, documents are replicated and distributed among the server nodes every 3 hours.

In our simulation, processing a web request comprises (1) redirection (if necessary), (2) waiting in the queue of the serving server node, (3) reading the file from disk. The connection establishment time and teardown time is neglected. The round-trip time of redirection is 100 ms [16]. The disk access time is about 19 ms and the disk transfer time about 21 MB/s [22]. We use two real traces of Web access. One is from a website mainly used for hosting personal homepages, called Data Set 1. Another, called Data Set 2, is obtained from The Internet Traffic Archive [20]. For simplicity, the documents in the same directory are grouped and these groups are used as basic units of replication and distribution.

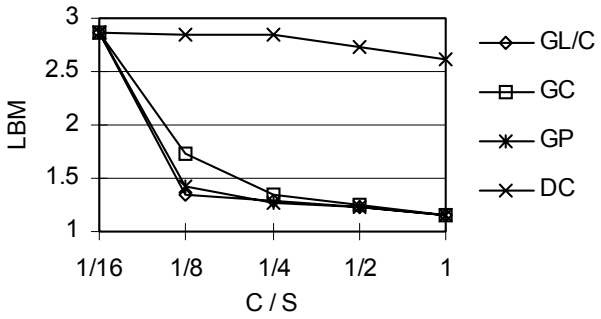

Figure 5. Load balancing performance with Data Set 1 (16 server nodes)

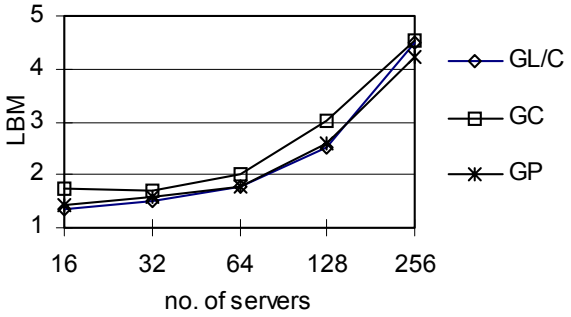

Figure 7. Load balancing performance with Data Set 1 (*C* / *S* = 1/8)

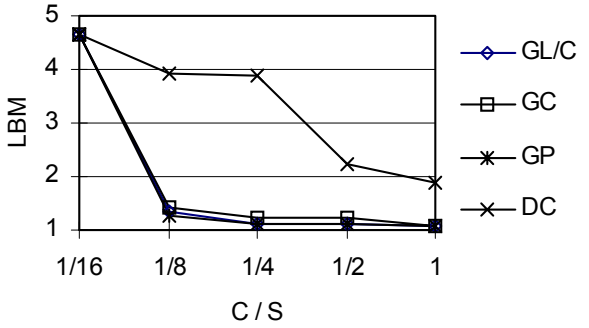

Figure 6. Load balancing performance with Data Set 2 (16 server nodes)

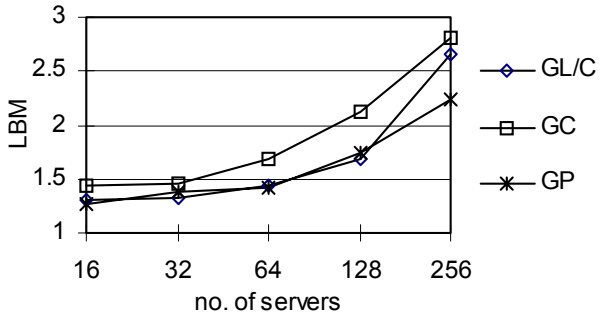

Figure 8. Load balancing performance with Data Set 2 (*C* / *S* = 1 / 8)

We simulated the algorithms presented in Section 3. Density algorithm is combined with Greedy-cost (GC), Greedy-load/cost (GL/C), Greedy-penalty (GP) respectively. For the purpose of comparison, we added a Dynamic scheme (DS). In this scheme, each server node owns a part of documents. Dynamically it examines the other servers' load and determines if they are under-loaded or overloaded or. It then replicates one of its documents to the under-loaded node or revokes one replica of its documents from the overloaded node. This scheme is similar to the one used in DC-Apache. In our simulation, the servers check load status every 10 minutes.

