On the Optimal Placement of Web Proxies in the Internet

Bo Li, Mordecai J. Golin and Giuseppe F. Italiano*and Xin Deng
Department of Computer Science
Hong Kong University of Science and Technology, Hong Kong

Kazem Sohraby
Bell Laboratories
Lucent Technologies, Holmdel, New Jersey, USA

Abstract

Web caching or web proxy has been considered as the prime vehicle of coping with the ever-increasing demand for information retrieval over the Internet, WWW being a typical example. Existing work on web proxy has primarily focused on content based caching; relatively less attention has been given to the development of proper placement strategies for the potential web proxies in the Internet. In this paper, we argue that the placement of wen proxies is critical to the performance and further investigates the optimal placement policy of web proxies for *a target web server* in the Internet. The objective is to optimize a given performance measure for the target web server subject to system resources and traffic pattern. Specifically, we are interested in finding the optimal placement of multiple web proxies (M) among potential sites (N) under a given traffic pattern. We show this can be modeled a *Dynamic Programming* problem. We further obtain the optimal solution for the tree topology using $O(N^3M^2)$ time.

1 Introduction

We have witnessed an explosive growth in the use of the World Wide Web (or web) in the past few years; there are many reasons behind this success, in particular, ease of use, the availability of standard tools for creating web documents and for navigating the web, timely dissemination of information, and the increased popularity of the Internet [1]. At the same time, this quick adoption also leads to its poor performance, as web clients often have to tolerate long response times. There are a number of factors contributing to this inefficiency such as server or network congestion during peak time, links with limited or inadequate bandwidth, and long propagation delay. Caching has been considered as one of the prime vehicles of coping with this inefficiency.

The basic principle behind caching is that it allows the retrieved documents to be kept close to clients; in web environment, this can reduce the response time of web services and alleviate the network congestion if any. There are several ways that documents can be cached for a web server including: web browser (client), web server itself and web proxy [17]. Caching at the clients' side has been implemented by most existing web browsers [2]. This can prevent a client from generating traffic to the same location repeatedly; for example both NCSA Mosaic and Netscape can save images and documents. Caching can also be deployed at the server side when a web server contains pointers to other web servers [17], this allows the web server to use a local copy fetching in advance to serve clients' requests, instead of having to forward the requests to remote server(s) each time. Unfortunately, both do little towards improving overall network performance [13]. Client side caching only saves one single client from consecutively fetching the same web documents. This, however, can not even restrict multiple clients of the same area from downloading the the same web files from the same web server during a short time interval. Server side caching only mitigates the problem of not forwarding requests further, but does nothing to alleviate potentially long access delay experienced by clients, nor able to resolve the possible congestion.

The most effective way of reducing the overall latency is the use of a web proxy, or proxy server (or

*The work was done when Dr. Italiano was a visiting professor at HKUST. He is with Dipartimento di Matematica Applicata ed Informatica, Università "Ca' Foscari" di Venezia, Italy.

0-7803-5417-6/99/$10.00 ©1999 IEEE.

simply proxy) [1]. A web proxy is an intermediate server acting as an caching agent between clients and server. If properly designed, proxy can significantly reduce the web access delay and alleviate the potential network congestion. Additionally, it also can reduce the server load, which may be critical during peak time. The effectiveness of the proxy is primarily determined by the *locality*, universally true for any cache. This locality depends a number of factors such as users' access patterns and cache configurations. The unique characteristics of web caching, *different* from conventional caching used in memory and distributed systems, is that the locality is also largely influenced by *the location of the web proxy.* Simply put, putting a web proxy in the "wrong" place is not only costly, but also does little to improve the system performance.

There has been considerable work on various aspect of web proxy, for example, traffic characterization [1, 6], the cache replacement algorithms [13, 15, 16], and server design [7, 10]. A number of recent publications have studied some new aspects of web caching. In [4], Bhattacharjee et al. proposes a self-organizing cache scheme in an active network [14]. It suggests associating small caches with switching nodes throughout the network, and considers the use of various self-organizing or active caching management strategies for organizing cache content. The issue of proper caches placement, however, is not addressed. Bestavros considers the problem of caching multiple web servers at a given location [3]. The objective is to maximize the fraction of the requests subject to the fixed storage space. The paper models the problem as a constrained-maximization problem, and obtains the solution using the Lagrange multiplier theorem. Sayal et al. consider the selection algorithms for replicated web servers [12], in which it concerns how a client chooses the "closest" replicated server based on either the hop count, ping round trip time or the HTTP request latency. The replicated servers' locations is assumed to be given, no proper placement is considered.

Finding the optimal placement of web proxies in a network like the Internet is a challenging task. Most existing proxies are placed in fairly "obvious" spots, e.g., the router for a LAN, the Internet service providers (ISP)' gateway, or some "strategic" locations [11] [2]. In addition, it has been shown that multiple web proxies are often needed in order to increase the locality, e.g., the hierarchical caching proposed in [5, 7]. Furthermore, this is also evident that many popular web sites have already employed a replication (mirroring) of the sites. For example, users can select among 101 different servers to access the Netscape browser, 15 different servers to access Yahoo within the US, 12 servers to access Lycos, 8 servers to access American Online, 7 servers to access Alta Vista and 3 servers to access Infoseek [12].

