A Tabu Search Algorithm for Object Replication in Distributed Web Server Systems

A. Mahmood

T. S. K. Homeed

Department of Computer Science

University of Bahrain

Kingdom of Bahrain

Abstract : Data replication has been considered a promising technique for improving performance of a large distributed web server (DWS) system. One of the key issues in the design of DWS is determining the optimal number and placement of objects (e.g. HTML documents) on the web servers. This paper specializes the basic tabu search algorithm for object replication problem that not only determines the number of replicas but also their placement in a web server system to minimize a cost function subject to storage and server load constraints. The performance of the proposed algorithm is evaluated and compared with four other algorithms through a simulation study and results are reported. The simulation results demonstrate that the proposed tabu search is an effective and superior algorithm to solve the object replication problem as compared to many other proposed algorithms.

Keywords*:* Tabu search algorithm, metaheuristics, object replication, WWW, distributed web-servers, document replication.

Dr. A. Mahmood got his MSc. 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 numerous research papers in international journals and conferences. He also served as a member of technical committees for a number of international conferences and has edited two conference proceedings. His research interests include distributed computing, real-time systems, WWW, and software engineering.

Dr. T. S. K. Homeed is an assistant professor in the department of Computer Science, University of Bahrain, Bahrain. He got his Ph.D. in Computer Science, from Bradford University, UK in 1997. Before joining Bahrain University in 2003, he was working with Sana'a University, Yemen. He has served as a member of technical committees of various international conferences. He is also member of editorial board of IAJIT. His research interests include WWW, software engineering, and parallel processing

1. Introduction

Demand placed on Web services continue to grow and Web systems are becoming more overloaded than ever before. As a result, users of popular web sites often experience poor response time or denial of a service (time-out error) if the supporting web-servers are not powerful enough. Since these sites have a competitive motivation to offer better service to their clients, the system administrators are constantly faced with the need to scale up site capacity. There are two options to scale-up Web services [1]. The first option, generally referred to as *hardware scale-up*, is to upgrade the Web server to a larger and powerful machine 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 client demand curve which characterizes the Web (the number of on-line users is growing at about 90% per annum) [2].

The second option to keep up with ever increasing request load and provide scalable Web services is to deploy a distributed Web server system (DWS) composed of multiple server nodes where some system component under the control of the content provider can route incoming requests among different servers [3]. The 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. The performance of such systems (e.g. latency, throughput, availability, hop counts, link cost, and delay) can also be improved by maintaining multiple copies of objects at various locations in DWS [1,3].

A proactive distributed web server system can decide where to place copies of a document to get the optimal or near optimal performance. However, 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 Web switches [2,5-7]. Although such systems are simple to implement but they could easily result in uneven load among the server nodes due to caching of IP addresses on the client side.

To achieve better load balancing as well as to avoid disk wastage, one can replicate part of the documents on multiple server nodes and requests can be distributed to achieve better performance [8-10]. However,

some rules and algorithms are then needed to determine the optimal number of replicas of each document/object and their locations in a DWS. Choosing the right number of replicas and their locations can significantly reduce web access delays and network congestion. In addition, it can reduce the server load which may be critical during peak time. Many popular web sites have already employed replicated server approach which reflects upon the popularity of this method [11].

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

This paper proposes a tabu search algorithm for replica placement in a distributed web server environment. The major motivation behind proposing a tabu search algorithm for objective replication problem is the fact that various researchers have shown that tabu search, if applied intelligently, produces better quality results as compared to many traditional heuristics in solving a variety of optimization problems [13]. In this paper, we specialize the basic tabu search into a specific algorithm for the object allocation and replication problem by turning the abstract concepts of tabu search, such as initial solution, solution space, neighborhood, etc, into more concrete, problem specific and implementable definitions so that better quality solution can be obtained as quickly as possible. We also give a detailed formulation of the system and cost models suitable for the application of the proposed algorithm. The results presented in section 6 demonstrate that the proposed algorithm out-performs some well-known algorithms proposed in the literature.

The rest of the paper is organized as follow: Section 2 reviews some of the existing work related to object replication in the web. Section 3 describes the system model and presents cost function. Section 4 gives an overview of the tabu search algorithm. The tabu search algorithm is specialized for object replication problem and a detail description of the algorithm is given in section 5. Section 6 presents our simulation results followed by conclusions in section 7.