4.2. Load Balancing Analyses

The Load Balance Metric (LMB) [4] is used as a performance metric for measuring load balancing results. We record the peak-to-mean ratio of server utilization every sampling period (10 minutes) during the simulation. The LBM value is obtained by calculating the weighted average of the peak-to-mean ratios measured, using the total server utilization at the sampling point as the weight. A smaller LBM value indicates better load balancing performance.

Figure 5 and Figure 6 present the load balancing performance of our scheme when the number of servers is fixed as 16. The y-axis is LBM value. The x-axis is *C* / *S*, where *C* is the storage capacity of each server node and *S* is the total size of the documents. We can 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 fully utilizes the disk space. Among the document distribution algorithms, GC performs worst in load balancing while GL/C and GP's performance is similar.

Next, we fix the storage capacity, and increase the number of server nodes from 16 to 256. The results are shown in Figure 7 and Figure 8. We notice that GL/C's and GP's performance is still close when the node number is not very large. When there are more than 128 nodes, however, GL/C appears to deteriorate faster than GP.

4.3. Traffic Analyses

We record the total number of bytes transferred inside the system each period (except the first period, as documents are randomly distributed without duplication initially). At the end of the simulation, the ratio between

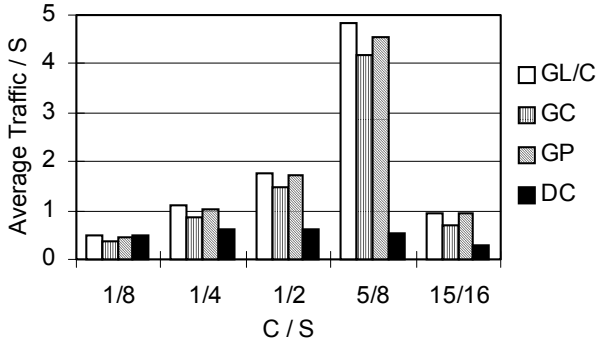

Figure 9. Average traffic with Data Set 1 (16 server nodes)

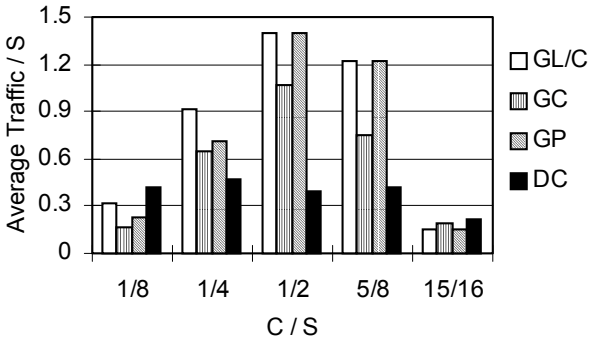

Figure 11. Average traffic with Data Set 2 (16 server nodes)

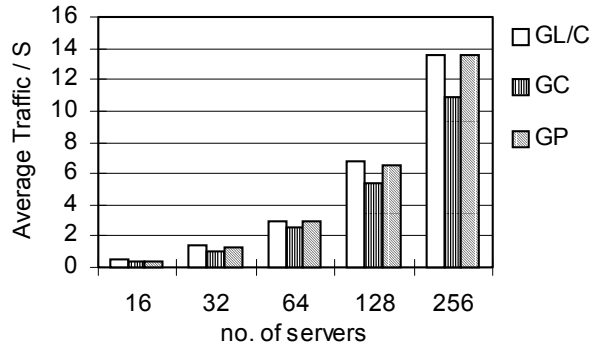

Figure 10. Average traffic with Data Set 1 (C / S = 1/8)

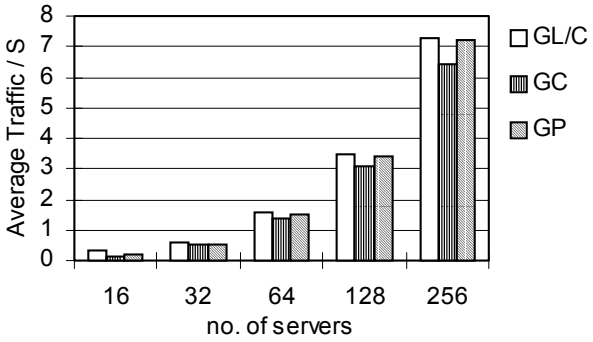

Figure 12. Average traffic with Data Set 2 (C / S = 1/8)

the average traffic each period and total size of Web documents S is computed. In the figures, y-axis represents this ratio.

