An $O(nh)$ Algorithm for Dual-Server Coordinated En-Route Caching in Tree Networks

Shihong Xu[1] and Hong Shen[2]

[1] School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa 923-1292 Japan
{s0520203}@jaist.ac.jp

[2] School of Computer Science
The University of Adelaide
SA 5005, Australia

Abstract

Dual-server coordinated en-route caching is important because of its basic features as multi-server en-route caching. In this paper, multi-server coordinated en-route caching is formulated as an optimization problem of minimizing total access cost, including transmission cost for all access demands and caching cost of all caches. We first discuss an algorithm for single-server en-route caching in tree networks and then show that this is a special case of another algorithm for dual-server en-route caching in tree networks whose time complexity is $O(nh)$.

1. Introduction

A dual-server network is a special type of multi-server network [21] where there are two servers in operation and a user can access files from any server they have access rights to. Dual-server networks can increase efficiency of transmission, reduce latency and provide redundancy.

Dual-server en-route caching is important because of its basic features as multi-server en-route caching. However, traditional optimization methods for en-route caching are mainly designed for networks with a single server and can't be directly applied to networks with more than one server. In this paper, we first formulate multi-server coordinated en-route caching as an optimization problem of minimizing total access cost, including transmission cost and caching cost, and then propose an algorithm for dual-server coordinated en-route caching in tree networks. As the result, we integrated two technologies, dual-server caching and en-route caching, to provide a better caching solution for network environments.

The rest of the paper is organized as follows. In section 2, related work on caching is discussed. In section 3, a formulation of the problem is proposed. Dynamic programming based solutions are presented in section 4. Section 5 is the conclusion.

2. Related Work

As an important technology to enhance content delivery and alleviate server load [5], [4], caching has attracted much attention. Significant effort has been made on the topic of optimizing cache performance [2], [22], [18], [3], cooperation among several caches [11], [15], [8], [7], and cache hierarchies [20], [17]. Recent studies have focused on the benefits of cooperative caching for distributed systems and large-scale systems [2], [12], [20]. In [23], web caching problem in a hierarchy of collaborating proxy servers is studied and a collaboration protocol is proposed to reduce duplicate caching between a proxy and its parent or higher level proxies in the hierarchy when making a caching decision. In [9], the authors examined three practical cooperative placement algorithms for large-scale distributed caches and showed that cooperative object placement could significantly improve performance compared to local replacement algorithms, particularly when the sizes of individual caches are small compared to those of the objects.

Recent advances on web caching [6] have enabled the development of a new caching architecture, called en-route web caching [1], [16], [10], [13]. The problem studied in [19] considered the coordinated en-route caching problem for linear topology, deciding the optimal locations for placing copies of an object among the en-route caches. This

Proceedings of the Seventh International Conference on
Parallel and Distributed Computing,Applications and Technologies (PDCAT'06)
0-7695-2736-1/06 $20.00 © 2006 IEEE

scheme, which optimizes the placement of objects on the path from the client to the server, has been shown to perform significantly better than other schemes that either object placement or replacement in individual caches only are considered. Paper [14] studied optimal methods for unconstrained and constrained en-route web caching in tree networks, which extended the scope of optimization to a tree rooted at the server and got better performance compared with [19]. At the same time, [14] solved constrained en-route web caching problem, including the constraints of caching *exactly K-copies* and *at most K-copies*.

The multi-server cache location problem in linear arrays has already been studied in [10]. However, our work focuses on dual-server coordinated en-route caching in tree networks where object placement and replacement polices are carried out in an coordinated fashion. We formulate multi-server coordinated en-route caching as an optimization problem of minimizing total transmission cost for all demands and caching cost of all caches. Then we present two algorithms, one for single-server en-route caching in tree networks and the other for dual-server en-route caching in tree networks.

3. Problem Formulation

First of all, we present a formulation of multi-server en-route caching in general networks which adopts the method in [10] to express the transmission cost of access demands.

We consider a network comprising n clients and m servers, where a client can issue requests to each of the servers. At each client there is a cache supporting coordinated en-route caching [19]. If an object *O* is cached at a client, requests for the object arriving at the client can be satisfied by the cache; otherwise, the requests are forwarded until they are satisfied by other caches or the server. The sets of clients and servers are denoted respectively by $C = \{c_i, 1 \leq i \leq n\}$ and $S = \{s_i, 1 \leq i \leq m\}$. Because cache space is limited (we assume it is always fully occupied), we need to remove some objects already cached before caching a new object at a client and this leads to miss penalty when these removed objects are requested later. We denote miss penalty of these objects by $p_c, c \in C$ and call it *caching cost* of object *O* at client c.