2. Related Work

The problem of replica placement in communication networks have been extensively studied in the area of file allocation problem (FAP) [14] and distributed database allocation problem (DAP) [15]. Both FAP and DAP are modeled as a 0-1 optimization problem and solved using various heuristics such as knapsack solution [16], branch-and-bound [17], and network flow algorithms [18]. 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 [19,20]. Kwok et al. [21] and Bisdikian Patel [22] 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 [23], the replica placement problem in CDN is formulated as an uncapacitated minimum K-median problem. In [24], different heuristics were proposed based on this K-median formulation to reduce network bandwidth consumption. The authors of [25] take storage constraint into consideration and reduce the knapsack problem to replica placement problem in CDNs. Li [11] proposed a suit of algorithms for determining the location of replica servers within a network. The objective of this paper is not to determine the placement of objects themselves but to determine the locations of multiple servers within a network such that the product of distance between nodes and the traffic traversing the path is minimized.

Wolfson et al. [26] proposed an adaptive data replication algorithm which can dynamically replicate objects to minimize the network traffic due to “read” and “write” operations. The proposed algorithm works on a logical tree structure and requires that communication traverses along the paths of the tree. They showed that the dynamic replication leads to convergence of the set of nodes that replicate the object. It, however, does not consider the issue of multiple object replications. Further, given that most objects in the Internet do not require “write” operation, the cost function based on “read” and “write” operations might not be ideal for such an environment.

Bestavros [27] 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. In [28], 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.

Tensakhti et al. [12] 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 replicas 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 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 document replication in extendable geographically distributed web server systems is addressed by Zhuo et al [1]. They proposed four heuristics to determine the placement of replica in a network. In addition, they presented an algorithm that determines the number of copies of each documents to be replicated depending on its usage and size. In [29], 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. [30] discussed the problems of replication proxy placement in a tree and data replication placement on the installed proxies given that maximum M proxies are allowed. The authors proposed algorithms to find number of proxies needed, where to install them and the placement of replicas on the installed proxies to minimize the total data transfer cost in the network. Karlsson et al. [31] developed a common framework for the evaluation of replica placement algorithms.

Heddaya and Mirdad [32] 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. [33] 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 [34], the author has surveyed distributed data management problems including distributed paging, file allocation, and file migration.

3. The System Models

A replicated Web consists of many sites/servers interconnected by a communication network. A unit of data to be replicated is referred 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/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. Associated with every node $v \in V$ is a set of nonnegative weights and each of the weights is associated with one particular web server. This weight can represent the traffic traversing this node v and going to web server i ($i = 1,2,\ldots,n$). This traffic includes the web access traffic generated at the local site that node v is responsible for and, also, the traffic that passes through 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 π. The requests are routed up the tree to the home site (i.e. root of the tree). Note that a route from a client to a site forms a routing tree along which document requests must follow. Focusing on a particular sever i, the access traffic from all

nodes leading to a server can be best represented by a tree structure if the transient routing loop is ignored [11,12,26]. 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. The spanning tree T_i rooted at a site i is formed by the clients that request objects from site i and the processors that are in the path π of the requests from clients to access object k at site i.

3.1. The Object Replication Model

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

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

$f_k^{i,j}$: 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, $f_k^{i,j}$, is the number of read requests for a certain period of time t issued at site i for object k to site j. 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) [12,31]. To determine the cost (latency), our proposed 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, 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 $T=1/(\mu - \lambda)$. 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.

3.2. The Cost Model

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

Suppose that the vertices of $G(V,E)$ issue read requests for an object and copies of that object can be stored at multiple vertices of G. Suppose that there are total n sites (web servers) and m objects. Let 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 object k at site i from site j, denoted by TC_k^i, is given by:

$$TC_k^i = \sum_{j=1}^{n} x_{jk} f_k^{i,j} C_k^{i,j} \tag{1}$$

The cost of serving requests for all the m objects at site i, denoted by TC^i, is given by

$$TC^i = \sum_{k=1}^{k=m} TC_k^i = \sum_{k=1}^{k=m} \left[\sum_{j=1}^{n} x_{jk} f_k^{i,j} C_k^{i,j} \right] \tag{2}$$

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

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

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 (3) subject to storage capacity, processing capacity, and minimum copy constraints. That is, we want to

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

Subject to

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

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

$$\sum_{k=1}^{m} x_{ik} . L_{ik} < P_i \quad \text{for all} 1 \leq i \leq n \tag{7}$$

$$x_{ik} \in \{0,1\}, \text{for all } i, j \tag{8}$$

In formula (5), the minimum number of safety copies for object k should be equal or greater that 1 (i.e. $b_k \geq 1$) This is necessary in case some failure of the servers may occur and/or we want 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.

4. Tabu Search

Tabu search is a well-known iterative procedure for solving discrete combinatorial optimization problems. It was first suggested by Glover [35] and since then, it has been successfully applied to obtain optimal and suboptimal solutions to such problems as scheduling, time tabling, travelling salesman, and layout optimization.

