Placement of Web-Server Proxies with Consideration of Read and Update Operations on the Internet

XIAOHUA JIA[1,3], DEYING LI[1], XIAODONG HU[1], WEILI WU[2] AND DINGZHU DU[2]

[1]*Department of Computer Science, City University of Hong Kong*
[2]*Department of Computer Science, University of Minnesota, USA*
Email: jia@cs.cityu.edu.hk

This paper investigates the optimal placement of proxies of a Web server on the Internet, with the consideration of both read and update operations to the data on the Web server. We first study the problem of optimal placement of k proxies in a system to minimize the total access cost to the Web server. Then, for an unknown number of proxies, we find the optimal number of proxies required in the system. The problems are formulated by using the dynamic programming method and the optimal solutions are obtained. Simulations have been conducted to evaluate the performance of the proposed algorithms and to demonstrate how the effectiveness of proxy placement is affected by various factors, such as network traffic load, number of proxies, read–write ratio and proxy hit ratio.

Received 18 April 2002; revised 25 October 2002

1. INTRODUCTION

With the fast growth of Web-related traffic on the Internet, most users have experienced the frustration of very slow responses when accessing some Web sites. Caching is a widely adopted technology to alleviate traffic congestion and improve the response time of Web servers. There are basically two types of Web caching [1]: client-based caching and server-based caching. Client-based caching is done at client proxies, which cache the documents recently accessed by the clients. Server-based caching is done at server proxies (also called edge-servers). A server proxy stores (partially) replicated data of a Web site and acts as a 'front end' of this Web server to its clients. When a client's request to access a Web server is intercepted by this server's proxy on its way to the server, the request will be served by the proxy, provided the requested data is available at this proxy. Server proxies aim to improve client response times, distributing the workload of the Web server and reducing the network traffic. This paper discusses the placement of server proxies. In the rest of the paper, proxies refer to server-based proxies.

Web pages maintained by a server need to be updated constantly, sometimes at a very high frequency. This is particularly true for those Web servers that provide on-line trading information or on-line auctions. By placing proxies in the network, it would take clients less time to retrieve data from the Web server. However, it would also increase the cost to keep the data at proxies consistent with the server whenever the data is updated. This paper discusses two sub-problems of the placement of proxies on the Internet.

(1) Given k proxies, find the optimal placement of the proxies in the network, such that the overall access cost (including both read and update costs) is minimized.
(2) For an unconstrained number of proxies, find the optimal number of proxies and their placement, such that the overall access cost is minimized.

We formulate the problems by using the dynamic programming method. Optimal results are obtained in polynomial time. Simulations have been conducted to evaluate the performance of the proposed algorithms and to demonstrate how the effectiveness of proxy placement is affected by various factors, such as network traffic load, number of proxies, read–write ratio and proxy hit ratio.

2. PROBLEM FORMULATION

The network is modeled by a connected graph $G(V, E)$, where V is a set of nodes and E the set of links of the network. For a link $(u, v) \in E$, $d(u, v)$ is the distance of the link. It can be the number of hops. Suppose our concerned Web server is s. Each node v is associated with a non-negative number $r(v)$, which is the access frequency to s. $r(v)$ can be the number of accesses during a certain period of time, combined with the data size retrieved from the server. Server s has an update frequency w, which is the number of update operations (combined with the updated data size) during a certain period of time. We assume that

[3]Corresponding author. Currently an adjunct professor with Computing School, WuHan University, Peoples Republic of China.

update operations must first be performed on the original data copies stored at the Web server, where they are then propagated to the proxies. That is, updates only come from the server and the proxies act as caches for data retrieval. This is a common configuration for a Web server and its proxies, because updating the contents of a Web page usually requires authorized administrators' approval. The proxies discussed in this paper are not mirrored Web servers [2]. Our problem is to find the optimal placement of proxies in the network such that the overall cost of accesses to s can be minimized.

The implied routing method in the Internet is the shortest path routing. It always takes the shortest paths to transmit requests from clients to s and to send the data back to the clients. Thus, if the routing is stable, the routes to all clients as viewed by s form a shortest path tree, where the root of the tree is s and the rest of the tree nodes are clients. Let T_s denote such a tree induced from the original network graph. Our problem is hence reduced to the placement of proxies in tree T_s.

The proxies discussed in this paper are *transparent en-route proxies* [3, 4, 5], where proxies are only placed along the routes from clients to the server and are transparent to the clients. A typical configuration is that proxies are co-located with routers and are maintained by network providers. This configuration has significant operational benefits, because proxies can be introduced easily into the existing network infrastructure. Moreover, *en-route* proxies are easier to manage and have less administration cost than replicated Web servers (mirrored Web servers). Almost all existing caching products include such a transparent operation mode [6].

In a real operation, an *en-route* proxy intercepts any request that passes through it and it either satisfies the request or forwards the request toward the server along the regular routing path. That is, if a client's request meets a proxy on its way to the server and the requested data is available at this proxy, it will be served directly by the proxy. Otherwise, the request has to go all the way to the server and is served by the server. It is important to note that a client may not be served by a nearby proxy if the proxy is not on the route between the client and the server, even though the client is physically close to the proxy. Taking an example in Figure 1, there are three proxies (in gray) placed in the tree. When a client at v_1 accesses s, the request will meet proxy p_1 and be served by p_1. However, a request from v_2 has to go all the way up to s and to be served there (even though v_2 is close to p_1), because p_1 is not on the route from v_2 to s.

Web pages on a server s may need to be updated constantly. When a Web page is updated, the updated data have to be transmitted to all proxies to keep the data stored at the proxies consistent with server s. There are basically two models for s to transmit updated data to the proxies: the multicast model and the unicast model. By using the multicast model, s multicasts the data to all proxies via a shortest path tree (SPT), where only one copy of the data traverses over the SPT to reach all proxies. The SPT is an induced graph of T_s, which contains all the proxies and the leaf nodes of the SPT are only proxies. While with the unicast model, s sends a copy of the updated data to each of the proxies individually. In this paper, we adopt the multicast model, because it is more efficient than the unicast model for multicasting data. Notice that the algorithms developed in this paper can be easily modified for the unicast model. Figure 2 is the SPT used by s to multicast updates to the proxies in the tree of Figure 1. When s multicasts data to p_1, p_2 and p_3, only one copy of the data traverses from s to v_3, where it is further forwarded to p_1 and p_2 individually.

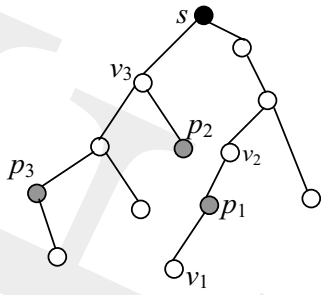

FIGURE 1. Access paths to a server with proxies.

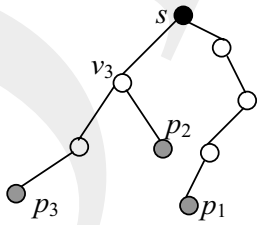

FIGURE 2. Shortest path tree for updating proxies.

By placing proxies at the proper locations in the network, it would enable most of clients' requests to be served at proxies, without letting them go further to the server. This would significantly reduce the traffic flow in the network and off-load the server. The number of proxies to be placed in the system will be a trade-off between the cost of data retrieval by clients and the cost of data updating by s.

We define the overall cost of accesses to server s as below. Let $\pi(u, v)$ be the path connecting u and v in T_s. We extend the distance function between nodes u and v along path $\pi(u, v)$ as:

$$d(u, v) = \sum_{(x,y)\in\pi(u,v)} d(x, y).$$

Let $p(v, s)$ denote the first proxy that is met while going from client v to s along tree T_s. $p(v, s)$ could be v itself when a proxy is placed at v, or it could be s when no proxy is met on its way to s. Suppose the hit ratio of a proxy is ρ. That is, $\rho r(v)$ of client's requests can be served by a proxy if the proxy is met and the remaining $(1-\rho)r(v)$ requests will be served by the server. Here we assume the upstream flow from a proxy has a hit ratio zero (i.e. a request will be served by the server if it is missed by a proxy). The hit ratio is an average value of many accesses to a proxy measured over a long period of time. We assume the hit ratios of all proxies

and for all clients are the same (i.e. ρ). Thus, the cost for client v to access s is

$$\rho r(v)d(v, p(v, s)) + (1 - \rho)r(v)d(v, s).$$

The total cost for all clients in T_s to access s is

$$\sum_{v \in T_s} (\rho r(v)d(v, p(v, s)) + (1 - \rho)r(v)d(v, s)).$$

Let P denote a set of proxies in the network and SPT(s, P) the shortest path tree rooted from s to reach all proxies in P. The overall cost to update the proxies is

$$w \sum_{(x,y) \in \mathrm{SPT}(s,P)} d(x, y).$$

The total cost of all clients in T_s to access s with a set of proxies P, denoted by $C(T_s, P)$, can thus be defined as:

$$C(T_s, P) = \sum_{v \in T_s} (\rho r(v)d(v, p(v, s)) + (1 - \rho)r(v)d(v, s)) + w \sum_{(x,y) \in \mathrm{SPT}(s,P)} d(x, y). \quad (1)$$

We consider two optimization problems of proxy placement. The first problem is, given a number k, to find the optimal placement of k proxies in the network, such that the total cost defined in Equation (1) is minimized. The second problem is, for an unconstrained number of proxies, to find the optimal number of proxies required in the system to make the total cost minimal.

