Optimal Content Placement For En-Route Web Caching[1]

Anxiao (Andrew) Jiang and Jehoshua Bruck

California Institute of Technology

Parallel and Distributed Systems Lab

MC 136-93

Pasadena, CA 91125

Phone: +1 (626) 395-8504

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

I. Introduction

Web caching is one of the main techniques solving the performance problems the World Wide Web faces today. WWW has been experiencing exponential growth in recent years [21], 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 [4] [9] [11] [13] [16] [19] [22].

Effective caching requires cooperative content management of web caches. Traditional caches include clients, proxies and servers. One common approach to coordinate caches is *Hierarchical Caching* [7] [16], where a cache hierarchy is set up and caches are located at different levels of the network, such as the client level, the institutional level, the regional level and the national level. When a request from a client is not satisfied by a cache, the cache redirects the request to a higher-level cache. A request always travels upwards in the cache hierarchy until it's satisfied by some cache or, if none of the caches it goes through has the requested file, by the web server. When the file is found and transmitted downward to the client, the file is usually cached in every cache in the hierarchy along the path. Another common approach to coordinate caches is *Distributed Caching* [12] [20], where only institutional caches are placed at the edge of the network which cooperate among themselves. In distributed caching, caches need to be aware of each other's content, which is realized through queries, exchanging content digests/summaries, or using hash functions that map files to caches. Some hybrid caching architectures also exist [16].

A new caching architecture, called *En-Route Caching* [3] [9] [15] [19], 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 [19]. 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 [8].

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] [14] [18] [22] [23]. Most policies make decisions on file placement and replacement for individual caches only. Some policies, such as MODULO [3], 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. [19]. 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 [19].

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 policies, including the coordinated en-route web caching sheme, 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 II, 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 III, 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 IV, implementation details are introduced. In Section V, simulation results showing the performance difference between the optimal scheme and other existing schemes are provided. In Section VI, we conclude this paper.

II. Modelling En-Route Web Caching

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

A. Caching Model and File Placement

The model we use in this paper closely follows the network model in [19]. 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,

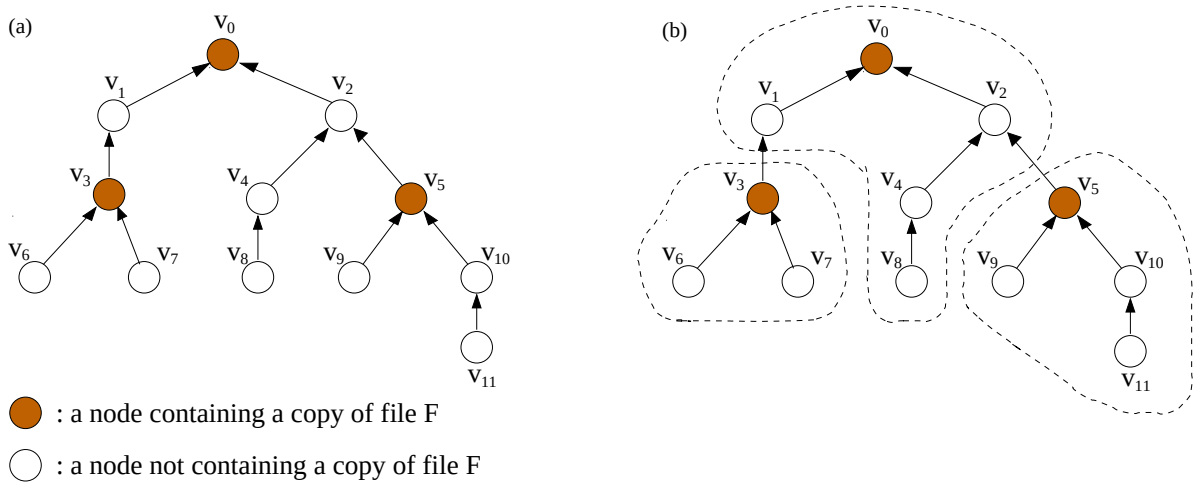

Fig. 1. (a) En-Route Web Caching (b) Subtrees Corresponding to Cached File Copies

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 [19], 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 [9] [10] [16] [19].

