Optimal Content Placement for En-Route Web Caching*

Anxiao (Andrew) Jiang and Jehoshua Bruck
California Institute of Technology
Parallel and Distributed Systems Lab
MC 136-93
Pasadena, CA 91125, U.S.A.
E-mail: {jax,bruck}@paradise.caltech.edu

Abstract

This paper studies the optimal placement of web files for en-route web caching. It is shown that existing placement policies are all solving restricted partial problems *of the file placement problem, and therefore give only sub-optimal solutions. A dynamic programming algorithm of low complexity which computes the optimal solution is presented. It is shown both analytically and experimentally that the file-placement solution output by our algorithm outperforms existing en-route caching policies. The optimal placement of web files can be implemented with a reasonable level of cache coordination and management overhead for en-route caching; and importantly, it can be achieved with or without using data prefetching.*

1. Introduction

Web caching is one of the main techniques solving the performance problems the World Wide Web faces today. WWW has been experiencing very fast growth in recent years, but long access latency can seriously hurt its popularity, especially for hot websites. Web caching dynamically stores popular files in different places of the Internet, thus decreasing the distance between clients and web content. It can significantly reduce network congestion, server load and access delay. A huge amount of research effort has been devoted to all aspects of web caching, and various caching schemes have been proposed [2] [5] [8] [10] [11].

Effective caching requires cooperative content management of web caches. Traditional caches include clients, proxies and servers. Two common approaches to coordinate caches are *Hierarchical Caching* and *Distributed Caching* [8], where caches are located at one or more levels of the network and file-requests are forwarded from low-level caches to higher-level caches or among caches at the same level. Some hybrid caching architectures also exist [8]. A new caching architecture, called *En-Route Caching* [1] [5] [7] [10], differs from hierarchical caching and distributed caching in that caches are associated with routing nodes and that a request is always forwarded from the client toward the web server along the *regular* routing path. Every en-route cache inspects the requests that pass through its associated routing node. If it has the requested file, it transmits the file to the client and the request is satisfied. Otherwise, it forwards the request along the regular routing path. En-route caching has the merit that it is transparent to both clients and servers, and requires no file location mechanisms such as broadcasting queries or exchanging content summaries [10]. So it's easy to manage in this sense. And it provides a much stronger capability to locate caches really *inside* the network, whose effectiveness has been shown [4].

File placement/replacement is a key technique that affects the effectiveness of caching. A large number of file placement/replacement policies are available for en-route caching [6] [11]. Most policies make decisions on file placement and replacement for individual caches only. Some policies, such as MODULO [1], consider the path from the cache (or server) containing the file to the client, and cache the file along the path using simple placement schemes. Recently a novel caching scheme, called *Coordinated En-Route Web Caching*, is proposed by Tang et al. [10]. The coordinated en-route web caching scheme *optimizes* the placement of files along the path from the cache (or server) to the client, and it requires moderately more coordination among the en-route caches. Its performance has been shown to be significantly better than the other policies [10].

This paper explores the file placement techniques for en-route web caching. We study file placement policies in a more general caching model, and show that existing

[1]This work was supported in part by the Lee Center for Advanced Networking at the California Institute of Technology.

policies, including the coordinated en-route web caching scheme, are all solving *restricted partial problems* of the placement problem, and therefore they give only sub-optimal solutions. We then present a dynamic programming algorithm which computes the optimal solution for file placement. It is shown both analytically and experimentally that the optimal solution given by our algorithm can be significantly better than the sub-optimal solutions given by other schemes. Implementation details are introduced, and it's shown that our scheme requires the same level of coordination among caches as the *coordinated en-route web caching scheme*. It is proven that the optimal placement can be implemented in an *independently successive way*—meaning that the file can be cached only in caches that it necessarily passes through, and successive independent computation and caching will aggregately give the optimal placement. Thus the optimal placement can be achieved with or without using prefetching (data pushing). That is a very important property desired by any caching scheme.

The rest of the paper is organized as follows. In Section 2, a general model for the file placement of en-route web caching is presented, and the performance of existing placement schemes and that of the optimal scheme are compared. In Section 3, the dynamic programming algorithm solving the optimal file placement problem is presented. The algorithm has complexity $O(|V|^2)$, where $|V|$ is the number of caches in consideration. In Section 4, implementation details are introduced. In Section 5, simulation results showing the performance difference between the optimal scheme and other existing schemes are provided. In Section 6, we conclude this paper.

2. Modelling En-Route Web Caching

In this section we model en-route web caching, and compare the performance of different file placement policies.

2.1. Caching Model and File Placement

The model we use in this paper closely follows the network model in [10]. We model the network as a graph $G = (V, E)$, where V is the set of routers each of which is associated with an en-route cache, and E is the set of network links. Each server or client is attached to a node in V. Without loss of generality, we assume there is only one server, and clients request for web files maintained by the server. A client's request goes along the path from the client to the server, and is satisfied by the first node on the path whose cache stores the requested file. The file from the cache is transmitted downstream along the same path to the client. For simplicity, symmetric routing is assumed here. (If the routing is asymmetric, then we let V only include those nodes on both upstream and downstream paths. Such a simplification is validated by Tang et al. in [10], where it is pointed out that for en-route caching, nodes not contained in both upstream and downstream paths are not appropriate locations for caching the file.) Routing paths from all clients to the server form a tree topology [5] [8] [10].