A demand is a request for object *O* from a client to a server and we denote by $f_{c,s}$ the frequency of a demand from c to s. We use $d(c,s), c \in C, s \in S$ to denote the sum of distances between c and s and call $d(c,s) \cdot f_{c,s}$ *transmission cost* of a demand from c to s.

We note that the formulation of en-route caching problem in [19] and [14] are described in a form of caching cost and caching gain at individual nodes. However, we formulate the problem in a form of total cost of the entire network, including transmission cost on all links and caching cost at all caches. Suppose A is the set of clients that cache the target object and $v \in A$ is the nearest node that caches the object on the path from c to s, then the transmission cost of a demand from c to s is the product of the frequency and the distance from the client to node v, which can be denoted by

$$\min_{v \in path[c,s] \cap (A \cup S)} d(c,v) \cdot f_{c,s}, \quad (1)$$

where $path[c,s]$ is the set of nodes on the path between c and s. The total access cost in the network is the sum of transmission cost of all demands and caching cost at all caches defined as (2). Our objective is to place copies in a subset $A, A \subseteq C$ so that the total access cost(2) is minimized.

$$\sum_{c \in C, s \in S} \min_{v \in path[c,s] \cap (A \cup S)} d(c,v) \cdot f_{c,s} + \sum_{c \in A} p_c \quad (2)$$

We define the problem of multi-server coordinated en-route caching as follows:

A newtwork can be represented by an undirected graph (V,E), where $V = \{c_1, c_2, ...c_n, s_1, s_2, ...s_m\}$ is the set of nodes comprising n clients and and m servers. Frequencies for all demands are denoted by set $F = \{f_{c,s}\} : C \times S \rightarrow R$, caching cost at all clients are denoted by $P = \{p_c\} : C \rightarrow R$. Our objective is to place copies in a subset $A \subseteq C$, so that the total access cost is minimized, which can be formulated as follows

$$G(F, A^*) = \min_A G(F, A) =$$
$$\min_A \{ \sum_{c \in C, s \in S} \min_{v \in path[c,s] \cap (A \cup S)} d(c,v) \cdot f_{c,s} + \sum_{v \in A} p_v \}, \quad (3)$$

where A is a feasible solution and A^* is an optimal solution.

If only one server is contained in the network, the model is for single-server problem which has already been solved by [19] and [14] for linear array and tree networks respectively. What we want to solve is the dual-server problem, that is, two servers are contained in the network. However, we can't get a solution to dual-server problem by simply composing the solutions of single-server problems one by one. This is because the solutions of single-server problems may contradict with each other. It's possible that optimal locations to place the copies in dual-server context may differ from the optimal locations in single-server context.

4. Dynamic Programming Solutions

Dual-server coordinated en-route caching is important because it's the basis of multi-server coordinated en-route caching. However, in the paper we first discuss two algorithms, one for dual-server en-route caching in linear array and the other for single-server en-route caching in tree networks before we concentrate on dual-server en-route caching in tree networks.

4.1. Solution to Single-Server En-Route Caching in Tree Networks

Before focusing on the dual-server problem in tree networks we solve the single-server problem first. This problem has been solved by [14] with a algorithm whose time complexity is $O(n^2)$ for an n-node tree. In this paper we present another algorithm whose time complexity is $O(nh)$, where h is the height of the tree.

We use a bottom-up dynamic programming approach enlightened by [10] to get a solution of a tree from the solutions of it's children. Let T_w denote the subtree rooted at node w, $C(w)$ denotes children set of node w, $G(w,l)$ means the total access cost to satisfy the requests issued by the clients in T_w optimally, where l is the distance from node w to the nearest cache on the path to the server. The optimal solution corresponding to $G(w,l)$ is denoted by $A(w,l)$.

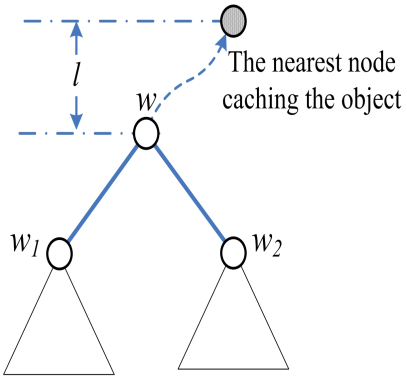

Figure 1. definition of G(w, l)

When we consider the relation between $G(w,l)$ for node w and that for it's children, we have the following theorem.

