Object Grouping and Replication Algorithms for Word Wide Web

A. Mahmood
Department of Computer Science
University of Bahrain
Kingdom of Bahrain
E-mail: amahmood@itc.uob.bh

Keywords: Data mining, document clustering, object replication, Web, distributed web-server system, document replication.

Received: December 4, 2004

This paper presents an algorithm to group correlated objects that are most likely to be requested by a client in a single session. Based on these groups, a centralized algorithm that determines the placements of objects to a cluster of web-servers is proposed to minimize latency. Due to the dynamic nature of the Internet traffic and the rapid changes in the access pattern of the World-Wide Web, we also propose a distributed algorithm where each site relies on some collected information to decide what object should be replicated at that site. The performance of the proposed algorithms is evaluated through a simulation study.

Povzetek: Grupiranje objektov na spletu.

1 Introduction

An ever-increasing popularity of Word Wide Web has brought a huge increase in traffic to popular web sites. As a result, users of such web sites often experience poor response time or denial of a service (time-out error) if the supporting web-servers are not powerful enough. Since these sites have a competitive motivation to offer better service to their clients, the system administrators are constantly faced with the need to scale up site capacity. There are generally two different approaches to achieving this [1]. The first approach is to use powerful server machines with advanced hardware support and optimized server software. Unfortunately, this approach is expensive and complicated one and the issue of scalability and performance may persist with ever increasing user demand.

The second approach, which is more flexible and sustainable, is to use distributed and a highly interconnected information system or distributed web server system (DWS). A distributed web server system is any architecture of multiple stand-alone web server hosts that are interconnected together and act as a logically single server [2]. A DWS is not only cost effective and more robust against hardware failure but it is also easily scalable to meet increased traffic by adding additional servers when required. In such systems, an object (a web page, a file, etc.) is requested from various geographically distributed. As the DWS spreads over a MAN or WAN, movement of documents between server nodes in an expensive operation [1]. Maintaining multiple copies of objects at various locations in DWS is an approach for improving system performance (e.g. latency, throughput, availability, hop counts, link cost, and delay etc.) [1-3].

Web caching attempts to reduce network latency and traffic by storing commonly requested documents as close to the clients as possible. Since, web caching is not based on the users' access patterns, the maximum cache hit ratio achievable by any caching algorithm is bounded under 40% to 50% [4].

A Proactive web server system, on the other hand, can decide where to place copies of a document in a distributed web server system. In most existing DWS systems, each server keeps the entire set of web documents managed by the system. Incoming requests are distributed to the web server nodes via DNS servers [5-7]. Although such systems are simple to implement but they could easily result in uneven load among the server nodes due to caching of IP addresses on the client side.

To achieve better load balancing as well as to avoid disk wastage, one can replicate part of the documents on multiple server nodes and requests can be distributed to achieve better performance [8-10]. However, some rules and algorithms are then needed to determine number of replicas of each document/object and their optimal locations in a DWS. Choosing the right number of replicas and their locations can significantly reduce web access delays and network congestion. In addition, it can reduce the server load which may be critical during peak time. Many popular web sites have already employed

replicated server approach which reflects upon the popularity of this method [11].

Choosing the right number of replicas and their location is a non-trivial and non-intuitive exercise. It has been shown that deciding how many replicas to create and where to place them to meat a performance goal is an NP-hard problem [12,13]. Therefore, all the replica placement approaches proposed in the literature are heuristics that are designed for certain systems and work loads.

This paper proposes a suit of algorithms for replica placement in a Web environment. The first two algorithms are centralized in nature and third is a distributed one. For distribution of requests, we take into account site proximity and access cost. A detailed formulation of the cost models and constraints is presented. Since most of the requests in web environment are read requests, our formulation is in the context of read-only requests.

The rest of the paper is organized as follow: Section 2 reviews some existing work related to object replication in the web. Section 3 describes the system model, centralized and distributed replications models and the cost function. Section 4 presents an algorithm to cluster highly correlated objects in a web environment. Section 5 presents a centralized and a distributed algorithm for object replication. Section 6 presents the simulation results and section 7 concludes the paper.

2 Related Work

The problem of replica placement in communication networks have been extensively studied in the area of file allocation problem (FAP) [14,15] and distributed database allocation problem (DAP) [16,17]. Both FAP and DAP are modeled as a 0-1 optimization problem and solved using various heuristics, such as knapsack solution [18], branch-and-bound [19], and network flow algorithms [20]. An outdated but useful survey of work related to FAP can be found in [14]. Most of the previous work on FAP and DAP is based on the assumption that access patterns are known a priori and remain unchanged. Some solutions for dynamic environment were also proposed [21-23]. Kwok et al. [24] and Bisdikian Patel [25] 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.