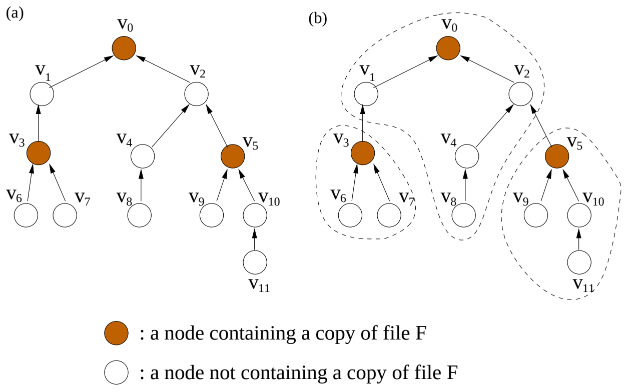
(a)
(b)
v0
v1
v2
v3
v4
v5
v6
v7
v8
v9
v10
v11
: a node containing a copy of file F
: a node not containing a copy of file F

Figure 1. (a) En-Route Web Caching (b) Subtrees Corresponding to Cached File Copies

An example of such a tree topology is shown in Figure 1(a). Here node v_0 is the router associated with the web server, while all other nodes are associated with en-route caches. For any web file F and every cache (or server) which contains the file F, the set of nodes in the network whose requests for F are satisfied by that particular node containing F form a subtree. Figure 1(b) shows the three subtrees corresponding to the three nodes containing the file F. Clearly in every such subtree, there is only one node containing the file F, which is the node closest to the server.

For a file F, we associate every edge $(u, v) \in E$ with a nonnegative cost $c(u, v, F)$, which represents the cost of transmitting a request for F and the corresponding response through edge (u, v). As in [10], the 'cost' here has a general meaning which can correspond to delay, data flow or request-processing cost. If a request goes through multiple edges, the total cost is considered to be the summation of the cost over each edge.

Consider a node A_0 which contains the file F. We use $U = \{A_0, A_1, A_2, \cdots, A_n\}$ to denote the set of nodes whose requests for F are satisfied by A_0. So nodes in U and the associated edges form a subtree—which we denote by T—of the network. We call A_0 the *root* of the subtree T. Let $f(A_i)$ $(1 \le i \le n)$ denote the rate of requests for F passing through node A_i (including the requests from A_i itself and from others). Then the total cost of the requests for F from nodes in U is $\sum_{i=1}^{n} f(A_i) \cdot c(A_i, P_{A_i}, F)$, where P_{A_i} is the parent of node A_i in the tree T.

Currently A_0 is the only node in the tree T which contains the file F. If the rates of requests for F are high, it's beneficial to cache more copies of F in the tree. However, because of the limited memory capacity of each cache, if F is to be stored in a cache, then one or more files in the cache will need to be removed in order to make room. Caching the file F at a node decreases the cost for accessing F in the future (referred to as *cost saving*), but increases the cost for accessing the files that are removed (referred to as *cost loss*).

Our goal is to minimize the access cost for both the file F and the files removed. Assume we select a set R of r nodes, $R = \{A_{j_1}, A_{j_2}, \cdots, A_{j_r}\} \subseteq U - \{A_0\}$, to cache F. Thus the cost for accessing F is reduced. Define B_i ($1 \leq i \leq r$) as the node that satisfies the following three requirements: (1) $B_i \in R \bigcup \{A_0\}$; (2) B_i is an ancestor of A_{j_i} in the tree T; (3) no node in R is both an ancestor of A_{j_i} and a descendant of B_i in the tree T. Then the *cost saving* here can be shown to be: $\sum_{i=1}^{r} \sum_{(u_1,u_2) \in PATH(A_{j_i}, B_i)} f(A_{j_i}) \cdot c(u_1, u_2, F)$, where $PATH(A_{j_i}, B_i)$ is the set of edges on the path between A_{j_i} and B_i.

Removing a file O from a node A_{j_i} will cause cost loss $\sum_{(u_1,u_2) \in PATH(A_{j_i}, C_{O,i})} f_{O,i} \cdot c(u_1, u_2, O)$, where $f_{O,i}$ is the rate of requests for O passing through node A_{j_i}, and $C_{O,i}$ is the node containing the file O which will satisfy the requests coming from A_{j_i} for the file O once O is removed from A_{j_i}. The cost loss of removing multiple files from a node A_{j_i} is simply the summation of the cost loss of removing each file from A_{j_i}.

Deciding which file to remove from a cache is the file replacement problem. There exist a large number of file replacement policies. In this paper, we adopt replacement policies that optimize access cost, such as LNC-R [9]. Let $l(A_{j_i})$ be the cost loss of removing files from node A_{j_i} to make enough room for storing file F. Then the total *cost loss* is $\sum_{i=1}^{r} l(A_{j_i})$.

The above cost-loss formula is used in [10], too. We would like to point out that strictly speaking, the cost loss of removing files at several nodes is not simply the summation of the cost loss of removing files at each node individually, if the same file is removed from at least two nodes and those two nodes are successive among the sites caching the file. However, files removed by cost-based replacement policies usually have very low access frequencies, therefore are sparsely populated among caches, which makes the above scenario unlikely to happen. So the formula above is a good approximation for the total cost loss.

Now we can define an '*optimal placement* of file F on tree T' as follows: an optimal placement of file F on tree T is to cache file F on a set of nodes $\{A_{j_1}, A_{j_2}, \cdots, A_{j_r}\} \subseteq U - \{A_0\}$ such that the *net* cost saving (cost saving minus cost loss) $\sum_{i=1}^{r} \sum_{(u_1,u_2) \in PATH(A_{j_i}, B_i)} f(A_{j_i}) \cdot c(u_1, u_2, F) - \sum_{i=1}^{r} l(A_{j_i})$ is maximized.

2.2. Performance Comparison of Placement Policies

There are lots of file placement policies available for en-route caching. For most of them, when a file is transmitted from a cache (or server) to a client, the file is cached on every node along the path. And at each individual node, some file replacement policy is used to evict files to create space for the newly cached file. Examples of such replacement policies include LRU, LFU, LRU-MIN, Hybrid, LNC-R, GD-Size, etc.. For some placement policies, the file is still cached along the path when it's being transmitted to the client, but each node on the path decides independently whether or not it's beneficial to cache the file, based on some key attribute or other admission control mechanisms. Some file placement policies cache files in a more coordinated way. An example is *MODULO caching* [1], which caches a file on nodes that are a fixed number of hops apart along the path between the server (or cache) and the client.