In this paper, we focus on two major factors: the overall traffic and access latency as described in [17]. The objective is to *minimize* the overall latency of searching *a target web server* subject to the system resources and traffic pattern. The algorithms proposed in this paper, however, also work for any objective function not just latency. For example it can be the number of hops between client and proxy, or an overall cost function if each link is associated with a cost. Specifically, we are interested in finding the optimal placement of multiple web proxies (M) among potential sites (N) under a given traffic pattern. The main complication is caused by the *dependencies* among the potential sites. Consider a potential site, say i, it can be in place between another potential site (j) and the web server. We define site i to be upstream of site j and j to be downstream of site i. Caching at any downstream site (j) modifies the traffic pattern of the upstream site (i). Unless the paths from all sites to the server are *disjoint*, in which the finding the optimal location becomes trivial, these dependencies significantly complicate the problem.

In [8], we show that the optimal placement of M proxies among N potential sites under a *linear topology* can be modeled as *dynamic programming problem* and its optimal solution can be obtained within $O(N^2M)$. In [9], we further studied a distributed caching placement algorithm in the content of active network [14]. We show the placement algorithm (non-optimal) can obtain suboptimal solution with significant less complexity ($NlogN$) for a tree topology. In this paper, we consider a tree topology with the target web server being the root of the tree. We show that the optimal placement problem can also be modeled as a *Dynamic Programming* problem, we obtain optimal solution using $O(N^3M^2)$ time;

The rest of the paper is organized as follows. We present the dynamic programming problem formulation in Section 2, and its numerical example in the

[1] We treat the replicated server or mirror site as a special case of web proxy, in which all documents of the original web server/site is cached.

[2] Noticeably, the ISP installed proxies are primarily for alleviating congestion, rather than reducing the latency.

next Section. We conclude the paper in Section 4 with discussions of on-going and future work.

2 The Problem Formulation

We start by describing the formulation of the problem that we will use in devising the dynamic programming algorithm for the tree topology. Let T be a rooted tree containing n nodes. We will always use r to specify the root of T. A node can have arbitrarily many children; we assume that these children are ordered from left to right, and that, given any two siblings u, v, we are able to say that u is to the left of v or that v to is the right of u.

Associated with every node $v \in T$ is a non-negative weight $w(v)$ representing the traffic traversing this node; associated with every edge (u, v) a non-negative distance $d(u, v)$, which can be interpreted as either latency, link cost, hop count and etc. We extend the distance function as follows. Denote the unique path from u to v in the tree by $\pi_{u,v}$. Then $d(u, v) = \sum_{(x,y)\in\pi_{u,v}} d(x, y)$ is the sum of the edge distances along the path.

Suppose now that we are a given a set of vertices $\mathcal{P} \subseteq T$ containing the root, i.e, $r \in \mathcal{P}$. Define $c(v, \mathcal{P})$ to be the lowest ancestor of v which is contained in $\mathcal{P}$, i.e., the first node in $\mathcal{P}$ that is seen while going up from v to r. Note that this node could be v itself. Now set

$$cost(T, \mathcal{P}) = \sum_{v\in T} w(v) \cdot d(v, c(v, \mathcal{P})). \qquad (1)$$

This cost will be the cost associated with the selection of the set $\mathcal{P}$ as proxies. It is the total weighted cost of servicing requests using the proxies located at $\mathcal{P}$. The problem that needs to be solved is the construction of $\mathcal{P}$ with $|\mathcal{P}| = m$ and $r \in \mathcal{P}$ that minimizes this cost. The practical implication of this, assuming that the edge distance $d(u, v)$ represents the latency, is to minimize the overall searching latency for the target web server given that M proxies are allowed. In the following, we will describe an $O(n^3m^2)$ dynamic programming algorithm for doing so. Before describing the algorithm, we need to introduce some preliminary definitions.

Let $v \in T$. We denote by T_v to the subtree of T rooted at v. We generalize the notion of being *left of* to non-sibling nodes. Given x and y in T, x is said to be to the *left of* y if there exist u and v such that $x \in T_u$, $y \in T_v$ and u and v are siblings with u being to the *left of* v. Note that if x and y are siblings then this reduces to the original definition, as $x \in T_x$ and $y \in T_y$. For example, in Figure 2 every node in the set $L_{u,v}$ is to the *left of* every node in the tree T_u.

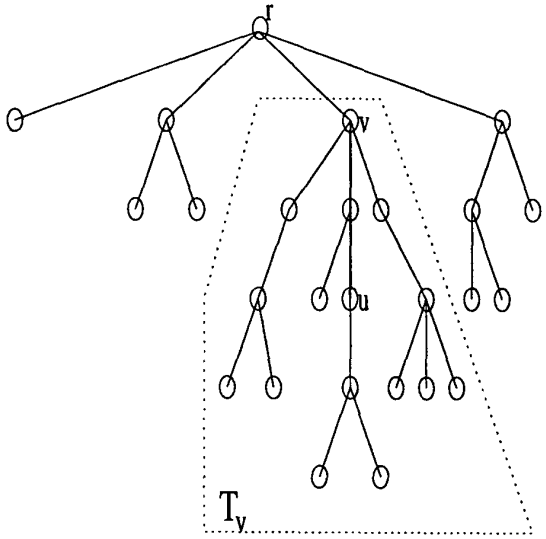

Figure 1: T_v is the subtree rooted at v.

