Computing and Informatics, Vol. 24, 2005, 371–390

OBJECT REPLICATION ALGORITHMS FOR WORLD WIDE WEB

Amjad Mahmood

Department of Computer Science
University of Bahrain
Kingdom of Bahrain
e-mail: `amahmood@itc.uob.bh`

Manuscript received 24 September 2004; revised 6 June 2005

Communicated by Jaroslav Pokorný

Abstract. Object replication is a well-known technique to improve the accessibility of the Web sites. It generally offers reduced client latencies and increases a site's availability. However, applying replication techniques is not trivial and a large number of heuristics have been proposed to decide the number of replicas of an object and their placement in a distributed web server system. This paper presents three object placement and replication algorithms. The first two heuristics are centralized in the sense that a central site determines the number of replicas and their placement. Due to the dynamic nature of the Internet traffic and the rapid change in the access pattern of the World-Wide Web, we also propose a distributed algorithm where each site relies on some locally collected information to decide what objects should be replicated at that site. The performance of the proposed algorithms is evaluated through a simulation study. Also, the performance of the proposed algorithms has been compared with that of three other well-known algorithms and the results are presented. The simulation results demonstrate the effectiveness and superiority of the proposed algorithms.

Keywords: Object replication, data replication, Web, distributed web-server system, data placement

1 INTRODUCTION

The recent phenomenal growth in the World Wide Web (the Web) has brought huge increase in the traffic to the poplar websites that occasionally reaches the limits of their capacity and consequently causes servers to be overloaded [1]. As a result, end users either experience 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 [2]. The first approach, generally referred to as *hardware scale-up*, is the use of powerful server machines 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 the clients' demand curve. Therefore, the issue of scalability and performance may persist with ever increasing user demand.

The second approach to keep up with ever increasing request load and provide scaleable services is to deploy a 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 [3]. The incoming requests are distributed among the server nodes to achieve load balancing. A DWS approach, generally referred to as *scale-out*, 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 multiple clients from different locations. As the DWS spreads over a MAN or WAN, movement of documents between server nodes is an expensive operation [2]. 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.) [2, 3, 4].

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 % [5]. A pro-active web server system, on the other hand, can decide where to place copies of an object 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 or request dispatchers [6, 7, 8]. Although such systems are simple to implement but 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 wastage of disk space, one can replicate part of the documents on multiple server nodes and requests can be distributed to achieve better performance [9, 10, 11]. However, some rules and algorithms are then needed to determine the 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 [12].

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 [13, 14]. Therefore, all the replica placement approaches proposed in the literature are heuristics that are designed for certain systems and work loads. The object-replication strategies proposed in the literature can be divided into two categories [14]: centralized and distributed. In centralized replication, the number of replicas is determined once and remains fixed for a fixed period of time until some central managing site executes re-allocation based on the access patterns and load on different sites. In distributed replication, the decision to replicate is taken locally at a site level in response to changes in the access pattern.

This paper proposes three algorithms for replica placement in a Web environment. The first two algorithms are centralized in nature and the third one is a distributed one. For distribution of requests, we take site proximity and access cost into account. 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 follows: Section 2 reviews some existing work related to object replication in the web. Section 3 describes the centralized and distributed replications models and the cost function. Section 4 presents two proposed centralized and one distributed object replication algorithms. Section 5 presents the simulation results and Section 6 concludes the paper.

2 RELATED WORK

There has been a considerable amount of work on Web performance, ranging from Web workload characterization [15, 16] to developing techniques to enhance web performance. Two common techniques for enhancing web performance are web caching and replication [17–19]. Data replication can be viewed as a special kind of "push-based" caching [20]. Compared with general caching, replication can take advantage of multicast for update dissemination [21]. Further, replication allows elegant placement schemes to be developed and employed based on the knowledge of mid-term and long-term access patterns.

The problem of replica placement in communication networks has been extensively studied in the area of file allocation problem (FAP) [22] and distributed database allocation problem (DAP) [23, 24]. The FAP can be defined as follows: given a network of M sites with different storage capacities and N files exhibiting different read and write frequencies from each site, allocate the files to the sites in order to maximize/minimize a given cost function subject to a number of constraints [25]. Both FAP and DAP are modeled as a 0-1 optimization problem and solved using various heuristics, such as knapsack solution [26], branch-and-bound [27], and network flow algorithms [28]. An outdated but useful survey of work related to FAP can be found in [22]. 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 [29–31]. Kwok et al. [32] and Bisdikian Patel [33] 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 [34], the replica placement problem in CDN is formulated as an uncapacitated minimum K-median problem. In [35], different heuristics were proposed based on this K-median formulation to reduce network bandwidth consumption. The authors of [36] take storage constraint into consideration and reduce the knapsack problem to replica placement problem in CDNs. Li [12] 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. [37] 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. This, 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 [38] 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 [39], 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 from the nearest replica only.

