Chapter XI
Object Grouping and Replication on a Distributed Web Server System

Amjad Mahmood
University of Bahrain, Kingdom of Bahrain

Taher S. K. Homeed
University of Bahrain, Kingdom of Bahrain

ABSTRACT
Object replication is a well-known technique to improve performance of a distributed Web server system. This paper first presents an algorithm to group correlated Web objects that are most likely to be requested by a given client in a single session so that they can be replicated together, preferably, on the same server. A centralized object replication algorithm is then proposed to replicate the object groups to a cluster of Web-server system in order to minimize the user perceived latency subject to certain constraints. Due to dynamic nature of the Web contents and users’ access patterns, a distributed object replication algorithm is also proposed where each site locally replicates the object groups based on the local access patterns. The performance of the proposed algorithms is compared with three well-known algorithms and the results are reported. The results demonstrate the superiority of the proposed algorithms.

INTRODUCTION
The phenomenal growth in the World Wide Web (Web) has brought about a huge increase in the traffic to popular Websites. This traffic occasionally reaches the limits of the sites’ capacity, causing servers to be overloaded (Chen, Mohapatra, & Chen, 2001). As a result, end users either experience a poor response time or denial of a service (time-out error) while accessing these sites.
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 the site capacity. There are generally two different approaches to achieving this (Zhuo, Wang, & Lau, 2003). The first approach, generally referred to as **hardware scale-up**, is the use of powerful servers with advanced hardware support and optimized server software. While hardware scale-up relieves short-term pressure, it is neither a cost effective nor a long-term solution, considering the steep growth in clients’ demand curve. Therefore, 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 a distributed Web-server system (DWS). 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 clients. As the DWS spreads over a metropolitan area network (MAN) or wide area network (WAN), movement of documents between server nodes is an expensive operation (Zhuo, Wang, & Lau, 2003). Maintaining multiple copies of objects at various locations in a DWS is an approach for improving system performance, such as latency, throughput, availability, hop counts, link cost, and delay (Kalpakis, Dasgupta, & Wolfson, 2001; Zhuo, Wang, & Lau, 2003).

There are two techniques used in maintaining multiple copies of an object: caching and replication. In Web caching, a copy of an object is temporarily stored at a site that accesses the object. The intermediate sites and proxies also may cache an object when it passes through them en route to its destination site. The objective of Web caching is 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 users’ access patterns, the maximum cache hit ratio achievable by any caching algorithm is bounded under 40-50% (Abrams, Standridge, Abdulla, Williams, & Fox, 1995). In addition, cached data have a time to live (TTL), after which the requests are brought back to the original site. Object replication, on the other hand, stores copies of an object at predetermined locations to achieve a defined performance level. The number of replicas to be created and their locations are determined by users’ access patterns. Therefore, the number of replicas and their locations may change in a well-controlled fashion in response to changes in the access patterns.

In most existing DWS, each server keeps the entire set of Web documents/objects managed by the system. Incoming requests are distributed to the Web server nodes via domain name system (DNS) servers or request dispatchers (Cardelino, Colajanni, & Yu, 1999; Colajanni & Yu, 1988; Kwan, Megath, & Reed, 1995; Baker & Moon, 1999). Although such systems are simple to implement, 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 (Li & Moon, 2001; Karlsson & Karamanolis, 2004; Riska, Sun, Smimi, & Ciardo, 2002). Choosing the right number of replicas and their locations is a nontrivial and nonintuitive exercise. It has been shown that deciding how many replicas to create and where to place them to meet a performance goal is an NP-hard problem (Karlsson & Karamanolis, 2004; Tenzakhti, Day, & Olud-Khaoua, 2004). Therefore, all the replica placement approaches proposed in the literature are heuristics that are designed for certain systems and work loads.