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 [19], 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 $f(A_i)$ $(1 \leq i \leq n)$ 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 [17]. 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

$$\sum_{i=1}^{r} l(A_{j_i}) \tag{1}$$

The above cost-loss formula is used in [19], 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 Formula (1) is a good approximation for the 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.

B. 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 [1], Hybrid [23], LNC-R [17], GD-Size [6], 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 [1] or other admission control mechanisms [2]. Some file placement policies cache files in a more coordinated way. An example is *MODULO caching* [3], 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 [19] 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.

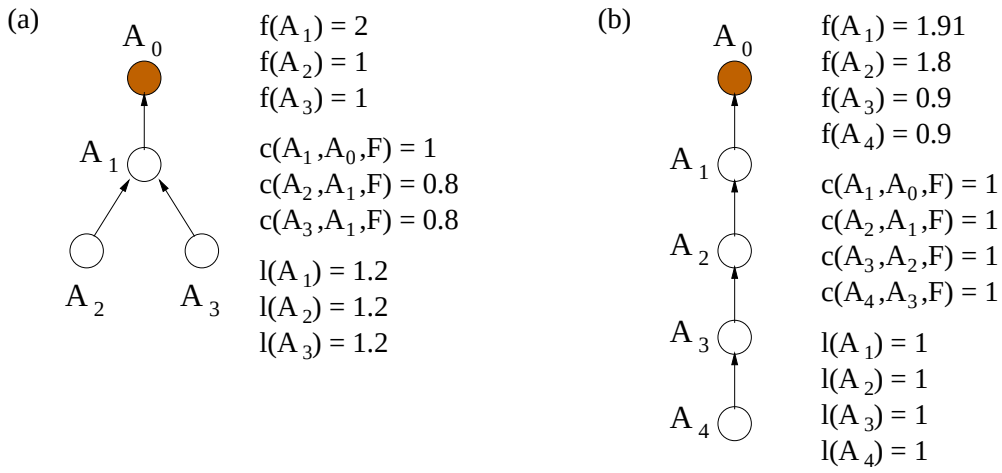

Fig. 2. File Placement on Trees

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 [19], and the *optimal placement* as defined in the previous subsection.

(a) A tree of 4 nodes, $\{A_0, A_1, \cdots, A_3\}$, is shown in Fig. 2(a). Here A_0 is the only node that contains a file F. The values of the parameters $f(A_i)$, $c(A_i, P_{A_i}, F)$ and $l(A_i)$ ($1 \leq i \leq 3$) are shown in Fig. 2(a), where $f(A_i)$ is the rate of requests for F passing through node A_i, $c(A_i, P_{A_i}, F)$ is the cost of transmitting a request for F and the corresponding response through the edge (A_i, P_{A_i}) (here P_{A_i} is the parent of A_i, e.g., $P_{A_1} = A_0$ and $P_{A_2} = P_{A_3} = A_1$), and $l(A_i)$ is the cost loss of removing enough files from A_i to make room for F. The requests for F issued by A_0 won't cause caching F on other nodes, and A_1 doesn't issue any request for F because $f(A_1) = f(A_2) + f(A_3)$. Assume A_2 issues a request for F first, and A_3 issues a request for F some time later.

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 $\sum_{i=2}^{3} \sum_{(u_1,u_2)\in PATH(A_i,A_0)} f(A_i) \cdot c(u_1,u_2,F) - \sum_{i=2}^{3} l(A_i) = 1.2$.

It's simple to see that actually no matter in which order A_2 and A_3 send requests for F, the scheme which 'caches a file on every node it passes through' and the 'Coordinated En-Route Web Caching scheme' will have the same placements as above once A_2 and A_3 has sent at least one request each, and their *net* cost saving will be 0 and 0.8 respectively, both worst than the optimal placement.