A general framework of Tabu search is given in Figure 1. Tabu search starts from some initial solution and attempts to determine a better solution in a manner of a "greatest descent neighborhood" search algorithm. The basic idea of the method is to explore the search space of all feasible solutions by a sequence of moves. A *move* is an atomic change, which transforms the current solution into one of its neighboring solutions. Associated with each move is a *move value*, which represents the change in the objective function value as a result of the move. Move values generally provide a fundamental basis for evaluating the quality of a move. At every iteration of the algorithm, an admissible best move is applied to the current solution to obtain a new solution to be used in the next iteration. A move is applied even if it is a non-improving one, i.e. it does not lead to a solution better than the current solution. To escape from local optima and to prevent cycling, a subset of moves is classified as tabu (or forbidden) for certain number of iterations. The classification depends on the history of the search, particularly as manifested in the *recency* or *frequency* that certain moves of solution components, called *attributes*, have participated in generating past solutions. A simple implementation, for example, might classify a move as tabu if the reverse move has been made recently. These restrictions are based on the maintenance of a short-term memory function (the *tabu list*), which determines how long a tabu restriction will be enforced, or alternatively, which moves are admissible at each iteration.

The *tabu tenure* (i.e. the duration for which a move will be kept tabu) is an important feature of tabu search, because it determines how restrictive the neighborhood search is. The tabu restrictions are not inviolable under all circumstances and a tabu move can be overridden under some conditions. A condition that allows such an override is called an aspiration criterion. For example, an aspiration criterion might allow overriding a tabu move if the move leads to a solution, which is the best obtained so far.

```
Tabu_search()
{
    initialize the short-term memory
    generate a starting solution s0
    s,s* = s0   /*s is current solution and s* is best solution found so
    far*/
    for(i=1; i<=maxtry; i++) {
        bestmovevalue=∞
        for (all candidate moves in the neighbourhood)
          if ( the candidate move is admissible) {
              obtain the neighbor solution s̄ by applying a candidate
              move to the current solution s
              movevalue=c(s̄)−c(s)    /* c(s)is the objective function
                                        to be optimized */
              if (movevalue<bestmovevalue){
                  bestmovevalue=movevalue
                  s'=s̄
              }
          }
    }
    update the short term memory function
    if (c(s')<c(s*)) s* = s'
    s=s'
}
```

Figure 1. A general Framework of Tabu Search

5. Object Placement and Replication with Tabu Search Algorithm

The replica placement problem described in section 4 reduces to finding a 0-1 assignment of the matrix X that minimizes the cost function (4) subject to a set of constraints (constraints (5) to (8)). The time complexity of this type of problems is exponential. In this section, we specialize the basic tabu search into a specific algorithm for the object allocation and replication problem. This implies turning the abstract concepts of tabu search, such as initial solution, solution space, neighborhood, move generaration, tabu criteria and others, into more concrete, problem, specific, and implementable definitions.

The core of the proposed tabu search algorithm for object replication is given in Figure 2. The input parameters to the algorithm are the number of objects (input parameter *no_of_objects*), number of servers in the network (input parameter *no_of_servers*), the network topology (input parameter *topology*), and maximum number of successful moves the algorithm makes before terminating (input parameter *maxmoves*).

The routine **tabu_search()** makes use of other routines to perform the changes in the solution space. Some important routines are **init_solution()**, **init_history()**, **compute_cost()**, **select_site()**, **select_best_move()** and **update_history()**. A detail description of these routines is given in the following sections. Other routines are self-explanatory and properly commented in the algorithm given in Figure 2. We also use two arrays N and R of sets that store the current allocation of all the objects to servers (called the *residence set*) and objects that are allocated to a specific node respectively.

The final output of the algorithm is an array of *residence sets* (a residence set R_k contains the server IDs on which object k is replicated). It is important to correctly define the parameters of the algorithm and to implement it as efficiently as possible to reduce the overall complexity and computation time. In the following sections, we describe how the important routines can be implemented.

```
residence_set
tabu_search(no_of_objects,no_of_servers,topology,maxmoves)
{
   object_set N[n];    //N[i] is set of objects assigned at site i
   residence_set R[m]; //R[i] is a set of nodes at which object i
                       // is assigned

   initial_solution(R,N,no_of_objects, no_of_sites); //initail
   schedule
   init_history(history);     //initialize history
   cost_value=compute_cost(R) //compute cost of residence set
   best_value=cost_value;     // used by aspiration criterion
   best_residence_set=R;      //set residence found so far
   nmoves=0;                  //moves made so far
   for(i=1; i<=maxtry; i++) { // maxtry > maxmoves
      siteinstance=select_site(no_of_servers);  //select a site
      neigh_size=get_neighbourhoodsize(n,topology);
      //select the neighborhood set of the selected site
      neighborhood=select_
      neighborhood(topology,siteinstance,neigh_size);
      object_instance=select_object(no_of_objects); //a random object

      best_move=select_best_move(no_of_objects, neighborhood,
                                          N[siteinstance], R)
      apply_move(best_move,R);
      new_cost=compute_cost(R);
      update(N,R)//update N w.r.t. new R
      if(not tabu(best_move.move) or new_cost < best_value)  {
           nmoves++;
           update_history(history,best_move.move);
           if (new_cost < best_value and ferasible(R)){
              best_residence_set=R;
               best_value = new_cost; }
      }
      else { undo_move(best_move,R); undo_object_set(best_move,N); }
      if nmoves>maxmoves break;
   } //for
   return best_residence_set;
}
```