Now suppose $u \in T_v$. Let $\pi_{u,v}$ be the unique path connecting u and v in T. We define

$$\begin{aligned} L_{u,v} &= \{x \in T_v : x \text{ is to the left of } u\} \\ T_{u,v} &= \{x \in T_v : x \notin T_u \cup L_{u,v}\} \end{aligned}$$

Intuitively, T_v is partitioned into $L_{u,v}$, T_u and $T_{u,v}$: $L_{u,v}$ consists of all the nodes of T_v which are to the left of u, T_u contains the node below u (u included), and $T_{u,v}$ contains the remaining nodes: i.e., nodes in $\pi_{u,v} - \{u\}$ plus all the nodes x such that u is to the left of x.

For $x \in T_{u,v}$ we further set

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

We now define

$$\begin{aligned} C(v, t) &= \min_{\mathcal{P}\subseteq T_v, |\mathcal{P}|=t, v\in\mathcal{P}} cost(T_v, \mathcal{P}) \\ C(u, v, t) &= \min_{\mathcal{P}\subseteq T_{u,v}, |\mathcal{P}|=t, v\in\mathcal{P}} cost(T_v, \mathcal{P}) \end{aligned}$$

Put in other words, $C(v, t)$ is the optimal cost of placing t proxies in T_v and $C(u, v, t)$ is the optimal

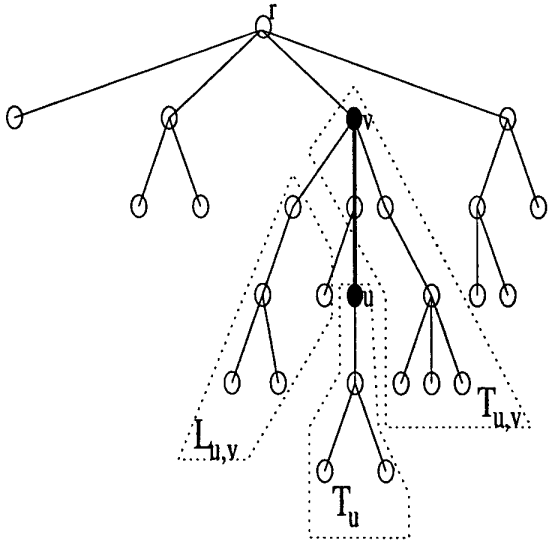

Figure 2: The heavy line is $\pi_{u,v}$. The set $L_{u,v}$ and the trees T_u and $T_{u,v}$ are labelled.

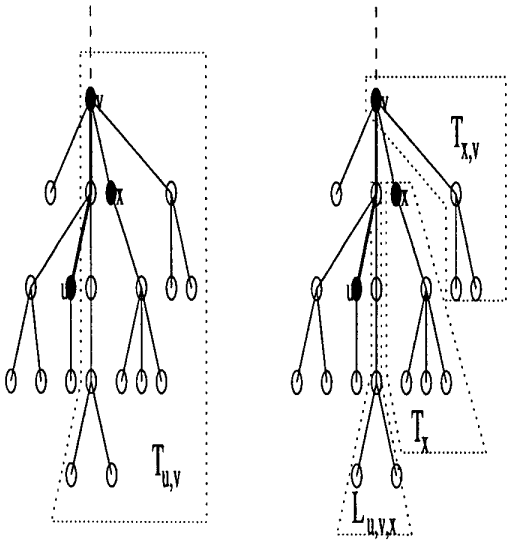

Figure 3: The heavy line is $\pi_{u,v}$. The Tree $T_{u,v}$ is shown in the left hand figure. Its decomposition into $L_{u,v,x}$, T_x and $T_{v,x}$ is illustrated in the right hand figure.

cost of placing t proxies in $T_{u,v}$. Our original problem can then be restated as finding $C(r,m)$ along with the set of proxies $\mathcal{P}$ that achieves it.

We are now almost ready to write out our dynamic programming equation. Before doing so we need one last pair of definitions

$$W(u,v) = \sum_{x \in L_{u,v}} w(x)d(x,v)$$

$$W(u,v,x) = \sum_{y \in L_{u,v,x}} w(y)d(s,v)$$

Let $\mathcal{P}'$ be any set of proxies such that $v \in \mathcal{P}'$ no proxies of $\mathcal{P}'$ are in $L_{u,v}$ and no proxies of $\mathcal{P}'$ are in $\pi_{u,v} - \{u,v\}$. Then for such a set of proxies v is the proxy that services $L_{u,v}$, i.e., for $x \in L_{u,v}$, we have $c(x,\mathcal{P}') = v$. This means that $W(u,v)$ is the total contribution to $cost(T_v,\mathcal{P}')$ given by the nodes in $L_{u,v}$. Similarly if $\mathcal{P}'$ is any set of proxies such that $v \in \mathcal{P}'$ no proxies of $\mathcal{P}'$ are in $L_{x,v}$ and no proxies of $\mathcal{P}'$ are in $\pi_{x,v} - \{x,v\}$ then $W(u,v,x)$ is the total contribution to $cost(T_{u,v},\mathcal{P}')$ given by the nodes in $L_{u,v,x}$.