Another important data replication problem exists in Content Delivery Networks (CDN). Unlike FAP and DAP, in a CDN, a unit of replication/allocation is the set of documents in a website that has registered for some global web hosting service. In [26], the replica placement problem in CDN is formulated as an uncapacitated minimum *K*-median problem. In [27], different heuristics were proposed based on this *K*-median formulation to reduce network bandwidth consumption. The authors of [28] take storage constraint into consideration and reduce the knapsack problem to replica placement problem in CDNs. Li [11] proposed a suit of algorithms for determining the location of replica servers within a network. The objective of this paper is not to determine the placement of objects themselves but to determine the locations of multiple servers within a network such that the product of distance between nodes and the traffic traversing the path is minimized.

Wolfson et al. [29] proposed an adaptive data replication algorithm which can dynamically replicate objects to minimize the network traffic due to “read” and “write” operations. The proposed algorithm works on a logical tree structure and requires that communication traverses along the paths of the tree. They showed that the dynamic replication leads to convergence of the set of nodes that replicate the object. It, however, does not consider the issue of multiple object replications. Further, given that most objects in the Internet do not require “write” operation, the cost function based on “read” and “write” operations might not be ideal for such an environment.

Bestavros [30] considered the problem of replicating contents of multiple web sites at a given location. The problem was formulated as a constraint-maximization problem and the solution was obtained using Lagrange multiplier theorem. However, the solution does not address the issue of selecting multiple locations through the network to do replication. In [31], the authors have studied the page migration problem and presented a deterministic algorithm for deciding on where to migrate pages in order to minimize its access and migration costs. This study, however, deals only with page migration assuming that the network has k copies of a page. In addition, it does not address the problem of adding and deleting replicas to the system and presents no special algorithm for replica selection. It only assumes that the reads are done only from the nearest replica.

Tensakhti et al. [13] present two greedy algorithms, a static and a dynamic one, for replicating objects in a network of web servers arranged in a tree-like structure. The static algorithm assumes that there is a central server that has a copy of each object and then a central node determines the number and location of replication to minimize a cost function. The dynamic version of the algorithm relies on the usage statistics collected at each server node. A test is performed periodically at each site holding replicas to decide whether there should be any deletion of existing replicas, creation of new replicas, or migration of existing replicas. Optimal place of replica in trees has also been studied by Kalpakis at el. [3]. They considered the problem of placing copies of objects in a tree network in order to minimize the cost of serving read and write requests to objects when the tree nodes have limited storage and the number of copies permitted is limited. They proposed a dynamic programming algorithm for finding optimal placement of replicas.

The problem of documents replication in extendable geographically distributed web server systems is addressed by Zhuo et al [1]. They proposed four heuristics to determine the placement of replica in a network. In addition, they presented an algorithm that determines the number of copies of each documents to be replicated depending on its usage and size. In [32] the authors also proposed to replicate a group of related documents as a unit instead of treating each document as a replication unit. They also presented an algorithm to determine the group of documents that have high cohesion, that is, they are generally accessed together by a client in a single session.

Xu el al. [33] discussed the problems of replication proxy placement in a tree and data replication placement on the installed proxies given that maximum *M* proxies are allowed. The authors proposed algorithms to find number of proxies needed, where to install them and the placement of replicas on the installed proxies to minimize the total data transfer cost in the network. Karlsson et al. [34] developed a common framework for the evaluation of replica placement algorithms.

Heddaya and Mirdad [35] have presented a dynamic replication protocol for the web, referred to as the Web Wave. It is a distributed protocol that places cache copies of immutable documents on the routing tree that connects the cached documents home site to its clients, thus enabling requests to stumble on cache copies *en route* to the home site. This algorithm, however, burdens the routers with the task of maintaining replica locations and interpreting requests for Web objects. Sayal el al. [36] have proposed selection algorithms for replicated Web sites, which allow clients to select one of the replicated sites which is close to them. However, they do not address the replica placement problem itself. In [37], the author has surveyed distributed data management problems including distributed paging, file allocation, and file migration.

3 The System Models

A replicated Web consists of many sites interconnected by a communication network. A unit of data to be replicated is referred as an object. Objects are replicated on a number of sites. The objects are managed by a group of processes called replicas, executing at replica sites. We assume that the network topology can be represented by a graph $G(V, E)$, in which $N = |V|$ is the number of nodes or vertices, and $|E|$ denotes the number of edges (links). Each node in the graph corresponds to a router, a switch or a web site. We assume that out of those *N* nodes there are *n* web servers as the information provider. Associated with every node $v \in V$ is a set of nonnegative weights and each of the weights is associated with one particular web server. This weight can represent the traffic traversing this node *v* and going to web server *i* ($i = 1,2,\ldots,n$). This traffic includes the web access traffic generated at the local site that node *v* is responsible for and, also, the traffic that passes through this it on its way to a target web server. Associated with every edge is a nonnegative distance (which can be interpreted as latency, link cost, or hop count, etc.).

