Optimal, Seamless Self-Assembly of Files in Linear Networks

Andreas Constantinides Ed Coffman

Department of Electrical Engineering
Columbia University
New York, NY 10027
Email: {andreas,egc}@ee.columbia.edu

April 17, 2006

Abstract

A file F is sited at the root of a tree network of nodes, each able to cache part or all of F. Each node may also have a client wanting to download F. Links between neighbors in the tree have given transmission delays associated with them, but there is no such delay from a node to its client, if it has one. The seamless, self-assembly paradigm requires that requesting nodes start receiving F immediately, receive it continuously until F is fully downloaded, and be implemented by means of a self-assembly protocol which is unaware of network structure beyond links to immediate neighbors. In the CLIENT problem, the nodes where clients are located are specified as part of the problem instance. For CLIENT, we show how to assign segments of F to the node caches in such a way that seamless self-assembly is realized and, simultaneously, the total cache size achieves a lower bound determined solely by the network link delays.

The lower bound also applies to solutions of the much harder ANY-CLIENT problem, in which client locations are *not* part of the problem instance; seamless self assembly of F must still occur wherever clients appear. However, the lower bound can not be achieved in general by solutions to ANY-CLIENT. By including a flooding component in the self-assembly paradigm, we prove that the lower bound can be achieved for the important special case of paths/chains. The paper concludes with a discussion of the many open problems arising in systems requiring seamless, or nearly seamless, self assembly of files.

1 Introduction

A new paradigm in information transmission was introduced in [5] for networks of cooperative cache servers[1]. Each file in the network has an origin server which stores the entire file, and all nodes of the network are potential *client nodes* for that file, i.e., nodes having clients wanting that file. A file segment of any size can be cached at any of the nodes. The downloading of a file to a client is *seamless* in that it incurs no delay, i.e., it begins when the request is made and continues without interruption until it is complete. Thus, seamless downloading can be viewed as a way to implement a maximum quality of service (QoS) guarantee to all clients of a network. We study this problem for a single file F, under a distributed, *self-assembly* constraint in which there is no global control and nodes are unaware of network structure beyond their immediate neighbors. File size and transmission times are discretized: F consists of $M \geq 1$ equal-size *units*; link transmission rates are all the same, and, for convenience, are taken to be one file unit per unit of time. Link transmission delays can vary, but are restricted to positive integers; they label the edges of the network. Seamless transmission of F to a client at the origin server is automatic, so for convenience, but without loss of generality, we consider only problems in which the origin server is not a client node.

This paper provides algorithms/protocols for seamless self-assembly of files at minimum achievable storage cost, i.e., with a minimum total number of file units cached in the nodes of the network. To realize this "perfect" retrieval performance, i.e., stream-like downloading with no initial delay, clients must be prepared to re-sequence segments of a file received out of order. Thus, if the file is a video, for example, the requesting node must experience an initial delay during which some portion of the video is buffered before streaming in the usual sequential sense can begin.

Seamless self-assembly is based on given, typically very simple, finite-state communication protocols at the nodes of the network. In the distributed setting the input to these protocols is confined to information available locally, and that available from immediate (single-hop) neighbors. The self-assembly process is started by requests issued by the clients wanting the file, which in turn initiates a downloading of the file segments cached at the corresponding client nodes; thereafter, the network autonomically schedules file-unit transmissions so that the remainder of the requested file assembles seamlessly at the client nodes. With self-assembly protocols given, an instance of the CLIENT problem is an edge-labeled network, a file size M, the identity of the origin-server node, and the subset of client nodes. The problem is to assign subsets of the file $F = \{1, \ldots, M\}$ to the node caches so that seamless downloading is achieved (F is fully assembled at time M at each client) at minimal storage cost, i.e., the total cache occupancy has the minimum achievable size. Removing the subset of clients from the above instance gives an instance of the ANY-CLIENT problem. In this case, the cache assignments must be such that, no matter which nodes have clients, seamless self assembly is achieved at each, and solutions must have minimal cost.