3. RELATED WORK

There has been considerable work in various aspects of Web caching. Most of the work focused on issues such as Web traffic characterization [7, 8], caching replacement policies [9, 10, 11] and performance analysis of caching [12, 13]. Glassman *et al.* presented one of the earliest attempts at the placement of Web caches in [14]. His method organizes satellite relays (proxy caches) into a tree-structured hierarchy, with cache misses in lower relays percolating up through higher relays until the requested object is found. More recent work in [4] discusses the placement of Web caches, aiming at minimizing the overall flow or the average delay. A k-cache location problem and a k-transparent *en-route* cache (TERC) problem were formulated. The optimal solutions for some regular topologies and some greedy heuristic algorithms for general networks were presented. The work on data replication at caches was reported in [5, 15, 16]. Wang *et al.* studied the optimal proxy prefix cache allocation to videos that minimizes the aggregate network bandwidth cost in [15]. In [5], Tang *et al.* studied data replication at *en-route* caches (proxy nodes). This assumes that the paths that clients access a Web server with form a tree and a novel algorithm was proposed by using dynamic programming for the optimal replication of data objects at a subset of proxy nodes along the path from clients to the server such that the access cost is minimized. Xu *et al.* discussed a combined problem of the placement of proxies and the replication of data objects at proxies in [16]. Three optimal algorithms were first proposed for the replication of a single object in a tree network, based which two efficient heuristics, namely aggregate access (AGGA) and weighted popularity (WPOP), were then proposed for the placement of proxies. Different from the work in [16], this paper discusses the placement of proxies without considering the replication of each individual object and proposes the optimal algorithms.

An active research topic has been the placement of Web proxies or the replication of Web servers. Li *et al.* presented a method for the placement of Web proxies in [17]. The problem is to place a set of proxies in a tree so that the overall latency of accessing the Web server is minimized. The problem was modeled as a dynamic programming problem and the optimal solution is obtained. The problem of placement of replicated Web servers on the Internet to minimize the overall access distance was discussed in [18, 19]. This was formulated into the p-median problem, which is NP-complete [20]. Several heuristic algorithms were proposed in [18, 19]. Sayal *et al.* presented some selection algorithms to access replicated Web servers in [21]. The objective was to choose the closest server replica for a client, based on either the hop counts, round trip time, or the actual HTTP request latency. Jamin *et al.* in [2] presented the problem of placing mirrored Web sites (replicated Web servers), aiming at minimizing the maximum distance between any client and a mirrored site. The problem is formalized as the k-center problem (which is again NP-complete). A greedy algorithm has been proposed to find a sub-optimal solution.

Along with the work of replication of Web servers and proxies, a new concept called the content distribution network (CDN) has emerged. Loosely speaking, CDN is an architecture of network proxy servers. Many issues of CDN, such as where the proxies are deployed, how clients' requests are routed to the proxies and how the data at proxies are updated, are left for implementations. The companies that run CDNs, such as Akamai [22] and Mirror Image [23] use proprietary algorithms for deploying proxy servers and assigning clients to them.

Most of the work concerning Web proxies and CDNs does not consider the update operations of Web servers. Keeping Web proxies consistent is similar to the classical problem of data (file) replication in database systems [24, 25, 26]. The problem of data replica placement has been extensively studied in the literature (see a survey in [25]). Most of the work studies the optimization of the access (communication) cost, access latency and data availability in terms of serving user's read and write requests. It has been proven that the optimal placement of data replicas in general networks is NP-complete [24, 27] and efficient algorithms can only be found in some special network topologies [27], such as tree, ring and bus. Murthy *et al.* [28] proposed an approximation algorithm for the placement of file replicas in general networks to minimize the overall access cost. An approximation ratio to the optimal result

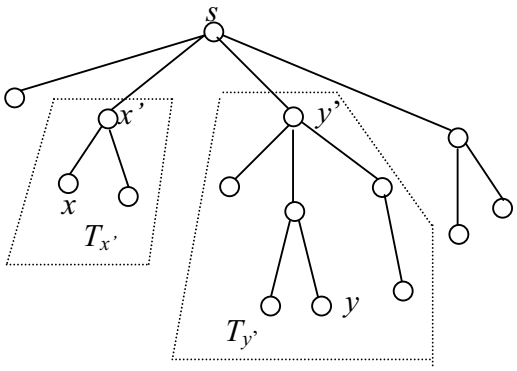

FIGURE 3. Left–right relationship of two tree nodes.

has been obtained. Stephens *et al.* simplified the problem by restricting the replicas to read-only in [29]. The problem is reduced to the p-median problem and can thus be solved by some well-known approximation algorithms.

4. OPTIMAL PLACEMENT OF k PROXIES

We formulate the problem and derive the solution by using the dynamic programming method. Let T_v denote a subtree of T_s rooted at a node v, $v \in T_s$. A non-leaf node in T_s can have an arbitrary number of children which are ordered from left to right, so that given any two siblings u and v, we are able to determine that u is to the left of v or that v is to the right of u. We generalize the notion of being left to non-sibling nodes as shown in Figure 3. Given x and y in T_s, x is said to be the left of y if there exist x' and y' such that $x \in T_{x'}$, $y \in T_{y'}$ and x' and y' are siblings with x' being to the left of y'.

Consider the dynamic programming formulation of placing proxies in a subtree T_v, $v \in T_s$. T_v can be partitioned into three subgraphs by dividing point u, $u \in T_v$, as shown in Figure 4: (1) the subgraph containing all nodes to the left of u; (2) T_u; and (3) the rest of T_v. This partitioning can be formally defined as

$$L_{v,u} = \{x : x \in T_v \text{ and } x \text{ is to the left of } u\},$$
$$R_{v,u} = \{x : x \in T_v \text{ and } x \notin T_u \cup L_{v,u}\}.$$

Taking the example of Figure 4, T_v is partitioned into $L_{v,u}$, T_u and $R_{v,u}$. $L_{v,u}$ contains the nodes in T_v which are to the left of u, T_u is T_v's subtree rooted at u and $R_{v,u}$ contains the rest of the nodes of T_v.

In formulating our dynamic programming equations, we need further partition $R_{v,u}$ to divide the problem space into smaller subtrees. We introduce $L_{v,u,x}$ as a partition of $R_{v,u}$ as below, where x is the partitioning point, $x \in R_{v,u}$:

$$L_{v,u,x} = \{y : y \in R_{v,u} \text{ and } y \text{ is to the left of } x\}.$$

Figure 5 shows a partitioning of $R_{v,u}$ into three parts: $L_{v,u,x}$, T_x and $R_{v,x}$.

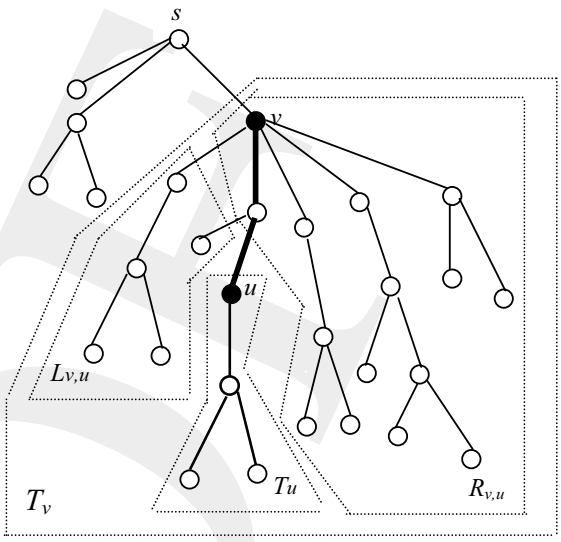

FIGURE 4. Partitioning of subtree T_v.