A client initiates a read operation for an object *k* by sending a read request for object *k*. The request goes through a sequence of hosts via their attached routers to the server that can serve the request. The sequence of nodes that a read request goes through is called a routing path, denoted by π. The requests are routed up the tree to the home site (i.e. root of the tree). Note that a route from a client to a site forms a routing tree along which document requests must follow. Focusing on a particular sever *i*, the access traffic from all nodes leading to a server can be best represented by a tree structure if the transient routing loop is ignored [11,13,29]. Therefore, for each web server *i*, a spanning tree T_i can be constructed rooted at *i*. Hence, *m* spanning trees rooted at *m* web servers represent the entire network. The spanning tree T_i rooted at a site *i* is formed by the clients that request objects from site *i* and the processors (clients) that are in the path π of the requests from clients to access object *k* at site *i*.

3.1 The Object Replication Models

In this paper, we consider two object replication models: centralized and distributed. In the centralized model, each read request for an object is executed at only one of the replicas, the best replica. If $\aleph_k$ is the set of sites that have a replica of object *k* and C_k^{i,LC_k^i} denotes the cost of accessing object *k* at site *i* from the least cost site (denoted by LC_k^i), then

$LC_k^i = i$, if a replica of *k* is locally available at *i*

$LC_k^i = j$ such that $C_k^{i,j}$ is minimum over all $j \in \aleph_k$, otherwise

That is, for a given request for an object *k* at site *i*, if there is a local replica available, then the request is serviced locally incurring a cost $C_k^{i,i}$, otherwise the request is sent to site *j* having a replica of object *k* with the least access cost.

In the centralized model, there is a central arbitrator that decides on the number of replicas and their placement based on the statistics collected at each site. Upon determining the placement of replicas for each object, the central arbitrator re-configures the system by adding and/or removing replicas according to the new placement determined by the arbitrator. The location of each replica is broadcasted to all the sites. In addition each site *i* keeps the following information:

LC_k^i: The least cost site to *i* that has a replica of object *k*.

$C_k^{i,j}$: The cost of accessing object *k* at site *i* from site *j* on π.

$f_k^{i,j}$: The access frequency of object *k* at site *i* from site *j* on π.

$\aleph_k$: The set of sites that have a replica of object *k*

In the distributed model, there is no central arbitrator. Similar to centralized model, for a given request for an object *k* at site *i*, if there is a local replica available, then the request is serviced locally incurring a cost $C_k^{i,i}$, otherwise the request is sent to site *j* having a replica of object *k* with the least access cost. After every time period *T*, each site makes the decision about acquiring or deleting a copy of an object based on the local statistics.

3.2. The Cost Model

Determining an optimal replication involves generating new allocations and determining their goodness. The evaluation is done in terms of an objective function subject to system constraints. The designation of an objective function reflects the view of goodness of object replication with respect to system design goals. It is not feasible to completely describe a system with just one objective function; instead the objective function should only capture the critical aspects of the system design. Also, the form and the parameters of the objective function should be proper. That is, if the objective function indicates that an allocation is better than the other one then the actual measurements should concur. Keeping in mind these considerations, we develop the objective function for object replication problem as follow:

Suppose that the vertices of *G* issue read requests for an object and copies of that object can be stored at multiple vertices of G. Let there are total *n* sites (web servers) and *m* objects. Let $f_k^{i,j}$ is the number of read requests for a certain period of time *t* issued at site *i* for object *k* to site *j* on π. Given a request for an object *k* at site *i*, if there is a local replica available, then the request is serviced locally with a cost $C_k^{i,i}$, otherwise the request is sent to site *j* having a least access cost replica of object *k* with a cost C_k^{i,LC_k^i} as explained earlier. If *X* is an $n \times m$ matrix whose entry $x_{ik} = 1$ if object *k* is stored at site *i* and $x_{ik} = 0$ otherwise, then the cost of serving requests for object *k* $(1 \le k \le m)$ at site *i* $(1 \le i \le n)$ is given by

$$TC_k^i = (1 - x_{ik}) f_k^{i,LC_k^i} C_k^{i,LC_k^i} + x_{ik} f_k^{i,i} C_k^{i,i} \quad (1)$$

The cost of serving requests for all the objects at site *i* is:

$$TC = \sum_{k=1}^{k=m} TC_k^i = \sum_{k=1}^{k=m} \left[(1 - x_{ik}) f_k^{i,LC_k^i} C_k^{i,LC_k^i} + x_{ik} f_k^{i,i} C_k^{i,i} \right] \quad (2)$$

Hence, the cumulative cost over the whole network for all the objects can be written as:

$$CC(X) = \sum_{i=1}^{n} \sum_{k=1}^{m} \left[(1 - x_{ik}) \sum_{LC_k^i} f_k^{i,LC_k^i} C_k^{i,LC_k^i} + x_{ik} f_k^{i,i} C_k^{i,i} \right] \quad (3)$$

Now, the replica placement problem can be defined as a 0-1 decision problem to find *X* that minimizes (3) under certain constraints. That is, we want to

$$\text{minimize } CC(X) = \min \sum_{i=1}^{n} \sum_{k=1}^{m} \left[(1 - x_{ik}) \sum_{LC_k^i} f_k^{i,LC_k^i} C_k^{i,LC_k^i} + x_{ik} f_k^{i,i} C_k^{i,i} \right] \quad (4)$$

Subject to

$$\sum_{i=1}^{n} x_{ik} \ge 1 \text{ for all } 1 \le k \le m \quad (5)$$

$$\sum_{k=1}^{m} x_{ik} s_k \le TS_i \text{ for all } 1 \le i \le m \quad (6)$$

$$x_{ik} \in \{0,1\}, \text{ for all } i, j \quad (7)$$

The first constraint specifies that each object should have at least one copy. If s_k denotes size of object *k* and TS_i is the total storage capacity of site *i* then the second constraint specifies that the total size of all the objects replicated at node *i* should not exceed its storage capacity.