The *Coordinated En-Route Web Caching scheme* presented recently in [10] is a file placement policy which *optimizes* the placement of the file along the whole path from the cache (server) to the client. It uses the same cost-saving and cost-loss formulas as in this paper, although they are written in different forms. The scheme considers a linear array (a path) instead of a tree. Thus it can be seen as a special (or restricted) case of the optimization problem considered in this paper.

All the file placement policies discussed above try to optimize the placement of a file on the path from a cache (server) to a client, some considering individual nodes only, while others considering the whole path. None of them considers the placement of a file over a tree. Let T denote the same tree as in the previous subsection, which consists of a *root* node containing the file F and all the nodes whose requests for F are serviced by the root. Although after enough requests for F from different nodes of T are sent, each of which causing a placement of F on a path, we will get a placement of the file F over the whole tree T, that placement is the aggregation of the placements on single paths which might be locally optimal but are globally sub-optimal. So the global placements on T of existing file placement policies are sub-optimal.

We use the following example to illustrate the sub-optimality of existing file placement policies.

Example : In this example, we consider three file placement policies: caching a file on every node the file passes through, the *Coordinated En-Route Web Caching scheme* presented in [10], and the *optimal placement* as defined in the previous subsection.

A tree of 4 nodes is shown in Fig. 2, where A_0 is the only node that contains a file F. Assume A_2 issues a request for F first, and A_3 issues a request for F some time later. (Note

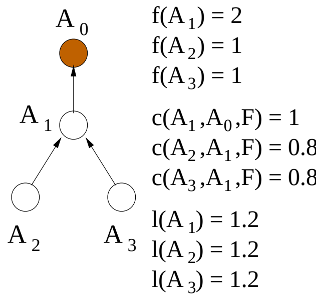

Figure 2. File Placement on Trees

that A_1 doesn't issue any request for F because $f(A_1) = f(A_2) + f(A_3)$.)

When the policy 'caching a file on every node the file passes through' is used, after both requests of A_2 and A_3 are satisfied, clearly F will be cached on all nodes in the tree, and the *net* cost saving is $\sum_{i=1}^{3} f(A_i) \cdot c(A_i, P_{A_i}, F) - \sum_{i=1}^{3} l(A_i) = 0$.

When the *Coordinated En-Route Web Caching scheme* is used, when A_2's request reaches A_0, A_0 computes the placement of F on the path between A_0 and A_2 which will maximize the *net* cost saving — and in this case the *net* cost saving will be maximized by placing F on A_1. So when A_0 sends F to A_2 in response to A_2's request, F is cached on A_1, which causes a *net* cost saving of $f(A_1) \cdot c(A_1, A_0, F) - l(A_1) = 0.8$. After this moment, every time A_i ($i = 2, 3$) issues a request for F, it will be satisfied by A_1; and placing F on A_i ($i = 2, 3$) will cause a *net* cost saving of $f(A_i) \cdot c(A_i, A_1, F) - l(A_i) = -0.4$. So when A_1 sends F to A_i ($i = 2, 3$) in response to A_i's request, F won't be cached on the path between A_1 and A_i (excluding the node A_1 which has already cached F). So the total *net* cost saving stabilizes to be 0.8.

It can be verified that the *optimal placement* of F, which maximized the *net* cost saving for the tree, is to cache F on nodes A_2 and A_3, whose corresponding *net* cost saving is $\sum_{i=2}^{3} \sum_{(u_1,u_2) \in PATH(A_i, A_0)} f(A_i) \cdot c(u_1, u_2, F) - \sum_{i=2}^{3} l(A_i) = 1.2$, better than the outputs of other policies. $\square$

It can be shown that even if the tree is a linear array, all the sub-optimal file placement polices discussed so far (those except the *optimal placement* policy for trees) can output non-optimal solutions, if the first file request is not issued by the end-node of the array. What's more, further analysis shows that in the worst case the following relative performance will be achieved for every sub-optimal file placement policy: the policy caches $O(n)$ more copies of the file than the optimal solution does, where n is the number of nodes in the tree; and the ratio between the net cost saving of the optimal solution and the net cost saving of that policy approaches ∞ (if the net cost saving of the policy is positive). For simplicity of this paper we omit this analysis.

3. Optimal File Placement Algorithm

In this section we formally define the optimal file placement problem for en-route web caching, and present a dynamic programming algorithm which gives the optimal solution. The notations used in this section will be slightly different from those in previous sections for simplicity.

Definition 1 *$T = (V, E)$ is a tree, where V is the set of vertices and E is the set of edges. The tree T has a vertex called its 'root'. For every vertex $v \in V$, $D(v)$ denotes the set of all the vertices that are descendants of v, and $C(v)$ denotes the set of all the vertices that are children of v. (So $D(v) \supseteq C(v)$.) For any two vertices $u \in V$ and $v \in V$, $PATH(u, v)$ denotes the set of all the edges on the path between u and v. For every edge $(u, v) \in E$, it is associated with a nonnegative parameter $c(u, v)$. For every vertex $v \in V$, it is associated with two nonnegative parameters, $f(v)$ and $l(v)$. For any vertex $v \in V$, $f(v) \geq \sum_{u \in C(v)} f(u)$.*

Let $w \in V$ be a vertex in the tree. Let r be a nonnegative integer, where $r \leq |D(w)|$. ($|D(w)|$ is the cardinality of the set $D(w)$.) Let $R = \{A_1, A_2, \cdots, A_r\} \subseteq D(w)$ be a set of r vertices. For $1 \leq i \leq r$, define B_i as the vertex that satisfies the following three requirements: (1) $B_i \in R \bigcup \{w\}$; (2) B_i is an ancestor of A_i; (3) no vertex in R is both an ancestor of A_i and a descendant of B_i. Then we define the objective function $\Delta cost(w : r : R)$ as $\Delta cost(w : r : R) = \sum_{i=1}^{r} \sum_{(u,v) \in PATH(A_i, B_i)} f(A_i) \cdot c(u, v) - \sum_{i=1}^{r} l(A_i)$. If $r = 0$, define $\Delta cost(w : 0 : \emptyset) = 0$. Finding r and R that maximize $\Delta cost(w : r : R)$ is referred to as the 'optimal placement problem corresponding to w'.