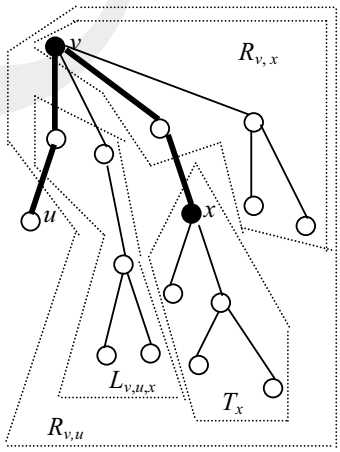

FIGURE 5. Partitioning of $R_{v,u}$.

The cost function defined in (1) can be rewritten as:

$$C(T_s, P) = \sum_{v \in T_s} \rho r(v) d(v, p(v, s)) + \sum_{v \in T_s} (1 - \rho) r(v) d(v, s) + w \sum_{(x,y) \in \mathrm{SPT}(s,P)} d(x, y).$$

Note that the second term of the above function is a constant. It is the amount of traffic flow that goes to s, which is irrelevant to the proxy placement P. For the simplicity of notation, we omit this term from the cost function (note that this part of cost is counted for all proxies and s in the simulations). The cost function becomes

$$C(T_s, P) = \sum_{v \in T_s} \rho r(v) d(v, p(v, s)) + w \sum_{(x,y) \in \mathrm{SPT}(s,P)} d(x, y). \quad (2)$$

To recursively define the placement of proxies in a tree, the server at the root is regarded as one of the proxies. This does not affect the generality of our problem, because the server provides the same service to users as the proxies.

For $u \in T_v$, we define

$$\begin{aligned} C(T_v, k) &= C(T_v, P_{\text{opt}}) = \min_{P \subseteq T_v, |P|=k, v \in P} \{C(T_v, P)\}, \\ C^*(R_{v,u}, k) &= C^*(R_{v,u}, P_{\text{opt}}) \\ &= \min_{P \subseteq T_v, |P|=k, v \in P} \{C^*(R_{v,u}, P)\}, \end{aligned} \tag{3}$$

where P_{opt} denotes the optimal placement of proxies in T_v (or $R_{v,u}$). $C(T_v, k)$ (or $C^*(R_{v,u}, k)$) is the minimal access cost by placing k proxies inT_v (or $R_{v,u}$).

We now formulate the dynamic programming equations for placing t proxies in T_v.

When $t = 1$, we always place the only proxy at root v. Thus, we have

$$C(T_v, 1) = \sum_{x \in T_v} \rho r(x) d(x, v).$$

When $t > 1$, we can always find a node u, $u \in T_v$ and $u \neq v$, which satisfies:

(1) a proxy is placed at u;
(2) no proxy is placed in $L_{v,u}$ ($L_{v,u}$ could be empty);
(3) no proxy is placed in $\pi(v, u) - \{v, u\}$.

T_v is partitioned into three parts, $L_{v,u}$, T_u and $R_{v,u}$, as shown in Figure 4. Then, we consider the placement in T_u and $R_{v,u}$ (note that no proxy is placed in $L_{v,u}$). Assume t' proxies are placed in T_u, $1 \leq t' \leq t-1$. So $t - t'$ proxies will be placed in $R_{v,u}$.

The cost for updating data at proxy u is $wd(v, u)$. This is because the updated data is multicast to all proxies along tree SPT(s, P) and the cost of transmitting the data to v is already counted (note that v is a proxy node). We only need to count the cost of propagating the data from v to u, which is $wd(v, u)$. Therefore, we have

$$C(T_v,t) = C(L_{v,u}) + C(T_u,t') + C^*(R_{v,u},t-t') + wd(v,u),$$

where

$$C(L_{v,u}) = \sum_{x \in L_{v,u}} \rho r(x) d(x, v).$$

This is the overall read cost for clients in $L_{v,u}$ and is a constant once v, u are given.

To obtain the optimal placement of t proxies in T_v, we need to consider all the possible partitioning points u, $u \in T_v$ and all possible values of t', $1 \leq t' \leq t-1$. The dynamic programming equation for proxy placement in T_v can be formulated as

$$C(T_v, t) = \begin{cases} \displaystyle\sum_{x \in T_v} \rho r(x) d(x, v) & \text{if } t = 1 \\ \displaystyle\min_{u \in T_v} \min_{1 \leqslant t' \leqslant t-1} \{C(L_{v,u}) + C(T_u, t') \\ \quad + C^*(R_{v,u}, t - t') + wd(v, u)\} & \text{if } t > 1. \end{cases} \tag{4}$$

In the above equation, $C(L_{v,u})$ is a constant and $C(T_u, t')$ is recursively defined by $C(T_v, t)$ in the same equation. We now consider the formulation of $C^*(R_{v,u}, t)$. The placement of proxies in $R_{v,u}$ is done in a similar way to T_v. $R_{v,u}$ is further partitioned into $L_{v,u,x}$, T_x and $R_{v,x}$, as shown in Figure 5, where x is a proxy node and no proxy is placed in $L_{v,u,x}$ or $\pi(v, x) - \{v, x\}$.

Regarding the cost of updating proxy x, because path $\pi(u, v)$ is already included in tree SPT(s, P) when considering the updating cost of proxy u, the cost for updating proxy x is thus $d(x, \pi(u, v))$, which is the distance from x to path $\pi(u, v)$ along the tree (see Figure 5). Similar to Equation (4), we obtain

$$C^*(R_{v,u}, t) = \begin{cases} \displaystyle\sum_{y \in R_{v,u}} \rho r(y) d(y, v) & \text{if } t = 1 \\ \displaystyle\min_{x \in R_{v,u}} \min_{1 \leqslant t' \leqslant t-1} \{C(L_{v,u,x}) + C(T_x, t') \\ \quad + C^*(R_{v,x}, t - t') + wd(x, \pi(u, v))\} & \text{if } t > 1. \end{cases} \tag{5}$$

By using Equations (4) and (5), $C(T_v, k)$ can be computed out recursively.

THEOREM 1. *Equations (4) and (5) are correct.*

Proof. We only prove the correctness of Equation (5). The proof of Equation (4) can be easily derived in the similar way.

When $t = 1$, there is only one proxy (which is the server) placed at node v. Since there is no cost for updating proxies in this case, the equation is trivially correct.

When $t > 1$, let $C^{*\prime}(R_{v,u},t)$ denote the value of the right-hand side formula of (5), i.e.

$$\begin{aligned} C^{*\prime}(R_{v,u}, t) = \min_{x \in R_{v,u}} \min_{1 \leqslant t' \leqslant t-1} \{&C_1(L_{v,u,x}) + C(T_x, t') \\ &+ C(R_{v,x}, t - t') + wd(x, \pi(u, v))\}. \end{aligned}$$

We need to prove $C^*(R_{v,u},t) = C^{*\prime}(R_{v,u},t)$.

First, we prove $C^*(R_{v,u},t) \geqslant C^{*\prime}(R_{v,u},t)$. Suppose P_{opt} is the optimal placement of t proxies in $R_{v,u}$. Thus, we have

$$C^*(R_{v,u},t) = C^*(R_{v,u}, P_{\text{opt}}). \tag{6}$$

Once P_{opt} is known, we can always find a proxy node x in $R_{v,u}$, such that:

(1) a proxy is placed at x;
(2) no proxy is in $L_{v,u,x}$ ($L_{v,u,x}$ could be empty);
(3) no proxy is in $\pi(v, x) - \{v, x\}$.

The cost for updating proxy node $x$is the distance from $x$to $\pi(u, v)$, i.e. $d(x, \pi(u, v))$. $R_{v,u}$ is partitioned into three parts by x: $L_{v,u,x}$, T_x and $R_{v,x}$. Then, we have

$$\begin{aligned} &C^*(R_{v,u}, P_{\text{opt}}) \\ &\quad = C(L_{v,u,x}) + C(T_x, P_{\text{opt}} \cap T_x) \\ &\qquad + C^*(R_{v,x}, P_{\text{opt}} \cap R_{v,x}) + wd(x, \pi(u, v)). \end{aligned} \tag{7}$$

```
Main () {       // T_s is the tree whose root is s.
    for v ∈ T_s, u ∈ T_v, t ← 1 to k do   // init
        C[T_v, t] ← ∞; C*[R_{v,u}, t] ← ∞;
    Place(T_s, k). // start recursive call of proxy placement
}
Place(T_v, t) {       // recursive procedure of placing t proxies in T_v
    if C [T_v, t] ≠ ∞ then       // C[T_v, t] has been computed before
        return C[T_v, t];
    if t = 1 then            // only one proxy in T_v
        P[T_v, 1] ← (v, 1);
        C[T_v, 1] = Σ_{i∈T_v} ρr(i)d(i, v); return C[T_v, 1];
    for u ∈ T_v do       // when t > 1, find opt'l partitioning point u
        C_{v,u,t} ← ∞;   // C_{v,u,t}: cost of T_v partitioned at u
        for t' ← 1 to t − 1 do   // find t': the opt'l #proxies in T_u
            tmp ← C[L_{v,u}] + Place(T_u, t') + Place*(R_{v,u}, t − t') + wd(u, v); // see (4)
            if tmp < C_{v,u,t} then
                C_{v,u,t} ← tmp;   p_{v,u,t} ← t';
        if C_{v,u,t} < C_1[T_v, t] then       // a better partitioning point u is found.
            P[T_v, t] ← (u, p_{v,u,t});
            C[T_v, t] ← C_{v,u,t};
    return C[T_v, t];
}
```

ALGORITHM 1.