Assume then we wish to find the optimal cost of placing t proxies in the tree T_v. If $t = 1$ this is trivial, since we are forced to place the only proxy available at v, and the total cost will clearly be $C(v,t) = \sum_{x \in T_v} w(x)d(x,v)$. If $t > 1$, computing $C(v,t)$ is much more difficult. However, assume that in this case we have the extra knowledge that (a) one proxy must be $u \in T_v$; (b) $t' < t$ proxies are overall placed in T_u; (c)no proxies are in $\pi_{u,v} - \{u,v\}$; and (c) no proxies are in $L_{u,v}$. Then this extra information can make the problem much easier for us, as $C(v,t) = (W(u,v) + C(u,t') + C(u,v,t-t'))$.

If we do not have this extra knowledge, then we have to minimize over all possible choices of u and t'. Doing this leads to the following dynamic programming equations (whose correctness will be shown later):

$$C(u,v,t) = \begin{cases} \sum_{x \in T_{u,v}} w(x)d(x,v) & \text{if } t = 1 \\ \min_{x \in T_{u,v}} \min_{0<t'<t} & \\ \left(W(u,v,x) + C(x,t') + C(v,x,t-t')\right) & \text{if } t > 1. \end{cases} \quad (2)$$

The second is $\forall v \in T$

$$C(v,t) = \begin{cases} \sum_{x \in T_v} w(x)d(x,v) & \text{if } t = 1 \\ \min_{u \in T_v} \min_{0<t'<t} & \\ \left(W(u,v) + C(u,t') + C(u,v,t-t')\right) & \text{if } t > 1 \end{cases} \quad (3)$$

To aid construction of the actual proxy sets we also, in equation (2) set $P(u,v,t) = (x,t')$ where $x \in T_{u,v}$ $0 < t' < t$ are the values that minimize the expression and in (3) set $P(v,t) = (u,t')$ where $u \in T_v$, $0 < t' < t$ are the values that minimize the expression.

We now state the major dynamic programming results:

Theorem 1 *Equation (3) is correct. Furthermore, a best way of placing $t > 1$ proxies in T_v is to place t' proxies the best way in T_u and $t - t'$ proxies the best way in $T_{u,v}$ where $P(v,t) = (u,t')$.*

Theorem 2 *Equation (2) is correct. Furthermore, a best way of placing $t > 1$ proxies in $T_{u,v}$ is to place t' proxies the best way in T_x and $t - t'$ proxies the best way in $T_{x,v}$ where $P(u,v,t) = (x,t')$.*

For the moment we will assume the correctness of the theorems (they will be proven at the end of this section) and concentrate on their implications.

The first implication is that, given the $P()$ table we can construct the best way of placing t proxies in T_v. To do this, we basically have to backtrack through the tables: for a given (v,t) pair if $t = 1$ just choose v as the proxy. Otherwise let $(u,t') = P(v,t)$ and build the best set $\mathcal{P}$ of t proxies for T_v by building the best set $\mathcal{P}_{t'}$ of t' proxies for T_u and building the best set $\mathcal{P}_{t-t'}$ of $t - t'$ proxies for $T_{u,v}$ and then combining $\mathcal{P} = \mathcal{P}_{t'} \cup \mathcal{P}_{t-t'}$. Similarly for a given (u,v,t) triple we can construct the best way of placing t proxies in $T_{u,v}$: if $t = 1$ just choose v as the proxy. Otherwise let $(x,t') = P(u,v,t)$ and build the best set $\mathcal{P}$ of t proxies for $T_{u,v}$ by building the best set $\mathcal{P}_{t'}$ of t' proxies for $T_{u,v}$ and building the best set $\mathcal{P}_{t-t'}$ of $t - t'$ proxies for $T_{x,v}$ and then combining $\mathcal{P} = \mathcal{P}_{t'} \cup \mathcal{P}_{t-t'}$. Working through the above we find that, given pre-computed $P()$, we can compute the optimal set of x proxies for $T = T_r$ in $O(m)$ time.

The second implication is that all the entries $C(v,t)$, $C(u,v,t)$ and the associated $P()$ entries can be calculated in a total of $O(n^3m^2)$ time. We now see this in further detail [3].

First of all, we have to compute in a preprocessing phase the $O(n^3)$ values $W(u,v)$ and $W(u,v,x)$. All these values can be calculated in a total of $O(n^3)$ time recursively. Once they are computed, they are stored in a matrix.