(b) A tree of 5 nodes, $\{A_0, A_1, \cdots, A_4\}$, is shown in Fig. 2(b). Here A_0 is the only node that contains a file F. Note that here the tree is also a path. The values of the parameters $f(A_i)$, $c(A_i, P_{A_i}, F)$ and $l(A_i)$ ($1 \leq i \leq 4$) are shown in Fig. 2(b). Assume the first request and the second request for F are issued by A_1 and A_4 respectively.

When the policy 'caching a file on every node the file passes through' is used, after both requests of A_1 and A_4 are satisfied, clearly F will be cached on all nodes in the tree, and the *net* cost saving is $\sum_{i=1}^{4} f(A_i) \cdot c(A_i, A_{i-1}, F) - \sum_{i=1}^{4} l(A_i) = 1.51$.

When the *Coordinated En-Route Web Caching scheme* is used, the request of A_1 will cause caching F on A_1 (which is the placement on the path between A_0 and A_1 that maximizes the *net* cost saving), and the request of A_4 will cause caching F on A_4 (which is the placement on the path between A_1 and A_4 that maximizes the *net* cost saving). It's simple to verify that afterwards no request for F issued by nodes in the tree will cause caching any extra copy of F in the tree. So the total *net* cost saving stabilizes to be $f(A_1) \cdot c(A_1, A_0, F) + \sum_{(u_1,u_2)\in PATH(A_4,A_1)} f(A_4) \cdot c(u_1,u_2,F) - l(A_1) - l(A_4) = 2.61$.

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_4, whose corresponding *net* cost saving is $\sum_{(u_1,u_2)\in PATH(A_2,A_0)} f(A_2) \cdot c(u_1,u_2,F) + \sum_{(u_1,u_2)\in PATH(A_4,A_2)} f(A_4) \cdot c(u_1,u_2,F) - l(A_2) - l(A_4) = 3.4$.

So the first two schemes both output placements worse than the optimal placement. And it can be seen that the *Coordinated En-Route Web Caching scheme*, which optimizes file placements on paths, can also output non-optimal solutions even if the tree is a path, if the first request doesn't come from the bottom node. (Clearly if A_4 issues the first request, then the *Coordinated En-Route Web Caching scheme* will output an optimal placement of F.)

(c) The two placements output by the '*Coordinated En-Route Web Caching scheme*' in (a) and (b) are both stable, in the sense that no further request will cause caching any extra copy of F in the tree, and that removing any already cached copy of F will cause a negative 'net cost saving'. It can be easily shown that the *Coordinated En-Route Web Caching scheme* can also output unstable (transient) file placements (e.g., consider the case in (a) where $f(A_1)$, $f(A_2)$ and $f(A_3)$ are changed to be 2.5, 0.5 and 2 respectively). It's also very easy to show that the scheme of 'caching a file on every node the file passes through' can also output both stable and unstable file placements. Experiments show that both stable and unstable placements

are common outputs of the above two schemes; and when being suboptimal, both kinds of placements fail to maximize the net cost saving for the tree.

□

Further analysis shows that in the worst case the following relative performance will be achieved for every file placement policy mentioned in this paper (excluding the optimal placement): 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.

III. 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 $\delta(w_1 : w_2 : 0 : \emptyset) = 0$.

□

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,

$$\begin{aligned}
& \Delta cost(u_0 : r' : R') \\
= & \sum_{v \in R'} \sum_{(v_1, v_2) \in PATH(v, B'_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in R'} l(v) \\
= & \sum_{i=1}^{n} \sum_{v \in S'_i} \sum_{(v_1, v_2) \in PATH(v, B'_v)} f(v) \cdot c(v_1, v_2) - \sum_{i=1}^{n} \sum_{v \in S'_i} l(v) \\
= & \sum_{i=1}^{n} \{ \sum_{v \in S'_i} \sum_{(v_1, v_2) \in PATH(v, B'_v)} f(v) \cdot c(v_1, v_2) - \sum_{v \in S'_i} l(v) \} \\
= & \sum_{i=1}^{n} \delta(u_0 : u_i : s'_i : S'_i)
\end{aligned}$$