Let $w_1 \in V$ and $w_2 \in V$ be two vertices in the tree, where w_1 is an ancestor of w_2. Let s be a nonnegative integer, where $s \leq |D(w_2)| + 1$. ($|D(w_2)|$ is the cardinality of the set $D(w_2)$.) Let $S = \{P_1, P_2, \cdots, P_s\} \subseteq D(w_2) \bigcup \{w_2\}$ be a set of s vertices. For $1 \leq i \leq s$, define Q_i as the vertex that satisfies the following three requirements: (1) $Q_i \in S \bigcup \{w_1\}$; (2) Q_i is an ancestor of P_i; (3) no vertex in S is both an ancestor of P_i and a descendant of Q_i. Then we define the objective function $\delta(w_1 : w_2 : s : S)$ as $\delta(w_1 : w_2 : s : S) = \sum_{i=1}^{s} \sum_{(u,v) \in PATH(P_i, Q_i)} f(P_i) \cdot c(u, v) - \sum_{i=1}^{s} l(P_i)$. If $s = 0$, define $\delta(w_1 : w_2 : 0 : \emptyset) = 0$.
$\square$

If we use v_0 to denote the root of the tree T, then the optimal file placement problem we're studying is simply the '*optimal placement problem corresponding to v_0*'.

Theorem 1 *Let $u_0 \in V$ be a vertex in tree $T = (V, E)$. Say u_0 has n ($n \geq 1$) children—$u_1, u_2, \cdots, u_n$. Suppose for $1 \leq i \leq n$, $s = s_i$ and $S = S_i$ are a solution that maximizes the function $\delta(u_0 : u_i : s : S)$. Then $r = \sum_{i=1}^{n} s_i$ and $R = \bigcup_{i=1}^{n} S_i$ are a solution that maximizes the function*

$\Delta cost(u_0 : r : R)$. *And* $\Delta cost(u_0 : \sum_{i=1}^n s_i : \bigcup_{i=1}^n S_i) = \sum_{i=1}^n \delta(u_0 : u_i : s_i : S_i)$.

Proof: Let r' be a nonnegative integer that is no greater than $|D(u_0)|$. Let $R' \subseteq D(u_0)$ be a set of r' vertices. For any vertex $v \in R'$, define B'_v as the vertex that satisfies the following three requirements: (1) $B'_v \in R' \cup \{u_0\}$; (2) B'_v is an ancestor of v; (3) no vertex in R' is both an ancestor of v and a descendant of B'_v.

For $1 \leq i \leq n$, define $S'_i = R' \cap (D(u_i) \cup \{u_i\})$. (Then obviously $\bigcup_{i=1}^n S'_i = R'$, and $S'_i \cap S'_j = \emptyset$ for any $1 \leq i \neq j \leq n$.) Define $s'_i = |S'_i|$ to be the cardinality of S'_i. (Then obviously $\sum_{i=1}^n s'_i = r'$.) By definition, $\Delta cost(u_0 : r' : R') = \sum_{v \in R'} \sum_{(v_1,v_2) \in PATH(v,B'_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in R'} l(v) = \sum_{i=1}^n \sum_{v \in S'_i} \sum_{(v_1,v_2) \in PATH(v,B'_v)} f(v) \cdot c(v_1, v_2) - \sum_{i=1}^n \sum_{v \in S'_i} l(v) = \sum_{i=1}^n \{\sum_{v \in S'_i} \sum_{(v_1,v_2) \in PATH(v,B'_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in S'_i} l(v)\} = \sum_{i=1}^n \delta(u_0 : u_i : s'_i : S'_i)$

For $1 \leq i \leq n$, the value of $\delta(u_0 : u_i : s'_i : S'_i)$ is maximized when $s'_i = s_i$ and $S'_i = S_i$. Therefore the value of $\Delta cost(u_0 : r' : R')$ is maximized when $r' = \sum_{i=1}^n s_i$ and $R' = \bigcup_{i=1}^n S_i$, and $\Delta cost(u_0 : \sum_{i=1}^n s_i : \bigcup_{i=1}^n S_i) = \sum_{i=1}^n \delta(u_0 : u_i : s_i : S_i)$.
□

Theorem 2 *Let u_{-1} and u_0 be two vertices in tree $T = (V, E)$, where u_{-1} is an ancestor of u_0. Say u_0 has n ($n \geq 1$) children—$u_1, u_2, \cdots, u_n$. Suppose for $1 \leq i \leq n$, $s = s_i$ and $S = S_i$ are a solution that maximizes the function $\delta(u_{-1} : u_i : s : S)$. Suppose $r = r_0$ and $R = R_0$ are a solution that maximizes the function $\Delta cost(u_0 : r : R)$. Then*

(1) if $\sum_{i=1}^n \delta(u_{-1} : u_i : s_i : S_i) \geq \sum_{(u,v) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(u, v) - l(u_0) + \Delta cost(u_0 : r_0 : R_0)$, then $s = \sum_{i=1}^n s_i$ and $S = \bigcup_{i=1}^n S_i$ are a solution that maximizes the function $\delta(u_{-1} : u_0 : s : S)$, and $\delta(u_{-1} : u_0 : \sum_{i=1}^n s_i : \bigcup_{i=1}^n S_i) = \sum_{i=1}^n \delta(u_{-1} : u_i : s_i : S_i)$;

(2) if $\sum_{i=1}^n \delta(u_{-1} : u_i : s_i : S_i) \leq \sum_{(u,v) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(u, v) - l(u_0) + \Delta cost(u_0 : r_0 : R_0)$, then $s = r_0 + 1$ and $S = R_0 \cup \{u_0\}$ are a solution that maximizes the function $\delta(u_{-1} : u_0 : s : S)$, and $\delta(u_{-1} : u_0 : r_0 + 1 : R_0 \cup \{u_0\}) = \sum_{(u,v) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(u, v) - l(u_0) + \Delta cost(u_0 : r_0 : R_0)$.