After this preprocessing, we can calculate the entries $C(v,t)$ and $C(u,v,t)$ along with their associated $P()$ values according to the dynamic programming equations (2) and (3). To do so, we first need to find a proper ordering in which to process these entries, so that at the time an entry $C(v,t)$ or $C(u,v,t)$ is calculated all of the entries that are used in its minimization equation (2) or (3) have already been calculated. There are many such orderings. One trivial one is to simply to sort the entries by increasing t value $t = 1, 2, 3, \ldots$. Note that this is a proper ordering because entries with $t > 1$ only access entries with smaller t values in their minimization in (2) and (3).

We now calculate all of the entries using this ordering. Since there are at most $O(n^2m)$ different entries to compute and each one of them requires at most $O(nm)$ time to evaluate, this immediately leads to an $O(n^3m^2)$ algorithm for calculating the entries. Combining this with the $O(m)$ time algorithm for building an optimal proxy set using the pre-calculated $P()$ tables, this leads to a $O(n^3m^2)$ algorithm for building an optimal proxy set.

It remains to prove the correctness of the theorems. We will actually only prove Theorem 1 since the proof of Theorem 2 is almost the same. To prove Theorem 1 first note that if $t = 1$ the proxy must be placed at v so the theorem is trivially true.

So now suppose that $t > 1$. Set

$$X(v,t) = \min_{u \in T_v} \min_{0<t'<t} \left(W(u,v) + C(u,t') + C(u,v,t-t') \right) \quad (4)$$

Let $\mathcal{P}$ be a set of t proxies such that $cost(T_v, \mathcal{P}) = C(v,t)$. Now let $\overline{u} \in \mathcal{P}$ be such that $\mathcal{P} \cap L_{\overline{u},v} = \emptyset$ and $\mathcal{P} \cap \pi_{\overline{u},v} = \{v\}$.

Set $\overline{t} = \mathcal{P} \cap T_{\overline{u}}$ to be the number of proxies in $T_{\overline{u}}$. Since there are no proxies in $L_{\overline{u},v}$ and $\pi_{\overline{u},v} - \{v, \overline{u}\}$ this means that

$$C(v,t) = W(\overline{u},v) + Cost(T_{\overline{u}}, \mathcal{P} \cap T_{\overline{u}}) + Cost(T_{\overline{u},v}, \mathcal{P} \cap T_{\overline{u},v}) \quad (5)$$

We claim that $Cost(T_{\overline{u}}, \mathcal{P} \cap T_u) = C(\overline{u}, \overline{t})$ since otherwise we could replace the $\overline{t}$ proxies in $\mathcal{P} \cap T_u$ by an optimal set of $\overline{t}$ proxies for $T_{\overline{u}}$ and achieve a lower cost for T_v. Similarly $Cost(T_{\overline{u},v}, \mathcal{P} \cap T_{\overline{u},v}) =$

[3] This section only describes the important details of the algorithm. We leave some of the implementation particulars, such as how to order nodes and edges, to the reader.

$C(\overline{u}, v, t-\overline{t})$ since otherwise we could replace the $t-\overline{t}$ proxies in $\mathcal{P} \cap T_{\overline{u},v}$ by an optimal set of $t-\overline{t}$ proxies for $T_{\overline{u},v}$ and find a lower cost for T_v. We note that this latter replacement might actually contain proxies on $\pi_{\overline{u},v}$ whereas before we assumed the path was proxy-free. This is not a problem since such a placement could only reduce the cost (by reducing the distance from some elements in $L_{\overline{u},v}$ to a proxy) which is not possible.

By plugging in $u = \overline{u}$ and $t = \overline{t}$ we have therefore just seen that

$$C(v,t) \geq X(v,t). \tag{6}$$

But now suppose $u \in T_v$ and $t' < t$ are arbitrary. Let $\mathcal{P}'$ be an optimal set of t' proxies for T_u and $\mathcal{P}''$ an optimal set of $t-t'$ an optimal set of t' proxies for $T_{u,v}$. Then

$$\begin{aligned} C(v,t) &\leq Cost(T_v, \mathcal{P}' \cup \mathcal{P}'') \\ &\leq W(u,v) + Cost(T_u, \mathcal{P}') + Cost(T_{u,v}, \mathcal{P}'') \\ &= W(u,v) + C(u,t') + C(u,v,t-t') \end{aligned}$$

so

$$C(v,t) \leq X(v,t). \tag{7}$$

Combining (6) and (7) yields

$$C(v,t) = X(v,t). \tag{8}$$

To complete the proof of Theorem 1 it remains to show that *a best way of placing $t > 1$ proxies in T_v is to place t' proxies the best way in T_u and $t-t'$ proxies the best way in $T_{u,v}$ where $P(v,t) = (u,t')$.*

We start by noting that

$$C(v,t) = W(u,v) + C(u,t') + C(u,v,t-t'). \tag{9}$$

Now let $\mathcal{P}'$ be any best way of placing t' proxies in T_u, $\mathcal{P}''$ any best way of placing $t-t'$ proxies in $T_{u,v}$. Set $\mathcal{P} = \mathcal{P}' \cup \mathcal{P}''$. By definition

$$cost(T_u, \mathcal{P}) = cost(T_u, \mathcal{P}') = C(u,t) \tag{10}$$

and

$$cost(T_{u,v}, \mathcal{P}) = cost(T_{u,v}, \mathcal{P}'') = C(u,v,t). \tag{11}$$