Suppose $|P_{\text{opt}} \cap T_x| = l$. Then, $|P_{\text{opt}} \cap R_{v,x}| = t - l$. Since the sub-placement $P_{\text{opt}} \cap T_x$ in T_x may not be the optimal (the same for $P_{\text{opt}} \cap R_{v,x}$ in $R_{v,x}$), according to the definitions in (2), we have

$$C(T_x, P_{\text{opt}} \cap T_x) \geqslant C(T_x, l),$$
$$C^*(R_{v,x}, P_{\text{opt}} \cap R_{v,x}) \geqslant C^*(R_{v,x}, t - l).$$

By substituting these into (7), we have

$$\begin{aligned} C^*(R_{v,u}, P_{\text{opt}}) &\geqslant C(L_{v,u,x}) + C(T_x, l) + C^*(R_{v,x}, t - l) \\ &\quad + wd(x, \pi(u, v)) \\ &\geqslant \min_{x \in R_{v,u}} \min_{1 \leqslant t' \leqslant t-1} \{C(L_{v,u,x}) + C(T_x, t') \\ &\quad + C^*(R_{v,x}, t - t') + wd(x, \pi(u, v))\} \\ &= C^{*\prime}(R_{v,u}, t). \end{aligned}$$

From (6), we thus have

$$C^*(R_{v,u,}t) \geqslant C^{*\prime}(R_{v,u,}t).$$

Next, we prove $C^*(R_{v,u,}t) \leqslant C^{*\prime}(R_{v,u,}t)$. For any $x \in R_{v,u}$, partition $R_{v,u}$ into three parts: $L_{v,u,x}$, T_x and $R_{v,x}$. For any $1 \leqslant t' < t$, let P^x_{opt} be the optimal placement of t' proxies in T_x, and $P^{v,x}_{\text{opt}}$ the optimal placement of $t - t'$ proxies in $R_{v,x}$. Since $P^x_{\text{opt}} \cup P^{v,x}_{\text{opt}}$ is a placement of t proxies in $R_{v,u}$ but may not be the optimal, we have

$$\begin{aligned} C^*(R_{v,u}, t) &\leqslant C^*(R_{v,u}, P^x_{\text{opt}} \cup P^{v,x}_{\text{opt}}) \\ &= C(L_{v,u,x}) + C(T_x, P^x_{\text{opt}}) + C^*(R_{v,x}, P^{v,x}_{\text{opt}}) \\ &\quad + wd(x, \pi(u, v)) \\ &= C(L_{v,u,x}) + C(T_x, t') + C^*(R_{v,x}, t - t') \\ &\quad + wd(x, \pi(u, v)). \end{aligned} \tag{8}$$

Since x and t' are arbitrary values in their valid domains, (8) implies

$$\begin{aligned} C^*(R_{v,u}, t) &\leqslant \min_{x \in R_{v,u}} \min_{1 \leqslant t' \leqslant t-1} \{C(L_{v,u,x}) + C(T_x, t') \\ &\quad + C^*(R_{v,x}, t - t') + wd(x, \pi(u, v))\} \\ &= C^{*\prime}(R_{v,u}, t). \end{aligned}$$

Thus, $C^{*\prime}(R_{v,u,}t) = C^*(R_{v,u,}t)$. □

The detailed algorithm is shown in Algorithm 1.

$C[T_v, t]$, a two-dimensional array, records the minimal cost of T_v by placing $t$proxies in it. Array $P[T_v, t]$ records the partitioning point of T_v—say u—and the number of proxies placed in T_u. In procedure *Place*(T_v, t), the outside *for-loop* on u searches for the optimal partitioning point u, which partitions T_v into $L_{v,u}$, T_u and $R_{v,u}$ as described above. The inner *for-loop* on t' searches for the optimal number of proxies to be placed in T_u.

The pseudo code of *Place*$^*(R_{v,u}, t)$ is omitted, because it is very similar to *Place*(T_v, t). It uses two arrays $C^*[R_{v,u}, t]$ and $P[R_{v,u}, t]$, both three dimensional, to hold the minimal cost of $R_{v,u}$ and the first dividing point x in $R_{v,u}$ and the

number of proxies in T_x, respectively. It uses Equation (5) to compute the cost $C^*[R_{v,u}, t]$.

The final result of proxy placement is derived from arrays $P[T_v, t]$ and $P[R_{v,u}, t]$. The search starts from the root, by locating the element of $P[T_s, k]$, which indicates the first dividing point for the initial tree T_s. Suppose $P[T_s, k] = (u, t)$, which means u is the optimal partitioning point of T_s and t proxies should be placed in T_u. Note that u is a proxy node. Then, (u, t) will be used as the index to locate an element in $P[T_u, t]$ (and in $P[R_{s,u}, k - t]$), which indicates the next dividing point of subtree T_u (and $R_{s,u}$). This dividing point is another proxy node. The operation is repeated for each subtree while the partitioning comes down from the root along the tree, until the number of proxies placed in the concerned subtree is one. At the end, we locate all proxy nodes.

THEOREM 2. *The program computing Equations (4) and (5) terminates in time $O(n^3k^2)$.*

Proof. The work of procedures $Place(T_v, t)$ and $Place^*(R_{v,u}, t)$ is to compute results in arrays $C[T_v, t]$ and $C^*[R_{v,u}, t]$, respectively. $C[T_v, t]$ and $C^*[R_{v,u}, t]$ have nk and n^2k different entries, respectively, and each entry is computed only once (it simply returns the value if it was computed before). Consider the time complexity of computing an entry in $C[T_v, t]$ or $C^*[R_{v,u}, t]$. It takes two layers of loops to compute an element in $C[T_v, t]$ (or $C^*[R_{v,u}, t]$)). The outside *for-loop* on $u \in T_v$ iterates $|T_v|$ times, where $|T_v| \leqslant n$ is the number of nodes in T_v. The inner *for-loop* on t' iterates t times, $t \leqslant k$. Thus, it takes at most $O(nk)$ times of comparison to compute an entry. Therefore, it takes $O((nk + n^2k) \times nk) = O(n^3k^2)$ time in total to compute all entries in $C[T_v, t]$ and $C^*[R_{v,u}, t]$. □

5. OPTIMAL NUMBER OF PROXIES

In this section, we will find the optimal number of proxies required in the system to make the total cost minimal. Let P be a set of proxies placed in T_v (the size of P is unknown), where v is always a proxy node (or the Web server). We define

$$\begin{aligned} C(T_v) &= \min_{P \subseteq T_v, v \in P} C(T_v, P) \\ C^*(R_{v,u}) &= \min_{P \subseteq R_{v,u}, v \in P} C^*(R_{v,u}, P). \end{aligned} \tag{9}$$

Considering the optimal placement of proxies in T_v, there are always two choices: placing no proxy in T_v (except v) or place some proxies in T_v (besides v). Let $C^0(T_v)$ denote the total cost in T_v with no proxy placed in it and $C^+(T_v)$ the total cost with some proxies placed in T_v. It is obvious to see that

$$C^0(T_v) = \sum_{x \in T_v} \rho r(x) d(x, v),$$

because the cost for updating proxies is zero.

Now, we consider $C^+(T_v)$. When there are proxies other than v in T_v, we can always find a node u, $u \in T_v$ and $u \neq v$, which satisfies:

(1) a proxy is placed at u;
(2) no proxy is placed in $L_{v,u}$ ($L_{v,u}$ could be empty);
(3) no proxy is placed in $\pi(v, u) - \{v, u\}$.