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,

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

$$= \sum_{i=1}^{n}\{\sum_{v\in S_i'}\sum_{(v_1,v_2)\in PATH(v,Q_v')} f(v)\cdot c(v_1,v_2) - \sum_{v\in S_i'} l(v)\}$$
$$= \sum_{i=1}^{n}\delta(u_{-1}:u_i:s_i':S_i')$$

$s = s'$ 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 $1 \le i \le n$, $s = s_i'$ 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,

$$\begin{aligned}
&\delta(u_{-1} : u_0 : s'' : S'')\\
=\ &\sum_{v\in S''}\sum_{(v_1,v_2)\in PATH(v,Q_v'')} f(v)\cdot c(v_1,v_2) - \sum_{v\in S''} l(v)\\
=\ &\sum_{i=1}^{n}\sum_{v\in S_i''}\sum_{(v_1,v_2)\in PATH(v,Q_v'')} f(v)\cdot c(v_1,v_2) + \sum_{(v_1,v_2)\in PATH(u_0,u_{-1})} f(u_0)\cdot c(v_1,v_2) - \sum_{i=1}^{n}\sum_{v\in S_i''} l(v) - l(u_0)\\
=\ &\sum_{(v_1,v_2)\in PATH(u_0,u_{-1})} f(u_0)\cdot c(v_1,v_2) - l(u_0) + \{\sum_{i=1}^{n}\sum_{v\in S_i''}\sum_{(v_1,v_2)\in PATH(v,Q_v'')} f(v)\cdot c(v_1,v_2) - \sum_{i=1}^{n}\sum_{v\in S_i'} l(v)\}\\
=\ &\sum_{(v_1,v_2)\in PATH(u_0,u_{-1})} f(u_0)\cdot c(v_1,v_2) - l(u_0) + \Delta cost(u_0 : \sum_{i=1}^{n} s_i'' : \bigcup_{i=1}^{n} S_i'')\\
=\ &\sum_{(v_1,v_2)\in PATH(u_0,u_{-1})} f(u_0)\cdot c(v_1,v_2) - l(u_0) + \Delta cost(u_0 : s'' - 1 : S'' - \{u_0\})
\end{aligned}$$

$s = s''$ 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 recurrence come from Theorem 1 and 2 respectively, and the second and fourth recurrence 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 $T = (V, E)$.

IV. 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.

A. 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 $f'(v) = f(v) - \bigcup_{u \in U(v)} f(u)$. (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 caching the file F which satisfies the requests for file F from all the nodes in T. Then $\Delta cost(A_0, r_0, R_0)$ is the *net* cost saving got by caching F on nodes in the set R_0. The *net* cost saving got by caching F on nodes in the set N is $\Delta cost(A_0, n, N)$. Define Δ_{add}^{max} as the maximum net cost saving that we can *additionally* get by caching F on more (including

zero) nodes in T when F has already been cached on the $n+1$ nodes in the set $\{A_0\} \cup N = \{A_i | 0 \leq i \leq n\}$. Since $r = r_0$ and $R = R_0$ is a solution that maximizes $\Delta cost(A_0, r, R)$ and $N \subseteq R_0$, clearly $\Delta cost(A_0, r_0, R_0) = \Delta cost(A_0, n, N) + \Delta_{add}^{max}$.