Theorem 1 *For a tree T_r, we have $G(F,A^*) = \sum_{w_i \in C(r)} G(w_i,1)$, where A^* is an optimal solution to Equation (2). For inner node w and distance l, we have*

$$G(w,l) = \min\{G_{without}(w,l), G_{caching}(w,l)\}, \quad (4)$$

$$A(w,l) = \begin{cases} \cup_{w_i \in C(w)} A(w_i, l+1) \\ (G_{without}(w,l) \le G_{caching}(w,l)) \\ \cup_{w_i \in C(w)} A(w_i,1) \cup \{w\} \\ (G_{without}(w,l) > G_{caching}(w,l)). \end{cases} \quad (5)$$

where $G_{without}(w,l) = \sum_{w_i \in C(w)} G(w_i, l+1) + l \cdot f_w$ is the minimum access cost in the case that the object is not cached at node w and $G_{caching}(w,l) = \sum_{w_i \in C(w)} G(w_i,1) + p_w$ is the minimum access cost in the case that the object is cached at node w.

Proof.

First, all subtrees of the root node are independent of each other and the distance from the root node to it's children is 1, so it's easy to find out that $G(F,A^*) = \sum_{w_i \in C(r)} G(w_i,1)$.

Then for inner node w:

(1) If $w \notin A^*$, because $G_{without}(w,l)$ is an optimal solution, we have $G_{without}(w,l) \le \sum_{w_i \in C(w)} G(w_i, l+1) + l \cdot f_w$. On the other hand $G(w_i, l+1), w_i \in C(w)$ are also optimal solutions, so we have $\sum_{w_i \in C(w)} G(w_i, l+1) \le G_{without}(w,l) - l \cdot f_w$. From two inequations, we have $G_{without}(w,l) = \sum_{w_i \in C(w)} G(w_i, l+1) + l \cdot f_w$and $A(w,l) = \cup_{w_i \in C(w)} A(w_i, l+1)$.

(2) If $w \in A^*$, in a similar way we have $G_{caching}(w,l) = \sum_{w_i \in C(w)} G(w_i,1) + p_w$ and $A(w,l) = \cup_{w_i \in C(w)} A(w_i,1) \cup \{w\}$.

From (1)(2), we know Theorem 1 is correct.

Based on Theorem 1, we have dynamic programming algorithm as follows which runs in bottom-up order.

Algorithm 1: $EWC1$
Step 1. Initialization:
$A(w,l) = \phi$;
$G(w,l) = l \cdot f_w$, for all w and l, $1 \le l \le h$;
Step 2. Running the algorithm in post-order traversal
for all non-leaf w, $1 \le l \le h_w$ **do**
if $\sum_{w_i \in C(w)} G(w_i, l+1) + l \cdot f_w \le \sum_{w_i \in C(w)} G(w_i,1) + p_w$ **then**
$G(w,l) = \sum_{w_i \in C(w)} G(w_i, l+1) + l \cdot f_w$
$A(w,l) = \cup_{wi \in C(r)} A(w_i, l+1)$
else
$G(w,l) = \sum_{w_i \in C(w)} G(w_i,1) + p_w$
$A(w,l) = \cup_{wi \in C(r)} A(w_i,1) \cup \{w\}$
endif
endfor.

Complexity Time

What we should compute is all $G(w,l), w \in C, 0 \le l \le h_w$ and time complexity of the algorithm is $O(nh)$. In the worst case $h = n$ and time complexity is $O(n^2)$ which is same as the method in [14]. In the best case $h = 1$, where each node is a leaf node except the root node r, time complexity is $O(n)$.

4.2. Solution to Dual-Server En-Route Caching in Tree Networks

The method used in previous section can be extended easily to solve multi-server case in $O(nh^m)$ time complexity through computing all combination of $G(w, l_1, l_2, ...l_m)$, where m is the number of servers. However, in this section,

Proceedings of the Seventh International Conference on
Parallel and Distributed Computing,Applications and Technologies (PDCAT'06)
0-7695-2736-1/06 $20.00 © 2006 IEEE

we only focus on dual-server network and present an algorithm whose time complexity is $O(nh)$. If only one server is contained in the network, the algorithm equals to $EWC2$; if the tree degenerates to a line, the algorithm is equivalent to $EWC1$.