This paper is restricted to the problem for networks no more general than trees, where downloading is in the root-to-leaf direction, the root is the origin server for F

[1] An earlier version of this work was presented at the 2^{nd} *Multidisciplinary International Conference on Scheduling: Theory and Applications*, New York, 2005.

and all client requests are made simultaneously at time 0. Downloading a unit from any node to its downstream neighbors can be done simultaneously in parallel. Initially, to keep our focus on core issues, we make the *small-network* assumption: The problem instance is such that, in a minimal cost solution, all clients are less than $M/2$ time units away from the root: This condition guarantees that at least one unit will be downloaded to every client from the root node. By the self-assembly protocols presented in Sections 2 and 3, this assumption implies that the total extra cache content required for minimal-cost solutions along any path will be less than the file size. At the ends of Sections 2 and 3, we remove the small-network assumption and modify our result to apply to the more general case.

In the remainder of the paper, δ_v denotes the transmission delay between non-root node v and its parent; $\delta_r \equiv 0$ for the root r. Let V_* be the set of nodes with clients, or with descendants that have clients. We show in Section 2 that $\sum_{v \in V_*} 2\delta_v$ is a lower bound to the total cache size needed for seamless self assembly by either CLIENT or ANY-CLIENT. It is the value of this bound that defines the term *minimal cost*. This lower bound is achieved by the self assembly protocol and cache assignment algorithm for the CLIENT problem given in Section 2. However, this bound can not be achieved in general for ANY-CLIENT under the self assembly protocol of CLIENT. Section 3 introduces a flooding component to the protocol and shows that minimal-cost self assembly of F at all client nodes is achieved for the special case of paths/chains. In general, cache assignments that minimize total cache size for ANY-CLIENT on trees under either of the protocols of Sections 2 and 3 require more than the minimal cost of the lower bound. This more general problem is beyond the scope of this paper, which is confined to problem instances with minimal cost solutions.

Motivation and Related Work: The caching concept has been widely applied, both to computer systems [15, 11] and to the Internet [16, 12, 9], as caches are among the simplest and most effective ways to reduce access times. One of the challenges that web caching currently faces is accommodating the increase in the average size of web pages created primarily by the addition of multimedia files. The number of requests for these larger objects can be expected to increase as more and more users switch to high-speed internet access. Unfortunately, the replacement policies of most currently implemented caching schemes, such as Least-Recently-Used (LRU), are oblivious to file sizes. This leads to huge files in the cache, and consequently, a decrease in the hit ratio of the cache. To alleviate this problem, the LRU-threshold algorithm [1] was introduced. By this algorithm, only files up to a specific size can be stored in the cache. The main limitation of this algorithm is that large files have to be fetched from the origin servers each time they are requested. This increases the average download time of requested files, usually referred to as average access latency.

Recently, the concept of partial caching [14] was introduced to overcome some of the disadvantages of LRU and LRU-threshold. The main idea of partial caching is to store an initial part of the file in the cache and to obtain the rest of the file from the origin server. In this way, the cache hit ratio increases and the average file access latency decreases. The paradigm of seamless downloading grew out of the partial caching concept. Another method to improve the performance of isolated caches is cache co-operation, (see [12], Chapter 9, for a summary of such schemes). Co-operative caching schemes effectively 'increase' the size of an isolated cache [17, 13], and hence they also decrease the access latency.

The problem of how to distribute content efficiently on a network in order to optimize different performance measures has been widely studied. In [7, 6, 8], the authors deal with the question of how to best place a fixed number of caches (web proxies) among potential sites in the network in order to optimize different parameters. In [4], the authors calculate the number of caches needed in order to minimize the overall cost of content distribution. In all of these papers, it is assumed that the requested content is replicated at all the caches. In our work, full replication is not needed, but all nodes are available as caches for content distribution.

The network under consideration in this paper is a set of co-operative caches (in our case they are arranged in a hierarchical structure [3], but they could just as well be arranged in a distributed structure [10]), and the goal is to have seamless downloading at all client nodes with minimal storage cost. Note that, without the use of partial caching, we would have to store the whole file at each client node in order to achieve the minimum achievable QoS guarantees mentioned earlier. However, with the use of partial caching, we are able to achieve seamless downloading by storing just a small fraction of the file at each node.

In the large literature dealing with models of network congestion, file downloading times become service times within a stochastic service (e.g., queueing) system. These studies bring out a strong motivation for seamless self assembly. For, as is well known, expected waiting times and numbers of waiting requests normally grow with the variance of service times. Thus, the zero-variance, constant downloading times of seamless transmission of a given file can be expected to reduce classical measures of congestion in the systems being modeled.