This article proposes two algorithms for replicating objects in a DWS environment. The first algorithm is centralized in the sense that a central site/server determines the allocation of objects. The distributed algorithm (DA), however, does
not require a central server, except for determining the object groups. A detailed formulation of cost model and constraints is presented. Since most of the requests in a Web environment are read requests, our formulation is in the context of read-only requests. We also propose an object grouping algorithm to group objects, which are likely to be accessed in a single session by a client, to improve the efficiency and performance of the proposed replication algorithm. It should be noted that other issues, such as consistency and fault tolerance, that generally arise in distributed systems have not been addressed in this article and have been dealt with elsewhere (Agrawal & Bernstein, 1991; Anderson, Breitbart, Korth, & Wool, 1998; Wolfson, Jajodia, & Huang, 1997). Wolfson et al. (1997) also propose how one copy serializability can be ensured by combining the data replication algorithms with two-phase-locking protocol in the presence of read-write requests.

The article proceeds in the following section with a discussion of the related work. It then describes our system model, followed by the proposed algorithms for grouping of highly correlated objects and object replication in a DWS environment. Subsequent sections present the results of our simulation study and conclusions.

SOME RELATED WORK

The object replication problem presented in this article is an extension of the classical data allocation problem (DAP) and file allocation problem (FAP) (Apers, 1998; Dowdy & Foster 1982). Both FAP and DAP are modeled as a 0-1 optimization problems and solved using various heuristics, such as the knapsack solution (Ceri, Martella, & Pelagatti, 1982) branch-and-bound (Fisher & Hochbaum, 1980), and network-flow algorithms (Chang & Liu, 1982). Most of the previous work on FAP and DAP is based on the assumption that access patterns are known a priori and remain unchanged. However, some solutions for dynamic environments also were proposed (Gavish & Sheng, 1990; Loikopoulos & Ahmed 2000). The work presented in this article differs from both FAP and DAP in a number of ways. First, FAP and a large number of DAP consider that each object (a file, a relation, a fragment of a relation) is read and updated independently, and, hence, each object can be replicated independently. Whereas, we consider the objects that are most likely to be accessed (read) together in a single session as a unit of replication. Apers (1998) addresses the replication of object in the presence of join queries. His algorithm not only finds the sites where the relations should be allocated but also decides where the join operations should be performed. However, in the context of a Web, there is no join operation as such. Rather a user navigates a set of pages following the hyperlinks and, hence, does not require a join operation. In addition, both FAP and DAP deal with the replication and deletion of files in response to a sequence of read and write requests to optimize communication costs. Since most Web requests are read requests, we provide a detailed formulation in the context of read queries only. In addition, our second algorithm dynamically reallocates objects when users’ access patterns change.

The problem of replica placement in communication networks has been extensively studied in the literature. Wolfson et al. (1997) proposed an adaptive data replication algorithm, which can dynamically replicate objects to minimize the network traffic due to “read” and “write” operations. 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” operations, the cost function based on “read” and “write” operations might not be ideal for such an environment.

Bestavros (1995) 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 the Lagrange multiplier theorem. However, the solution does not address the issue of selecting multiple locations through the network to do replication. Tensakhti et al. (2004) present two greedy algorithms, static and dynamic, 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 placement of replicas in trees also has been studied by Kalpakis et al. (2001). 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 document replication in extendable geographically DWSs is addressed by Zhuo et al. (2003). They proposed four heuristics to determine the placement of a replica in a network. In addition, they presented an algorithm that determined the number of copies of each document to be replicated, depending on its usage and size. In Zhuo, Wang, & Lau (2002), 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, Li, and Lee (2002) 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 the 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. Heddaya and Mirdad (1997) 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 \textit{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, Breitbart, Scheurermann, and Vingralek (1998) proposed selection algorithms for replicated Web sites, which allow clients to select one of the replicated sites that is close to them. However, they do not address the replica placement problem itself. Mahmood (2005a, b) proposed a series of algorithms to place objects in a DWS in a tree-like topology.