The cost for updating proxy node u is $d(u, v)$. T_v is partitioned into $L_{v,u}$, T_u and $R_{v,u}$, as shown in Figure 4. Then, we have

$$C^+(T_v) = C(L_{v,u}) + C(T_u) + C^*(R_{v,u}) + wd(u, v)$$

where

$$C(L_{v,u}) = \sum_{x \in L_{v,u}} \rho r(x) d(x, v).$$

This is the overall read cost for clients in $L_{v,u}$ and is a constant once v, u are given. When computing the optimal result of $C^+(T_v)$, all possible dividing points u, $u \in T_v$, should be considered. We can thus formulate $C^+(T_v)$ as

$$C^+(T_v) = \min_{u \in T_v} \{C(L_{v,u}) + C(T_u) + C^*(R_{v,u}) + wd(u, v)\}.$$

Then, $C(T_v)$ can be computed by:

$$\begin{aligned} C(T_v) &= \min\{C^0(T_v), C^+(T_v)\} \\ &= \min\left\{ \sum_{x \in T_v} \rho r(x) d(x, v), \min_{u \in T_v} \{C(L_{v,u}) \right. \\ &\qquad \left. + C(T_u) + C^*(R_{v,u}) + wd(u, v)\} \right\}. \end{aligned} \tag{10}$$

In the above equation, $C(L_{v,u})$ is a constant and $C(T_u)$ is recursively defined by $C(T_v)$ in the same equation. The placement in $R_{v,u}$ is done in the similar way to T_v. We can choose to place no proxy in $R_{v,u}$ (the cost in this case is denoted by $C^{*0}(R_{v,u})$) or place some proxies in $R_{v,u}$ (the cost in this case is denoted by $C^{*+}(R_{v,u})$). When computing $C^{*+}(R_{v,u})$, $R_{v,u}$ is further partitioned into $L_{v,u,x}$, T_x and $R_{v,x}$, as shown in Figure 5, where x is a proxy node and no proxy is placed in $L_{v,u,x}$ or path $\pi(v, x) - \{v, x\}$. Similar to Equation (10), we obtain

$$\begin{aligned} C^*(R_{v,u}) &= \min\{C^{*0}(R_{v,u}), C^{*+}(R_{v,u})\} \\ &= \min\left\{ \sum_{x \in R_{v,u}} \rho r(x) d(x, v), \min_{x \in R_{v,u}} \{C(L_{v,u,x}) \right. \\ &\qquad \left. + C(T_x) + C^*(R_{v,x}) + wd(x, \pi(u, v))\} \right\}. \end{aligned} \tag{11}$$

Equations (10) and (11) give the result about where the proxies should be placed in order to achieve the minimal total cost defined in Equation (2). By counting the number of proxies which should be placed in the network, we know the optimal number of proxies required in the system.

THEOREM 3. *Equations (10) and (11) are correct.*

Proof. We only prove the correctness of Equation (11). The proof of Equation (10) can be derived in the same way.

```
Main () {      // T_s is tree T whose root is s.
    for v ∈ T_s, u ∈ T_v, // init
        C[T_v] ← ∞; C*[R_{v,u}] ← ∞;
    Place(T_s). // start recursive call of proxy placement
}
Place(T_v) {      // recursive procedure of placing proxies in T_v
    if C[T_v] ≠ ∞ then      // C[T_v] has been computed before
        return C[T_v];
    C^0[T_v] ← Σ_{x∈T_v} ρr(x)d(x, v);    // cost of placing no proxy in T_v
    C^+[T_v] ← ∞;   // cost of placing some proxies in T_v
    for u ∈ T_v do      // find opt'l partitioning point u
        tmp ← C[L_{v,u}] + Place(T_u) + Place*(R_{v,u}) + wd(u, v); // see (10)
        if tmp < C^+[T_v] then
            C^+[T_v] ← tmp;      p_v ← u;
    if C^+[T_v] < C^0[T_v] then // see if T_v needs proxy or not
        C[T_v] ← C^+[T_v]; P[T_v] ← p_v;
    else
        C[T_v] ← C^0[T_v]; P[T_v] ← ∅;
    Return C[T_v];
}
```

ALGORITHM 2.

From the definition in (11),

$$\begin{aligned} C^*(R_{v,u}) &= \min_{P\subseteq R_{v,u}} C^*(R_{v,u}, P) \\ &= \min\left\{C^*_{|P|=1}(R_{v,u}, P), \min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P)\right\} \\ &= \min\left\{\sum_{x\in R_{v,u}} \rho r(x)d(x, v), \min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P)\right\}. \end{aligned} \tag{12}$$

Comparing (11) with (12), it can be seen that we need to prove

$$\begin{aligned} &\min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P) \\ &\quad = \min_{x\in R_{v,u}} \{C(L_{v,u,x}) + C(T_x) \\ &\qquad + C^*(R_{v,x}) + wd(x, \pi(u, v))\}. \end{aligned} \tag{13}$$

Similar to the proof of Theorem 1, we first prove the inequality '$\geqslant$' of (13). Suppose P_{opt} is the optimal placement in $R_{v,u}$, which is

$$\min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P) = C^*(R_{v,u}, P_{\text{opt}}). \tag{14}$$

Once P_{opt} is known, we can always find a proxy node x in $R_{v,u}$, such that:

(1) a proxy is placed at x;
(2) no proxy is in $L_{v,u,x}$ ($L_{v,u,x}$ could be empty);
(3) no proxy is in $\pi(v, x) - \{v, x\}$.

The cost for updating proxy x is $d(x, \pi(u, v))$. $R_{v,u}$ is partitioned into three parts by x: $L_{v,u,x}$, T_x and $R_{v,x}$. Then, we have:

$$\begin{aligned} C^*(R_{v,u}, P_{\text{opt}}) &= C(L_{v,u,x}) + C(T_x, P_{\text{opt}} \cap T_x) \\ &\quad + C^*(R_{v,x}, P_{\text{opt}} \cap R_{v,x}) + wd(x, \pi(u, v)) \\ &\geqslant C(L_{v,u,x}) + C(T_x) + C^*(R_{v,x}) \\ &\quad + wd(x, \pi(u, v)) \\ &\geqslant \min_{x\in R_{v,u}} \{C(L_{v,u,x}) + C(T_x) + C^*(R_{v,x}) \\ &\quad + wd(x, \pi(u, v))\}. \end{aligned} \tag{15}$$

From (14) and (15), we have

$$\begin{aligned} &\min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P) \\ &\quad \geqslant \min_{x\in R_{v,u}} \{C(L_{v,u,x}) + C(T_x) + C^*(R_{v,x}) \\ &\qquad + wd(x, \pi(u, v))\}. \end{aligned} \tag{16}$$

Now, we prove the inequality '$\leqslant$' of formula (13). For any $x \in R_{v,u}$, let P^x_{opt}, $P^{v,x}_{\text{opt}}$ be the optimal placement in T_x, $R_{v,x}$, respectively. Since $P^x_{\text{opt}} \cup P^{v,x}_{\text{opt}}$ is a placement in $R_{v,u}$ but may not be the optimal, we have

$$\begin{aligned} &\min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P) \\ &\quad \leqslant C^*(R_{v,u}, P^x_{\text{opt}} \cup P^{v,x}_{\text{opt}}) \\ &\quad \leqslant C(L_{v,u,x}) + C(T_x, P^x_{\text{opt}}) + C^*(R_{v,x}, P^{v,x}_{\text{opt}}) \\ &\qquad + wd(x, \pi(u, v)). \end{aligned} \tag{17}$$

Since x is an arbitrary value in its valid domain, (17) implies

$$\begin{aligned} &\min_{P\subseteq R_{v,u},|P|>1} C^*(R_{v,u}, P) \\ &\quad \leqslant \min_{x\in R_{v,u}} \{C(L_{v,u,x}) + C(T_x, P^x_{\text{opt}}) + C^*(R_{v,x}, P^{v,x}_{\text{opt}}) \\ &\qquad + wd(x, \pi(u, v))\}. \end{aligned} \tag{18}$$

By (16) and (18), we proved (12) is correct. Thus, (11) is correct. □

The detailed algorithm is shown in Algorithm 2.

Algorithm 2 uses an array $C[T_v]$ that records the minimal cost of T_v. Array $P[T_v]$ records the first partitioning point of T_v, which is a proxy node if it is not empty. If $P[T_v]$ is empty, it means that no proxy is placed in T_v. In procedure $Place(T_v)$, when considering the case of placing some proxies in T_v, the *for-loop* on u searches for the optimal partitioning point u, which partitions T_v into $L_{v,u}$, T_u and $R_{v,u}$ as described above. The code for $Place^*(R_{v,u})$ is omitted, which is very similar to $Place(T_v)$. It uses array $P[R_{v,u}]$ to record the proxy nodes and Equation (10) to compute the cost array $C^*[R_{v,u}]$.

