Object Replication in Distributed Web Server Systems with a Hybrid Tabu Search Algorithm

Amjad Mahmood and Taher, S. K. Homeed

Faculty of Information Technology, University of Bahrain, Kingdom of Bahrain
amahmood@itc.uob.bh, tskhomeeed@itc.uob.bh

ABSTRACT

One of the key issues in the design of a distributed web server system (DWS) is determining the optimal number of replicas and their placement on the web servers. This paper presents a hybrid tabu search (HTS) algorithm for replica placement in a DWS environment. We model the object replication problem as a 0-1 optimization problem and specialize the tabu search into a specific algorithm for solving this 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. In addition, we hybridize the tabu search algorithm with simulated annealing algorithm to speed up the convergence time of the algorithm without compromising the solution quality. Through a simulation study and comparison with well-known replica placement algorithms, we demonstrate the applicability and effectiveness of our hybrid algorithm.

Keywords: *Tabu Search, Simulated Annealing, Object Replication, WWW, Distributed Web-Servers, Data Replication*

1. INTRODUCTION

As the usage of web services grows, the number of accesses to many popular web sites is ever increasing and occasionally reaches the limits of their capacity. As a consequence, end-users of these web sites often experience poor response time or denial of service (time-out error). For certain types of web applications (e.g. e-commerce), this could result in a sizeable revenue losses. Therefore, the system administrators of these sites are constantly faced with the need to scale up the site capacity to offer better service to their clients.

To construct a powerful web server system, one could either use a powerful machine with advanced hardware support and optimized server software, or a collection of machines working together as a distributed web server system (DWS) [17]. The first approach is expensive one and the issue of scalability and performance may persist with ever increasing user demand. On the other hand, a DWS is not only cost effective and more robust against hardware failure but it is also easily scalable to meet increased traffic by adding additional servers when required. The performance of such systems (e.g. latency, throughput, availability, hop counts, link cost, and delay) can further be improved by maintaining multiple copies of objects at various locations [6,17]. Incoming requests are distributed to the web servers via switches or DNS servers [2]. Many popular web sites have already employed replicated server approach which reflects upon the popularity of this method [10].

Choosing the right number of replicas and their locations 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 [7,15]. Therefore, all the replica placement approaches proposed in the literature are heuristics that are designed for certain systems and work loads.

There has been a number of studies on replication of Web contents. Wolfson et al. [16] proposed an adaptive data replication algorithm which can dynamically replicate an object 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. It, however, does not consider the issue of multiple object replications. Furthermore, the performance of their algorithm for general network topology is not clear. Bestavros [1] formulated the problem 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. Tenzakhti et al. [15] proposed 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 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/site.

Heddaya and Mirdad [4] 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. Mahmood [12] proposed a series of algorithms for object replication in distributed web server systems. The author, however, considers the read requests only on a tree-like topology. Optimal placement of replica in trees has also been studied by Kalpakis at el. [6]. Due to successful application of meta-heuristics (e.g. genetic algorithm, tabu search etc.) to a variety of optimization problems, these algorithms have also been used to solve object replication problem. Sayal el al. [11] 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. A tabu search algorithm for object replication is proposed in [13].

This paper extends the work presented by the authors in [13] and proposes a hybrid tabu search (HTS) algorithm for replica placement in a DWS environment. The major motivation behind proposing a HTS algorithm 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. In addition, we hybridize the tabu search algorithm with simulated annealing algorithm to speed up the convergence time of the algorithm without compromising the solution quality.

2. THE SYSTEM MODEL

A distributed Web server system consists of a number of sites interconnected by a communication network. A unit of data to be replicated is referred as an object. An object can be an XML/HTML page, an image file, a relation, etc. Each object is identified by a unique identifier and may be replicated on a number of sites. The objects are managed by a group of processes called replicas, executing at replica sites. We assume that the network topology can be represented by a graph $G(V, E)$. Each node in the graph corresponds to a router, a switch or a web site. We assume that out of a total N nodes there are n web servers as the information provider. Associated with every node $v \in V$ is a set of non-negative. 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 non-negative distance (which can be interpreted as latency, link cost, or hop count, etc.).