Also note that for all $x \in T_v$ the furthest node away (above) x is v so $d(x, c(x,\mathcal{P})) \leq d(x,v)$. In particular, summing over all $x \in L_{u,v}$, this means that

$$\sum_{x \in L_{u,v}} w(x) d(x, c(x,\mathcal{P})) \leq \sum_{x \in L_{u,v}} w(x) d(x,v) = W(u,v).$$

Combining this with (10), (11) and (9) yields

$$cost(T_v, \mathcal{P}) \leq W(u,v) + C(u,t') + C(u,v,t-t') = C(v,t).$$

But, by definition, $C(v,t)$ is the minimum possible cost achievable by $t = |\mathcal{P}|$ proxies so $cost(T_v, \mathcal{P}) = C(v,t)$, $\mathcal{P}$ is an optimal proxy set and we are done.

3 Numerical Examples

We consider an example given in Figure 4, there are $n = 11$ nodes (including the root being the server) and we're interested in finding $m = 4$ (the root is a default proxy). The node weight and edge weight are given as $(w(v), d(u,v))$ and node u is the parent of node v. For instance, the node 11 has weight $w(11) = 0.1$ and edge $d(6,11) = 9.0$. The algorithm basically needs to find the leftmost node in the optimal solution v, and $C(v,t)$ is the optimal cost of placing t proxies in T_v and $C(u,v,t)$ is the optimal cost of placing t proxies in $T_{u,v}$. This is done by tracing all v and then recursively finding the optimal cost within the two subtrees, i.e., T_v and $T_{u,v}$, respectively.

There are many ways to implement the proposed dynamic programming algorithm. Our implementation allows us to be able to obtain the solution for 2000 nodes and 100 proxies case within reasonable amount of time and space. A straightforward implementation can simply sort all subtrees by the size, and then proceeds by solving the proxy allocation problem in the smallest subtree, and moves up to the bigger subtrees.

The Table 1 shows the optimal solutions for $m = 1, 2, 3, 4$. The algorithm assumes that the root of the tree is a default proxy, therefore the algorithm tries to find out $m-1$ proxies in the rest of the tree. A row in the Table 1 corresponds to either T_v or $T_{u,v}$. For example, $v = 9$ refers to the subtree T_9 (node 9 itself is the only node in the subtree T_9). The row $v = 1, u = 2$ refers to the subtree $T_{2,1}$. The column in the Table 1 corresponds to the number of proxy (or proxies) needed, with the first column being the default proxy at the root. The entries in the Table are given as $(u, t, C(v,m))$ for the row of

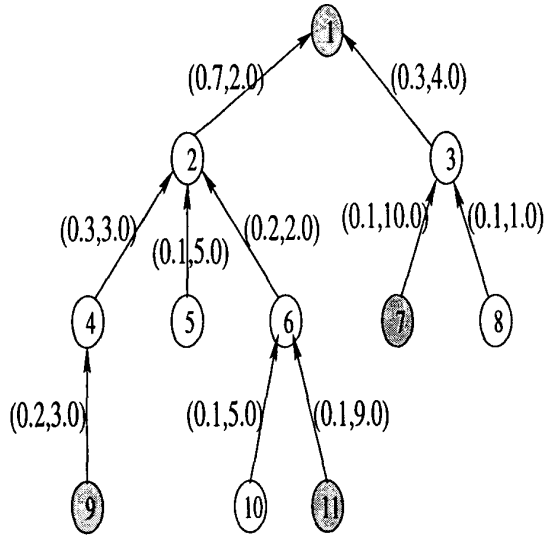

Figure 4: A tree topology example. Node and edge weights are given as $(w(u), d(u,v))$