**THE SYSTEM MODEL**

A replicated Web consists of many sites/servers interconnected by a communication network. A unit of data to be replicated is referred as an \textit{object}. An object can be an XML/HMTL page, an image file, a relation, and so forth. Each object is identified by a unique identifier and may be replicated on a number of sites/server. 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. Associ-
ated 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 it on its way to a target Web server. Associated with every edge is a nonnegative distance, which can be hop count, data transmission rate, or the economic cost of the path between two nodes.

A client initiates a read operation for an object \( k \), by sending a read request for \( k \). 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 \( \pi \). The requests are routed up the tree to the home site (i.e., root of the tree). Focusing on a particular sever \( i \), the access traffic from all nodes leading to a server can be represented best by a tree structure, if the transient routing loop is ignored (Li, 1999; Tenzakhti et al., 2004). Therefore, for each Web server \( i \), a spanning tree \( T_i \), rooted at \( i \), can be constructed. Hence, \( m \) spanning trees rooted at \( m \) Web servers represent the entire network.

**Object Replication Model**

In this article, we consider two object replication models: centralized and distributed. In a centralized model, a central arbitrator determines the number and placement of replicas. Upon determining the placement of replicas for each object, the central arbitrator reconfigures the system by adding and/or removing replicas, according to the new placement determined by the arbitrator. The location of each replica is broadcast to all the sites. In a distributed model, the central arbitrator only performs the global grouping of objects that is broadcast to all the sites. Each site itself periodically determines which object groups should be replicated locally to minimize the locally perceived latency. In both models, each site \( i \) keeps the following information:

- \( C_{ik}^{ij} \): The cost of accessing object \( k \) at site \( i \) from site \( j \).
- \( f_{ik}^{ij} \): The access frequency of object \( k \) at site \( i \) from site \( j \).
- \( N_k \): The set of sites that have a replica of object \( k \).

The traffic frequency, \( f_{ik}^{ij} \), is the number of read requests for a certain period of time \( t \) issued at site \( i \) for object \( k \) to site \( j \). This frequency includes the number of requests issues from site \( i \) and the request for object \( k \) passing through in its way to \( j \). This traffic can easily be monitored and recorded by using the existing technologies.

There are a number of methods that have been proposed to calculate cost (latency) (Tenzakhti et al. 2004). To determine the cost (latency), our preferred algorithm proceeds as follows. Each replica site \( j \) maintains a count \( c \) of the total number of requests it receives in a period of time \( t \). The arrival rate, \( \lambda \), at the replica is given by \( \lambda = c/t \). Assuming that each replica site has an exponential service time with an average service rate of \( \mu \), then the time \( T \) that a request will spend at the replica site (waiting + processing time) is the well-known \( M/M/1 \) queuing result \( T = 1/(\mu - \lambda) \). Periodically, each replica site computes its \( T \) and broadcasts it to all the sites in its tree. Upon receiving this value from site \( j \), site \( i \) would add to it the average latency involved in receiving data from \( j \) and broadcast this new value to its neighbors other than \( j \). The latency will reach at all the sites in a recursive way. The added communication cost (latency) can be obtained by having each site periodically query its neighbors and determining this cost.

**The Cost Model**

Suppose that vertices of \( G(V,E) \) issue read requests for an object and copies of that object can be stored
Object Grouping and Replication on a Distributed Web Server System

at multiple vertices of $G$. Suppose that there are total $n$ sites (Web servers) and $m$ objects. Let $X$ be 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 object $k$ at site $i$ from site $j$, denoted by $TC_k^i$ is given by:

$$TC_k^i = \sum_{j=1}^{n} x_{ik} f_k^{i,j} C_k^{i,j}$$  \hspace{1cm} (1)

The cost of serving requests for all the $m$ objects at site $i$, denoted by $TC_i$, is given by:

$$TC_i = \sum_{k=1}^{m} TC_k^i = \sum_{k=1}^{m} \left[ \sum_{j=1}^{n} x_{ik} f_k^{i,j} C_k^{i,j} \right]$$ \hspace{1cm} (2)