4 Object Grouping

Almost all the proposed object/document placement and replication algorithms for web on web servers decide about the placement/replication of a complete web site or individual objects comprising a web site. Both of these methods are not realist. It has been shown in various studies that each group of users generally accesses a subset of related pages during a single session. Therefore, it is logical to group documents which have high correlation – that is, the documents that are very likely to be requested by a client in a single session. This would reduce the HTTP redirection throughout a HTTP session and hence improve the response time. Each group then can be replicated on web servers as a unit hence reducing the search space.

In this section, we propose an algorithm to group objects that are highly correlated in the sense that they have high

probability of being accessed by a client in a single session. The proposed algorithm is an adaptation of the algorithm proposed in [38]. The major difference is that the algorithm in [38] produces non-overlapping groups, that is, each document is placed in a single group but the proposed algorithm may include an object in more than one group. This is particularly important since different users may request for different correlated objects during each session. Also, we use multiple sessions, instead of a single session, originating from a client to obtain object groups for the reasons explained.

The proposed algorithm groups the objects into correlated object clusters based on the user access patterns which are stored in the system access log files. An access log file typically includes the time of request, the URL requested, and the machine from which the request originated (i.e. IP address of the machine). Below, we explain major steps of the algorithm.

1. First the log file is processed and divided into sessions where a session is a chronological sequence of document requests from a particular machine in a single session. We assume that each session spans over a finite amount of time. It is important to note that the log file may have multiple sessions for the same user. This gives a better picture of the usage pattern of a user. Also, note that we have to make sure that each request from a machine should be recorded in the log file to obtain an accurate access pattern of users. This can be accomplished by disabling caching, that is, every page sent to a machine contains a header saying that it expires immediately and hence browsers should load a new copy every time a user views that page.
2. In step 2, we create a correlation matrix. The correlation between two objects O_1 and O_2 is the probability that they are accessed in the same user session. To calculate correlation between O_1 and O_2, we scan the log file and count the number of distinct sessions in which O_1 was accessed after O_2 ($count(O_1,O_2)$) and calculate $p(O_1|O_2)=count(O_1,O_2)/s(O_1)$, where $p(O_1|O_2)$ is the probability of a client visiting O_1 if it has already visited O_2 and $s(O_1)$ is the number of sessions in which O_1 was accessed by a client. Similarly, we compute $p(O_2|O_1)=count(O_2, O_1)/s(O_2)$, where $p(O_2|O_1)$ is the probability of O_2 being accessed after O_1 in a session, $count(O_2, O_1)$ is the number of sessions in which O_2 is accessed after O_1 and $s(O_2)$ is the total number of sessions in which O_2 is assessed. The correlation between O_1 and O_2 is the $min(p(O_1|O_2), p(O_2|O_1))$ to avoid mistaking a asymmetric relationship for a true case of high correlation.
3. At step three, we first create a graph corresponding to correlation matrix in which each object is a vertex and each non-zero cell of the correlation matrix is mapped to an edge. The length of an edge is equal to the correlation probability between two vertices. The edges with a small value are removed from the graph. We then group documents by identifying cliques in the graph. A clique is a subgraph in which each pair of vertices has an edge between them. The algorithm to identify cliques if given in figure 1. The algorithm always starts with a pair of vertices that have the longest edge between them. Both of these vertices are included in the group and edge is removed. Then we examine the rest of the vertices that have not been included in the group and select the next best vertex (a vertex with the highest edge value) that is connected to the vertices already included in the group and include it in the group. In this way we choose the objects that are highly correlated. The size of the clique is bounded by the longest session of its members since there is no need of including an object to a group if it is not accessed in the longest session. Each vertex that is not included in any of groups is included in a separate group having that vertex as its only member.

```
R = {vertices connected to at least one edge}
while (R ≠ φ) {
    Find the longest edge in R with vertices O1 and O2
    V = { O1 , O2}
    G = R \ V, C = φ
    l=maximum size of V
    while ( |V| ≤ l ) {
        for (each vertex O in G) {
            if (O is connected to all vertices in V ){
                Record the shortest edge between o and vertices in V
                Add O to V
            }
        }
        if (C ≠ φ ) {
            Choose the vertex O whose shortest edge to V is longest
            Add O to V
            Delete O from G and R
            C = φ
            l=l+1
        }
        else {
            delete O1 and O2 O1 and O2 from G and R
            break
        }
}
Construct a group for each remaining vertex
```

Figure 1.Object grouping algorithm

5 Object Placement and Replication Algorithms

The replica placement problem described in the previous section reduces to finding 0-1 assignment of the matrix X that minimizes the cost function subject to a set of constraints. The time complexity of this type of problems is exponential. In the next section, we present our proposed centralized object replication algorithms.