Now, let us think about the location of a server in a tree network. It's certain that if a server is located in a tree, it must be in a leaf node or root node. (If there is a server located at an inner node, the tree can be decomposed so that the inner node becomes a leaf node or the root node.) For such a tree with two servers contained, we adjust the shape of tree as follows so that some property can be achieved.

(1) We adjust the location of one server so that the node becomes the root of the tree. We call the node *root-server node*.

(2) For the other server, if it's not the most left node, we swap the node with his brother (the most left one). We swap the upper nodes (parent, grandfather ...) of the server in the same way until the server becomes the left-most node in the tree. We call the server node *leaf-server node*.

In process(1), the height of the tree changes. Suppose h' is the height of the tree before adjusting and h after adjusting, then we have $0 < h \leq 2h'$. In both process (1) and (2), the connectivity never changes and the network adjusted is an isomorphic structure to the original network. We can get the original network from the latter in the reversed process of (1) and (2). We call the tree adjusted as a *dual-server tree*.

Now, we consider the solution of dual-server en-route caching in a weighted tree network from listing the difference between the single server case and two-server case as follows:

(1) In the dual-server case, the root server has only one child (otherwise the network can be further decomposed) and the leaf server is located at the most left node in tree.

(2) In the dual-server case, a client can issue demands to two servers.

Here, we denote *root server* by s and the *leaf server* by s'. f_w is the frequency of demand from node w to server s, f'_w is the frequency of demand from node w to server s'. $G(w, l)$ means the optimal access cost to satisfy the requests issued by the clients in T_w optimally, where l is the distance from node w to the nearest cache on the path to *root server* s and the corresponding optimal solution is $A(w, l)$. Let $L(w, l)$ denote the distance from w to the nearest client caching the object in the path to *leaf server* s' as Figure() shows.

Based on the analysis above, we have the following theorem.

Theorem 2 *For a dual-server tree T_r defined before, in the case that the object is cached at node w, we have $L(w, l) = 0$; otherwise in the case that the object is not cached at node w, we have*

$$L(w,l) = \begin{cases} L(w', l+1) + 1 \\ (w \in path[s, s']) \\ L(w', l - D(w) + 1) + 1 + D(w) \\ (w \notin path[s, s'] \ \ and \ \ l > D(w)) \\ l (w \notin path[s, s'] \ \ and \ \ l \leq D(w)) \end{cases} \quad (6)$$

, where $D(w)$ is the distance between w and the nearest node w^* in $path[s, s']$($w^* = w$ when $w \in path[s, s']$). w' is the child of w^* in $path[s, s']$.

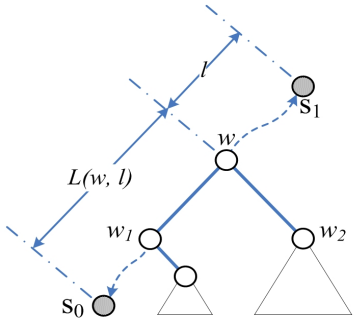

Figure 2. Definition of L(w, l)

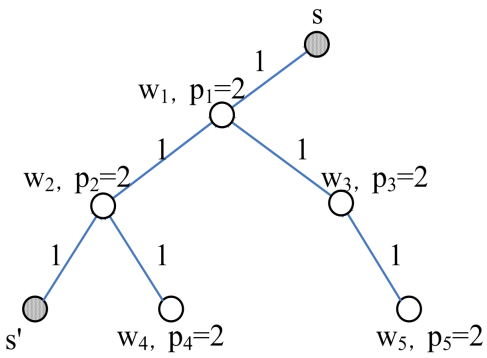

Figure 3. An Example

Proof. If node w is on the path$[s, s']$, the nearest client caching the object from node w to leaf server s must be on the path$[s, s']$ too; otherwise it can be on path$[s, w^*]$ or path$[w^*, w]$, where $w^* = path[s, s'] \cap path[w^*, w]$ is the first node from w to s.

(1) if node w is on path$[s, s']$ and w' is the child of w in path$[s, s']$, we have $L(w, l) = L(w', l + 1) + 1$.