Figure 2. The core of Tabu Search Task Scheduling Algorithm

5.1. Initial Solution

The routine **init_solution()** uses a greedy algorithm to get the initial solution. It proceeds as follows: For each object *k* whose storage and processing requirements are less than the storage and processing capacity of site 1, calculate the profit (in terms of cost function) of putting a copy of object *k* on site 1. Sort the objects in descending order of their profits. Starting with object 1 in the sorted list, replicate objects one by one on site *i* until all the objects are replicated or there is no more capacity at site *i* to hold any other object. Repeat the same procedure for remaining *m-1* sites. The complete algorithm is given in Figure 3. Note that this method ensures that the initial solution satisfies all the constraints and hence is a feasible solution.

```
initial_slution(R, N, no_of_objects, no_of_nodes)
{
   initilize(N);        // array of empty object sets
   initialize(R);       // array of empty residence sets
   for (i=1;i<=no_of_nodes;i++)
      for (j=1;i<=no_of_objects;j++){
      if (size[j] <= storage_capacity[i] and load[j]<=
      load_capacity[i])
      {
         object[j].profit=calculate_proft(i,j);
         object[j].id=j;
      }
      sort_descending(object);
      j=1;
      while (storage and load constraint satisfied) {
           assign_object(object[j].id,i);  //assign object to site i
           N[i]=N[i] ∪ {object[j].id}
           R[object[j].id]=R[object[j].id] ∪ {i}
           j++;
      } //while
   } //for
}
```

Figure 3. Algorithm to Find Initial Solution

5.2. Move Generation

For our replication algorithm, the set of all possible solutions consists of all the possible permutations of the objects subject to the constraints. To generate a new allocation, it is possible to move an object from one site to another site (object migration) or put an additional copy of the object on a node (object replication).

The proposed algorithm (shown in Figure 2) generates a new move in four steps. First, **select site ()** routine randomly selects a site. Then **find neighborhood ()** finds a subset of sites from the given network topology that should be considered for object replication and migration. In our simulation, it was observed that the neighborhood should include all the sites that are directly connected to the site selected by **select site ()** routine for efficient and better quality solution. However, generally, any number of nodes can be selected as a neighborhood. Then the routine **select_best_move ()** (given in Figure 4) selects the best valid move in the neighborhood set. The routine considers the following moves:

1. *(migrate,object,n,neighborhood)* – That is, migrating an object from site *n* to another site in the neighborhood.
2. *(replication,object,n,neighborhood)* – That is, replicating an object already residing on site *n* to another site in the neighborhood.
3. *(add,object,n)* – That is, adding an object not currently replicated at site *n*.

The best move selected by the routine is applied to the current solution to obtain the new solution by calling routine **apply move ()**.

```
move_struct select_best_move(n, neighbourhood, N, R)
{
   // n is randomly selected node
   // N is set of objects assigned to node n
   // R is the array of residence set
   // move_struct is struct that contain move and profit
   // as its attributes

    best_move.profit=0;
    for (each p in neighborhood) {
        profit=evaluate_move(N,transfer,i,n,p);
        if (profit > best_move.profit and move is valid) {
            best_move.profit=profit;
            best_move.move=(transfer,i,n,p);
        }
        profit=evaluate_move(replicate,i,n,p);
        if (profit > best_move.profit and move is valid) {
            best_move.profit=profit;
            best_move.move=(transfer,i,n,p);
        }
    } //for
    o=select_node_not_in(N);
    profit=evaluate_move(add,o,n);
    if (profit > best_move.profit and move is valid) {
        best_move.profit=profit;
        best_move.move=(transfer,i,n,p);
    }
   return best_move;
}
```

Figure 4. Algorithm to Select the Best move

5.3. Tabu Lists

The chief mechanism for exploiting memory in tabu search is to classify a subset of moves in the neighborhood as forbidden (tabu). This classification depends on the history of the search, particularly manifested in the recency or frequency that certain moves or solution components have participated in generating past solutions. We use the following tabu criteria to make certain moves tabu.