Define $\Delta_{S|N}$ as the *additional* net cost saving we can get by caching F on nodes in the set $S \subseteq V - \{A_i | 0 \leq i \leq n\}$ when F has already been cached on the $n+1$ nodes in the set $\{A_i | 0 \leq i \leq n\}$. Partition S into $n+1$ subsets S_0, S_1, $\cdots$, S_n, where $S_i = S \cap V_i$ for $0 \leq i \leq n$. It's simple to see that when F is cached on the $n+1$ nodes in the set $\{A_i | 0 \leq i \leq n\}$, for any node $v \in V$, its request for F will be satisfied by A_i ($0 \leq i \leq n$) if and only if $v \in V_i$, because when $v \in V_i$, A_i is the nearest ancestor of v that caches F or A_i is the same node as v. Therefore the net cost saving got by caching F on nodes in $S_i \subset V_i$ in the subtree T_i is independent of the net cost saving got by caching F on nodes in $S_j \subset V_j$ in the subtree T_j for any $0 \leq i \neq j \leq n$. When F is cached on the $n+1$ nodes in the set $\{A_i | 0 \leq i \leq n\}$, clearly for any node $v \in V$, the rate of requests for F passing through node v (including the requests from v itself and from others) equals $f'(v)$ — so for $0 \leq i \leq n$, the additional net cost saving got by caching F on nodes in $S_i \subset V_i$ equals $\Delta' cost(A_i, |S_i|, S_i)$. So $\Delta_{S|N} = \sum_{i=0}^{n} \Delta' cost(A_i, |S_i|, S_i)$. For $0 \leq i \leq n$, the value of $\Delta' cost(A_i, |S_i|, S_i)$ achieves its maximum when $|S_i| = r_i'$ and $S_i = R_i'$. So the value of $\Delta_{S|N}$ achieves its maximum, which is Δ_{add}^{max}, when $|S| = \sum_{i=0}^{n} r_i'$ and $S = \bigcup_{i=0}^{n} R_i'$. The conclusion of Theorem 3 naturally follows.

□

With Theorem 3 it can be shown that the optimal placement of a file on a tree can be fulfilled without prefetching (data pushing) for en-route caching. Let the tree in consideration be the tree T in Theorem 3. When a node v in the tree issues a request for F, the request will reach the root A_0, and A_0 will carry out computation to find out the optimal locations to cache F, which is the set of nodes R_0. However, the system only needs to cache F on the path between A_0 and v. To be specific, we can make the set N, as defined in Theorem 3, to be the set of nodes not only in R_0 but also on the path between A_0 and v, and cache F only in the nodes in N. After that, the rate of requests for F passing through any node u in the tree becomes $f'(u)$. And then, when a node v' issues a request for F, the request will reach one of the $n+1$ nodes in the set $\{A_0\} \cup N$ — say A_i ($0 \leq i \leq n$) — and A_i will compute the optimal locations to cache F in the subtree T_i. Then again F only needs to be cached on the path between A_i and v'. This process can keep going on, 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 T is not only optimal for all the subtrees encountered, but also optimal for the whole tree T itself. So the file F only needs to be cached on the nodes it necessarily passes through, which is done when F is transmitted from the node answering the corresponding request for F to the node that issued the request, and doesn't need to be cached on any node outside the path (which is called *prefetching*

or *data pushing*). The aggregation of the placements on different paths forms the optimal placement for the whole tree. 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.

B. Implementation

The caching scheme is implemented as follows. Each cache v containing a file F maintains information on the topology of the tree which consists of the set of nodes whose requests for F are satisfied by v and the corresponding set of links (edges), the request rate $f(v')$ for F passing through every node v' in the tree and its corresponding eviction cost $l(v')$, and the cost $c(u, u')$ of every edge (u, u') in the tree. Every time a request for F is sent from a node v' to a node v which contains F, each node on the path between v and v' piggybacks its ID, the cost for F of the edge between itself and its parent — which is estimated based on the delays of transmitting files of sizes similar to that of F from its parent to itself recently — and its eviction cost for F. v' also piggybacks a time stamp recording the time when the request was issued. The cache v builds the tree topology based on the piggybacked information, builds a table which records the time when the most recent few requests for F from each node were issued, and records the nodes' eviction costs and the costs of edges. Then when v computes the optimal placement of F on the tree, it can estimate the request rate of each node based on the request history using a sliding window technique [18] [19], estimate the eviction cost for F of each node based on the eviction costs for a few typical file sizes of that node recorded recently, and estimate the cost of every edge for F based on the costs of the edge for a few typical file sizes recorded recently. For a network containing hundreds of thousands of web files (or more), the information collected by v is quite recent and its estimation is usually well updated. Then v sends the file F to v' along with a field indicating which nodes on the path should cache F, and F is cached as indicated.