5.1 Centralized Greedy Algorithm

Our first algorithm is a polynomial time greedy algorithm that is executed at a central server and decides the placements of replicas for each object. The algorithm proceeds as follows: First all the objects groups are organized in descending value of their density to make sure that the objects that are heavily accessed are assigned to the best server. For each object, we determine the number of replicas that should be assigned to various servers using the algorithm proposed in [32] (R_k denotes the number of replica each object k should have). The first object in a group is assigned to most suitable server and then all the other objects in the same group are allocated to the same server if it has enough capacity. The idea is that the documents in the same group have high probability of being accessed in the same session by a client; therefore, keeping them together will improve the response time. If an object cannot be assigned to the same server then we find a server with minimum access cost and assigned the object on that server. After a copy of an object is assigned, then we assign the remaining replica of each object to best servers not having a copy of that object and have the capacity for that object. The complete algorithm is given in figure 2.

5.2. Distributed Object Replication Algorithm

The algorithm presented in the previous section are centralized in the sense that a central arbitrator collects all the necessary statistics, determines the placement of the objects, and reconfigures the system in accordance with the newly determined allocation. This might involve removing/deleting replicas and adding or migrating replicas by the central arbitrator. However, in the distributed model, there is no central arbitrator. Rather, each site determines for itself which objects it should add/remove based on the current replica placement and

```
Group objects using object group algorithm
Arrange object groups in descending order of their density
Arrange objects in each group in descending order of their density
Determine the number of replicas for each object
for (k=1; k<= no_of_objects; k++) replica_assigned_k=0
for g = 1 to no_of_groups {
        while (G_i ≠ φ ) {
            k = first_object_in _G_i
            A = k // set of objects allocated to j
            if (k has not been allocated) {
                    j = site with minimum value of (2) such that no constraint is violated if a
                     replica of k is allocated to j
                     Allocate k at j
                    replica_assigned_k = replica_assigned_k + 1
            }
            G_i = G_i  - k
            while (j has capacity and G_i ≠ φ and ) {
                  k = first_object_in _G_i
                  Allocate k at j
                  replica_assigned_k = replica_assigned_k + 1
                  G_i = G_i  - k
                  A= A ∪ k
            }
           for ( each k in A ) {
           for (r= replica_assigned_k;  r ≤ R_k ; r++) {
                    Find a site i not having a replica of k and has minimum value of C_k^{j,i} and if a
                    replica of k is assigned at j and no constraint is violated
                     Assigned k at j
                    replica_assigned_k = replica_assigned_k + 1
            } }
    }
}
```

Figure 2: Proposed replication algorithm (algorithm 1)

for (each object $k \in K_i$) {

If (server *i* is the only server having a replica of object *k*)

$profit_k$ = *a_max_number*

else

$profit_k = \left|(TC_k^i - f_k^{i,i} C_k^{i,i}) / s_k\right|$

// TC_k^i then the cost of serving requests for object *k*

// at site *i* from the least cost site $j \neq i$

}

for (each object $k \in \xi_i$) {

$profit_k = \left|(TC_k^i - f_k^{i,i} C_k^{i,i}) / s_k - C_k^{i,LC_k^i} / s_k\right|$

}

Sort all the objects in $K_i \cup \xi_i$ in descending order of their profit values.

Replicate the objects from the sorted list one by one until there is space available on server *i*.

Figure 4. The proposed distributed algorithm (algorithm 2)

locally collected statistics as described in section 3.1.

Our proposed distributed object replication algorithm is a polynomial time greedy algorithm where each site keeps the replicas of those objects that are locally evaluated to be the best replicas. Assume that *X* (an $n \times m$ matrix) represents the current object replication. Initially *X* can be determined by using the algorithm proposed in the previous section. If K_i and ξ_i is the set of objects that are replicated at site *i* and the set of objects that are not replicated at site *i* respectively then each site, after every time period *t*, determines which objects it should add/remove based on the current replica placement and locally collected statistics using the following proposed algorithm. The algorithm first calculates the unit loss/profit of removing the local replicas and then the unit profit of having replicas of those objects which are not available locally. It then sorts all the objects in descending values of their profit and replicates top *n* objects which it can accommodate without violating the constraints. The complete algorithm is given in figure 4.

6 Experimental Results

This section presents some performance measures obtained by simulation of the proposed algorithms. We have run several simulation trials. In each simulation run, we model the web as a set of trees having 100-600 sites. The total objects to be replicated were 2000 in all the simulation runs. We use different object sizes which follows a normal distribution. The average object size is taken as 10 KB and maximum size was taken as 100KB. About 64% objects sizes were in the range of 2KB and 16KB. The storage capacity of a server was set randomly in such a way that total storage of all the servers was enough to hold at least one copy of each object at one of the servers. In each trial, we run the replica placement algorithms for 200,000 requests for different objects. We created log files by generating requests for objects for multiple sessions. This log file was used to group objects. The same log file was used by the proposed algorithms to collect various statistics.