5.3.1 Recency Tabu

There are three kinds of moves in our algorithm as explained before The first one is (*migrate, object,n,m*) and has a reverse move expressed as (*migrate,object,m,n*). The second move is (*replicate,object,n,m*) with a reverse move (*replicate,object,m,n*). The last move is (*add,object,n*) and has a reverse move (*migrate,object,n,m*). A move is prohibited (or is tabu) if its reverse has been executed recently. Unlike standard tabu search in which the value of tabu tenure is fixed, we determine the tabu tenure of a move through a feedback (reactive) mechanism during the search. The tabu tenure of a move is equal to one at the beginning (the inverse move is prohibited only at the next iteration), and it increases only when there is evidence that diversification is needed, and it decreases when this evidence disappears. In detail, the evidence that diversification is needed is signaled by the repetition of previously visited configurations. All configurations found during the last I iterations of the search are stored in memory (use of a queue is a possible implementation strategy). After a move is executed, the algorithm checks whether the current configuration has already been found and it reacts accordingly (tabu tenure of the move increases if a configuration is repeated, it decreases if no repetition occurred during a sufficiently long period).

5.3.2. Same Cost Value

This is a powerful tabu to diversify the search when stuck in local optima. This tabu is triggered if the same value of the cost function has been obtained during the last c iterations, where c is specified by the user. When this tabu is switched on, all solutions with a cost value equal to the cost value selected during that period are tabued. The user can also specify the tenure of the tabu.

5.4. Aspiration Criterion

Tabu conditions based on the activation of some move attributes may be too restrictive and result in forbidding a whole set of unvisited moves which might be attractive. The aspiration criterion allows overriding a tabu move under certain conditions. The proposed algorithm overrides a tabu restriction if the move leads to a solution better than the best found so far. This is the only aspiration criterion used by the proposed algorithm.

6. Experimental Results

This section presents some performance measures obtained by simulation of the proposed algorithm. Since the object replication algorithm is NP-complete, computing the exact (optimal) solution is too computationally intensive to be useful in practice. Similar to other studies, we compare the performance of our algorithm with four well-known algorithms proposed in the literature. These are: random allocation algorithm [23], greedy algorithm [12], hot spot [23], and Max-router fan-out placement algorithm [24].

In our simulation, we generated two types of random network topologies: random trees and random graphs. The reason for selecting tree network topology is the fact that most of the distributed web server systems are organized as trees. We use the random graph topology to show the effectiveness and applicability of the proposed algorithm to any other type of topology that might be suitable for a particular web server system. To generate a random tree, we wrote a program that takes two parameters: the total number of nodes and the maximum degree of a tree node. Starting from the root node, we recursively create random children until the total number of nodes specified is reached. In our simulation, we used trees having 100-600 nodes and the maximum degree of 3-10 (a tree with a higher number of nodes has a higher degree).

To generate random graphs, we use the GT-ITM internetwork topology generator [36]. In particular we use three network models: pure random, Waxman, and Transit-Stub. In the pure random model, vertices are distributed at random locations in a plan and an edge is added between a pair of vertices with a probability p. In the Waxman model, the probability of an edge from node u to v is given by $p(u,v)=\alpha e^{-d/(\beta L)}$, where $0 < \alpha$ and $\beta \leq 1$ are the parameters of the model, d is the Euclidean distance from u to v and L is the maximum distance between any two nodes. The Transit-Stub model generates hierarchical graphs by composing interconnected transit and stubs (for details see [37]). We use a variety of parameters for each network model.

The total objects to be replicated were varied from 1000 to 2000. In each simulation run, we used different object sizes which follow a normal distribution. The average object size is taken as 10 KB and maximum size was taken as 100KB. About 64% of objects sizes were in the range of 2KB and 16KB. The storage capacity of a server was set randomly in such a way that total storage of all the servers was enough to hold at least one copy of each object at one of the servers. In each trial, we run the replica placement algorithms for 200,000 requests for different objects. We created log files by generating requests for objects for multiple sessions. This log file was used by the proposed algorithm to collect various statistics.

During a simulation run, each site keeps a count c of the total number of requests it receives for an object. The latencies are updated periodically for each replica using the formula $T = 1/(\mu - \lambda)$ where λ is the average arrival rate and μ is the average service time. Exponential service time is assumed with an average service rate of 100 transactions/second. The value of T is propagated to the clients in the shortest path spanning tree. The cost (latency) at different sites is computed as 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. The average number of iterations for tabu search algorithm was fixed from 15000 to 150000 depending on the problem size. The number of successful moves when the algorithm was forced to terminate was fixed at 1/2 of the total number of iterations. The minimum neighborhood size was taken equal to or more than the number of sites directly connected to a site whose neighborhood is to be searched.

To compare the algorithms, we run each of the algorithms for the same input parameters (network topology, network size, number of objects, size of each object, etc) as described above and compute the value of the cost function given by (3). Each algorithm is run for a number of times by varying different input parameters and the average cost value (also referred to as latency) is then compared. A similar approach has been used by almost all the other researchers.