The information a cache v maintains for a file F is the information on the whole corresponding tree. So on average each node needs to store only tens of bytes of information related to F. Also some of the information, such as network topology, can be shared for different files. The amount of data the nodes store for maintaining the information related to files are evened by the large number of files and nodes in the network. The amount of data piggybacked to requests can be reduced by hashing the nodes' IP addresses into short IDs. In the *Coordinated En-Route Web Caching scheme* [19], the types of data transmitted and stored for computing file placements on paths are quite similar to those in this scheme. 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*.

V. 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. This simplified model serves as a valuable first step toward the simulation over the original networks with many files, provides us with a useful tool for studying and comparing the different behaviors of different file placement policies, which is important for understanding the placement algorithms at a deeper level, and reveals ample results on the relative performance of different file placement policies by itself. Extensive simulation experiments have been done over a large set of tree topologies and wide parameter ranges.

The tree network is randomly generated using a node-degree probability distribution function (p.d.f.), which specifies the probability distribution of each node's degree (all nodes use the same p.d.f.). Starting with a single node, the tree grows by determining the degree of each existing node until the tree reaches the desired size. Being consistent with the Tiers model [5], 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 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. Requests for the file in consideration are generated only by MAN nodes and may be transmitted through both MAN and WAN nodes. An eviction cost $l(v)$ is associated with every node v, and its value changes from time to time, each time randomly generated independent of other nodes and its own historical values. 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. That is validated by the fact that the eviction cost of a node is largely determined by the characteristics and placements of the huge number of files other than the one file in consideration, and thus is only remotely correlated to the file in consideration; and once the cost loss of removing the file from a certain cache becomes so insignificant that it's smaller that the cache's eviction cost, requests for other files passing through the cache would have caused the file to be removed.

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), the *Coordinated En-Route Web Caching scheme* [19] which optimizes the file placement on paths, and the scheme which caches the file in every cache the file passes through. Experiments have been performed extensively for a large number of tree network topologies and wide ranges of the parameters (e.g., network size, percentage of WAN nodes and link costs). 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 four experiments as examples.

Let $U(x, y)$ denote the uniform distribution between x and y. Let $f(n)$ denote the probability that a node has degree n ($n = 1, 2, 3, \cdots$). Then the parameters of the four experiments are as shown in Table 1. Note that the four experiments are run independently. Therefore the four networks in the four experiments have totally different topologies, and their specific parameter assignments are also independent of each other.

Parameter	Experiment 1 and 3	Experiment 2 and 4
Total number of nodes	200	300
p.d.f. of node-degree	$f(1) = 0.1, f(2) = 0.4,$ $f(3) = 0.25, f(4) = 0.25$	$f(1) = 0.1, f(2) = 0.4,$ $f(3) = 0.25, f(4) = 0.125,$ $f(5) = 0.125$
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	$U(1, 1.2)$	$U(1, 1.2)$
Window size for request-rate estimation	5	5

Table 1: Parameters of Four Experiments

In all the four 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). Each experiment has 9 stages, while in the i-th stage ($1 \leq i \leq 9$) the average number of requests issued by every MAN node per second is a random number with distribution $U(0.001 \cdot 10^{(i-1)/4}, 0.009 \cdot 10^{(i-1)/4})$. (In each experiment, the network topology, the set of WAN nodes and the delay of each link remain the same at the 9 different stages.) The performances of the three schemes in experiment 1 and 2 are shown in Fig. 3(a) and (b). To get a better view of the relatively performance of the file placement scheme we propose and the Coordinated En-Route Web Caching scheme, we show the performances of only those two schemes in experiment 3 and 4 in Fig. 3(c) and (d). In those figures, '*Tree*', '*Path*' and '*Node*' respectively mean the file placement scheme we propose, the Coordinated En-Route Web Caching scheme and the scheme which caches the file in every cache the file passes through, because those three schemes optimize the file placement on a tree, a path and single nodes respectively.