During a simulation run, each site keeps a count *c* of the total number of requests it receives for an object. The latencies are updated periodically for each replica using the formula $T = 1/(\mu - \lambda)$ where λ is the average arrival rate and μ is the average service time. Exponential service time is assumed with an average service rate of 100 transactions/second. The value of *T* is propagated to the clients in the shortest path spanning tree. The cost (latency) at different sites is computed as follows: At the replica site, the average arrival rate is computed and the latency $T = 1/(\mu - \lambda)$ is broadcast to all the sites of the tree rooted as this replica. At a site *i* of the tree, the communication cost (set randomly) at the neighboring site *j* from which *T* is propagated is added to *T*. This quality will be the cost of accessing the replica from site *i*. At the end of every 20,000 requests, the mean latency required to service all the 20,000 requests is calculated and used as a performance measure of the simulated algorithms.

We studied the performance of our proposed algorithms and compared it with that of random allocation algorithm [28] and greedy algorithm [13]. The random algorithm stores replicas at randomly selected nodes subject to system constraints. The number of replicas for each object was determined by density algorithm [1]. We pick one replica at a time with uniform probability and one node also with uniform probability; and store that replica at that node. If the node already has a replica of that object or allocation of replica at that node violates any of the constraints then another node is selected randomly until the replica is placed at a node. Since the object placement problem is NP-Complete and hence optimal solution cannot be obtained for large problems in a

reasonable amount of time, the random algorithm provides a good basic on which we can determine how good a heuristic performs than that of a simple random algorithm.

Figure 5 shows the average latency for all the simulation runs for different tree sizes. The figure shows that the average latency decreases for all the algorithms as the number of sites increases in the system. This is because of the fact that as the number of sites increases, more replica of an object can be placed. Also, note that the performance of algorithm 1 and algorithm 2 is comparable demonstrating the effectiveness of the distributed algorithm. The figure 6 shows the average performance of the algorithms for all the system configurations. It is evident that the proposed algorithms perform, on average, better than the greedy and the random algorithm.

To demonstrate how algorithm 2 adapts to the access patterns, we performed a set of experiments. The initial allocation was obtained by randomly placing replicas of each object as explained before. After each 20000 requests, the algorithm is run on each site. We observed the improvement in latency, first by calculating the latency if no reallocation of objects is done and then by allowing the algorithm to adjust the replication using the statistics. The results are shown in figure 7. It is evident from the figure that the algorithm reduces the latency every time it is executed. Initially the improvement is significantly high since the initial allocation was obtained randomly. After a number of runs, the performance of algorithm 2 is comparable with that of algorithm 1.

7 Conclusions

Object replication on a cluster of web servers is a promising technique to achieving better performance. However, one needs to determine the number of replicas of each object and their locations in a distributed web server system. Choosing right number of replicas and their location is a non-trivial problem. In this paper, we presented an object grouping algorithm and two object replication algorithms. The object grouping algorithm groups web objects based on the client access patterns stored in access log file. The documents that are correlated and have high probability of being accessed by a client in a single session are put into the same group so that they can be allocated, preferably on the same server. The first proposed for object replication is a centralized one in the sense that a central site determines the replica placement in a graph to minimize a cost function subject to the capacity constraints of the sites. The second algorithm is a distributed algorithm and hence does not need a central site for determining object placement. Rather, each site collects certain statistics and decisions are made locally at each site on the objects to be stored at the site. Taken each algorithm individually, simulation results show that each algorithm improves the latency of the transactions performed at different sites as the number of sites is increased. A comparison of the proposed algorithms with greedy and random algorithms demonstrates the superiority of the proposed algorithms.