A client initiates a read operation for an object k by sending a read request for object k. The request goes through a sequence of hosts via their attached routers to the server that can serve the request. The sequence of nodes that a read request goes through is called a routing path, denoted by π. The requests are routed up the tree to the home site (i.e. root of the tree). Note that a route from a client to a site forms a routing tree along which document requests must follow. Focusing on a particular sever i, the access traffic from all nodes leading to a server can be best represented by a tree structure if the transient routing loop is ignored [10]. 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 (clients) that are in the path π of the requests from clients to access object k at site i.

2.1. OBJECT REPLICATION MODEL

We consider a centralized object replication model in which there is a central site 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 site re-configures the system by adding and/or removing replicas according to the new placement scheme. The location of each replica is broadcasted to all the sites. In addition, each site i keeps the following information:

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

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

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

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

Each read request for an object is executed at only one of the replicas, the best replica. If $\aleph_k$ is the set of sites that have a replica of object k and C_k^{i,LC_k^i} denotes the cost of accessing object k at site i from the least cost site (denoted by LC_k^i), then

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

$LC_k^i = j$ such that $C_k^{i,j}$ is minimum overall $j \in \aleph_k$, 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.

2.2. THE COST MODEL

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 m objects to be replicated. 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. If X is an $n \times m$ matrix whose entry $x_{ik} = 1$ if object k is stored at site i and $x_{ik} = 0$ otherwise, then the cost of serving requests for object k $(1 \le k \le m)$ at site i $(1 \le i \le n)$ is given by:

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

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

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

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

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

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

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

Subject to

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

$$\sum_{k=1}^{m} x_{ik} . L_{ik} < P_i \quad \text{for all } 1 \le i \le n \quad (6)$$

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

If s_k denotes size of object k and TS_i is the total storage capacity of site i then the first constraint specifies that the total size of all the objects replicated at node i should not exceed its storage capacity. The second constraint specifies that the processing load brought by all the objects assigned at node i (L_{ik} denotes the processing load of object k at node i) should not exceed the total capacity of a node (denoted by P_i).

3. THE ALGORITHM

In this section, we specialize the basic tabu search [3] into a specific algorithm for the object replication problem described in section 2. This implies turning the abstract concepts of tabu search, such as initial solution, solution space, neighborhood, move generation, tabu criteria and others, into more concrete and implementable definitions.

In standard tabu search algorithm [3], every non-improving move is accepted if it is not tabued. This increases the solution search space and consequently tabu search may take longer time to produce a solution Therefore, to reduce the solution search space and improve the efficiency of the proposed algorithm, we accept a non-improving move with a probability which is a function of temperature, the best objective function value found so far and the new value of the objective function after the move is applied to the current solution (an idea borrowed from simulated annealing [5]).

The core of the proposed hybrid tabu search algorithm for object replication is given in Figure 1. The input parameters to the algorithm are the number of objects (input parameter no_of_objects), number of servers in the network (no_of_servers), the network topology (topology), maximum number of successful moves the algorithm makes before terminating (maxmoves) and initial value of temperature (input parameter temperature). The final output of the algorithm is an array of residence sets (a residence set Rk contains the server IDs on which object k is to be replicated).

The tabu_search() makes use of other routines to perform the changes in the solution space (these are select_site,get_neighborhoodsize,select_neighborhood, select_object, select_best_move and apply_move), the evaluation of a solution (using compute_cost), updating checking whether a non-improving move should be accepted (using isvalid), updating the tabu list (using update_history), rejecting a move (using undo_move) and generating initial solution and initializing tabu list (using initial_solution and init_history respectively). A detail description of these routines is given in the following sections. We also use two arrays *N* and *R* of sets that store the current location of all the objects (called the residence set) and objects that are allocated to a specific node, respectively. The algorithm terminates when either the maximum number of iterations has been completed or maximum allowed moves (controlled with variable maxmoves) have been made. The algorithm may also be terminated if no successful move has been made for last *n* iterations (*n* maybe a user supplied parameter).