Tensakhti et al. [14] 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. [4]. 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. [2]. 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 document to be replicated depending on its usage and size. In [40] 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. [20] 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. Karlsson et al. [41] developed a common framework for the evaluation of replica placement algorithms.

Heddaya and Mirdad [42] 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. [43] 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 [44], the author has surveyed distributed data management problems including distributed paging, file allocation, and file migration.

3 THE SYSTEM MODELS

A distributed Web server system consists of many servers interconnected by a communication network. A unit of data to be replicated is referred to as an *object*. An object can be a XML/HMTL page, an image file, a relation, etc. Each object is identified by a unique identifier and may be replicated on a number of sites/servers. 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 $|E|$ is the number of edges (links). Each node in the graph corresponds to a router, a switch or a web site/server. We assume that out of these N nodes there are n web servers acting as information providers. 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 node v and going to web server i ($i = 1, 2, \ldots, n$). The 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).

Like all other similar studies [2, 4, 14, 17, 37], we do not include the individual clients that send read requests for objects in our model for two reasons. First, clients are neither information provider nor under the direct control of the distributed web server system and hence cannot be modeled explicitly [7]. Second, we only have to know the total number of read requests from all clients to the individual servers for each object. Each read request is either served locally by the server or it is forwarded to the server that can satisfy 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 [12, 14, 37]. 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.

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 that resides at the least cost server (denoted by LC_k^i) from site 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. The decision is based on the statistics collected at each server 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 server sites. In addition, each server i keeps the following information:

- LC_k^i: The least cost site with a replica of object k when k is accessed from i.
- $LC_k^{i,j}$: 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.

The traffic frequency, $f_k^{i,j}$, is the number of read requests for a certain period of time t issued at site i for object k that resides at site j. This frequency includes the number of requests received by site i from the individual clients and the requests for object k passing through it in their way to j from other servers [4]. This traffic can easily be monitored and recorded by using the existing technologies [4, 6].

A number of methods has been proposed to calculate cost (latency) of accessing an object from a server [13, 14]. We calculate the latency as follows: each replica site j maintains a count c of the total number of requests it receives in a time period t. The arrival rate, λ, 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 μ, then the time T, a request will spend at the replica site (waiting + processing time) is the well-known $M/M/1$ queuing result $T = 1/(\mu - \lambda)$ [13]. 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 all the sites in a recursive way. The added communication cost can be obtained by having each site periodically query its neighbors and determining this cost.

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 server 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 follows.

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 be total n sites (web servers) and m objects. Let $f_k^{i,j}$ be 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 $(1 \leq k \leq m)$ at site i $(1 \leq i \leq n)$ is given by

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

The cost of serving requests for all the objects at site i is given by

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

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

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

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

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

Subject to

$$\sum_{i=1}^{n} x_{ik} \geq 1 \text{ for all } 1 \leq k \leq m \tag{5}$$

$$\sum_{k=1}^{m} x_{ik} s_k \leq TS_i \text{ for all } 1 \leq i \leq m \tag{6}$$

$$x_{ik} \in \{0, 1\}, \text{ for all } i, j. \tag{7}$$

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 PLACEMENT ALGORITHMS

The replica placement problem described in the previous section is reduced 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 sub-section, we present our proposed centralized object replication algorithms.

4.1 Centralized Object Placement Algorithms

For our first algorithm, we convert our model into a state-space search problem and then introduce the use of algorithm $A*$ from artificial intelligence theory. Before explaining the algorithm, we define the following terms.

Definition 1. We say that an assignment X is *incomplete* if not all the objects and their replicas are assigned under assignment X. Otherwise, X is a *complete assignment.*

Definition 2. A *state* is defined by a list of n sets $(S_1, S_2, \ldots, S_n)$ where S_i contains the objects assigned to site i.