The result of proxy placement can be obtained by searching the arrays $P[T_v]$ and $P[R_{v,u}]$. The search starts from the root s. The value of $P[T_s]$ indicates the first dividing point for initial tree T_s. Suppose $P[T_s] = u$. This means that u is a proxy node. Then, we search the proxies in T_u and $R_{s,u}$, which can be found in $P[T_u]$ and $P[R_{s,u}]$, respectively. When searching the proxies in $R_{s,u}$, suppose $P[R_{s,u}] = x$. Then, x is another proxy node and we further look at $P[T_x]$ and $P[R_{s,x}]$ for the proxies in T_x and $R_{s,x}$, because $R_{s,u}$ is divided into $L_{s,u,x}$, T_x and $R_{s,x}$ and no proxy is in $L_{s,u,x}$. The operation is repeated until no proxy is found in the searched subtree, i.e. the value in $P[T_v]$ (or $P[R_{v,u}]$) is empty. At the end, we locate all proxy nodes.

THEOREM 4. *The program for computing Equations (10) and (11) terminates in time* $O(n^3)$.

Proof. Procedures $Place(T_v)$ and $Place^*(R_{v,u})$ compute arrays $C[T_v]$ and $C^*[R_{v,u}]$, respectively. Looking at the program code of $Place(T_v)$, it takes $|T_v|$ times of comparisons (see the *for-loop* on $u \in T_v$), where $|T_v| \leqslant n$ is the number of nodes in T_v. It thus takes at most $O(n)$ time to compute an entry in $C[T_v]$ (or $C^*[R_{v,u}]$). $C[T_v]$ and $C^*[R_{v,u}]$ have n and n^2 entries, respectively, and each entry is computed only once in the program. Therefore, it takes $O((n + n^2) \times n) = O(n^3)$ time in total to compute all the entries in $C[T_v]$ and $C^*[R_{v,u}]$. □

6. NUMERICAL EXAMPLE, SIMULATION RESULTS AND DISCUSSIONS

6.1. Numerical example

To understand the results produced by the optimal placement algorithm, we run our algorithms on an example graph as shown in Figure 6. The distances of links and the read frequencies of client nodes are marked in the figure. The update frequency originating from server s is 12. According to the result produced by our algorithm, one proxy is required (marked in grey) to make the overall cost minimal.

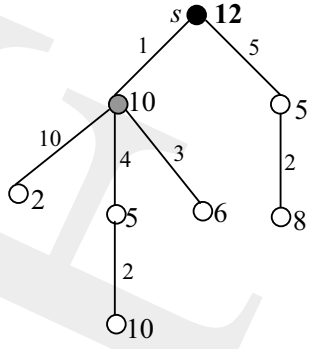

FIGURE 6. An example of optimal proxy placement.

6.2. Simulation setup

Extensive simulations have been conducted to evaluate the performance of the algorithms. Network topologies used in the simulations are generated using the Inet topology generator, downloaded from http://topology.eecs.umich.edu/inet/. It generates random network topologies following the characteristics of the Internet topology reported in [30]. The default size of the networks used in the simulations is of 3037 nodes (i.e. $n = 3037$), except for the simulations in Figure 12 shown later. The read frequency of a node $r(v)$ is randomly generated in the range of $[0, 100]$. The number of update operations originating from server s, i.e. w, is $\alpha \sum_{v \in T_s} r(v)$, where α is the ratio of update and read operations (we call it the read–write ratio for simplicity).

When a network graph is generated with the parameters of links and nodes, all the simulations are conducted on this graph. In each run of the simulation, a server node, s, is randomly picked up from the graph and a shortest path tree T_s, which is rooted from $s$and linking all the nodes in the graph, is computed. Then, we run the placement algorithms on T_s.

6.3. Performance comparisons of placement algorithms

The main purpose of using proxies is to reduce the traffic flow of Web accesses on the network. The access cost defined in this paper is the access frequency times the distance. Reducing this access cost is equivalent to the reduction of traffic flow in the network. We use a traffic-reduction ratio to measure the effectiveness of placing proxies, which is defined as

$$R = \frac{C(T_s, \emptyset) - C(T_s, P)}{C(T_s, \emptyset)} \times 100\%,$$

where $C(T_s, \emptyset)$ and $C(T_s, P)$ are the cost with no proxies and the cost with a set of P proxies, respectively.

Three placement algorithms were simulated: the optimal algorithm, a greedy algorithm and a random placement algorithm. The greedy algorithm places each of the k proxies independently from the others. That is, it places one proxy at the 'optimal location' and modifies the traffic flow accordingly in the network. The 'optimal location' means the proxy node that results in the maximal reduction of the cost defined in (1). Then, it places the next proxy in the same way, without changing the location of previously placed proxies. This operation is repeated until all k proxies are placed.

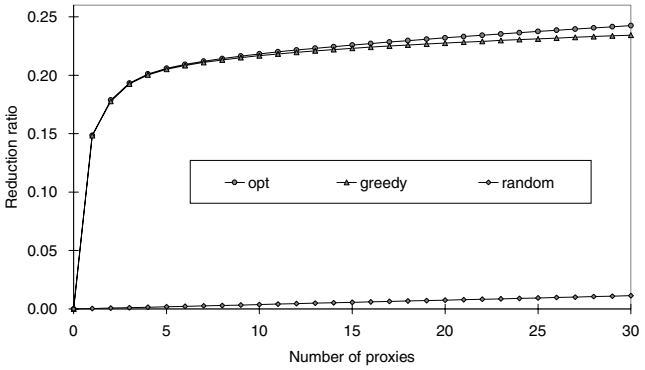

FIGURE 7. R versus k.

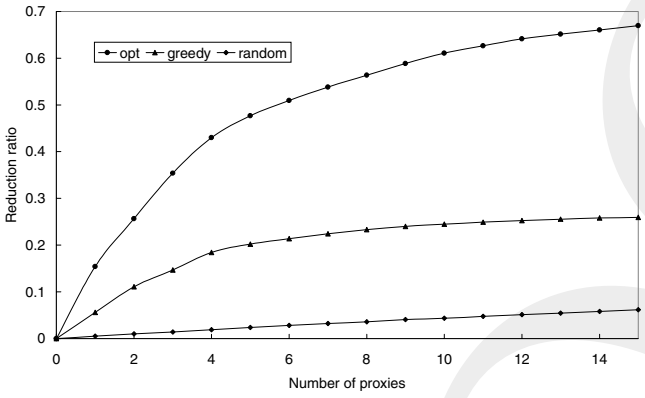

FIGURE 8. R versus k for Bell Lab's Web server.

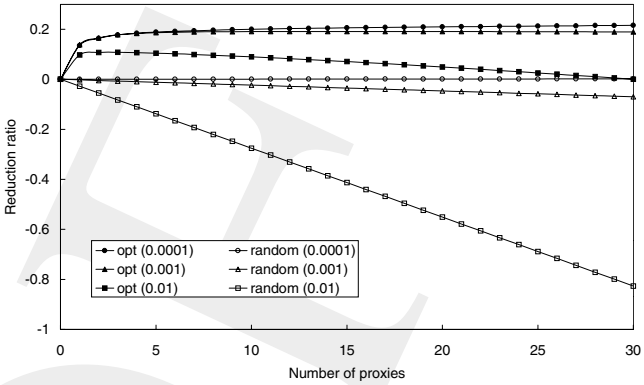

FIGURE 9. R versus k with $\alpha = 0.0001$, 0.001 and 0.01.

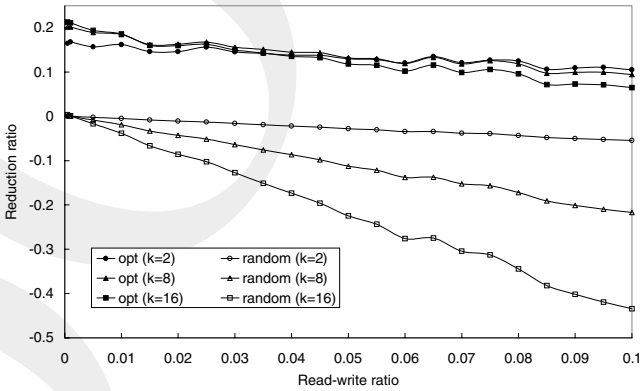

FIGURE 10. R versus α with $k = 2$, 8 and 16.

We first look at the case where update operations are not considered (i.e. $\alpha = 0$). This group of simulations is designed to compare the performance (i.e. the traffic reduction ratio R) against the number of proxies. The results are shown in Figure 7. From the curves in Figure 7, the following observations can be made.

(1) The performance of greedy algorithm is very close to the optimal algorithm. In fact, the difference is usually within 10% throughout all the simulations under the same simulation input. For the clarity of presenting simulation results, we will drop the curves of the greedy algorithm from Figures 9–14 shown later.
(2) R rises sharply at a small number of proxies ($k \leqslant 5$). It becomes quite stable when k reaches about seven. This result tells us that with a small number of proxies installed in the system, the traffic flow can be reduced to a significant level.
(3) R does not change much as the access frequency changes. We have simulated the performance under different access frequencies, but the R values remain more or less the same as we increase the access frequencies (the results were not presented in the figure due to over-crowding).