We can see from Figure 9 and Figure 10 that 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 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. It is easy to understand that GC, which cares most about communication cost, incurs the least cost. We find that when the storage capacity is large, the traffic caused by GL/C and GP is almost the same.

As the number of nodes increases, the total storage space increases, therefore, the traffic in the system increases. From Figure 10 and Figure 11, we can see that GC still causes least traffic, and the traffic caused by GL/C and GP get closer as the number of nodes increases.

The actual time needed to move the documents

$$t \approx \text{total bytes} /(B * M)$$

where M is the number of servers and B is the bandwidth between any two server nodes. Therefore, if we assume that the bandwidth is 1 MB/s and total size of the documents is 1G, moving documents would take no more than several minutes. Since during this period, the DWS system can continue to serve requests with documents not in the move, its performance would not be substantially affected.

In the figures, DS's average traffic may be smaller than that of our scheme. But since its period is much shorter, its total traffic is actually larger than ours. This may be because that it frequently replicates a document and then revokes it.

From the simulation results, we see that our document distribution scheme can achieve better load balancing in a geographically DWS system and generate less traffic than the dynamic scheme. Among the document distribution algorithms, GC's load balancing performance is not as good as that of GP and GL/C. However, GC generates the least internal traffic. GL/C needs shortest computing time. Its load balancing performance is best in most cases and only generates a little more traffic than GP. When the number of server nodes is large, however, GL/C performs much worse than GP. GP balances the load well but it requires more computation than the others. A suitable algorithm can be chosen according to the practical situation of a geographically DWS system.

5. Related Work

Much research work has been done on ways to keep a balanced load in geographically DWS systems.

Various DNS based scheduling techniques have been proposed. The NCSA scalable web server depends on round-robin DNS to dispatch requests [11], while [9] found that the DNS policies combined with a simple feedback alarm mechanism could effectively avoid overloading the server nodes. Adaptive TTL algorithm [8] was proposed to address the uneven client request distribution and heterogeneity of server capacities. The main problem with these techniques is that DNS only has a limited control on the requests reaching the Web servers, due to the caching of IP address in intermediate DNS servers and client caches.

The content-aware requests distribution strategy LARD [16] makes it possible to balance the load among the server nodes through partitioning the Web documents among the server nodes, with or without replication. DCWS [5] makes use of a graph-based Web document-partitioning algorithm. Each document resides on its home server at first and can be migrated to a co-op server for load balancing reason. To redirect client requests from the home server to the co-op server, all hyperlinks pointing to the document are modified. However, 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.

DC-Apache [12] is similar to DCWS, except that documents are replicated instead of migrated among the server nodes. Each document has a home server that keeps its original copy. Every time the number or locations of copies of a document change, the document's home server needs to regenerate all the hyperlinks pointing to this document based on global load information. This operation requires substantial computation.

Riska et. al. observed that directing tasks of similar size to the same server reduces the slowdown in a web server and proposed a load balancing policy ADAPTLOAD [18] which partitions the documents among the server nodes according to their sizes. How to effectively choose parameters of the policy still needs more work.

In RobustWeb [14], each document has the same number of replicas. The replicas are placed on the server nodes based on past access pattern to equalize the servers load. Multiple copies of a document may have different weights of redirection, and the requests are assigned to them in a weighted round-robin way. Instead of moving documents like we do, in RobustWeb, only the weights of the copies are computed periodically. When the access pattern change dramatically, however, it's difficult to maintain load balancing using this method.