(2) if node w is not on path$[s, s']$ and the nearest client caching the object is in the path$[s, w^*]$, then $L(w, l) =$

Proceedings of the Seventh International Conference on
Parallel and Distributed Computing,Applications and Technologies (PDCAT'06)
0-7695-2736-1/06 $20.00 © 2006 IEEE

$L(w^*, l - D(w)) + D(w)$. Because w^* is in path$[s, s']$, we have $L(w^*, l - D(w)) = L(w', l - D(w) + 1) + 1$ according to (1), where w' is the child of w^* in path$[s, s']$. At last, we have $L(w, l) = L(w', l - D(w) + 1) + 1 + D(w)$.

(3) if node w is not on the path$[s, s']$ and the nearest client caching the object is on path$[w^*, w]$, it's easy to see that $L(w, l) = l$

Now that we have already known the distances from w to s and s', we can give a theorem for the optimal solution as follows.

Theorem 3 *For a dual-server tree T_r defined before, we have $G(F, A^*) = G(w', 1)$, where A^* is an optimal solution to Equation (2), w' is the child of root node r. For node w and distance l, we have*

$$G(w, l) = \min\{G_{without}(w, l), G_{caching}(w, l)\} \quad (7)$$

$$A(w, l) = \begin{cases} \cup_{w_i \in C(w)} A(w_i, l + 1) \\ (G_{without}(w, l) \leq G_{caching}(w, l)) \\ \cup_{w_i \in C(w)} A(w_i, 1) \cup \{w\} \\ (G_{without}(w, l) > G_{caching}(w, l)) \end{cases} \quad (8)$$

where $G_{without}(w, l) = \sum_{w_i \in C(w)} G(w_i, l + 1) + f_w \cdot l + f'_w \cdot L(w, l)$ is the optimal access cost in the case that the object is not cached at node w and $G_{caching}(w, l) = \sum_{w_i \in C(w)} G(w_i, 1) + p_w$ is the optimal access cost in the case that the object is cached at node w.

Proof. Similar to Theorem(1) and is left out because of space limitations.

Based on Theorem 3, we present a dynamic programming algorithm which runs in post-order(left-right-root) traversal.

Algorithm 2: $EWC2$

Step 1. Initialization:

Adjust the tree so that s' is the most left node, s is the root node and the height of tree is h. $A(w, l) = \phi$;

$G(s', l) = 0$, for each distance l, $1 \leq l \leq h$;

$L(s', l) = 0$, for each distance l, $1 \leq l \leq h$;

Step 2. Running the algorithm in post-order traversal

for all node w and distance l, $1 \leq l \leq h_w$ **do**

if $w \in path[s, s']$ **then**

$L(w, l) = L(w', l + 1) + 1$

else if $w \notin path[s, s']$ **and** $l > D(w)$ **then**

$L(w, l) = L(w', l - D(w) + 1) + 1 + D(w)$

else if $w \notin path(s, s')$ **and** $l \leq D(w)$ **then**

$L(w, l) = l$

endif;

$M = \sum_{w_i \in C(w)} G(w_i, l + 1) + L(w, l) f'_w + l f_w$;

$N = \sum_{w_i \in C(w)} G(w_i, 1) + p_w$;

if $M \leq N$ **then**

$G(w, l) = M$

$A(w, l) = \cup_{w_i \in C(w)} A(w_i, l + 1)$

else

$L(w, l) = 0$

$G(w, l) = N$

$A(w, l) = \cup_{w_i \in C(w)} A(w_i, 1)\{w\}$

endif;

enddo.

Time Complexity

Computing $G(w, l)$, where $w \in C, 1 \leq l \leq h$, results in the time complexity of the algorithm to be $O(nh)$. In the case that only one server is contained in the tree, we suppose $L(w, l) = f'_w = 0$ then this algorithm equals to EWC2. In the case that the tree degenerates to a linear array, we suppose $l = i$ and $L(w, l) = r - l - i$ then this algorithm becomes EWC1.

Now, we give an example for dual-server coordinated en-route caching to show the process how algorithm EWC3 works. Consider a tree network consisting of five client $w_i, 1 \leq i \leq 5$ and two servers s_0, s_{n+1} as Figure **??** shows. All demands from clients to servers are $\{f'_1 = f'_2 = f'_3 = f'_4 = f'_5 = f'_6 = 1, f_1 = f_2 = f_3 = f_4 = f_5 = f_6 = 2\}$, caching cost at each client $p_1 = p_2 = p_3 = p_4 = p_5 = p_6 = 4$. Running the algorithm EWC3, we can get an optimal solution $G(F, A^*) = 17, A^* = \{w_2, c_3\}$. The process is as follows:

Table 1. D'(w, l)

l	D'(5,l)	D'(4,l)	D'(3,l)	D'(2,l)	D'(1,l)
0	0	0	0	0	0
1	1	1	1	1	1/2
2	2	2	2/3	1	
3	3/4	2			

Table 2. G(w, l)

l	G(5,l)	G(4,l)	G(3,l)	G(2,l)	G(1,l)
0	4	4	7*	7*	18
1	3*	3*	9	9	17*
2	6	6	15/17	13	
3	9/10	8			

5. Conclusions

Deploying multiple servers and en-route caching is an important technology to improve the scalability of networks. However, previous work on en-route caching is mostly devoted in single-server networking environment and can't be applied to networks with more than one server.