One reason that the optimal algorithm does not show a significant performance gain over the greedy algorithm is because of the even distribution of traffic. In the above simulations, the read frequency, $r(v)$, of a node is randomly generated. Therefore, the density of traffic is relatively even over the whole network. In the real system, the access frequencies to a Web site are usually different from region to region. Figure 8 shows results computed from a real network and traffic accessing Bell Lab's Web site (www.bell-labs.com). The data were logged on 14 Jan 1998 [4]. From Figure 8, we can see that the optimal algorithm is much more effective than the greedy algorithm in the real traffic situation. When the traffic is uneven, placing proxies at some high traffic locations would yield a larger reduction of access cost than in the network with even traffic. That is why the optimal algorithm demonstrates higher superiority in an uneven traffic network by placing a limited number of proxies at the most needed locations.

Next, we introduce the update cost to the proxies in the simulations. This group of simulations aims at analyzing how R is affected by three major factors, namely k (number of proxies), α (read–write ratio) and ρ (hit ratio). These three factors interfere with each other in real situations. Figure 9 shows R versus k, where ρ was fixed at 40%. The sets of three curves in Figure 9 represent α values at 0.0001, 0.001 and 0.01, respectively. Figure 10 shows R versus α, where ρ was fixed at 40% and the sets of three curves represent k values at 2, 8 and 16, respectively. Figure 11 shows R versus ρ, where α was fixed at 0.001 and k values were set at 2, 8 and 16. From Figures 9–11, we can make the following observations.

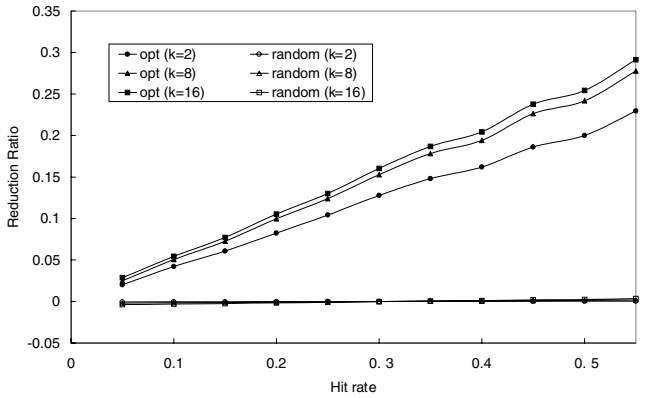

FIGURE 11. R versus ρ with $k = 2$, 8 and 16.

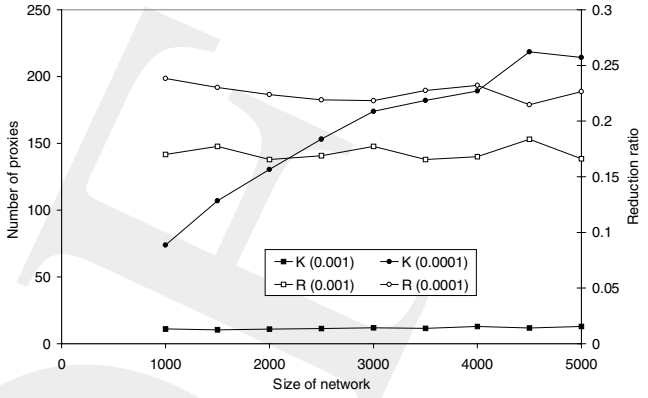

FIGURE 12. k and R versus n with $a = 0.001$, 0.0001.

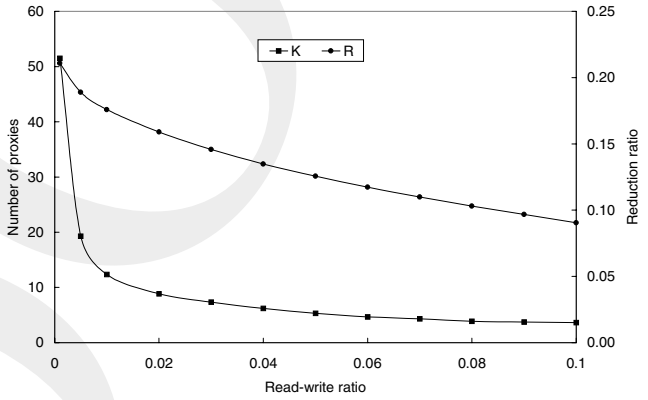

FIGURE 13. k versus α.

(1) When the cost for updating proxies is counted, placing proxies at random locations does not have any effect in reducing the network traffic, but incurs extra cost in updating the proxies. From Figures 9–11, we can see that the curves for the random algorithm at various network situations are at or below zero. That is, the system will pay performance penalties if proxies are placed randomly in the network.
(2) From Figure 9, we can see that by using the optimal algorithm, R increases sharply when k is small and becomes saturated when k reaches about 5. This suggests that it is more cost-effective to place a small number of proxies in the system. For the curve where $\alpha = 0.01$, R even starts dropping at $k = 3$. This indicates that placing too many proxies would degenerate the system performance if the update frequency to the Web data is relatively high.
(3) R decreases as α increases (see Figure 10). However, R improves significantly as the ρ increases (see Figure 11). The results in Figure 11 justify the efforts of improving the hit-ratio of data stored at proxies, which is primarily important to the reduction of traffic flow.

6.4. Simulations on finding the optimal number of proxies

The optimal number of proxies required in a network depends on n, α and ρ. This group of simulations was designed to see how the optimal number of proxies is affected by these three parameters and the corresponding traffic reduction ratio. In Figures 12–14, there are two y-axes, the left-hand side is the optimal number of proxies required in the system (denoted by k) and the right-hand side is the corresponding traffic reduction ratio (denoted by R) by placing the optimal number of proxies. The curves are in pairs (a k-curve and an R-curve) in these figures.

Figure 12 shows the optimal number of proxies versus n with $\rho = 40\%$. The two pairs of k and R curves are for $\alpha = 0.001$ and 0.0001, respectively. From the figure, we can see the k-curve for $\alpha = 0.001$ remains almost flat as the network size increases (in fact, the k values on this curve are between 10 and 12). The k-curve for $\alpha = 0.0001$ shows a stable increase as the network size increases. It is worth pointing out that the two R-curves are also quite flat even when more proxies are required for larger networks. The simulation results suggest that by placing the optimal network proxies in the system, the traffic reduction ratio is dependent on the read–write ratio (the R-curve for $\alpha = 0.001$ is lower than that for $\alpha = 0.0001$). However, it remains almost unchanged as the network size increases. The reason might be that when considering the optimal number of proxies required in the system, the algorithm always strikes the balance between the saving of read traffic and the cost incurred by the write traffic whatever size of the network.

Figure 13 shows the optimal number of proxies versus α with $\rho = 40\%$. The k-curve shows that k decreases as α increases, which is consistent with the results observed in Figure 9. The k-curve drops quickly when α is small and remains quite stable from the point at $\alpha = 0.02$. This suggests that the need of proxies drops dramatically as the update to the Web data frequency increases. From the decline of the k-curve, we can predict that it would eventually reach zero when α is increased to a point where the cost for updating the proxies outweighs the benefit the proxies bring to the system.

Figure 14 shows the optimal number of proxies versus ρ with $\alpha = 0.001$. The k-curve shows a stable increase of k as the hit-ratio increases and the R-curve shows an increase of R at a similar pace to k. This suggests that placing more

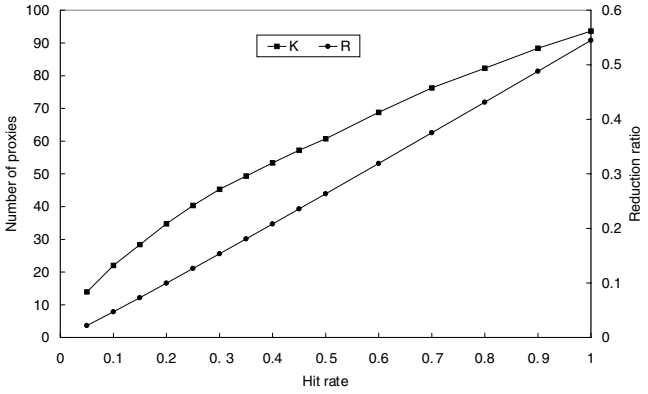

FIGURE 14. k versus ρ.

proxies should come together with the improvement of cache hit-ratio; otherwise proxies cannot work effectively. This confirms the results presented in Figure 11.