Finally, we mention the intriguing parallels between our self-assembly process and those in biocomputation at nanoscale, where files are the structures/crystals created by distributed growth processes, the node structures play the role of templates, and the file units are the elementary particles/molecules of the growth processes. The initial distribution of cache contents can be thought of as the stoichiometry of the reactants in, say, DNA-based computing. See Adleman et. al. [2] for an introduction to combinatorial self-assembly problems in molecular science; the paper also contains many references to related research in this field.

2 The ANY-CLIENT Problem

Now suppose we remove the subset of clients from the problem instance, but require that, no matter which nodes are chosen as clients, a minimal cost solution be obtained. Under the distributed self-assembly protocol for the CLIENT problem, such solutions can not be found in general. If nodes have some information about structure (and hence violate the self assembly paradigms of interest here), e.g., each node knows its distance from the root, then seamlessness can be achieved at minimal cost. However, we leave this more elaborate subject to a future paper currently in preparation.

Here, we introduce a *flooding* mechanism which allows us to retain the on-demand self-assembly property, with the only substantial change being in the initial condition:

The process not only begins with client nodes sending requests upstream, but also sending their cache units to all downstream neighbors, irrespective of whether they have downstream clients. Also, when a node v receives a file unit (necessarily from upstream), it immediately forwards it to all of its downstream neighbors, if any. Thus, the flooding self assembly protocol is as follows.

1. At time 0, every client node v sends a request signal to its upstream neighbor $p(v)$ and begins supplying its own client. In addition, v begins flooding its cached file units to all of its downstream neighbors, if any.

2. When a non-root node v first receives requests (necessarily from downstream neighbors), it forwards a request immediately to $p(v)$; and if it has not received file units from upstream yet, it begins the flooding of its cache to all of its downstream neighbors.

3. When a node v receives a file unit (necessarily from upstream), it immediately forwards it to all of its downstream neighbors, if any, and also supplies it to its own client, if it has one.

4. Forwarding at a node v takes precedence over downloading file units from v's cache.

We show below that, if the tree is simply a path, then a minimal cost solution to ANY-CLIENT exists under the flooding paradigm, i.e., seamless reception sequences achieving the lower bound on storage cost assemble file units at each client node. Note that, according to the distributed protocol, a *non-client* node starts participating in the transmission of the file as soon as it receives either a request from downstream, or starts receiving a file unit from upstream.

Define $\lambda_{xy} := \delta_{xy} - \delta_x$ to be the total delay from node x down to node y, recalling that $\delta_r \equiv 0$ for the root r, so that $\lambda_{rv} = \delta_{rv}$. Denote the set of client nodes by W, consider a node $x \in W$ that has a full (reception) sequence from time 0 through M, and consider a node v, which is in T_x but not necessarily in W. Then, because of the nature of the flooding mechanism, the following holds :

Fact 1. *The first $M - \lambda_{xv}$ units appearing in the sequence of client node x appear in the same order but in the last $M - \lambda_{xv}$ positions of the sequence of node v.*

2.1 The Problem on a Path of Length K

We first consider the ANY-CLIENT problem on a path of length K under the flooding paradigm, where the transmission delay from any non-root node v to its parent is 1; this assumption will simplify the analysis, yet be easily extended to general integer delays. For convenience, label the nodes $1, 2, \ldots, K$, with node 1 being the child of the root, r.

Since node v might be the sole client for the file, then by Lemma 1, S_v has to have at least $2\delta_v$ file units, and the total number of distinct file units in the caches at the nodes along the path from client v to the root r must be at least $2\delta_{rv}$. Without loss of generality set up the following

Allocation: Put $S_i = \{2i-1, 2i\}$, for $1 \le i \le K$.

Adopt the convention that, if a node i starts supplying the units of its cache, then it does so in reverse order, i.e., it transmits unit $2i$ then unit $2i-1$.