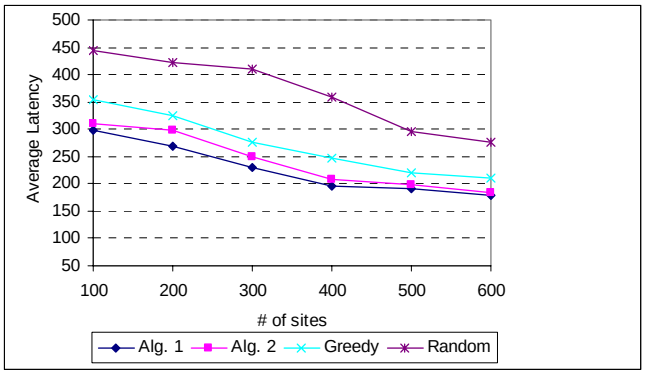

Figure 5. Mean latency for different tree sizes

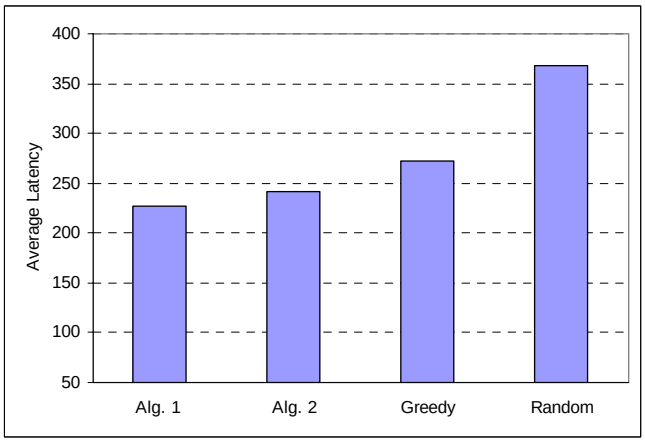

Figure6. Average latency for all simulation runs

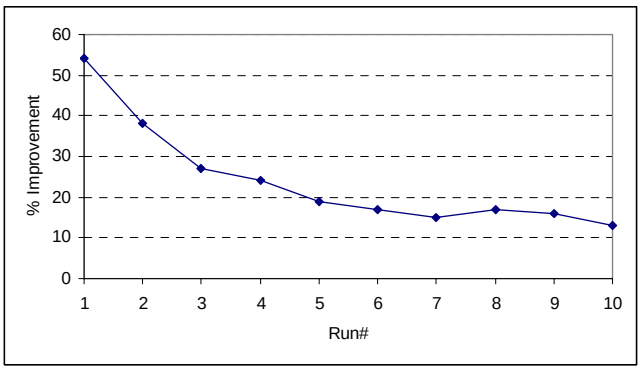

Figure 7. Average % improvement in latency achieved by algorithm 2

References

[1] ZHUO, L., WANG, C-L. and LAU, F. C. M., Document Replication and Distribution in Extensible Geographically Distributed Web Servers, J. of Parallel and Distributed Computing, Vol. 63, 2003, No. 10, pp. 927-944.

[2] PHOHA, V. V., IYENGAR S. S. and KANNAN, R., Faster Web Page Allocation with Neural

Networks. IEEE Internet Computing, Vol. 6, 2002, No. 6, pp. 18-25.

[3] KALPAKIS, K., DASGUPTA, K. and WOLFSON, O., Optimal Placement of Replicas in Trees with Read, Write and Storage Costs. IEEE Trans. On Parallel and Distributed Systems, Vol. 12, 2001, No. 6, pp. 628-637.

[4] ABRAMS, M., STANDRIDGE, C. R., ABDULLA, G., WILLIAMS, S., and FOX, E. A., Caching Proxies: Limitations and Potentials. Proc. 4th International World Wide Web Conference, Boston, Dec. 1995, pp. 119-133.

[5] CARDELLINI, V. COLAJANNI, M., and YU, P. S., Dynamic Load Balancing on Web-Server Systems. IEEE Internet Computing, Vol. 3, 1999, No. 3, pp. 28-39.

[6] COLAJANNI, M., YU, P. S., Analysis of Task Assignment Policies in Scalable Distributed Web Server Systems. IEEE Trans. On Parallel and Distributed Systems, Vol. 9, 1988, No. 6, pp. 585-600.

[7] KWAN, T. T., MCGRATH, R. E. and REED, D. E., NCSA's World Wide Web Server: Design and Performance. IEEE Computer, Vol. 28, 1995, No. 11, pp. 68-74.

[8] BAKER, S. M. and MOON, B., Scalable Web Server Design for Distributed Data Management. Proc. Of 15th Int. Conference on Data Engineering, Sydney, March 1999, pp. 96-110.

[9] LI, Q. Z. and MOON, B., Distributed Cooperative Apache Web Server. Proc. 10th Int. World Wide Web Conference, Hong Kong, May 2001.

[10] RISKA, A. Sun, W., SMIMI, E, and CIARDO, G., ADATPTLOAD: Effective Load Balancing in Clustered Web Servers Under Transient Load Conditions. Proc. 22nd Int. Conf. on Distributed Systems, Austria, July 2002.

[11] LI, B., Content Replication in a Distributed and Controlled Environment. J. of Parallel and Distributed Computing, Vol. 59, 1999, No. 2, pp. 229-251.

[12] KARLSSON, M. and KARAMANOLIS, C., Choosing Replica Placement Heuristics for Wide-Area Systems. International Conference on Distributed Computing Systems (ICDCS) 2004, available at http://www.hpl.hp.com/personal/Magnus_Karlsson.

[13] TENZAKHTI, F., DAY, K. and OLUD-KHAOUA, M., Replication Algorithms for the Word-Wide Web. J. of System Architecture, Vol. 50, 2004, pp. 591-605.

[14] DOWDY, L. and FOSTER, D., Comparative Models of the File Assignment Problem. Computer Surveys, Vol.14, 1982, No. 2, pp. 287-313.