6.5. Discussions on placement assumptions and results

6.5.1. Stability of routing and traffic

The effectiveness of the placement algorithm depends on the stability of the Internet routing methods. If the routes on the Internet are stable, the routes used by clients to access the Web server would form a shortest path tree, rooted from the server. Studies in [31, 32] point out that in reality the Internet routing is stable. It was found that 80% of routes change at a frequency lower than once a day. Krishnan *et al.* carried out another study in [4], which traced the routes from Bell Lab's Web server to 13,533 destinations. It was found that almost 93% of the routes are actually stable during their experiments. These studies justify our assumption on the stability of Internet routing, which leads to the reduction of placing proxies in an arbitrary network to the problem of placing them in a tree structure.

The placement of *en-route* proxies in the routers requires static configuration work. Once a proxy is configured inside a router, it is expected that the proxy would stay with the router for a relatively long period of time. Studies in [4] pointed out although the client population changes significantly from time to time, the traffic flow in the outgoing tree from the Web server to the clients remains pretty much stable in the branches that carry most of the traffic. This indicates the optimal locations for the proxies do not change by much as time progresses, because the topology of the tree network and the traffic pattern on the network are quite stable.

6.5.2. Placement pattern of proxies

In our algorithm, the Web server uses a multicast model to propagate updates to all proxies. Since our cost function only considers the traffic flow of read and write operations, in the case where the number of proxies was not limited, the only constraint that prohibits introducing too many proxies is the cost incurred for update operations. This will create a situation that once a proxy is placed at node u, it implies that every node on path $\pi(u, s)$ is qualified to have a proxy, because by placing proxies on these nodes it does not incur any extra cost for update operations, but reduces the cost for read operations originating from or passing through those nodes. This pattern is inevitable for placing an unconstrained number of data replicas in a tree structure if the multicast model is used to propagate update operations [27]. However, in real situations, the cost function would take many other factors into account, such as monetary cost of setting up a proxy and its maintenance cost. In such cases, the above concerned pattern of placing proxies will disappear.

7. CONCLUSIONS

We have investigated the optimal placement of Web proxies to minimize the overall access cost, with the consideration of both read and update operations. The solution to this problem can significantly alleviate the Web access traffic in the Internet and improve the performance of Web servers. The original contributions of this paper are as follows.

(1) Formulating the problem of placing k proxies in the system. The optimal solution can be obtained in $O(n^3k^2)$, where k is the number of proxies and n the number of nodes in the system.
(2) Formulating the problem of finding the optimal number of proxies required in the system, given the read frequencies of all clients and the update frequency of the server. The optimal solution can be obtained in $O(n^3)$.

Simulations have been conducted to demonstrate the performance and illustrate the results of our optimal placement algorithms.

ACKNOWLEDGEMENTS

This work is supported by CityU grant No. 7001182, NSF China under grant No. 60273071, and the National 973 InfoTech and High Performance Software of China under grant No. G1998030402.

REFERENCES

[1] Bestavros, A. (1997) WWW traffic reduction and load balancing through server-based caching. *IEEE Concurrency*, 56–67.
[2] Jamin, S., Jin, C., Kurc, A. R., Raz, D. and Shavitt, Y. (2001) Constrained mirror placement on the Internet. In *Proc. IEEE InfoCom'01*, Anchorage, Alaska, April 22–26, pp. 31–40.
[3] Chatel, M. (1996) *Classical Versus Transparent IP Proxies*. RFC 1919, March.
[4] Krishnan, P., Raz, D. and Shavitt, Y. (2000) The cache location problem. *IEEE/ACM Trans. Networking*, **8**, 568–582.
[5] Tang, X. and Chanson, S. (2002) Coordinated en-route Web caching. *IEEE Trans. Computers*, **51**, 595–607.
[6] Davison, B. D. Proxy cache comparison. http://www.web-cashing.com/proxy-comparison.html.

[7] Arlitt, M. F. and Williamson, C. L. (1997) Internet Web servers: workload characterization and performance implications. *IEEE Trans. Networking*, **5**, 631–645.

[8] Braun, H. W. and Claffy, K. C. (1995) Web traffic characterization: an assessment of the impact of caching documents from NCSA's Web server. *Comput. Networks ISDN Syst.*, **28**, 37–51.

[9] Cao, P. and Irani, S. (1997) Cost-aware WWW proxy caching algorithms. In *Proc. of Usenix Symp. Internet Technologies and Systems*, California, December, pp. 193–206.

[10] Dilley, J. and Arlitt, M. (1999) Improving proxy cache performance: analysis of three replacement policies. *IEEE Internet Comput.*, **3**, 44–50.

[11] Rizzo, L. and Vicisano, L. (2000) Replacement policies for a proxy cache. *IEEE/ACM Trans. Networking*, **8**, 158–170.

[12] Abrams, M. *et al.* (1995) Caching proxies: limitations and potentials. In *Proc. 4th Int. Conf. WWW*, Boston, USA, December, pp. 312–319.

[13] Bolot, J. and Hoschka, P. (1996) Performance engineering of the World Wide Web: application to dimensioning and cache design. In *Proc. 5th Int. Conf. WWW*, Pairs, France, May, pp. 6–10.

[14] Glassman, S. (1994) A caching relay for the World Wide Web. In *Proc. of 1st Int. Conf. on WWW*, Amsterdam, pp. 69–76.

[15] Wang, B., Sen, S., Adler, M. and Towsley, D. (2002) Optimal proxy cache allocation for efficient streaming media distribution. In *IEEE Infocom'02*, New York, June 23–27, pp. 1726–1735.

[16] Xu, J., Li, B. and Lee, D. (2002) Placement problems for transparent data replication proxy services. *IEEE J. Selected Areas Commun.*, **20**, 1399–1413.

[17] Li, B., Golin, M. J., Italiano, G. F. and Deng, X. On the optimal placement of Web proxies in the Internet. In *Proc. IEEE InfoCom'99*, New York, USA, March 21–25, pp. 1282–1290.

[18] Chen, F. and Li, B. (1999) Replicated servers allocation for multiple information sources in a distributed environment. In *Proc. 8th IEEE Int. Conf. Computer Communications and Networks (IC3N'99)*, USA, October, pp. 168–173.

[19] Qiu, L., Padmanabhan, V. and Voelker, G. (2001) On the placement of Web server replicas. *IEEE InfoCom'01*, Anchorage, Alaska, April 22–26, pp. 1587–1596.

[20] Kariv, O. and Hakimi, S. L. (1979) An algorithmic approach to network location problems. II: The p-medians. *SIAM J. Appl. Math.*, **37**, 539–560.

[21] Sayal, M., Breitbart, Y., Scheuermann, P. and Vingralek, R. (1998) Selection algorithms for replicated web servers. In *Proc. of Internet Server Performance Workshop* (in conjunction with ACM SIGmetrics'98), June.

[22] Akamai Technologies. http://www.akamai.com.

[23] Mirror Image. http://www.mirror-image.com.

[24] Ceri, S., Martella, G. and Pelagatti, G. (1982) Optimal file allocation in a computer network: a solution method based on the knapsack problem. *Computer Networks*, **6**, 345–357.

[25] Dowdy, L. W. and Foster, D. V. (1982) Comparative models of the file assignment problem. *ACM Comput. Survey*, **14**, 287–313.

[26] Irani, K. B. and Khabbaz, N. G. (1982) A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems. *IEEE Trans. Computer*, **31**, 419–434.

[27] Wolfson, O. and Milo, A. (1991) The multicast policy and its relationship to replicated data placement. *ACM Trans. Database Syst.*, **16**, 181–205.

[28] Murthy, K., Kam, J. and Krishnamoorthy, M. S. (1983) An approximation algorithm to the file allocation problem in computer networks. In *Proc. of 2nd ACM Symposium on Principles of Database Systems*, pp. 258–266.

[29] Stephens, A. B., Yesha, Y. and Humenik, K. (1995) Optimal allocation for partially replicated database systems on tree networks. *Appl. Math. Lett.*, **8**, 71–76.

[30] Faloutsos, M., Faloutsos, P. and Faloutsos, C. (1999) On power-law relationships of the Internet topology. In *Proc. ACM SIGCOMM*, August.

[31] Labovitz, C., Malan, G. R. and Jahania, F. (1997) Internet routing instability. In *ACM SIGCOMM'97*, August, pp. 115–126.

[32] Paxson, V. (1997) End-to-end routing behavior in the Internet. *IEEE/ACM Trans. Networking*, **5**, 601–615.

Annotations from bxg032.pdf

Page 12

Annotation 1
Au:
Any volume number for reference [1]?
Please supply publisher details for references [2,9,12-15,17-19,21,28,30,31] if available.