Ng at el. [15] included the prefetching feature in their EWS system. In this system, documents that are always accessed together are grouped and placed on the same server node. Only the first request of a session has to go through the redirection server, thus cutting down on the redirection overhead. Load balancing is achieved by using a revised document placement algorithm of the one used in RobustWeb. Our work can be considered a derivative from theirs by taking disk utilization and communication cost into account. The algorithms we propose in this paper can be deployed in EWS.

Recently there has been an increase in interest in replica placement in Content Delivery Networks (CDN) that offer hosting services to Web content providers [2,10,17]. Although the problem formulation in CDN is very similar to ours, it mainly focuses on minimizing clients' latency or total bandwidth consumption, and not balancing the load among the servers.

6. Conclusion and Future Work

In this paper, we study how to replicate and distribute the documents in a geographically DWS system to achieve load balancing. In contrast with existing work, we also take the communication cost caused by distributing the replicas into consideration. We prove that even without load balancing constraint, minimizing this cost in homogeneous DWS systems is NP-hard.

We propose a document distribution scheme which periodically replicates the documents and distributes the replicas. In this scheme, we utilize the concept of "density" of a document to decide number of replicas for each document. Several distribution algorithms are proposed and they use the available information from different perspectives to reduce internal traffic of the geographically DWS system. Our scheme is compared with a dynamic scheme using real log files. The results show that our scheme could balance the load in a DWS system more efficiently during run-time and causes less traffic. We also discuss the difference between the distribution algorithms and the situations for which they are suitable.

Our next step is to incorporate geographical information into our document distribution scheme. We aim at a geographically DWS system which would automatically copy a document to a location where it is in most demand, while maintaining load balancing and minimizing communication cost. Such a geographically DWS system would reduce access latencies and be most suitable for Web sites where different parts of the content are of interest to people from different geographical locations.

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*, Boston, USA, December 1995, pp 119-133.

[2] A. Aggarwal and M. Rabinovich, "Performance of Dynamic Replication Schemes for an Internet Hosting Service". *Technical Report*, AT&T Labs, October 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*, Philadelphia, USA, May 1996, pp.160 – 169.

[4] 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*, San Diego, USA, March 1999, pp. 159-169.

[5] S.M. Baker, and B. Moon, "Scalable Web Server Design for Distributed Data Management". In *Proc. of 15th International Conference on Data Engineering*, Sydney, Australia, March 1999, pp. 96.

[6] V. Cardellini, M. Colajanni and P.S. Yu, "Dynamic Load Balancing on Web-Server Systems". In *IEEE Internet Computing*, vol. 3, No. 3, May/June 1999, pp 28-39.

[7] L.C. Chen and H.A. Choi, "Approximation Algorithms for Data Distribution with Load Balancing of Web Servers". In *Proc. of the 3rd IEEE International Conference on Cluster Computing (CLUSTER'01)*, Newport Beach, USA, October 2001.

[8] M. Colajanni and P.S. Yu, "Adaptive TTL Schemes for Load Balancing of Distributed Web Servers". In *ACM Sigmetrics Performance Evaluation Review*, 25(2):36--42, September 1997.

[9] M. Colajanni, P.S. Yu, and D.M. Dias, "Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems". In *IEEE Trans. Parallel and Distributed Systems*, vol. 9, No. 6, June 1998, pp. 585-600.

[10] 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.

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

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

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

[14] 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*, Durham, USA, October 1997, pp. 97-106.

[15] C.P. Ng and C.L. Wang, "Document Distribution Algorithm for Load Balancing on an Extensible Web Server Architecture". In *Proc. of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, May 2001.

[16] V.S. 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*, San Jose, USA, October 1998, pp. 205-216.

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

[18] A. Riska, W. Sun, E. Smirni, and G. Ciardo, "ADATPTLOAD: effective balancing in clustered web servers under transient load conditions". In *Proc. of 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, July 2002.

[19] C.S. Yang and M.Y. Luo, "A Content Placement and Management System for Distributed Web-Server Systems". In *Proc. of 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, Taipei, Taiwan, April 2000.

[20] ClarkNet-HTTP, http://ita.ee.lbl.gov/contrib/ClarkNet-HTTP.html

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

[22] Seagate ST360020A, http://www.seagate.com