Let the set of client nodes be $W := \{x_1, x_2, \ldots, x_j\}$, and assume that node x_i is closer to the root than node x_{i+1} (the x_i's form an increasing sequence). Since client x_1 is the closest client to the root, it does not receive a flood from another client node. Thus, x_1 first receives $2x_1$ units in the sequence $2x_1, 2x_1-1, \ldots, 1$ whose length is the time taken for the units from the root to reach x_1. Immediately then, we have

Fact 2. *Client x_1 will have seamless downloading if the units $2x_1+1, \ldots, M$ appear (in any order) in the last $M-2x_1$ positions of its reception sequence.*

Consider two successive client nodes, x_i and x_{i+1}. First, client x_{i+1} receives some units from the caches (including its own) of the nodes on its path to the root, and then it starts receiving the flood from x_i. In particular, client x_{i+1} receives $x_{i+1}-x_i$ units from above in the sequence $2x_{i+1}, 2x_{i+1}-1, \ldots, x_{i+1}+x_i+1$ whose length is the time taken for a flood from x_i to reach x_{i+1}. Thus, we have

Fact 3. *If client x_i has seamless downloading and the last $x_{i+1}-x_i$ positions of its reception sequence contain the units $2x_{i+1}, 2x_{i+1}-1, \ldots, x_{i+1}+x_i+1$, then x_{i+1} also has seamless downloading with a reception sequence starting with $2x_{i+1}, 2x_{i+1}-1, \ldots, x_{i+1}+x_i+1$ and ending with the first $M-(x_{i+1}-x_i)$ units in x_i's reception sequence.*

The construction below, to be called the Self Assembly Flooding (SAF) construction, begins with a reception sequence for x_1 that is the basis of a seamless, minimal cost solution. In particular, since the flood from x_1 reaches all other clients, we use this sequence to construct the sequences of the remaining clients. The latter construction satisfies Fact 3 and thus ensures seamless reception at all client nodes.

The SAF construction of client reception sequences: Client x_1 first receives $2x_1$ units in the sequence $2x_1, 2x_1-1, \ldots, 1$. At the end of its reception sequence, client x_1 receives $x_j - x_1$ units in the following sequence:

$$2x_j, 2x_j-1, \ldots, x_{j-1}+x_j+1, \ldots, 2x_2, 2x_2-1, \ldots, x_1+x_2+1$$

The remaining units of the file can be specified in any order during the time interval $2x_1$ through $M-(x_j-x_1)$.

From the construction of the sequence for x_1, iteratively construct reception sequences at all remaining client nodes by the construction described in Fact 3.

$\square$

Example 3: As an example in the flooding, self-assembly paradigm of the ANY-CLIENT problem, consider a path of length 6, where the transmission delay from any non-root node v to its parent is 1. For convenience, label the nodes $a, b, \ldots, f$, with node a being the child of the root, r. Let $M = 15$, and without loss of generality set up the following allocation: $S_a = \{1, 2\}$, $S_b = \{3, 4\}$, $S_c = \{5, 6\}$, $S_d = \{7, 8\}$, $S_e = \{9, 10\}$, and $S_f = \{11, 12\}$.

We give examples with the 3 different sets of clients $\{b, e, f\}$, $\{c\}$, and $\{a, f\}$. As can be seen, with the flooding mechanism, a minimal cost solution exists for each case. The reception sequences for the 3 sets of clients are given in Figures 4, 5, and 6 in the order the sets are given above.

a:				5	6	7	11	13	14	15	12	10	9	8	
b:	$\underline{4}$	$\underline{3}$	2	1	5	6	7	11	13	14	15	12	10	9	8
c:		4	3	2	1	5	6	7	11	13	14	15	12	10	9
d:			4	3	2	1	5	6	7	11	13	14	15	12	10
e:	$\underline{10}$	$\underline{9}$	8	4	3	2	1	5	6	7	11	13	14	15	12
f:	$\underline{12}$	10	9	8	4	3	2	1	5	6	7	11	13	14	15

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 time, t

Figure 1: The reception sequences when b, e, and f are clients for the file

a:					7	8	9	10	11	12	13	14	15		
b:				2	1	7	8	9	10	11	12	13	14	15	
c:	$\underline{6}$	$\underline{5}$	4	3	2	1	7	8	9	10	11	12	13	14	15
d:		6	5	4	3	2	1	7	8	9	10	11	12	13	14
e:			6	5	4	3	2	1	7	8	9	10	11	12	13
f:				6	5	4	3	2	1	7	8	9	10	11	12

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 time, t

Figure 2: The reception sequences when node c is a client for the file

□

The main result for ANY-CLIENT is as follows.