The cumulative cost, $TC$, of serving all the objects over the whole network can be written as:

$$TC(X) = \sum_{i=1}^{n} \sum_{k=1}^{m} \sum_{j=1}^{n} x_{ik} f_k^{i,j} C_k^{i,j}$$ \hspace{1cm} (3)

If $b_k$, $s_k$, $TS_i$, $L_{ik}$, and $P_i$ denote the minimum number of safety copies of object $k$, size of object $k$, total storage capacity at site $i$, processing load of object $k$, and total processing capacity of site $i$, respectively, then the replica placement problem can be defined as a 0-1 decision problem to find $X$ that minimizes Equation 3, subject to storage capacity, processing capacity, and minimum copy constraints. That is, we want to:

$$\min TC(X) = \min \sum_{i=1}^{n} \sum_{k=1}^{m} \sum_{j=1}^{n} x_{ik} f_k^{i,j} C_k^{i,j}$$ \hspace{1cm} (4)

Subject to

$$\sum_{i=1}^{n} x_{ik} \geq b_k \text{ for all } 1 \leq k \leq m; \hspace{1cm} (5)$$

$$\sum_{k=1}^{m} x_{ik} s_k \leq TS_i \text{ for all } 1 \leq i \leq n; \hspace{1cm} (6)$$

$$\sum_{k=1}^{m} x_{ik} L_{ik} < P_i \text{ for all } 1 \leq i \leq n; \hspace{1cm} (7)$$

$$x_{ik} \in \{0, 1\}, \text{ for all } i, j \hspace{1cm} (8)$$

In Equation 5, the minimum number of safety copies for object $k$ should be equal or greater than 1 (i.e., $b_k \geq 1$). This is necessary in case some failure of the servers occurs, and/or we want a different minimum number of copies for each object. Note that each object should have at least one copy in the network. The second constraint specifies that the total size of all the objects replicated at node $i$ should not exceed its storage capacity. The third constraint specifies that the processing load brought by all the objects assigned at node $i$ should not exceed the total processing capacity of a node.

**OBJECT GROUPING**

Almost all the proposed object/document placement and replication algorithms for Web servers decide the placement/replication of a complete Web site or individual objects comprising a Web site. Both of these methods are not realistic. It has been shown in various studies that each group of users generally accesses a subsets of related pages during a single session. Therefore, it is logical to group documents that 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 an 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
Object Grouping and Replication on a Distributed Web Server System

Figure 1. Object grouping algorithm

Step 1. Process the log file (as explained below)
Step 2. Create a correlation matrix (as explained below)
Step 3. Create a clique using:

\[
R = \{ \text{vertices connected to at least one edge} \}
\]

while \((R \neq \emptyset)\) {

- Find the longest edge in \(R\) with vertices \(O_1\) and \(O_2\)

\[V = \{ O_1, O_2 \}\]

\[G = R \setminus V, C = \emptyset\]

\[l = \text{maximum size of } V\]

while \((|V| \leq l)\) {

- for (each vertex \(O\) in \(G\)) {
  
  if \((O\) is connected to all vertices in \(V\)) {
    
    Record shortest edge between \(O\) and vertices in \(V\)

    Add \(O\) to \(V\)
  }

- if \((C \neq \emptyset)\) {
  
  Choose the vertex \(O\) whose shortest edge to \(V\) is longest

  Add \(O\) to \(V\)

  Delete \(O\) from \(G\) and \(R\)

  \[C = \emptyset\]

  \[l = l + 1\]

} else {

  delete \(O_1\) and \(O_2\) from \(G\) and \(R\)

  break

} // while

Construct a group for each remaining vertex

} // // if

The proposed algorithm is an adaptation of the algorithm proposed in Perkowitz and Etzioni (1998). The major difference is that the algorithm in Perkowitz and Etzioni produces nonoverlapping 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 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 (Mahmood, 2005a) groups the objects into correlated object clusters based on the user’s 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). The complete algorithm is given in Figure 1.