Proof: Let $s = s'$ and $S = S'$ be a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$ given the condition that $u_0 \notin S$. Let $s = s''$ and $S = S''$ be a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$ given the condition that $u_0 \in S$. Clearly either $s = s'$ and $S = S'$, or $s = s''$ and $S = S''$, is a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$.

For every vertex $v \in S'$, define Q'_v as the vertex that satisfies the following three requirements: (1) $Q'_v \in S' \cup \{u_{-1}\}$; (2) Q'_v is an ancestor of v; (3) no vertex in S' is both an ancestor of v and a descendant of Q'_v. Similarly, we define Q''_v for every vertex $v \in S''$.

For $1 \leq i \leq n$, define $S'_i = S' \cap (D(u_i) \cup \{u_i\})$. (Then obviously $\bigcup_{i=1}^n S'_i = S'$, and $S'_i \cap S'_j = \emptyset$ for any $1 \leq i \neq j \leq n$.) Define $s'_i = |S'_i|$ to be the cardinality of S'_i. (Then obviously $\sum_{i=1}^n s'_i = s'$.)

For $1 \leq i \leq n$, define $S''_i = S'' \cap (D(u_i) \cup \{u_i\})$. (Then obviously $S'' = (\bigcup_{i=1}^n S''_i) \cup \{u_0\}$, and $S''_i \cap S''_j = \emptyset$ for any $1 \leq i \neq j \leq n$.) Define $s''_i = |S''_i|$ to be the cardinality of S''_i. (Then obviously $s'' = 1 + \sum_{i=1}^n s''_i$.)

We analyze the following two cases.

(1) By definition, $\delta(u_{-1} : u_0 : s' : S') = \sum_{v \in S'} \sum_{(v_1,v_2) \in PATH(v,Q'_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in S'} l(v) = \sum_{i=1}^n \sum_{v \in S'_i} \sum_{(v_1,v_2) \in PATH(v,Q'_v)} f(v) \cdot c(v_1, v_2) - \sum_{i=1}^n \sum_{v \in S'_i} l(v) = \sum_{i=1}^n \{\sum_{v \in S'_i} \sum_{(v_1,v_2) \in PATH(v,Q'_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in S'_i} l(v)\} = \sum_{i=1}^n \delta(u_{-1} : u_i : s'_i : S'_i)$.

$s = s'$ and $S = S'$ is a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$ given the condition that $u_0 \notin S$. So for $1 \leq i \leq n$, $s = s'_i$ and $S = S'_i$ is a solution that maximizes the function $\delta(u_{-1} : u_i : s : S)$, just as the solution $s = s_i$ and $S = S_i$ is. So $\delta(u_{-1} : u_0 : s' : S') = \sum_{i=1}^n \delta(u_{-1} : u_i : s_i : S_i)$. By definition, we know $u_0 \notin \bigcup_{i=1}^n S_i$, so $s = \sum_{i=1}^n s_i$ and $S = \bigcup_{i=1}^n S_i$ is also a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$ given the condition that $u_0 \notin S$.

(2) By definition, $\delta(u_{-1} : u_0 : s'' : S'') = \sum_{v \in S''} \sum_{(v_1,v_2) \in PATH(v,Q''_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in S''} l(v) = \sum_{i=1}^n \sum_{v \in S''_i} \sum_{(v_1,v_2) \in PATH(v,Q''_v)} f(v) \cdot c(v_1, v_2) + \sum_{(v_1,v_2) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(v_1, v_2) - \sum_{i=1}^n \sum_{v \in S''_i} l(v) - l(u_0) = \sum_{(v_1,v_2) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(v_1, v_2) - l(u_0) + \{\sum_{i=1}^n \sum_{v \in S''_i} \sum_{(v_1,v_2) \in PATH(v,Q''_v)} f(v) \cdot c(v_1, v_2) - \sum_{i=1}^n \sum_{v \in S''_i} l(v)\} = \sum_{(v_1,v_2) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(v_1, v_2) - l(u_0) + \Delta cost(u_0 : \sum_{i=1}^n s''_i : \bigcup_{i=1}^n S''_i) = \sum_{(v_1,v_2) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(v_1, v_2) - l(u_0) + \Delta cost(u_0 : s'' - 1 : S'' - \{u_0\})$.

$s = s''$ and $S = S''$ is a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$ given the condition that $u_0 \in S$. So $r = s'' - 1$ and $R = S'' - \{u_0\}$ is a solution that maximizes the function $\Delta cost(u_0 : r : R)$, just as the solution $r = r_0$ and $R = R_0$ is. So $\delta(u_{-1} : u_0 : s'' : S'') = \sum_{(v_1,v_2) \in PATH(u_0,u_{-1})} f(u_0) \cdot c(v_1, v_2) - l(u_0) + \Delta cost(u_0 : r_0 : R_0)$. Clearly $s = r_0 + 1$ and $S = R_0 \cup \{u_0\}$ is also a solution which maximizes the function $\delta(u_{-1} : u_0 : s : S)$ given the condition that $u_0 \in S$.