The state-space search tree is built as follows:

1. First of all, we determine the number of replicas for each object according to its popularity and the available storage space using the density algorithm proposed in [2]. The algorithm first defines the density of each object as access rate of that object divided by its size. The larger the object's density, the more replicas it should have. Each replica of an object is identified by a unique replica number. If there are m objects and each object i has r_i replicas, then total number of objects to be replicated is given by $R = \sum_{i=1}^{m} r_i$.
2. Arrange the replicas in non-increasing order of their density. This is to make sure that the objects having higher density value are replicated first to the best available sites. The reason for ordering the objects with respect to their density value is that the objects which are heavily accessed by the clients and have relatively smaller size are considered first for allocation and would have a greater chance of being allocated to the best site without violating the constraints. A similar ordering heuristic has been used in [45] and it was shown that this heuristic produces better quality solution and converges faster than other ordering heuristics.
3. The root node has a state $(\emptyset, \emptyset, \ldots, \emptyset)$ indicating that none of the objects has been assigned to any server.
4. At each subsequent level, nodes are expanded by assigning next replica in the sorted list to every server to generate n nodes of the next level. This procedure ends when all the objects have been assigned.

Note that in a state-space search tree, all the internal nodes correspond to incomplete assignments and all the leaf nodes correspond to complete assignments. Our goal is to find a leaf node in the tree with an assignment X such that CC given by (3) is minimum.

Definition 3. In the state-space search tree, if state of a node corresponding to an assignment violates any of the constraints, then the node is called an *invalid node.* Otherwise the node is called a *valid node.*

For object replication problem, the state-space search tree can be trimmed down due to the existence of invalid nodes. Algorithm $A*$ searches a subtree of the entire state space search tree to find a path from the root node to a goal/leaf node. It uses a heuristic evaluation function $f(v)$ to order nodes for expansion. At each iteration (step 3 of the algorithm), we select a node v with the minimum $f(v)$ value. If node v is not a leaf node (i.e. node v corresponds to an incomplete assignment), then it is expanded by assigning next replica in the sorted list to every server to generate n nodes of the next level. However, if v is a leaf node (i.e. it corresponds to a complete assignment), then v is accepted as a solution and no further expansion is required. The heuristic evaluation function $f(v)$ which is used to order the nodes is defined as

$$f(v) = g(v) + h(v) \tag{8}$$

where $g(v)$ is the path cost from root node to node v, and $h(v)$ is an estimation of the minimum cost path from node v to a leaf/goal node. If node v corresponds to an incomplete assignment with s replicas (out of total n replicas) already assigned, then $g(v)$ is defined as

$$g(v) = CC(X_v^s) \tag{9}$$

$$CC(v) = \sum_{i=1}^{n} \sum_{k=1}^{s} \left[(1 - x_{ik}) f_k^{i,LC_k^i} C_k^{i,LC_k^i} + x_{ik} f_k^{i,i} C_k^{i,i} \right] \tag{10}$$

which is a form of (3) when X is substituted by X_v^s (X_v^s denotes the partial assignment matrix of replicas at node v when only s replicas out of total n have been allocated to node v). This gives the path cost from the root node to node v. We define $h(v)$ for a node v corresponding to a replica assignment $1, 2, \ldots, s$ as

$$h(v) = \sum_{k=s+1}^{R} \min(f_k^{i,i} C_k^{i,i}). \tag{11}$$

Note that in (11) minimum has to be taken by assigning each of the remaining replicas to one of the n sites without violating any constraint. The proposed algorithm is given in Figure 1.

Note that the worst case time complexity of the algorithm is exponential. Also, the function $f(v)$ is quite complicated and involves a large number of computations. Furthermore, the number of replicas for each object has to be known. Thus, the algorithm is suitable only for relatively small systems with known number of replicas for each object. Therefore, there is a need for efficient heuristics to find suboptimal allocations in a reasonable computational time.