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.

3.1. 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. Note that this method ensures that the initial solution satisfies the storage and load constraints (e.g. constraint 1 and 2) and hence is a feasible solution.

```
residence_set  tabu_search(no_of_objects, no_of_servers, topology,maxmoves,temperature)
{
        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);      //initial allocation
        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;                         //residence set found so far
        nmoves=0;                                      //moves made so far
        naccepted=0;                                  //no of non-imrpoving moves accepted so far
        for(i=1; i<=maxtry; i++) {                    // maxtry > maxmoves
            siteinstance=select_site(no_of_servers);  //select a site
            neigh_size=get_neighborhoodsize(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);
            update(N,R)                        //update N w.r.t. new R
            new_cost=compute_cost(R);   delta=new_cost-best_value;
            if (delta < 0 and feasible(R) )                    //aspiration criteria
            {
                nmoves++;
                update_history(history,best_move.move);
                best_residence_set=R;  best_value = new_cost
            }
            else {
                valid=isvalid(delta,temperature);    // acceptance criteria of simulated annealing
                if (valid and feasible(R)) {                      //conditionally accept the move
                        nmoves++; naccepted++
                        update_history(history,best_move.move);
                } else { undo_move(best_move,R); undo_object_set(best_move,N); }
            if (nmoves > maxmoves)  break;
            if (naccepted >=MAXCHANGE) {
                    temperature *=alpha;                      //update temperature
                    naccpeted=0;
            }
        } //for
        return best_residence_set;
}
```