Either $s = \sum_{i=1}^n s_i$ and $S = \bigcup_{i=1}^n S_i$, or $s = r_0 + 1$ and $S = R_0 \cup \{u_0\}$, is a solution that maximizes the function

$\delta(u_{-1} : u_0 : s : S)$. Which of them is the solution that maximizes the function $\delta(u_{-1} : u_0 : s : S)$ depends on whether $\delta(u_{-1} : u_0 : \sum_{i=1}^{n} s_i : \bigcup_{i=1}^{n} S_i)$ is greater or less than $\delta(u_{-1} : u_0 : r_0 + 1 : R_0 \cup \{u_0\})$. Now it's easy to see that Theorem 2 holds.

□

Theorem 1 and 2 show how an optimization problem on placement can be decomposed into subproblems. Based on those two theorems, the *optimal file placement problem* can be solved with a dynamic programming algorithm.

We first define a few notations.

Definition 2 *Given a vertex $w \in V$ of the tree $T = (V, E)$, define r_w^{opt} and R_w^{opt} to be a pair of parameters such that the solution '$r = r_w^{opt}$ and $R = R_w^{opt}$' maximizes the function $\Delta cost(w : r : R)$. And define Δ_w^{opt} as $\Delta_w^{opt} = \Delta cost(w : r_w^{opt} : R_w^{opt})$.*

Given two vertices $w_1 \in V$ and $w_2 \in V$ of the tree $T = (V, E)$, where w_1 is an ancestor of w_2, define s_{w_1,w_2}^{opt} and S_{w_1,w_2}^{opt} to be a pair of parameters such that the solution '$s = s_{w_1,w_2}^{opt}$ and $S = S_{w_1,w_2}^{opt}$' maximizes the function $\delta(w_1 : w_2 : s : S)$. And define δ_{w_1,w_2}^{opt} as $\delta_{w_1,w_2}^{opt} = \delta(w_1 : w_2 : s_{w_1,w_2}^{opt} : S_{w_1,w_2}^{opt})$. □

Now we present the recurrences of the dynamic programming algorithm:

- If a vertex u_0 in tree $T = (V, E)$ has $n \geq 1$ children—u_1, u_2, $\cdots$, u_n—then $r_{u_0}^{opt} = \sum_{i=1}^{n} s_{u_0,u_i}^{opt}$, $R_{u_0}^{opt} = \bigcup_{i=1}^{n} S_{u_0,u_i}^{opt}$, and $\Delta_{u_0}^{opt} = \sum_{i=1}^{n} \delta_{u_0,u_i}^{opt}$.

- If a vertex u_0 in tree $T = (V, E)$ has no child, then $r_{u_0}^{opt} = 0$, $R_{u_0}^{opt} = \emptyset$, and $\Delta_{u_0}^{opt} = 0$.

- For two vertices u_{-1} and u_0 in tree $T = (V, E)$, where u_{-1} is an ancestor of u_0, if u_0 has $n \geq 1$ children—u_1, u_2, $\cdots$, u_n—then $\delta_{u_{-1},u_0}^{opt} = \max\{\sum_{i=1}^{n} \delta_{u_{-1},u_i}^{opt}, \sum_{(u,v)\in PATH(u_0,u_{-1})} f(u_0) \cdot c(u,v) - l(u_0) + \Delta_{u_0}^{opt}\}$. If $\sum_{i=1}^{n} \delta_{u_{-1},u_i}^{opt} \geq \sum_{(u,v)\in PATH(u_0,u_{-1})} f(u_0) \cdot c(u,v) - l(u_0) + \Delta_{u_0}^{opt}$, then $s_{u_{-1},u_0}^{opt} = \sum_{i=1}^{n} s_{u_{-1},u_i}^{opt}$ and $S_{u_{-1},u_0}^{opt} = \bigcup_{i=1}^{n} S_{u_{-1},u_i}^{opt}$; otherwise, $s_{u_{-1},u_0}^{opt} = r_{u_0}^{opt} + 1$ and $S_{u_{-1},u_0}^{opt} = R_{u_0}^{opt} \cup \{u_0\}$.

- For two vertices u_{-1} and u_0 in tree $T = (V, E)$, where u_{-1} is an ancestor of u_0, if u_0 has 0 child, then $\delta_{u_{-1},u_0}^{opt} = \max\{\sum_{(u,v)\in PATH(u_0,u_{-1})} f(u_0) \cdot c(u,v) - l(u_0), 0\}$. If $\sum_{(u,v)\in PATH(u_0,u_{-1})} f(u_0) \cdot c(u,v) - l(u_0) > 0$, then $s_{u_{-1},u_0}^{opt} = 1$ and $S_{u_{-1},u_0}^{opt} = \{u_0\}$; otherwise, $s_{u_{-1},u_0}^{opt} = 0$ and $S_{u_{-1},u_0}^{opt} = \emptyset$.

The first and third recurrences come from Theorem 1 and 2 respectively, and the second and fourth recurrences can be easily seen to be correct. If we use v_0 to denote the root of the tree $T = (V, E)$, then the *optimal file placement problem* is to find $r_{v_0}^{opt}$ and $R_{v_0}^{opt}$, and to cache $r_{v_0}^{opt}$ copies of the file on nodes in the set $R_{v_0}^{opt}$. The dynamic programming algorithm can be shown to have complexity $O(|V|^2)$, where $|V|$ is the number of vertices in tree $T = (V, E)$.

4. Implementation of Optimal Content Placement for En-Route Caching

In this section we show how the optimal file placement can be fulfilled without prefetching (data pushing) for en-route web caching, and introduce the implementation details of the caching scheme.