In the first set of experiments, we compared the algorithms for tree topology. Figure 5 shows the average latency for different tree sizes. The figure shows that the average latency decreases for all the algorithms as the number of sites increases in the system. This is because of the fact that as the number of sites increases, more replica of an object can be placed. Also, note that the performance of the proposed algorithm is better (i.e. tabu search has the minimum latency value) than the other four algorithms demonstrating the effectiveness of the proposed algorithm. However, there were about 3.5% cases when tabu search produced inferior quality solution as compared to another algorithm. Also, there were about 24% cases when the proposed tabu search algorithm produced solutions which were more than 50% better than the closest ones found by any other algorithm. The figure 6 shows the average performance of the algorithms for all the simulation runs for tree topologies with 100-600 nodes. The figure clearly demonstrates that the proposed algorithm outperforms the rest of the four algorithms.

In the second set of experiments, we repeated our simulation for pure random, Waxman, and Transit-Stub graph topologies. The results (figure 7 and 8) show a trend similar to the results obtained for tree topology. The proposed algorithm performs better than the other four algorithms. The average of all the simulation runs (both for trees and graphs) are shown in figure 9 and figure 10. These results show that the proposed algorithm out-performs all other algorithm, no matter what network topology and other parameters are used. We also noted in our simulation that when the network topology is changed while keeping all the other parameters constant, the proposed algorithm was able to find the solutions better than other four algorithms in more than 95% of cases. Also, the change in the bandwidth did not have any negative impact on the relative performance of the proposed algorithm.

In our simulation, we also studied the effect of change in the access frequencies on different algorithms and the results of some simulation runs are shown in figure 11 and 12. The results were obtained by evaluating the percentage improvement in the latency (cost function) when the object access frequencies (i.e. the request arrival rate at various nodes) were changed, for a simulation run. The request arrival rate for objects at the network nodes were changed either randomly, by a constant factor or it follows a Poisson distribution. The proposed algorithm adjusted the replication of object better than all the other algorithms and was able to find better object replication matrix consistent with the results reported in this paper.

Conclusions

Object replication on a cluster of web servers is a promising technique to achieving better performance. However, one needs to determine the number of replicas of each object and their locations in a distributed web server system. Choosing right number of replicas and their locations is a non-trivial problem. In this paper, we modeled the object replication and allocation as a 0-1 optimization problem. Then a tabu search algorithm is proposed to obtain solution to this problem. A detailed description of the proposed algorithm and its implementation considerations are discussed. The proposed algorithm has been compared with four other algorithms through a simulation study. A comparison of the proposed algorithm with four other algorithms demonstrates the superiority of the proposed algorithm.