Below, we explain the major steps in 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 is 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
## Object Grouping and Replication on a Distributed Web Server System

2. At Step 2, we create a correlation matrix. The correlation between two objects $O_i$ and $O_j$ is the probability that they are accessed in the same user session. To calculate the correlation between $O_i$ and $O_j$, we scan the log file and count the number of distinct sessions in which $O_i$ was accessed after $O_j$ ($\text{count}(O_j, O_i)$) and calculate $p(O_j | O_i) = \text{count}(O_j, O_i) / s(O_j)$, where $p(O_j | O_i)$ is the probability of a client visiting $O_j$, if it has already visited $O_i$ and $s(O_j)$ is the number of sessions in which $O_j$ was accessed by a client. Similarly, we compute $p(O_i | O_j) = \text{count}(O_i, O_j) / s(O_i)$, where $p(O_i | O_j)$ is the probability of $O_i$ being accessed after $O_j$ in a session; $\text{count}(O_i, O_j)$ 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 accessed. The correlation between $O_1$ and $O_2$ is the $\min(p(O_2 | O_1), p(O_1 | O_2))$ to avoid mistaking a asymmetric relationship for a true case of high correlation.

3. At Step 3, we first create a graph corresponding to correlation matrix in which each object is a vertex and each nonzero 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 is 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 for including an object in 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.

## OBJECT 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 as well as distributed object replication algorithms.

The proposed centralized algorithm is a polynomial time greedy algorithm that is executed at a central server/arbiter and decides the placement of replicas for each object. The algorithm proceeds as follows. First, all of the object groups are organized in descending order of their density values 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 Zhuo et al. (2002) where $R_k$ 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 assign the object to 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 having the capacity for that object. The complete algorithm is given in Figure 2.

In the second proposed algorithm, a central arbitrator is used to group the objects using the object grouping algorithm. These groups are sent to each site using any of the available techniques, such as push technology (Franklin & Zdonik, 1998). Each site then determines which groups should be locally replicated based on the object usage statistics collected locally. The DA first removes all the objects previously replicated at a site, except those objects that do not have a replica at any other site. It then sorts the object groups in descending order of their local density values, where the local density value for a group $G_i$ is calculated as:

$$Density(G)=\frac{(\text{Total size of objects in } G)}{(\text{Total local read requests for objects in } G_i)}$$

The object groups are allocated one at a time (in the sorted order), provided enough space is available on the server, and no other constraint is violated. In case there is not enough space available to accommodate the whole group, some of the objects with high local access frequency are replicated. The complete algorithm is given in Figure 3.

In the proposed DA, the central arbitrator performs object regrouping either after a predefined time interval or on specific events (such as additional reads to objects).
or removal of new objects), making the object groups more adaptive to the changing access patterns and objects. Each site also periodically determines the new replication schemes when it observes changes in the local access patterns. Each site also can perform object grouping based on the local logs, hence eliminating the need of a central arbitrator altogether.

**EXPERIMENTAL RESULTS**

Comparing the performance of the proposed algorithm to the optimal solution obtained by an exhaustive search would have been the best way to illustrate the merits of our algorithms. An exhaustive search, though, is able to provide the optimum solutions, within a reasonable running time, only for small-sized problems. Since small problem sizes have little practical meaning, we followed the approach used by other researchers and compared our results with those obtained by other well-known algorithms. These algorithms are the random allocation algorithm (Kangasharju, Roberts, & Ross, 2002), the greedy algorithm (Tenzakhti et al. 2004), and a DA also proposed in Tenzakhti et al. (2004). In the random placement, replicas are placed at random with uniform probability among all the sites in the network. It can be considered as an “upper bound” placement method in a sense that an efficient replica placement method should always be better than the random placement (Radoslavov, Govindan, & Estrin, 2001).