Figure 1: The core of hybrid tabu search object allocation and replication

```
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 2: Algorithm to select the best move

3.2. MOVE GENERATION

For our replication algorithm, set of possible solutions consists of all 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 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. Then the routine select_best_move() selects the best valid move in the neighborhood set (see Figure 2). 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().

3.3. 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.

3.3.1 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).

3.3.2. 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 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 this tabu.

3.4. ASPIRATION CRITERION

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.

3.5. ACCEPTANCE OF NON-IMPROVING MOVES AND CONTROL PARAMETER T

The standard tabu search algorithm generates a move and accepts it even if it is a non-improving one (provided it is not tabu). However, accepting moves which are far from the current best solution may make the algorithm to evaluate many inferior solutions resulting in high number of moves before producing any solution. In contrast to the tabu search algorithm, simulated annealing (SA) algorithm [9] selects a non-improving solution with a probability that is a function of temperature and function values for the best solution found so far and the new solution obtained after the move.

The probability is generally computed following the Boltzmann distribution. That is, if $CC(X_{best})$ is the function value of the best solution found so far and $CC(X_{new})$ is the function value of the new solution obtained by applying the best move, then probability of accepting a non-improving move (p) is given by:

$$p = e^{-\Delta/T} \tag{8}$$

Where

$$\Delta = CC(X_{best)}) - CC(X_{new}) \tag{9}$$

The temperature T is decreased during the search process, thus at the beginning of the search the probability of accepting uphill moves is high and it is gradually decreased to a simple iterative improvement method. The choice of an appropriate cooling schedule is crucial to the performance of the algorithm. The cooling schedule defines the value of T_k at each iteration k (or after a number of iterations). One of the most commonly used cooling schedule follows a geometric law: $T_{k+1}=\alpha T_k$, where $\alpha \in (0,1)$, which corresponds to an exponential decay of the temperature. In our algorithm, we use $\alpha = 0.9/0.95$ as proposed in [8].

The proposed algorithm accepts a maximum number of non-improving moves (proportional to the dimension of the problem) for each value T_k of the control parameter. We have chosen to limit the number *MAXCHANGE* of moves that are accepted for a particular value of T_k before it is lowered proportionally to T_k according to the formula $T_{k+1}=\alpha T_k$.

It is particularly important that T_0, the starting value of T, be sufficiently large to allow almost all non-improving move be accepted at the start of process and then lowering it down in such a way that almost no non-improving move is accepted towards the end of the process. A simple way of empirically selecting the starting temperature is to initially sample the search space with a random walk to roughly evaluate the average and the variance of the objective function values [5].

4. EXPERIMENTAL RESULTS

This section presents some performance measures obtained by simulation of the proposed algorithm. In each simulation run, we model the web as a tree having 100-600 nodes. The total objects to be replicated were 2000 in all the simulation runs. The average object size was taken as 10 KB and maximum size was taken as 100KB and follow the pareto distribution. 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.

During a simulation run, latencies are calculated as described in [15] Exponential service time is assumed with an average service rate of 100 transactions/second. At the end of every 20,000 requests, the mean latency of all the requests is calculated and used as a performance measure. The number of iterations was fixed from 5000 to 50000 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. The starting temperature was selected using a random walk in the solution space of different problems.

We studied the performance of our proposed algorithm and compared it with that of random allocation algorithm [14], greedy algorithm [15] and hot spot [14]. Figure 3 shows the average latency for all the simulation runs for different tree sizes. The figure shows that the average latency decreases for all the algorithms as the number of sites increases in the system. This is because of the fact that as the number of sites increases, more replica of an object can be placed. Also, note that the performance of the proposed algorithm is better than other three algorithms demonstrating the effectiveness of the proposed algorithm. Figure 4 shows the average performance of the algorithms for all the system configurations. It is evident that the proposed algorithm performs, on average, better than the greedy, random algorithm and hot spot.