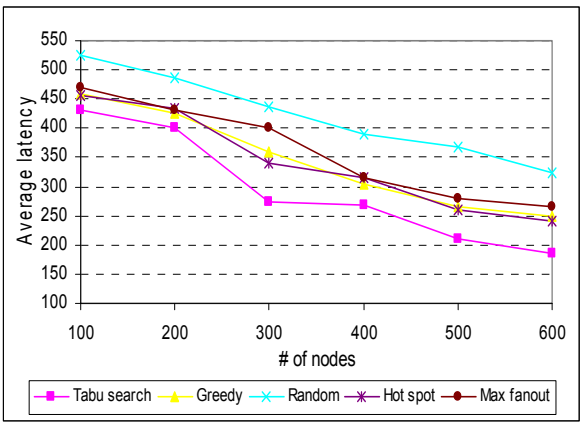

Figure 5. Average Latency for Different Tree Sizes

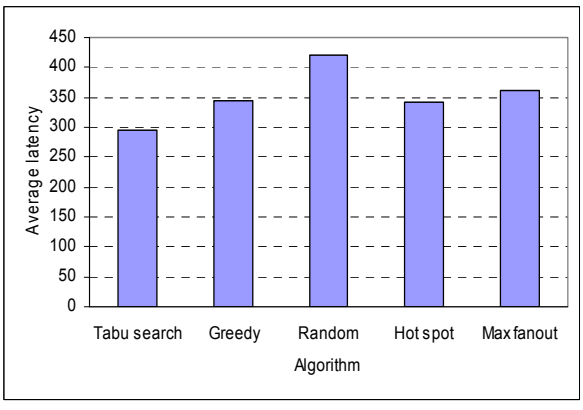

Figure 6. Average Latency for all Simulation Runs for Tree Topology

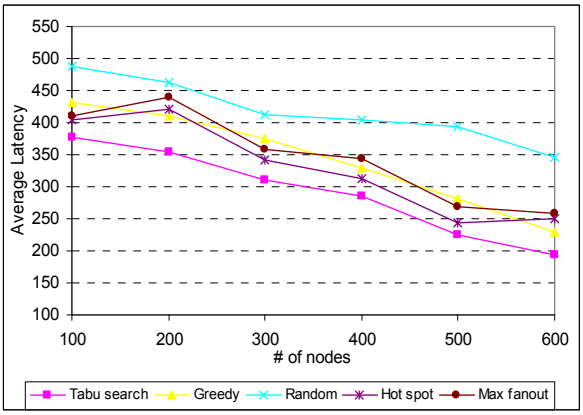

Figure 7. Average Latency for Different Graph Sizes

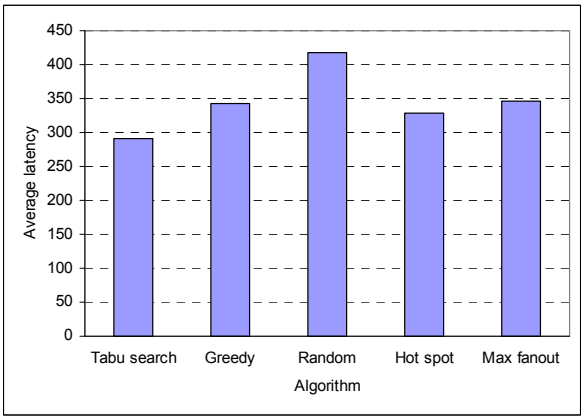

Figure 8. Average Latency for all Simulation Runs for Graph Topologies

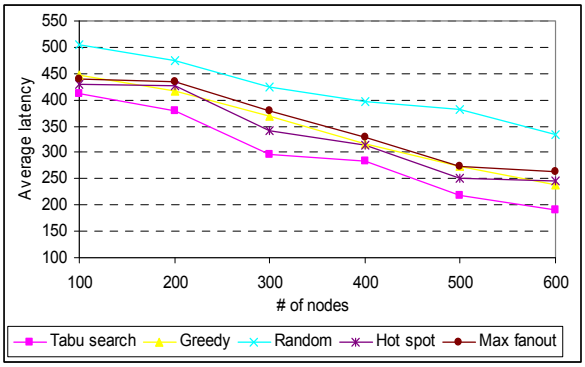

Figure 9. Average Latency for all Simulation Runs

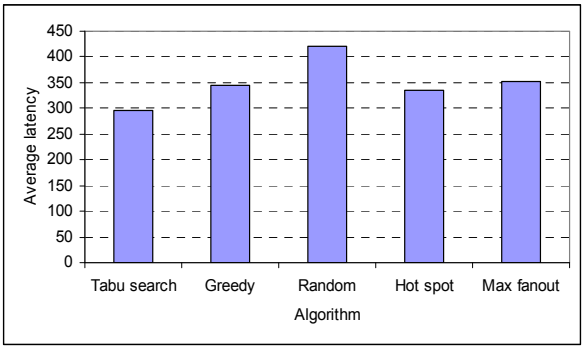

Figure 10. Average Latency for Different Algorithms for all the Simulation Runs

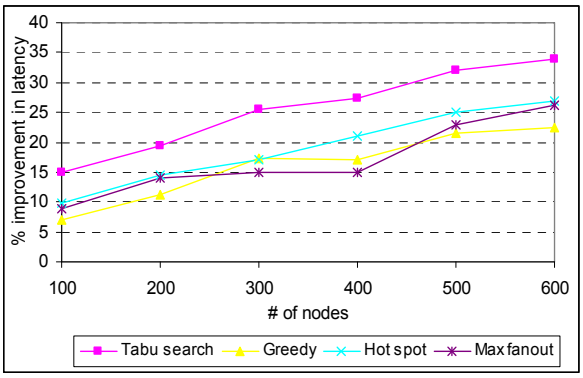

Figure 11. Percentage Improvement for Different Network Sizes in Latency when Object Arrival Rates are Changed