Our second algorithm (algorithm 2) is a greedy algorithm that proceeds as follows: for each site i, we assign an object k to site i such that putting a copy of k

```
Step 1. Arrange the replicas in non-increasing order of their density.
Step 2. Put root node $r = (\emptyset, \emptyset, \ldots, \emptyset)$ on a list called OPEN and set $f(r) = 0$.
Step 3. Repeat
            Find a node $v$ in the list OPEN with a minimum $f$ value.
            If $v$ is not a leaf node then
               Expand node $v$ by assigning the next replica in the list to each
               of the $n$ sites to generate $n$ successor nodes of node $v$.
               Move all the valid nodes to list OPEN and calculate their $f$ values
               according to (8), (10) and (11)
            Endif
        Until node $v$ is a leaf node
Step 4. Return the assignment associated with node $v$ as a solution.
```

Fig. 1. The proposed algorithm 1

at i has the maximum unit profit provided no constraints are violated. The profit is calculated as:

$$P_k^i = \begin{cases} \left(f_k^{i,LC_k^i} C_k^{i,LC_k^i} - f_k^{i,i} C_k^{i,i}\right)/s_k & \text{if at least one replica of object } k \\ & \text{has already been assigned,} \\ \left(f_k^{i,WC_k^i} C_k^{i,WC_k^i} - f_k^{i,i} C_k^{i,i}\right)/s_k & \text{otherwise.} \end{cases} \tag{12}$$

Note that equation (12) gives unit profit that is obtained when object k is replicated at site i. The first term $(f_k^{i,LC_k^i} C_k^{i,LC_k^i})$ is the current total cost of accessing object k that resides at the least cost site (LC_k^i) by site i. The second term $f_k^{i,i} C_k^{i,i}$ is the cost of accessing the object if a local copy is made (note that the second term will not be zero since we consider that accessing a local copy incurs processing cost). Equation (12) is based on an optimistic heuristic [46] and gives the expected profit of storing a replica at site i. The profit is calculated by taking the difference between the local access cost and the cost of accessing the object from a least cost site, denoted by WC_k^i, where a replica of i may be placed.

We keep putting the copies of objects at i as explained before until the remaining capacity at site i is enough to hold the smallest object that has not been assigned to i. Note that this algorithm does not guarantee that each object will be allocated to at least one site since heavily accessed objects may get higher number of copies leaving not enough space for less frequently accessed objects. Therefore, as the last step of the greedy algorithm, we perform an adjustment operation if all of the objects have not been allocated. The adjustment operation is carried out by a branch-and-bound algorithm that adds unallocated objects to suitable sites by removing some of the replicas. The branch-and-bound algorithm works on the principle that an unallocated object can only be allocated if one or more already allocated objects are deleted from a node. Since deleting any of the already allocated nodes will increase the total cost, therefore reducing the loss of these deletions will result in a better allocation.

Let K_i and ξ be set of objects that have been allocated to site i and set of objects that has not been allocated to any site, respectively. If $\xi \neq \emptyset$, then B & B must be executed. The main steps of the B & B algorithm are described in Figure 2.

```
Step 1. Find $k \in \xi$ that has the highest density.
Step 2. Generate $n$ branches by making forced allocation of object $k$ on each site
        $i$ (i.e. $x_{ik} = 1$). This generates $n$ sub-problems, one for each site.
        Order the sub-problems in descending profit values given by equation (12).
Step 3. Solve the first sub-problem, maintaining for other sites the previously obtained
        allocation. The sub problem is solved as follows:
        Order the objects allocated to site $i$ increasing the value of $P_k^i$. Delete
        the objects one by one subject to the constraints until addition of object $k$
        at site does not violate any of the constraints. Calculate the net loss
        of all deletions made so far.
Step 4. If replication of object $k$ at site $i$ violates any of the constraints, undo step 3,
        close the branch and go to step 7.
Step 5. If all the objects are allocated then end else go to step 1.
Step 6. If the net loss is less than incumbent (initially a large number) then store this
        allocation as a candidate solution and update incumbent to net loss else close
        the branch. Intuitively, a branch is closed when all of its sub-branches are
        closed.
Step 7. If all branches are closed then terminate the algorithm with candidate
        allocation as a solution, else go to step 3.
```

Fig. 2. The proposed B & B algorithm

The ordering criteria for deciding which object should be considered first and at which site it should be allocated first (step 1 and 2) are based on a simple heuristic that a heavily required object should be allocated first at a site where its usage requirements are high. This may help in early production of a good feasible solution which, in turn, makes the bound test more efficient by early fathoming of branches that do not lead to a better solution [46].