In our simulation, we used trees having 100-600 nodes with a maximum degree of 15 (a tree with a higher number of nodes has a higher degree). Objects to be replicated were created so as to resemble a generic Web work load (Braford, Bestarov, Bradley, & Crovella, 1999), that is, object sizes followed a pareto distribution and their popularity followed the Zipf law (Zipf, 1949). The minimum object size was 4K. The total number of objects to be replicated varied from 500-2,500, depending on the number of sites in the network. The total number of requests considered for large network instances were 1 million, while for medium sized networks, it was 100,000. Two distinct cases were considered for request generation. In the first case, each site had the same probability of requesting an object, while in the second case requests were normally distributed in order to measure the performance of the algorithms at the presence of hot spots. We created log files by generating requests for objects for multiple sessions. These log files were used to group objects. The same log files were used by the proposed algorithms to collect various statistics. Total site capacity was taken proportional to the total size of all the objects, which was a random value between \((T/4)\%\) and \((3T/2)\%\), where \(T\) is total size of all the objects.
Figure 4. Mean latency for different tree sizes

Figure 5. Average latency for all simulation runs

Figure 6. Effect of site capacity on mean latency for the proposed algorithms
Figure 7. Effect of number of objects on mean latency for the proposed algorithms

Figure 8. Average % improvement achieved by the proposed algorithms

Figure 9. Performance of the proposed grouping algorithm
We used the average latency (i.e., average time required to serve a read request) as a measure of performance. The latency for accessing an object $k$ from node $i$ to node $j$ is calculated as follows. 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 $\lambda$ is the average arrival rate and $\mu$ is the average service time. Exponential service time is assumed with an average service rate of 100 to 150 transactions/second for different servers. The cost (latency) is computed at the end of each simulation run as discussed in the object replication model.

In the first set of experiments, we assessed the performance of the proposed algorithms as compared to other algorithms mentioned before. We fixed the site capacity to $T=30\%$. Figure 4 shows the average latencies 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 replicas of an object can be placed. Also, note that the performance of proposed algorithms is better than all other algorithms.

In the second set of experiments, we investigated the impact of site capacity and number of objects on solution quality of the proposed algorithms in terms of average latency. Figure 6 shows that the average latency decreases as the site capacity increases. The average latency tends to decrease significantly as the site capacity increases, while after a certain point where most read intensive objects are replicated, adding more storage space results in only marginal performance improvements. A similar trend was seen when the number of objects were increased, while keeping other parameters constant. Figure 7 shows the effect of increasing the number of objects for a network of 100 sites (similar results were obtained for other configurations).

In the third set of experiments, we investigated the impact of the increase in the request frequencies. We periodically increased the access frequencies of each site-object pair randomly by 40-200%. The proposed algorithm is then run to determine the new replication schemes. 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 average percentage improvement in the latency is shown in Figure 8. It can be observed that the algorithms improve the replication schemes every time access frequencies are changed by adding, removing, or migrating objects to more appropriate sites.

We also investigated the effectiveness of the grouping algorithm. The objects were grouped randomly, using the proposed algorithm and putting a single object in each group. The proposed algorithms were run for each type of grouping. The results are shown in Figure 9. It can be seen in the figure that we obtained better replications when objects were grouped using the proposed algorithm. Random grouping of objects performed the worst while the performance of allocating a single object (i.e., groups with one object only) was better than the randomly grouping of objects.

**CONCLUSION**

Object replication on a cluster of Web servers is a promising technique for achieving better performance. However, one needs to determine the number of replicas of each object and their locations in a DWS. Choosing the right number of replicas and their locations are nontrivial problems. In this article, we presented an object
Object Grouping and Replication on a Distributed Web Server System

grouping algorithm and two object replication algorithms. The object grouping algorithm groups Web objects based on the users’ access patterns stored in Web log file. The documents that are highly 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 object replication algorithm is a centralized one, in the sense that a central site determines the replica placement to minimize a cost function subject to the capacity and other constraints. The second proposed algorithm is a distributed one, allowing each site to determine its locally optimal replication schemes, based on locally collected statistics. Taking 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, random, and DA demonstrates the superiority of the proposed algorithms.

REFERENCES