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. In all the four experiments, compared to the '*Path*' scheme, the '*Tree*' scheme saves the average access latency by 6 to 33

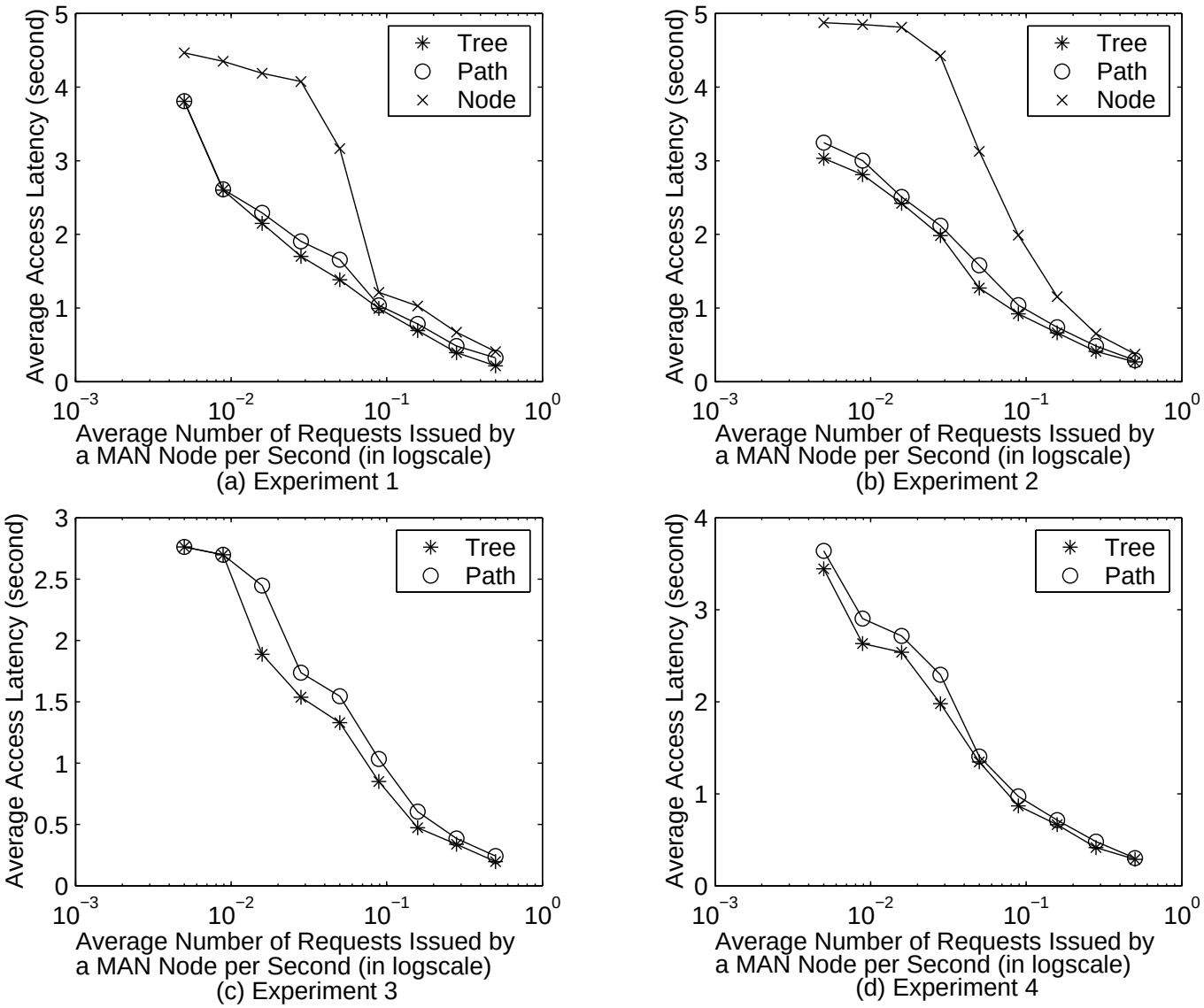

Fig. 3. Average access latency vs. average request rate

percent in most of the stages, which can be regarded as substantial.

VI. 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 can perform substantially better than other existing file placement policies for en-route caching.