We also studied the effect of change in the access frequencies. The access frequencies for objects were changed by -20% to 50%. Each algorithm was run again to find the new replication schemes and percentage savings in the cost function. The results are shown in figure 5 and figure 6. It is evident from the results that the proposed algorithm adjusted the replication schemes better than all the other algorithms.

We also compared the proposed hybrid tabu search with the standard tabu search algorithm. We observed that the hybrid algorithm was able to find better or same quality solution in almost all the cases. In about 68% cases, the standard tabu search was able to find the solutions of the same quality as that of hybrid algorithm after making, on an average of 19.3%, extra moves as compared to the hybrid algorithm. This show the effectiveness of hybridizing the tabu search with simulated annealing algorithm.

We also studied the convergence of the proposed algorithm by starting with different initial solutions. The simulation results showed that in more than 95.8% of cases, the proposed algorithm was able to find (sub)optimal solutions. This shows that the proposed algorithm converges to the (sub)optimal solution regardless of the quality of the initial solution. However, when the proposed algorithm for generating initial solution was used to obtain the starting solution, the hybrid tabu search found the (sub)optimal solutions, in almost 39% cases, with fewer number of moves. This justifies the use of a good starting solution.

We also studied the effect of neighborhood size on the accuracy of the proposed algorithm. The simulation results show that as the neighborhood size increases, the algorithm finds better or same quality solutions in less number of moves. The worst performance was observed when a single move is randomly generated in the neighborhood of the current solution. When a best move is selected by evaluating all the moves of an object in the neighborhood containing all the nodes directly connected to the selected node, (sub)optimal solutions were obtained in less number of moves. Increasing the size of the neighborhood further did not show any significant improvement in the solution quality.

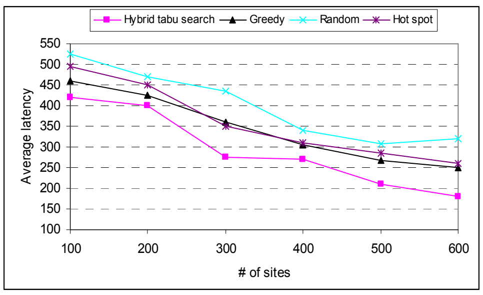

Figure 3: Mean latency for different tree sizes

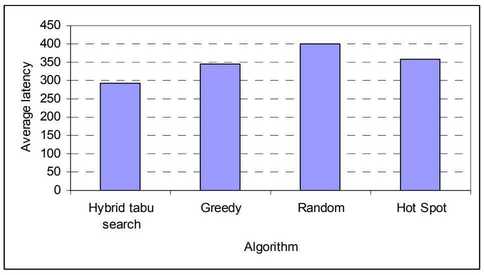

Figure: Average latency for all simulation runs

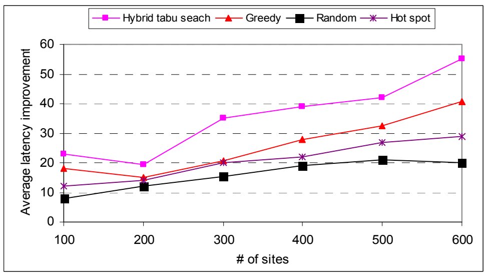

Figure 5: Average improvement

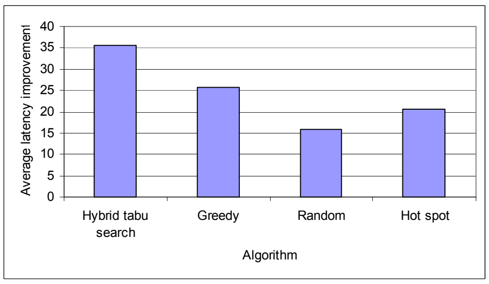

Figure 6: Average improvement for all runs

6. CONCLUSIONS

Object replication in a distributed web server system is a promising technique to achieving better performance. In this paper, we modeled the object replication as a 0-1 optimization problem. Then a hybrid 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 three other algorithms through a simulation study. A comparison of the proposed algorithm demonstrates the superiority of the proposed algorithm.

REFERENCES

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

[2] 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, pp. 585-600, 1988.

[3] Glover, F., "Tabu Search: a Tutorial", *Interfaces*, vol. 20, no. 1, pp. 74-94, 1990.

[4] Heddaya, 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*, pp. 160-168, 1997.

[5] Ingber, L., "Adaptive Simulated Annealing (ASA): Lessons Learned," *J. Control and Cybernetics*, vol. 25, pp. 33-54, 1996.

[6] 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, pp. 628-637, 2001.

[7] Karlsson, M. and Karamanolis, C., "Choosing replica Placement Heuristics for Wide-Area Systems", *International Conference on Distributed Computing Systems*, available at http://www. hpl.hp.com/personal/Magnus_Karlssn, 2004.

[8] Kirkpatrick, S. and Gellat, C. D., "Optimization by Simulated Annealing: Quantitative Studies," *J. Statistics of Physics*, vol. 34, pp. 975-986, .1984.

[9] Kirkpatrick, S., Gellat, C. D., and Vechhi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-680, 1983.

[10] Li, B., "Content Replication in A Distributed and Controlled Environment," *J. of Parallel and Distributed Computing*, vol. 59, pp. 229-251, 1999.

[11] Sayal, M., Breitbart, Y., Scheurermann. P. and Vingralek, R., "Selection of Algorithms for Replicated Web Sites," *Performance Evaluation Review*, vol. 26, No. 1, pp. 44-50, 1998.

[12] Mahmood, A., "Object Replication Algorithms for World Wide Web," *Computing and Informatics*, vol. 24, no. 4, pp. 371-390, 2005.

[13] Mahmood, A., "A Tabu Search Algorithm for Object Replication in Distributed Web Server Systems," *Studies in Informatics and Control*, vol. 14, no. 2, pp 85-98, 2005.

[14] Qiu, L., Padmanabham, V. N. and Voelker, G. M., "On the Placement of Web Server Replicas," *Proc. of 20th IEEE INFOCOM, Anchorage, USA*, pp. 1587-1596, 2001.

[15] Tenzakhti, F., Day, K. and Olud-Khaoua, M., "Replication Algorithms for the Word-Wide Web," *J. of System Architecture*, vol. 50, pp. 591-605, 2004.

[16] Wolfson, O., Jajodia, S. and Huang, Y., "An Adaptive Data Replication Algorithm," *ACM Trans. Database Systems*, vol. 22, no. 2, pp. 255-314, 1997.

[17] 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, pp. 927-944, 2003.