Theorem 1. *The SAF construction results in a minimal cost solution to ANY-CLIENT for paths under the flooding, self-assembly paradigm.*

Proof: For the case of unit delays the allocation is obviously of minimal cost; and the proof of seamlessness follows by induction on the number j of clients with Fact 2 and Fact 3 providing the basis and induction step.

a:	$\underline{2}$	$\underline{1}$	3	4	5	6	7	13	14	15	12	11	10	9	8
b:		2	1	3	4	5	6	7	13	14	15	12	11	10	9
c:			2	1	3	4	5	6	7	13	14	15	12	11	10
d:				2	1	3	4	5	6	7	13	14	15	12	11
e:					2	1	3	4	5	6	7	13	14	15	12
f:	$\underline{12}$	$\underline{11}$	10	9	8	2	1	3	4	5	6	7	13	14	15

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 time, t

Figure 3: The reception sequences when a, and f are clients for the file

For paths with general delays, replace each node v with a delay $\delta_v \geq 1$ by a sequence σ_v of δ_v consecutive unit-delay nodes. Next, set up the cache allocation for a minimal cost solution to the constructed unit-delay problem. Finally, for every node v combine the cache content of all of the nodes in σ_v and make it the cache content of the last node of the sequence, leaving the other nodes of the sequence with empty caches. Only the nodes with non-zero cache content are eligible to be clients, and it is easy to see that any subset of these nodes will have seamless self assembly in a minimal cost solution under the flooding, self-assembly paradigm.

□

The sacrifice usually made by flooding techniques is the stopping problem: how does one discontinue the flood when the initial objective has been met? For example, in the ANY-CLIENT path case, all flooding below the most distant client is useless; and in the case of trees the wasted downloading can obviously be much greater. If all nodes can be synchronized then the solution to this problem is simple: each node takes no further action beyond time M. But to avoid a full flood to the entire structure, some expansion of the sequencing protocol has to be made. We leave these issues as directions for further research.

2.2 The Problem on a Tree

The tree in Figure 7 shows that, in the flooding paradigm, a minimal cost solution does not always exist for tree structures. Without loss of generality create the following allocation: $S_a = \{1, 2\}$, $S_b = S_d = \{3, 4\}$, $S_c = \{5, 6\}$. Let $M = 8$.

Assume that at time $t = 0$, the clients at nodes c and d request the file. The client at node d will first receive the units from its cache, and then receive the units from the cache of node a. Note that the units from the cache of a will also start arriving at node c at time $t = 3$. Thus, there is not enough time for node c to receive both units 3 and 4 from the cache of node b. In time units 4 through 7, node a should receive units $\{5, 6, 7, 8\}$ so that node d can receive all of the file in the first M time units. These units will also go down to client c because of the flooding mechanism. Thus, client c

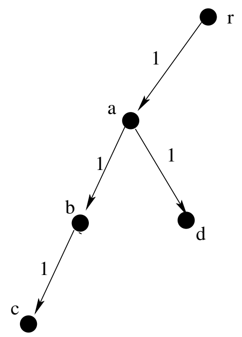

Figure 4: The tree under consideration

cannot receive the entire file in the first M time units and a seamless solution does not exist.

3 Concluding Remarks

Directions for further research, staying within the domain of this paper include the obvious generalizations of our assumptions. For example, we are currently working on solutions to ANY-CLIENT for trees under distributed protocols that relax the assumptions of the flooding paradigms.

Problems like CLIENT and ANY-CLIENT serve as tests of the complexity of many more general problems which are our eventual objective. For example, a problem that begs consideration is CLIENT-RELEASE in which the times the client nodes make their requests are allowed to be arbitrary integers which become part of the problem instance. Thus, an instance of the CLIENT-RELEASE problem consists of the edge-labeled tree, an integer M denoting the number of units in F, the subset of nodes that are clients and their respective release times, (i.e., the times the client nodes make their requests), which are allowed to be arbitrary integers. The following example on a path shows the complexity of the CLIENT-RELEASE problem.