(v,m) (u,v,m)	m = 1	m = 2	m = 3	m = 4
v=1	(–, –, 7.5)	(9, 1, 5.9)	(9, 1, 4.5)	(9, 1, 3.2)
v=2	(–, –, 3.8)	(9, 1, 2.6)	(9, 1, 1.5)	(9, 1, 0.8)
v=3	(–, –, 1.1)	(7, 1, 0.1)	(7, 1, 0.0)	
v=4	(–, –, 0.6)	(9, 1, 0.0)		
v=5	(–, –, 0.0)			
v=6	(–, –, 1.4)	(11, 1, 0.5)	(10, 1, 0.0)	
v=7	(–, –, 0.0)			
v=8	(–, –, 0.0)			
v=9	(–, –, 0.0)			
v=10	(–, –, 0.0)			
v=11	(–, –, 0.0)			
v=1 u=2	(–, –, 2.3)	(7, 1, 0.9)	(3, 2, 0.1)	(3, 3, 0.0)
v=1 u=3	(–, –, 0.0)			
v=1 u=4	(–, –, 5.2)	(7, 1, 3.8)	(11, 1, 2.5)	(10, 1, 1.6)
v=1 u=5	(–, –, 4.5)	(7, 1, 3.1)	(11, 1, 1.8)	(10, 1, 0.9)
v=1 u=6	(–, –, 2.3)	(7, 1, 0.9)	(3, 2, 0.1)	(3, 3, 0.0)
v=1 u=7	(–, –, 0.5)	(8, 1, 0.0)		
v=1 u=8	(–, –, 0.0)			
v=1 u=9	(–, –, 5.2)	(7, 1, 3.8)	(11, 1, 2.5)	
v=1 u=10	(–, –, 3.6)	(7, 1, 2.2)	(11, 1, 0.9)	
v=1 u=11	(–, –, 2.3)	(7, 1, 0.9)	(3, 2, 0.1)	
v=2 u=4	(–, –, 2.3)	(11, 1, 1.2)	(10, 1, 0.5)	(5, 1, 0.0)
v=2 u=5	(–, –, 1.8)	(11, 1, 0.7)	(10, 1, 0.0)	(6, 3, 0.0)
v=2 u=6	(–, –, 0.0)			
v=2 u=9	(–, –, 2.3)	(11, 1, 1.2)	(10, 1, 0.5)	(5, 1, 0.0)
v=2 u=10	(–, –, 1.1)	(11, 1, 0.0)		
v=2 u=11	(–, –, 0.0)			
v=3 u=7	(–, –, 0.1)	(8, 1, 0.0)		
v=3 u=8	(–, –, 0.0)			
v=4 u=9	(–, –, 0.0)			
v=6 u=10	(–, –, 0.9)	(11, 1, 0.0)		
v=6 u=11	(–, –, 0.0)			

Table 1: The optimal solution for the example tree topology

T_v, and $(x, t, C(u,v,m))$ for the row of $T_{u,v}$. For example, the entry of the row $v = 6$ (the subtree T_6) and the column $m = 2$ is $(11, 1, 0.5)$ $((u, t, C(v,m)))$. This means that within the subtree T_6, 2 (m) proxies need to be chosen, one is the root, i.e., node 6, the other is node $u = 11$, and the cost is 0.5 ($w(10) * d(10,6) = 0.5$ since node 10 is not a proxy node). Take another example, consider the entry of the row $v = 1$ $u = 2$ ($T_{2,1}$) and the column $m = 3$, which shows $(3, 2, 0.1)$ $((x, t, C(u,v,m)))$. This implies that within the subtree $T_{2,1}$, $m = 3$ proxies need to be chosen, and node $x = 3$ is one of the proxies in the optimal solution (root node 1 being another). It also shows that under the subtree T_3, $t = 2$ nodes will be chosen as proxies (including node $x = 3$) in the optimal solution. Now let's look at the T_3 subtree (row $v = 3$) with $m = 2$ (column $m = 2$), the entry shows $(7, 1, 0.1)$, meaning node 7 is the other proxy in the optimal solution. The cost 0.1 is obtained from $w(8) * d(8,3)$ since node 8 is not chosen as a proxy.

Now let's focus on the solution shown in the Table 1 for finding $m = 4$ proxies including the root in the tree topology given in Figure 4. The entry of $v = 1$ and $m = 4$ in the Table 1 is $(9, 1, 3.2)$, which means that the leftmost proxy in the optimal solution is node 9, and in the subtree T_9, only one proxy will be chosen in the optimal solution, i.e., the node 9. We next proceed to consider the subtree $T_{9,1}$, now the $m = 3$. The correspondent entry in the Table 1 is $(11, 1, 2.5)$. This implies that in the subtree of $T_{9,1}$, node 11 is the leftmost proxy in the optimal solution, and it is the only node of the optimal solution in the subtree T_{11}. Now it moves the subtree $T_{11,1}$, and $m = 2$, the entry shows node 7 is the node in the optimal solution. We don't need to proceed further, since the last node of the optimal solution is the root, i.e., node 1. Therefore, the $m = 4$ solution is nodes $1, 7, 9$ and 11 shown as "shaded" nodes in Figure 4. The Figure 5 re-captures the above described process.

Notice the process illustrated above is a top-down approach, but the actual implementation of the algorithm, as pointed out earlier, can be done bottom-up by calculating all the smaller subtrees first before moving up to the bigger subtrees.

4 Conclusions

In this paper, we investigate the optimal placement policy of web proxies for a target web server in the

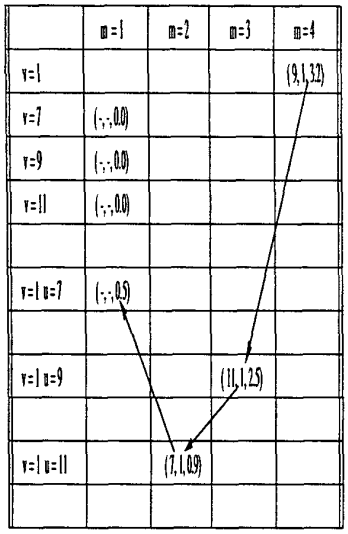

Figure 5: The optimal solution for the example tree topology

Internet. The objective is to minimize the overall latency of searching the target web server subject to the network resources and traffic pattern. Our contributions are 1) formulating the optimal proxy placement problem into a *Dynamic Programming* problem; 2) obtaining an optimal solution for the tree topology using $O(N^3M^2)$;

The model proposed here can be easily extended to handle the following two cases:

- Hierarchical caching, in which the down-stream proxies only hold a subnet of the documents of the up-stream proxies. In such cases each down-stream proxy can only block a portion of the traffic. This can be handled by re-defining the notation $w(i)$ (or $w(u)$) to be only the percentage of the traffic that the i^{th} node's cache can serve.
- Different link bandwidth. This can be dealt with by incorporating the link bandwidth into the distance $L(i)$ (or $d(u,v)$), e.g., assigning larger value to slower links.