[15] CHU, W. W., Optimal File Allocation in a Multiple Computer System. IEEE Trans. On Computers, Vol. 18, 1969, No. 10, pp. 885-889.

[16] OZSU, M. T. and VALDURIEZ, P., Principles of Distributed Database System. Englewood Cliff, N. J.: Prentice Hall, 1999.

[17] APERS, P. G. M., Data Allocation in Distributed Database Systems. ACM transactions on Database Systems, Vol. 13, 1998, No. 3, pp. 263-304.

[18] CERI, S., MARTELLA, G. and G. PELAGATTI, G., Optimal File Allocation in a Computer Network: A Solution Method Based on Knapsack Problem. Computer Networks, Vol. 6, 1982, No. 11, pp. 345-357.

[19] FISHER, M. K. and HOCHBAUM, D. S., Database Location in Computer Networks. J. ACM, Vol. 27, 1980, No. 10, pp. 718-735.

[20] CHANG, S. K. and LIU, A. C., File Allocation in Distributed Database. Int. J. Computer Information Science. Vol. 11, 1982, pp. 325-340.

[21] AWERBUCH, B., BARTAL, Y. and A. FIAT, A., Competitive Distributed File Allocation. Proc. 25th Annual ACM Symposium on Theory of Computing, Victoria, May 1993, pp. 164-173.

[22] LOIKOPOULOS, T. and AHMED, I., Static and Dynamic Data Replication Algorithms for Fast Information Access in Large Distributed Systems. 20th IEEE conference on Distributed Computing Systems, Taipei, 2000.

[23] GAVISH, B. and SHENG, O. R. L., Dynamic File Migration in Distributed Computer Systems. Comm. of ACM, Vol. 33, 1990, No. 1, pp. 177-189.

[24] KWOK, Y. K., KARLAPALEM, K., AHMED, I. and PUN, N. P., Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems. IEEE J. Selected Areas of Communications, Vol.17, 1996, No. 7, pp. 1332-1348.

[25] BISDIKIAN, C. and PATEL, B., Cost-Based Program Allocation for Distributed Multimedia-On-Demand Systems. IEEE Multimedia, Vol. 3, 1996, No. 3, pp. 62-76.

[26] QIU, L., PADMANABHAM, V. N. and VOELKER, G. M., On the Placement of Web Server Replicas. In Proc. Of 20th IEEE INFOCOM, Anchorage, USA, April 2001, pp. 1587-1596.

[27] RADOSLAVOV, P., GOVINDAN, R. and ESTRIN, D., Topology Informed Internet Replica Placement. Proc. 6th Int. workshop on Web Caching and Content Distribution, Boston, June 2001, Available at http://www.cs.bu.edu/techreports/2001-017-wcw01-proceedings.

[28] KANGASHARJU, J., ROBERTS, J. and ROSS, K. W., Object Replication Strategies in Content Distribution Networks. Computer Communications, Vol. 25, 2002, No. 4, pp. 367-383.

[29] WOLFSON, O. JAJODIA, S. and HUANG, Y., An Adaptive Data Replication Algorithm. ACM Trans. Database Systems, Vol. 22, 1997, No. 2, pp. 255-314.

[30] BESTAVROS, A.: Demand-Based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems. Proc. IEEE Symp. On Parallel and Distributed Processing, 1995, pp. 338-345.

[31] BARTAL, Y., CHARIKAR, M. and INDYK, P., On Page Migration and Other Relaxed Task Systems. Theory of Computer Science, Vol. 281, 2001, No. 1, 2001, pp. 164-173.

[32] ZHUO, L., WANG, C-L., and LAU F. C. M., Document Replication and Distribution in Extensible Geographically Distributed Web Servers. 2002, Available at http://www.cs.hku.hk/~clwang/papers/JPDC-EGDWS-11-2002.pdf

[33] XU, J., LI, B. and LEE, D. L., Placement Problems for Transparent Data Replication Proxy Services. IEEE J. on Selected Areas in Communications, Vol. 20, 2002, No. 7, pp. 1383-1398.

[34] KARLSSON, M., KARAMANOLIS, C. and MAHALINGAM, M., A Framework for Evaluating Replica Placement Algorithms. Tech. Rep. HPL-2002, HP Laboratories, July 2002, http://www.hpl.hp.com/personal/magnus_karlsson.

[35] HEDFDAYA, A. and MIRDAD, S., Web Wave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents. Proc. 17th IEEE int. Conf. On Distributed Computing Systems, 1997, pp. 160-168.

[36] SAYAL, M., BREITBART, Y, SCHEURERMANN, P. and VINGRALEK, R., Selection of Algorithms for Replicated Web Sites. Performance Evaluation Review, Vol. 26, 1998, No. 1, pp. 44-50.

[37] BARTEL, Y., Distributed Paging. Proc. Dagstuhl Workshop On-line Algorithms, 1997, pp. 164-173.

[38] PERKOWITZ, M. and ETZIONI, O., Adaptive Web Sites: Automatically Synthesizing Web Pages. Proc. AAAI'98, 1998, pp. 722-732.