4.2 Distributed Object Placement Algorithm

The algorithms 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. Also, all the servers need to send the locally collected statistics to the central arbitrator whenever a new allocation is to be determined and hence causes extra network load. Nevertheless, the centralized object replication model is useful when the number of servers and the statistics to be sent to the central arbitrator is small and the reallocation is determined at off-peak time. On the other hand, 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 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 either by placing a single copy of each object to a randomly selected server or by using a simple greedy algorithm [34, 36]. After every time period t, each site i determines which objects it should add/remove based on the current replica placement and locally collected statistics using the following proposed algorithm. If there is only one replica of an object k that resides at site i, then this replica is not considered for deletion. For all other objects, the algorithm calculates the unit profit of adding/removing the replicas. The profit of keeping a replica of object k at site i is calculated by taking the difference of cost of accessing the object from the least cost site $LC_k^i \neq i$ (i.e. the cost of accessing the object if the local replica is removed and it is accessed from a least cost server) and the present cost of accessing the replica locally, as explained in Section 4.1. Similarly, the profit of adding a local replica is calculated by taking the difference between the present cost of accessing the replica from the least cost site and the cost of accessing the replica if a replica is kept locally. The algorithm 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 3.

```
Step 1. for each object $k$.
            If server $i$ is the only server having a replica of object $k$ then
              $profit_k = a\_max\_number$
            else
              $profit_k = \left| \left( f_k^{i,LC_k^i} C_k^{i,LC_k^i} - f_k^{i,i} C_k^{i,i} \right) / s_k \right| \quad // \; LC_k^i \neq i$
Step 2. Sort all the objects in descending order of their profit values.
Step 3. Replicate the objects from the sorted list (obtained at step 2) one by one until
        there is no space available on server $i$.
```

Fig. 3. The proposed distributed algorithm (algorithm 3)

5 EXPERIMENTAL RESULTS

This section presents some performance measures obtained by simulation of the proposed algorithms. Since the object replication algorithm is NP-complete, computing the exact (optimal) solution is too computationally intensive to be useful in practice. Similarly to other studies, we compare the performance of our algorithms with two well-known algorithms proposed in the literature.

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 2 000 in all the simulation runs. We use different object sizes which follow a normal distribution. The average object size is taken as 10 KB and maximum size was taken as 100 KB. About 64 % objects sizes were in the range of 2 KB and 16 KB. 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. The number of requests for different objects from all the clients to the individual servers was generated randomly and different statistics was collected as described in Section 3.1.

During a simulation run, each site keeps a count c of the total number of requests it received 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 rate. 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 discussed in Section 3.1. 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. Note that the communication cost depends on the bandwidth of the link between two sites. The bandwidth of a link was selected randomly from {52 kbps, 256 kbps, 100 Mbps,1 Gbps}.

We studied the performance of our three proposed algorithms and compared it with that of random allocation algorithm [34], simple greedy algorithm [14], and Max-router fan-out algorithm [35]. 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 [2]. 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 an 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. The simple greedy algorithm and its variations are the most commonly used approaches and hence a comparison with it also provides an indication of goodness of a heuristic. The Max-router fan-out algorithm is based on the assumption that the nodes with large number of fan-out are eventually the closest to all other nodes and therefore they are a better choice for replica location [35].

Figure 4 shows the average latency for all the simulation runs for different tree sizes. The minimum and maximum latencies for various system configurations for different algorithms are shown in Table 1. Figure 4 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 average performance of algorithm 2 and algorithm 3 is comparable demonstrating the effectiveness of the distributed algorithm for larger tree sizes. Algorithm 1 performs better than all the algorithms. The performance of the random algorithm was worst of all the algorithms, as expected. However, Max-router fan-out algorithm performed, on average, better than the greedy algorithm and worse than the three proposed algorithms. It was also observed that algorithm 1 and algorithm 2 were able to find the same solution for 58 % of the runs, whereas algorithm 3 was a little inferior to them. It was also observed that in 21.5 % of cases, all the algorithms (except the random one) found the solutions of the same quality in terms of the value of the cost function. There were only 3.5 % of cases when the max-router fan-out produced solution better than algorithm 3. Algorithm 1 and algorithm 2 were always able to find solutions which were either equal or better than all the other algorithms in terms of the value of the cost function. However, the running time of algorithm 1 was noted to be much higher than the rest of the algorithms. In fact, in some cases, the running time of algorithm 1 was more than 100 times greater than those of algorithm 2 and algorithm 3.

Figure 5 shows the average performance of the algorithms for all the system configurations. It is evident from the results that the proposed algorithms perform, on average, better than the greedy, random and max-router fan-out algorithms.