4.1. Optimal Placement without Prefetching

Theorem 3 *Let $T = (V, E)$ be the tree considered in Definition 1, and let A_0 be its root. Let $r = r_0$ and $R = R_0$ be a solution that maximizes $\Delta cost(A_0, r, R)$, and let $N = \{A_1, A_2, \cdots, A_n\} \subseteq R_0$ be an arbitrary subset of R_0. Decompose T into $n + 1$ subtrees, which we denote by $T_0 = (V_0, E_0)$, $T_1 = (V_1, E_1)$, $\cdots$, $T_n = (V_n, E_n)$, according to the following three rules: (1) $V = \cup_{i=0}^{n} V_i$, and $V_i \cap V_j = \emptyset$ for any $0 \leq i \neq j \leq n$; (2) for $0 \leq i \leq n$, $A_i \in V_i$; (3) for any node $v \in V - \{A_i | 0 \leq i \leq n\}$, if $A_j \in \{A_i | 0 \leq i \leq n\}$ is an ancestor of v and the path between v and A_j doesn't contain any node in the set $\{A_i | 0 \leq i \leq n, i \neq j\}$, then $v \in V_j$.*

For any node $v \in V$, define $U(v)$ as the maximal set that satisfies the following two requirements: (1) $U(v) \subseteq D(v) \cap N$, (here $D(v)$ is the set of all the nodes that are descendants of v in the tree T, as defined in Definition 1); (2) for every node $u \in U(v)$, the path between v and u doesn't contain any node in the set $D(v) \cap N - \{u\}$.

For any node $v \in V$, define $f'(v)$ as $f'(v) = f(v) - \sum_{u\in U(v)} f(u)$. (For the definition of $f(v)$, see Definition 1.)

For any i such that $0 \leq i \leq n$, for any nonnegative integer r' such that $r' \leq |V_i| - 1$, for any set $R' = \{a_1, a_2, \cdots, a_{r'}\}$ such that $R' \subseteq V_i - \{A_i\}$, define the objective function $\Delta' cost(A_i : r' : R')$ as $\Delta' cost(A_i : r' : R') = \sum_{j=1}^{r'} \sum_{(u,v)\in PATH(a_j,b_j)} f'(a_j) \cdot c(u,v) - \sum_{j=1}^{r'} l(a_j)$, where b_j ($1 \leq j \leq r'$) is defined as the node that satisfies the following three requirements: (1) $b_j \in R' \bigcup \{A_i\}$; (2) b_j is an ancestor of a_j; (3) no vertex in R' is both an ancestor of a_j and a descendant of b_j. If $r' = 0$, define $\Delta' cost(A_i : 0 : \emptyset) = 0$.

For $0 \leq i \leq n$, let $r' = r_i'$ and $R' = R_i'$ be a solution that maximizes the value of $\Delta' cost(A_i : r' : R')$. Then $r = n + \sum_{i=0}^{n} r_i'$ and $R = N \cup (\bigcup_{i=0}^{n} R_i')$ is a solution that maximizes the value of $\Delta cost(A_0, r, R)$, namely, $\Delta cost(A_0, n + \sum_{i=0}^{n} r_i', N \cup (\bigcup_{i=0}^{n} R_i')) = \Delta cost(A_0, r_0, R_0)$.

Proof: Consider T as the tree network where A_0 is the only node originally caching the file F. The maximum net cost saving of caching F on T can be got by first caching F on nodes in N, and then caching F on nodes in $R_0 - N$. When F is cached on nodes in N, the rate of requests for F passing through any node v, which was originally $f(v)$, becomes $f'(v)$, and T can be partitioned into $n + 1$ subtrees each of which containing exactly one copy of F on its node closest to A_0. It can be seen that the net cost saving got by caching additional copies of F on any of the $n + 1$ subtrees is independent of any of the other n subtrees. So this theorem can be seen to hold.

□

Theorem 3 shows that the optimal placement of a file on a tree can be fulfilled without prefetching (data pushing) for en-route caching. When a cache containing a file F receives a request for F issued by a node v, the cache computes the locations to cache F that is optimal for the whole tree. However, F only needs to be cached on nodes that not only belong to the optimal locations but also belong to the path between the cache and v, which is fulfilled when F is transmitted from the cache to v. Such a caching process can keep going on in the tree, and eventually when no request for F will cause F to be cached on any additional node, by Theorem 3 the placement of F on the tree is optimal for the whole tree. So the file F never needs to be cached outside the path it necessarily passes through, which is called *prefetching* or *data pushing*. Prefetching is many times considered overly-active or unnecessary for caching; and having a caching scheme which doesn't have to use prefetching is certainly desirable. However, it's simple to see that the optimal placement here can also be achieved while using prefetching.

4.2. Implementation

The caching scheme is implemented as follows. Each cache v containing a file F maintains information about the tree rooted at v corresponding to the file F. The information consists of the tree topology and the parameters $f(v')$, $l(v')$ for every node v' and $c(u, u')$ for every edge (u, u') in the tree. Every time a request for F is sent to the cache v, the nodes that the request passes through piggybacks to the request-packet the information about the path that the request traverses; and v maintains the recent information. Upon receiving the request, v computes the optimal locations in the tree, and caches F on the path leading to the node that issued the request, when F is transmitted to that node. The information nodes piggyback to the request and the information used for computing optimal locations by v can be estimated from data for the file F as well as for files of sizes similar to that of F. For a network containing hundreds of thousands of web files (or more), such estimated information is usually well updated. The implementation is very similar to that of the *Coordinated En-Route Web Caching scheme* [10], with the major difference that in [10], the maintained information for a file is evenly divided and stored by all nodes in the corresponding tree, while here all the information is stored by the root of the tree. However, the amount of data stored on different nodes can be well evened by the large number of files and nodes in the network. The information piggybacked to requests is similar to that in [10]. So it's easy to show that the extra storage and transmission overheads of this scheme is at the same level of those of the *Coordinated En-Route Web Caching scheme*.

5. Simulation

The emphasis of our simulation is to compare the *relative performance* of the optimal file placement scheme with existing en-route caching schemes. Instead of simulating over a network containing a large number of web files, we simulate for a single file and over a sub-network which is the tree rooted at the server permanently maintaining the file. Being consistent with the Tiers model [3], the network consists of a WAN (wide area network) in the middle and a number of MANs (metropolitan area networks) attached to it. The WAN is seen as a backbone network where no server or client (source of requests) is attached. An en-route cache is attached to every WAN and MAN node. The single server containing the file in consideration is chosen randomly from the MAN nodes. An eviction cost $l(v)$ is associated with every node v, and its value changes from time to time. To simulate the removal of the file from a node it has been cached on, every time if the cost loss of removing the file from a cache is smaller than the cache's eviction cost, the file will be evicted from the cache.