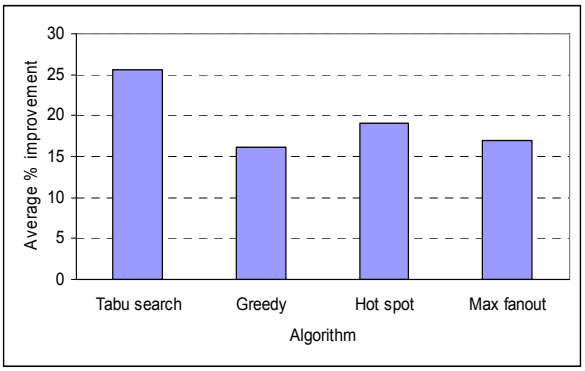

Figure 12. Percentage Improvement in Latency for all the Simulation Runs

REFERENCES

1. ZHUO, L., WANG, C-L. and LAU, F. C. M., **Document Replication and Distribution in Extensible Geographically Distributed Web Servers**. J. of Parallel and Distributed Computing, Vol. 63, 2003, No. 10, pp. 927-944.
2. CARDELLANI, V., CASALICCHIO, E., COLJANI, M., and YU, P. S., **The State of the Art in Locally Distributed Web-Server Systems**, IBM Research report RC22209, 2001.
3. PHOHA, V. V., IYENGAR S. S. and KANNAN, R., **Faster Web Page Allocation with Neural Networks**. IEEE Internet Computing, Vol. 6, No. 6, 2002, pp. 18-25.
4. KALPAKIS, K., DASGUPTA, K. and WOLFSON, O., **Optimal Placement of Replicas in Trees with Read, Write and Storage Costs**. IEEE Trans. On Parallel and Distributed Systems, Vol. 12, No. 6, 2001, pp. 628-637.
5. CARDELLINI, V. COLAJANNI, M., and YU, P. S., **Dynamic Load Balancing on Web-Server Systems**. IEEE Internet Computing, Vol. 3, No. 3, 1999, pp. 28-39.
6. COLAJANNI, M., YU, P. S., **Analysis of Task Assignment Policies in Scalable Distributed Web Server Systems**. IEEE Trans. On Parallel and Distributed Systems, Vol. 9, No. 6, 1988, pp. 585-600.
7. KWAN, T. T., MCGRATH, R. E. and REED, D. E., **NCSA's World Wide Web Server: Design and Performance**. IEEE Computer, Vol. 28, No. 11, 1995, pp. 68-74.
8. BAKER, S. M. and MOON, B., **Scalable Web Server Design for Distributed Data Management**. Proc. Of 15th Int. Conference on Data Engineering, Sydney, March 1999, pp. 96-110.
9. LI, Q. Z. and MOON, B., **Distributed Cooperative Apache Web Server**. Proc. 10th Int. World Wide Web Conference, Hong Kong, May 2001.
10. RISKA, A. Sun, W., SMIMI, E, and CIARDO, G., **ADATPTLOAD: Effective Load Balancing in Clustered Web Servers under Transient Load Conditions**. Proc. 22nd Int. Conf. on Distributed Systems, Austria, July 2002.
11. LI, B., **Content Replication in a Distributed and Controlled Environment**. J. of Parallel and Distributed Computing, Vol. 59, No. 2, 1999, pp. 229-251.
12. TENZAKHTI, F., DAY, K. and OLUD-KHAOUA, M., **Replication Algorithms for the Word-Wide Web**. J. of System Architecture, Vol. 50, 2004, pp. 591-605.
13. PORTO, S. C. S. and RIBEIRO, C. C., **A Tabu Search Approach to Task Scheduling on Heterogeneous Processors Under Precedence Constraints**. Int. J. of High Speed Computing, Vol. 17, No. 2, 1995, 110-123.
14. DOWDY, L. and FOSTER, D., **Comparative Models of the File Assignment Problem**. Computer Surveys, Vol.14, No. 2, 1982, pp. 287-313.
15. APERS, P. G. M., **Data Allocation in Distributed Database Systems**. ACM transactions on Database Systems, Vol. 13, No. 3, 1988, pp. 263-304.
16. CERI, S., MARTELLA, G. and G. PELAGATTI, G., **Optimal File Allocation in a Computer Network: A Solution Method Based on Knapsack Problem**. Computer Networks, Vol. 6, No. 11, 1982, pp. 345-357.
17. FISHER, M. K. and HOCHBAUM, D. S., **Database Location in Computer Networks**. J. ACM, Vol. 27, No. 10, 1980, pp. 718-735.
18. CHANG, S. K. and LIU, A. C., **File Allocation in Distributed Database**. Int. J. Computer Information Science. Vol. 11, 1982, pp. 325-340.
19. AWERBUCH, B., BARTAL, Y. and A. FIAT, A., **Competitive Distributed File Allocation**. Proc. 25th Annual ACM Symposium on Theory of Computing, Victoria, May 1993, pp. 164-173.
20. GAVISH, B. and SHENG, O. R. L., **Dynamic File Migration in Distributed Computer Systems**. Comm. of ACM, Vol. 33, No. 1, 1990, pp. 177-189.

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

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

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

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

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

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

27. 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.

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

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

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

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

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

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

34. BARTEL, Y., **Distributed Paging**. Proc. Dagstuhl Workshop On-line Algorithms, 1997, pp. 164-173.

35. GLOVER, F., **Tabu search: a Tutorial.** Interfaces. Vol. 20, No. 1, 1990, pp. 74-94.

36. CALVERT, K. AND ZEGURA, E., **GT-Internetwork Topology Models (GT-ITM)**. http://www.cc.gatech.edu/fac/ellen.zegura/gt-itm.

37. ZEGURA, E. W., CALVERT, K. and BHATTACHARJEE, S., **How to Model an Internetwork**, INFOCOM'96, 1996.