We also studied the effect of replica replacement to establish the fact that replacement is beneficial when there is a change in the object access pattern from different sites. Figure 6 shows the results obtained in our simulation. For each tree, we ran the simulation for a number of times, each time changing the access patterns randomly. Then we calculated the average latency without changing the object placement and replication. We determined the new object placement and reconfigured the system according to the new placement determined by each algorithm and calculated the average improvement in latency. Figure 7 clearly shows that there has been improvement after system reconfiguration. Algorithm 1 again outperforms algorithm 2 and algorithm 3, whereas the performance of algorithms 2 and 3 is comparable on the average.

# of sites	Algorithm 1		Algorithm 2		Algorithm 3		Greedy		Random		Max fan-out	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
100	237	298	265	335	255	338	277	410	350	543	337	396
200	239	280	239	306	229	305	242	379	330	485	221	357
300	210	225	215	250	219	252	236	310	305	420	210	289
400	185	217	185	230	199	247	190	285	270	405	185	267
500	170	215	170	228	182	249	195	255	260	375	197	239
600	155	180	165	205	165	217	185	238	205	318	157	213

Table 1. Maximum and minimum latencies (in msec) for some simulation cases

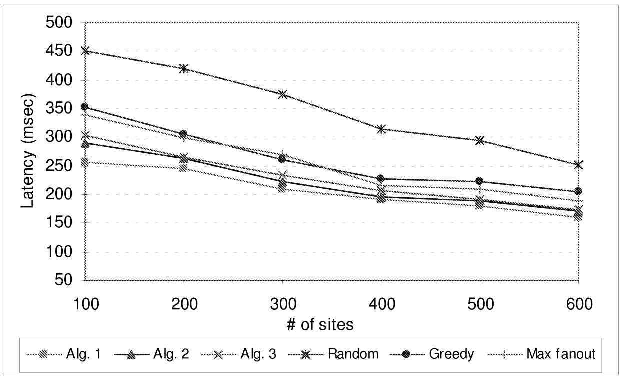

Fig. 4. Mean latency for different tree sizes

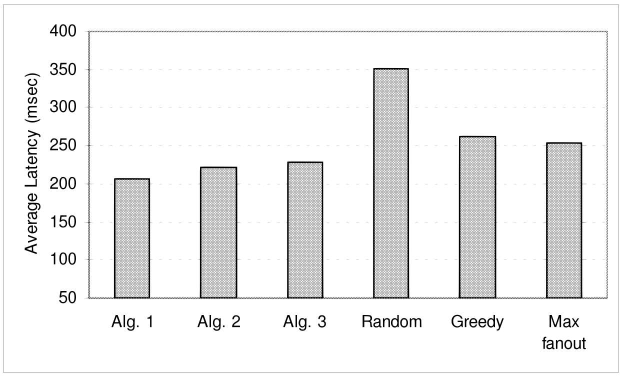

Fig. 5. Average latency for all simulation runs

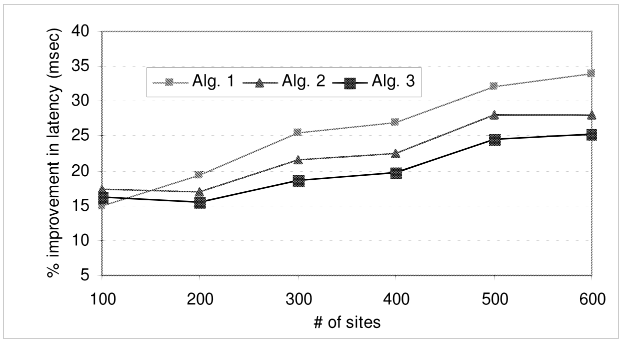

Fig. 6. Latency improvement obtained for different tree sizes

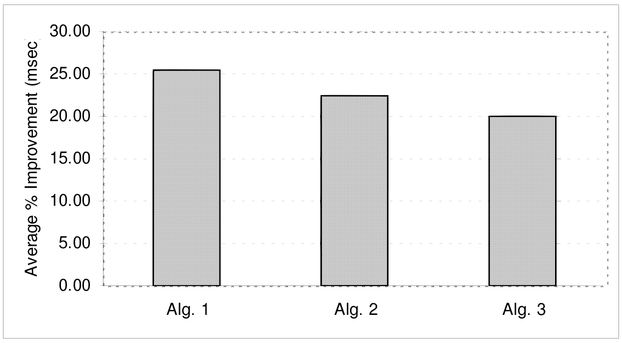

Fig. 7. Average latency improvement obtained by different algorithms

6 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 the right number of replicas and their locations is a non-trivial problem. In this paper, we presented three new algorithms for object replication in a distributed web-server environment. The first two algorithms are centralized 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 third algorithm is a distributed algorithm and hence does not need a central site for determining object placement. 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, random and max-router fan-out algorithms demonstrates the superiority of the proposed algorithms.