We simulate three caching schemes: the scheme using the optimal file placement on trees (but each time the file is only cached along a path) — which we shall call the *tree scheme*, the Coordinated En-Route Web Caching scheme [10] which optimizes the file placement on paths — which we shall call the *path scheme*, and the scheme which caches the file in every cache the file passes through — which we shall call the *node scheme*. Extensive experiments have been performed for a large number of randomly generated tree networks (of reasonable characteristics) and wide ranges of parameters. It turns out that the relative performance of the three schemes is quite similar for different network topologies and parameters. Therefore we only show the results of two experiments as examples. Let $U(x, y)$ denote the uniform distribution between x and y. Then the parameters of the two experiments are as shown in Table 1.

Parameter	Experiment 1	Experiment 2
Total number of nodes	200	300
Ratio of WAN nodes to MAN nodes	1:1	1:1
Delay of WAN links	$U(0.41, 0.51)$ second	$U(0.41, 0.51)$ second
Delay of MAN links	$U(0.06, 0.08)$ second	$U(0.06, 0.08)$ second
Eviction cost	$U(1, 1.2)$	$U(1, 1.2)$

Table 1: Parameters of Two Experiments

In both experiments, we increase the request rate of MAN nodes, and observe how the average access latency changes when the request rate increases (which means the file becomes more and more popular). The performances of the three schemes are shown in Fig. 3. It can be seen that both the '*Tree scheme*' and the '*Path scheme*' perform much better than the '*Node scheme*', while the performance difference between the '*Tree scheme*' and the '*Path scheme*' is comparatively smaller. The figures imply that the '*Path*' scheme is a big improvement on the '*Node*' scheme, and the '*Tree*' scheme further improves the performance by optimizing the file's placement even better.

6. Conclusions

In this paper we show that existing file placement policies for en-route caching are all solving restricted partial problem of the original file placement problem, thus give only sub-optimal solutions. A low-complexity dynamic-programming algorithm which outputs the optimal solution is presented. It's shown that the optimal placement of web files can be implemented without prefetching. And both analysis and simulations show that the optimal file placement solution perform better than other existing file placement policies for en-route caching.

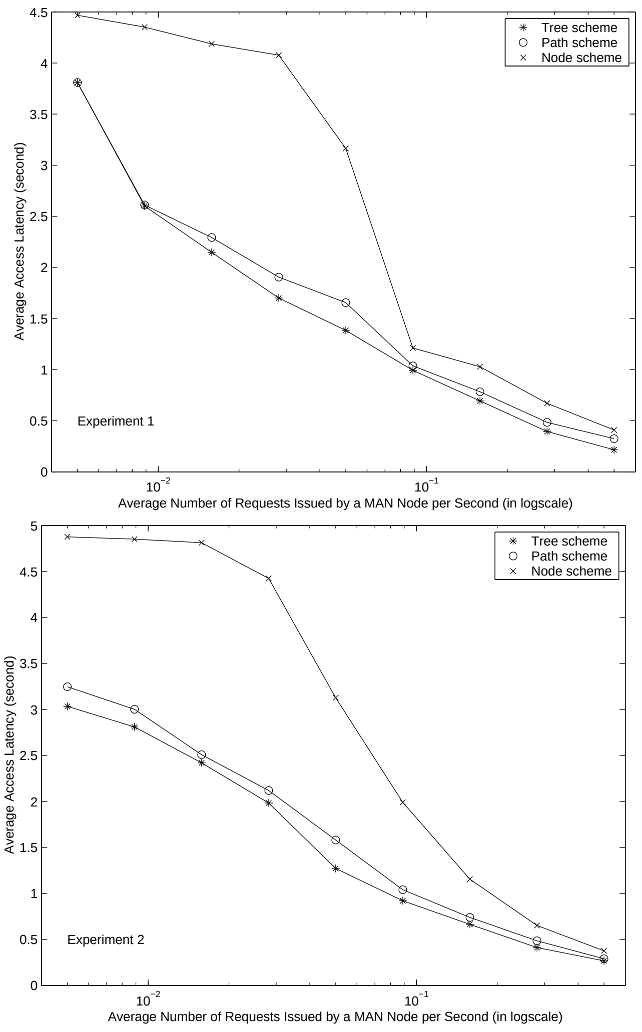

Figure 3. Average access latency vs. average request rate

References

[1] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura. Self-organizing wide-area network caches. In *Proceedings of IEEE INFOCOM'98*, pages 600–608, 1998.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *Proc. IEEE INFOCOM'99*, pages 126–134, 1999.

[3] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Comm. Magazine*, 35(6):160–163, 1997.

[4] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *Proceedings of the ACM SIGCOMM*, pages 239–243, 1993.

[5] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, 2000.

[6] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Trans. Networking*, 8(2):158–170, 2000.

[7] P. Rodriguez and S. Sibal. Spread: scalable platform for reliable and efficient automated distribution. *Computer Networks*, 33(1-6):33–49, 2000.

[8] P. Rodriguez, C. Spanner, and E. W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9(4):404–418, 2001.

[9] P. Scheuermann, J. Shim, and R. Vingralek. A case for delay-conscious caching of web documents. *Computer Networks and ISDN Systems*, 29(8-13):997–1005, 1997.

[10] X. Tang and S. T. Chanson. Coordinated en-route web caching. *IEEE Trans. Computers*, 51(6):595–607, 2002.

[11] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world wide web documents. In *Proceedings of ACM SIGCOMM'96*, pages 293–305, 1996.