REFERENCES

[1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams and E. A. Fox, “Caching proxies: limitations and potentials,” in Proceedings of the 4th International WWW Conference, Boston, MA, Dec. 1995.

[2] C. Aggarwal and J. L. Wolf and P. S. Yu, “Caching on the world wide web,” IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pp. 94–107, 1999.

[3] S. Bhattacharjee, K. L. Calvert and E. W. Zegura, “Self-organizing wide-area network caches,” in Proceedings of IEEE INFOCOM’98, pp. 600–608, 1998.

[4] 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, pp. 126–134, 1999.

[5] K. L. Calvert and M. B. Doar and E. W. Zegura, “Modeling Internet Topology,” IEEE Comm. Magazine, vol. 35, no. 6, pp. 160–163, 1997.

[6] P. Cao and S. Irani, “Cost-aware WWW proxy caching algorithms,” in Proc. USENIX Symp. Internet Technology and Systems, pp. 193–206, 1997.

[7] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz and K. J. Worrell, “A hierarchical Internet object cache,” in Proc. 1996 USENIX Ann. Technical Conf., pp. 153–163, 1996.

[8] P. B. Danzig, R. S. Hall and M. F. Schwartz, “A case for caching file objects inside internetworks,” in Proceedings of the ACM SIGCOMM, pp. 239–243, 1993.

[9] P. Krishnan, D. Raz and Y. Shavitt, “The cache location problem,” IEEE/ACM Transactions on Networking, vol. 8, no. 5, pp. 568–582, 2000.

[10] B. Li, M. J. Golin, G. F. Italiano, X. Deng and K. Sohraby, “On the optimal placement of web proxies in the Internet,” in Proceedings of IEEE INFOCOM’99, pp. 1282–1290, 1999.

[11] V. N. Padmanabhan and L. Qiu, “The content and access dynamics of a busy web site: findings and implications,” in Proc. ACM SIGCOMM’00, pp. 111–123, 2000.

[12] D. Povey and J. Harrison, “A distributed Internet cache,” in Proc. 20th Australian Computer Science Conf., Sydney, Australia, Feb. 1997.

[13] M. Rabinovich and H. Wang, “Dhttp: an efficient and cache-friendly transfer protocol for web traffic,” in Proc. IEEE INFOCOM’01, pp. 1597–1606, 2001.

[14] L. Rizzo and L. Vicisano, “Replacement policies for a proxy cache,” IEEE/ACM Trans. Networking, vol. 8, no. 2, pp. 158–170, 2000.

[15] P. Rodriguez and S. Sibal, “Spread: scalable platform for reliable and efficient automated distribution,” Computer Networks, vol. 33, no. 1-6, pp. 33–49, 2000.

[16] P. Rodriguez, C. Spanner and E. W. Biersack, “Analysis of web caching architectures: hierarchical and distributed caching,” IEEE/ACM Transactions on Networking, vol. 9, no. 4, pp. 404–418, 2001.

[17] 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, pp. 997–1005, 1997.

[18] J. Shim, P. Scheuermann and R. Vingralek, “Proxy cache design: algorithms, implementation, and performance,” IEEE Trans. Knowledge and Data Eng., vol. 11, no. 4, pp. 549–562, 1999.

[19] X. Tang and S. T. Chanson, “Coordinated en-route web caching,” IEEE Trans. Computers, vol. 51, no. 6, pp. 595–607, 2002.

[20] R. Tewari, M. Dahlin, H. M. Vin and J. S. Kay, “Design considerations for distributed caching on the Internet,” in Proc. 19th IEEE Int’l Conf. Distributed Computing Syetems, pp. 273–284, 1999.

[21] J. Wang, “A survey of web caching schemes for the Internet,” ACM SIGCOMM Computer Comm. Rev., vol. 29, no. 5, pp. 36-46, 1999.

[22] 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, pp. 293–305, 1996.

[23] R. P. Wooster and M. Abrams, “Proxy caching that estimates page load delays,” Computer Networks and ISDN Systems, vol. 29, no. 8-13, pp. 977–986, 1997.