Acknowledgement

Part of this work was carried out while the author was at King's College London, UK. I would like to thank all those who provided us with all the help I needed to carry out this work.

REFERENCES

[1] Chen, X.—Mohapatra, P. —Chen, H.: An Admission Control Scheme for Predictable Server Response Time for Web Accesses. In Proceedings of the 10th WWW Conference, Hong Kong, May 2001, pp. 45–54.

[2] Zhuo, L.—Wang, C.-L.—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.

[3] Phoha, V. V.—Iyengar, S. S.—Kannan, R.: Faster Web Page Allocation with Neural Networks. IEEE Internet Computing, Vol. 6, 2002, No. 6, pp. 18–25.

[4] Kalpakis, K.—Dasgupta, K.—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.

[5] Abrams, M.—Standridge, C. R.—Abdulla, G.—Williams, S.—Fox, E. A.: Caching Proxies: Limitations and Potentials. Proc. 4th International World Wide Web Conference, Boston, Dec. 1995, pp. 119–133.

[6] Cardellini, V.—Colajanni, M.—Yu, P. S.: Dynamic Load Balancing on Web-Server Systems. IEEE Internet Computing, Vol. 3, 1999, No. 3, pp. 28–39.

[7] 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.

[8] Kwan, T. T.—Mcgrath, R. E.—Reed, D. E.: NCSA's World Wide Web Server: Design and Performance. IEEE Computer, Vol. 28, 1995, No. 11, pp. 68–74.

[9] Baker, S. M.—Moon, B.: Scalable Web Server Design for Distributed Data Management. Proc. of 15th Int. Conference on Data Engineering, Sydney, March 1999, pp. 96–110.

[10] Li, Q. Z.—Moon, B.: Distributed Cooperative Apache Web Server. Proc. 10th Int. World Wide Web Conference, Hong Kong, May 2001.

[11] Riska, A.—Sun, W.—Smimi, E.—Ciardo, G.: ADATPTLOAD: Effective Load Balancing in Clustered Web Servers Under Transient Load Conditions. Proc. 22nd Int. Conf. on Distributed Systems, Austria, July 2002.

[12] Li, B.: Content Replication in a Distributed and Controlled Environment. J. of Parallel and Distributed Computing, Vol. 59, 1999, No. 2, pp. 229–251.

[13] Karlsson, M.—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`.

[14] Tenzakhti, F.—Day, K.—Olud-Khaoua, M.: Replication Algorithms for the Word-Wide Web. J. of System Architecture, Vol. 50, 2004, pp. 591–605.

[15] Arlit, M. F.—Williamson, C. L.: Internet Web Servers: Workload Characterization and Performance Implications. IEEE Trans. On Networking, Vol. 5, 1997, No. 11, pp. 560–571.

[16] Padmanabhan, V. N.—Qiu, L.: The Content and Access Dynamics of a Busy Web Site: Findings and Implications. SIGCOMM 2000, August-September 2000.

[17] Wolfson, O.—Milo, A.: The Multicast Policy and Its Relationship to Replicated Data Placement. ACM Trans. On Database Systems, Vol. 16, 1991, No. 1, pp. 181–205.

[18] Rabinovich, M.—Rabinovich, I.—Rajaraman, R.—Aggrawal, A.: A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service. Proc. IEEE ICDCS '99, May 31–June 4, 1999, pp. 101–113.

[19] Krishnan, P.—Raz, D.—Shavitt, Y.: The Cache Location Problem. IEEE/ACM Trans. Networking, Vol. 8, 2000, No. 10, pp. 568–582.

[20] Xu, J.—Li, B.—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.

[21] Yu, H.—Breslau, L.—Shenker, S.: Scaleable Web Cache Consistency Architecture. Proc. ACM SIGCOMM '99, Aug. 1999, pp. 163–174.

[22] Dowdy, L.—Foster, D.: Comparative Models of the File Assignment Problem. Computer Surveys, Vol. 14, 1982, No. 2, pp. 287–313.