In this paper we formulate multi-server coordinated en-roue caching as an optimization problem of minimizing total access cost and present several algorithms for different cases. Our algorithm for dual-server en-route caching in tree networks can also be used in dual-server linear array and single-server tree networks.

Our method can be easily extended to the multi-servers case, though the time complexity will then rise to exponential. However, we have reduced the complexity for the dual-server case after some local optimization. It's a challenging task for us to optimize the multi-server case and reduce it's complexity to polynomial. Another challenging task for our future research is to present an approximate method to solve the problem with multiple servers efficiently and the problem in arbitrary topologies. The techniques of applying dynamic programming shown in the paper can serve as useful tools for deriving such solutions in the general case.

References

[1] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura. Self-organizing wide-area network caches. In *INFOCOM (2)*, pages 600–608, 1998.

[2] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, 1996.

[3] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *SIGCOMM*, pages 241–253, 1998.

[4] M. Dahlin, R. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Operating Systems Design and Implementation*, pages 267–280, 1994.

[5] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *SIGCOMM*, pages 239–248, 1993.

[6] B. D. Davison. A web caching primer. *IEEE Internet Computing*, 5(4):38–45, July/August 2001.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[8] S. Gadde, M. Rabinovich, and J. S. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *Workshop on Hot Topics in Operating Systems*, pages 93–98, 1997.

[9] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical coorperative caching. In *Proc. 10th Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 586–595, 1999.

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

[11] P. Krishnan and B. Sugla. Utility of co-operating Web proxy caches. *Computer Networks and ISDN Systems*, 30(1–7):195–203, 1998.

[12] A. Leff, J. L. Wolf, and P. S. Yu. Replication algorithms in a remote caching architecture. *IEEE Trans. Parallel Distrib. Syst.*, 4(11):1185–1204, 1993.

[13] B. Li, X. Deng, M. J. Golin, and K. Sohraby. On the optimal placement of web proxies in the internet: The linear topology. In *HPN '98: Proceedings of the IFIP TC-6 Eigth International Conference on High Performance Networking*, pages 485–495, Deventer, The Netherlands, The Netherlands, 1998. Kluwer, B.V.

[14] K. Li, H. Shen, F. Y. L. Chin, and S. Q. Zheng. Optimal methods for coordinated enroute web caching for tree networks. *ACM Trans. Inter. Tech.*, 5(3):480–507, 2005.

[15] R. Malpani, J. Lorch, and D. Berger. Making world wide web caching servers cooperate. *Proceedings of the Fourth International WWW Conference*, pages 107–117, 1995.

[16] P. Rodriguez and S. Sibal. SPREAD: Scalable platform for reliable and efficient automated distribution. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):33–49, 2000.

[17] P. Rodriguez, C. Spanner, and E. Biersack. Analysis of web caching architectures: Hierarchical and distributed caching, 2001.

[18] J. Shim, P. Scheuermann, and R. Vingralek. Proxy cache algorithms: Design, implementation, and performance. *Knowledge and Data Engineering*, 11(4):549–562, 1999.

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

[20] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Design considerations for distributed caching on the internet. In *International Conference on Distributed Computing Systems*, pages 273–284, 1999.

[21] S. Venkataraman, J. Naughton, and M. Livny. Remote load-sensitive caching for multi-server database systems. In *14th International Conference on Data Engineering (ICDE'98)*, 1998.

[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 *Procedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, 1996.

[23] P. S. Yu and E. A. MacNair. Performance study of a collaborative method for hierarchical caching in proxy servers. *Computer Networks and ISDN Systems*, Vol. 30:215–224, 1998.

Proceedings of the Seventh International Conference on
Parallel and Distributed Computing,Applications and Technologies (PDCAT'06)
0-7695-2736-1/06 $20.00 © 2006 IEEE