We are currently working on two extensions: one is to reduce the algorithm complexity for the tree topology. Preliminary result indicates that this can be easily brought down to $O(N^2M^2)$. The second extension is to consider multiple web servers to be cached at multiple locations. We have done some initial assessment which reveals that the proposed dynamic programming formulations in this paper are not suitable for the multiple target web servers case. Clearly the single target web server proxies placement problem adressed in this paper is a special case of the multiple target web servers case, in which each target web server forms its own spanning-tree topology. The interesting aspect is that these multiple spanning trees share the same set of nodes except the root (target web servers), i.e., the potential proxy locations (or the set of routers where the proxies can be potentially associated with [4, 9]). The edges of each spanning-trees, however, are not necessarily the same. Most importantly, The optimal solution for one tree in general is not the optimal solution for any other tree.

We will consider a number of issues in the future work, including more realistic web traffic distribution, for example capturing the actual workload characterization [1], or considering the Zipf distribution used by Bestarov [3]. How to incorporate the dynamic nature of the traffic into the proxy placement mechanism remains a challenge.

References

[1] M. F. Arlitt and C. L. Williamson, "Internet Web Servers: Wordload Characterization and Performance Implications," *IEEE Transactions on Networking*, Vol. 5, No. 5, October 1997.

[2] M. Baentsch, L. Baum, G. Molters. S. Rothkugel and P. Sturm, "World Wide Web Caching: The Application-Level View of the Internet," *IEEE Communications Magazine*, Vol. 35, No. 6, June 1997.

[3] A. Bestavros, "WWW Traffic Reduction and Load Balancing Through Server-based Caching," *IEEE Concurrency*, January 1997.

[4] S. Bhattacharjee, K. L. Calvert and E. W. Zegura, "Self-Organized Wide-Area Network Caches," *IEEE Infocom'98*, San Francisco, March 1998.

[5] C. Bowman, P. Danzig, D. Hardy, U. Manber and M. Schwartz, "The Harvest Information Discovery and Access System," *Computer Networks and ISDN Systems*, Vol. 28, No. 1-2, December 1995.

[6] H-W. Braun and K. C. Claffy, "Web Traffic Characterization: An Assessment of the Impact of Caching Documents from NCSA's Web Server," *Computer Networks and ISDN Systems*, Vol. 28, No. 1-2, December 1995.

[7] S. Glassman, "A Caching Relay for World Wide Web," *Computer Networks and ISDN Systems*, Vol. 27, No. 2, November 1994.

[8] B. Li, X. Deng, M. Golin and K. Sohraby, "On The Optimal Placement of Web Proxies in the Internet: Linear Topology," *the 8th IFIP Conference on High Performance Networking (HPN'98)*, Vienna, Austria, September 1998.

[9] B. Li, X. Deng, M. Golin and K. Sohraby, "Dynamic and Distributed Web Caching in Active Networks," *Asia Pacific Web Conference'98 (APWeb'98)*, Beijing, China, September 1998.

[10] A. Luotonen and K. Altis, "World Wide Web Proxies," *Computer Networks and ISDN Systems*, Vol. 27, No. 2, November 1994.

[11] M. Nabeshima, "The Japan Cache Project: An Experiment on Domain Cache," *Computer Networks and ISDN Systems*, Vol. 29, No. 8-13, September 1997.

[12] M. Sayal, Y, Breitbart, P. Scheuermann and R. Vingralek, "Selection Algorithms for Replicated Web Servers," *Internet Server Performance Workshop in conjunection with ACM Sigmetrics'98/Performance'98*, Madison, WI, June 1998

[13] P. Scheuermann, J. Shim and R. Vingralek, "A Case for Delay-Conscious Caching of Web Documents", *Computer Networks and ISDN Systems*, Vol. 29, No. 8-13, September 1997.

[14] D. Tennenhouse, J. Smith, W. Sinncoskie, D. Wetheral and G. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, Vol. 35, No. 1, January 1997.

[15] S. William, M. Abrams, C. Standridge, G. Abdulla and E. Fox, "Removal Policies in Network Caches for World-Wide Web Documents," *ACM Sigmetrics'96*, Stanford University, Palo Alto, CA, 1996.

[16] R. Wooster and M. Abrams, "Proxy Caching That Estimates Page Load Delay," *The 6th World Wide Web Conference*, Santa Clara, CA, 1997.

[17] N. Yeager and R. McGrath, Web Server Technology, *Morgan Kaufman*, 1996.