[23] Ozsu, M. T.—Valduriez, P.: Principles of Distributed Database System. Englewood Cliff, N. J.; Prentice Hall, 1999.

[24] Apers, P. G. M.: Data Allocation in Distributed Database Systems. ACM Transactions on Database Systems, Vol. 13, 1998, No. 3, pp. 263–304.

[25] Chu, W. W.: Optimal File Allocation in a Multiple Computer System. IEEE Trans. on Computers, Vol. 18, 1969, No. 10, pp. 885–889.

[26] Ceri, S.—Martella, 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.

[27] Fisher, M. K.—Hochbaum, D. S.: Database Location in Computer Networks. J. ACM, Vol. 27, 1980, No. 10, pp. 718–735.

[28] Chang, S. K.—Liu, A. C.: File Allocation in Distributed Database. Int. J. Computer Information Science. Vol. 11, 1982, pp. 325–340.

[29] Awerbuch, B.—Bartal, Y.—Fiat, A.: Competitive Distributed File Allocation. Proc. 25th Annual ACM Symposium on Theory of Computing, Victoria, May 1993, pp. 164–173.

[30] Loikopoulos, T.—Ahmed, I.: Static and Dynamic Data Replication Algorithms for Fast Information Access in Large Distributed Systems. 20^{rmth} IEEE Conference on Distributed Computing Systems, Taipei, 2000.

[31] Gavish, B.—Sheng, O. R. L.: Dynamic File Migration in Distributed Computer Systems. Comm. of ACM, Vol. 33, 1990, No. 1, pp. 177–189.

[32] Kwok, Y. K.—Karlapalem, K.—Ahmed, I.—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.

[33] Bisdikian, C.—Patel, B.: Cost-Based Program Allocation for Distributed Multimedia-On-Demand Systems. IEEE Multimedia, Vol. 3, 1996, No. 3, pp. 62–76.

[34] Qiu, L.—Padmanabham, V. N.—Voelker, G. M.: On the Placement of Web Server Replicas. In Proc. of 20th IEEE INFOCOM, Anchorage, USA, April 2001, pp. 1587–1596.

[35] Radoslavov, P.–Govindan, R.—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`.

[36] Kangasharju, J.—Roberts, J.—Ross, K. W.: Object Replication Strategies in Content Distribution Networks. Computer Communications, Vol. 25, 2002, No. 4, pp. 367–383.

[37] Wolfson, O.—Jajodia, S.—Huang, Y.: An Adaptive Data Replication Algorithm. ACM Trans. Database Systems, Vol. 22, 1997, No. 2, pp. 255–314.

[38] 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.

[39] Bartal, Y.—Charikar, M.—Indyk, P.: On Page Migration and Other Relaxed Task Systems. Theory of Computer Science, Vol. 281, 2001, No. 1, pp. 164–173.

[40] Zhuo, L.—Wang, C.-L.—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`.

[41] Karlsson, M.—Karamanolis, C.—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`.

[42] Hedfdaya, A.—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.

[43] Sayal, M.—Breitbart, Y.—Scheurermann, P.—Vingralek, R.: Selection of Algorithms for Replicated Web Sites. Performance Evaluation Review, Vol. 26, 1998, No. 1, pp. 44–50.

[44] Bartel, Y.: Distributed Paging. Proc. Dagstuhl Workshop On-line Algorithms, 1997, pp. 164–173.

[45] Mahmood, A.: Task Allocation Algorithms for Maximizing Reliability of Heterogeneous Distributed Computing Systems. Control and Cybernetics, Vol. 30, 2001, No. 1, pp. 115–130.

[46] Mahmood, A.: Adaptive File Allocation in Distributed Systems: A Cybernetics Approach, Ph. D. Thesis, University of London, UK, 1994.

Amjad Mahmood received his M. Sc. in computer science from QAU, Pakistan in 1989 and a Ph. D., also in computer science, from the University of London, UK in 1994. Before joining the University of Bahrain in 2000, he worked with King Saud University, Saudi Arabia (1999–2000), Philadelphia University, Jordan (1997–1999) and National University of Science and Technology, Pakistan (1995–1997). He has published over 45 research papers in international journals and conferences. He also served as a member of technical committees for a number of international conferences and journals and has edited two conference proceedings. His research interests include distributed computing, real-time systems, WWW, and software engineering.