Consider a path of length 3, where the one-way transmission delay from any non-root node v to its parent is 1, and all three nodes are clients. For convenience, label the nodes a, b, c, with node a being the child of the root, r. Denote the release time of node v by t_v, and denote the time at which node v first receives a request (including the one from its own client) by t_v^*. (Note that because the release times are allowed to be arbitrary integers, t_v^* does not necessarily have to be equal to t_v.) Let $M = 7$, $t_c = 0$, $t_a = 5$, and $t_b = 6$. Thus, it follows that $t_c^* = 0$, $t_b^* = 1$, and $t_a^* = 2$. Each node has to cache the minimum number of file units needed to cover the two-way communication delay with its parent, and given the above release times we see that these $2\delta_v$ units have to be disjoint. Let S_v denote the set cached at node v, and without loss of generality do the following allocation: $S_a = \{1, 2\}$, $S_b = \{3, 4\}$, and $S_c = \{5, 6\}$. The reception sequence at node c has to begin (at time $t = 0$) with the units in S_c, followed by the units in S_b, followed by the units in S_a, followed by unit 7 that originates from the root.

Thus, it is easy to see that client node c can have seamless self assemblhy. We will now show that with the given allocation scheme, seamless self assembly is impossible at both clients a and b.

Now, at time $t = 5$ the reception sequence at node a has to begin with the units in $S_a = \{1, 2\}$ followed by 5 units originated from the root, and at time $t = 6$ the reception sequence at node b has to begin with the units in $S_b = \{3, 4\}$ followed by 5 units sent to b from a. Remember at this point that a node can only send the same unit to its children and to its own client if it has one. Also note that the time interval during which node a will receive 5 units from the root, (this interval is the time units $8, 9, \ldots, 12$), coincides with the time interval it has to send 5 units to node b. In order for node a to have seamless downloading, it has to receive units $\{3, \ldots, 7\}$ from the root. However, in order for node b to have seamless self assembly, node a has to send to b in the same time interval the units $\{1, 2, 5, 6, 7\}$. Thus, we see that it is impossible for both clients to have seamless self assembly with the given allocation scheme.

Another interesting question that we are pursuing is the relation between streaming and seamlessness: From the allocations in the set of minimal cost solutions, which allocation also minimizes the maximum delay until streaming can begin at each client? Multiple-file problems, trees where file units can also go upstream, structures more general than trees, and non-homogeneous link transmission rates are among the many directions for future research in seamless self assembly that are worth pursuing.

References

[1] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of 4th International World Wide Web Conference*, pages 119–133, Boston, USA, December 1995.

[2] Len Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, David Kempe, Pablo Moisset de Espanés, and Paul Wilhelm Karl Rothemund. Combinatorial optimization problems in self-assembly. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 23–32, New York, NY, USA, 2002. ACM Press.

[3] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, 1996.

[4] Israel Cidon, Shay Kutten, and Ran Soffer. Optimal allocation of electronic content. *Computer Networks*, 40(2):205–218, 2002.

[5] E. G. Coffman, Andreas Constantinides, Dan Rubenstein, Bruce Shepherd, and Angelos Stavrou. Content distribution for seamless transmission. *SIGMETRICS Perform. Eval. Rev.*, 32(2):31–32, 2004.

[6] P. Krishnan, Danny Raz, and Yuval Shavitt. The cache location problem. *IEEE/ACM Trans. Netw.*, 8(5):568–582, 2000.

[7] Bo Li, Mordecai J. Golin, Guiseppe F. Italiano, and Xin Deng. On the optimal placement of web proxies in the internet. In *Proceedings of IEEE INFOCOM*, pages 1282–1290, New York, March 1999.

[8] Keqiu Li and Hong Shen. Optimal placement of web proxies for tree networks. In *Proceedings of the 2004 IEEE Conference of e-Technology, e-Commerce and e-Service*, pages 479–486, March 2004.

[9] S. V. Nagaraj. *Web Caching and Its Applications*. Kluwer Academic Publishers, Inc., Norwell, MA, USA, 2004.

[10] Dean Povey and John Harrison. A distributed internet cache. In *Proceedings of the 20th Australasian Computer Science Conference*, Sydney, Australia, February 1997.

[11] Steven A. Przybylski. *Cache and memory hierarchy design: a performance-directed approach*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1990.

[12] Michael Rabinovich and Oliver Spatschek. *Web caching and replication*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[13] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Trans. Netw.*, 9(4):404–418, 2001.

[14] Subhabrata Sen, Jennifer Rexford, and Donald F. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE INFOCOM*, pages 1310–1319, New York, March 1999.

[15] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, 1982.

[16] Jia Wang. A survey of web caching schemes for the internet. *SIGCOMM Comput. Commun. Rev.*, 29(5):36–46, 1999.

[17] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.