

A novel cache distribution heuristic algorithm for a mesh of caches and its performance evaluation

Jorge Escorcia^a, Dipak Ghosal^b, Dilip Sarkar^{a,*}

^aDepartment of Computer Science, University of Miami, Coral Gables, FL 33124, USA

^bDepartment of Computer Science, University of California, Davis, CA 95616, USA

Received 24 October 2000; revised 4 May 2001; accepted 29 May 2001

Abstract

The widespread use of the Internet has created two problems: document retrieval latency and network traffic. Caching of documents ‘close’ to users has helped alleviate both problems. Different caching policies have been proposed/implemented to make best use of limited available cache at each caching server. A mesh of caching servers, aided by different data diffusion algorithms and the natural hierarchical structure of the Internet topology, has increased ‘virtual’ size of cache. Yet the size of available cache is small compared to the total size of all documents served, and remains a major resource constraint. In this work, we looked at how to improve document download time, by distributing a fixed amount of total storage in a network or mesh of caches. The intuition behind our cache distribution approach is to give more storage to the caching nodes in the network, which experience more traffic, in the hope that this will reduce the average latency of document retrieval in the network. A heuristic was developed to estimate traffic at each cache of a network. From this traffic estimation, each cache then receives a corresponding percentage of the total storage capacity of the network. Through extensive simulation it is found that the proposed cache distribution algorithm can reduce latency up to 80% over prior work that includes both Harvest-type and demand-driven data diffusion algorithms. Furthermore, the best improvement was achieved in a cache range that corresponds to practical, real world cache ranges. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Heuristic algorithm; Internet topology; NetCache; World Wide Web

1. Introduction

As the (World Wide) Web connects information sources or origin servers around the world, users from everywhere attempt to access these sources. Due to the ease and low cost of access, both the number of users and documents are growing dramatically [21]; per user network traffic is also increasing. When a user accesses a faraway document, the time it takes to download (latency) could be unacceptably high. Moreover, when a user or a group of users connected to a network-neighbourhood repeatedly access a faraway document, the traffic on the network, and hence the demand for bandwidth is increased.

An effective method for reducing both latency and network traffic is to use shared caches [1,10,18,19,25]. A mesh is a set of cooperative caches connected as a network, and they can directly or indirectly share documents stored in them. Some of these caches provide direct access to the

clients; these are called proxy caches. Others are called caching servers and provide direct or indirect access to other caches (see Section 4.2 for more details about network model). As a mesh, caches provide the effect of a ‘large virtual’ cache with a higher hit rate, improved access to the origin servers, reduction in document downloading time, and reduction in network traffic.

The two most commonly employed mesh caching methods are (a) hierarchical and (b) directory-based methods [23,25]. In the directory-based method, when a request for a document arrives at a caching server, if the caching server has the document in its local cache, it serves it from there. Otherwise, the caching server queries a mapping service to locate the document in the caching mesh. In the case of a hit, the document is fetched from the peer-caching server. In the case of a miss, the document is fetched from the origin server.

In hierarchical caching, the caching servers in a mesh are (logically) organized in a hierarchy. National caches connected to the backbone network are on the top of the hierarchy [2,8,9]. Regional caches are connected to national caches and occupy the next lower level in the hierarchy.

* Corresponding author. Tel.: +1-305-284-2256; fax: +1-305-284-2264.

E-mail addresses: ghosal@cs.ucdavis.edu (D. Ghosal), sarkar@cs.miami.edu (D. Sarkar).

Subregional caches are connected at the next lower level to regional caches, and institutional caches serve as the ‘gateways’ to subregional caches. This hierarchy naturally fits with the standard topology of the Internet. Because of this natural hierarchical topology, the directory based caching method could be implemented at one or two levels of the hierarchy, and not at all levels of the hierarchy.

To exploit the hierarchical topology of the Internet, the Harvest project originated a data diffusion scheme (will be referred to as Scheme 1 in this paper) that has led to popular commercial products (such as NetCache) and public-domain products (such as Squid) [12]. The Harvest [14], or Harvest-derived data diffusion schemes replicate documents on the mesh hierarchy using a simple strategy: whenever a document is retrieved from an origin or a caching server, it is replicated on every cache along the path to the client. Although this data diffusion scheme greatly reduces the latency and network traffic, it can be improved many ways. One would get significant improvement, by not replicating relatively ‘unpopular’ documents, if it would replace a ‘popular’ document [21].

Demand driven data diffusion schemes replicate documents only if they are relatively popular at that cache. There are two approaches to achieve this replication process. In a top-down protocol, a document is replicated at a cache’s child if it appears to be very popular and may create a hot-spot. This protocol is aimed at avoiding hot-spots, and will work well in conjunction with an appropriate hashing technique [4,5,13,15]. To implement this technique, significant changes will be necessary in currently used caching strategies.

The bottom-up approach replicates a document at a cache only when a relatively large number of requests for a document is observed [21]. In Section 2.2, we present this demand driven data diffusion strategy and we refer to this as Scheme 2.

Even if we had the best/optimal data diffusion scheme, another practical problem needs to be addressed: the size of available storage or cache. We use cache size and storage capacity interchangeably. Analysis of access logs from Ref. [22] shows that the storage capacity at a server (unless specified, will refer to a proxy or a caching server) is less than three percent of all documents served by it [16]. Thus, one could think that storage capacity is a limited resource. In our study, we assume that total amount of available storage on all the servers in a mesh of caches is constant. Moreover, it is assumed that available storage can be distributed among the servers as desired. Under these assumptions, a natural question is to ask: how to distribute total available storage to the servers for minimizing latency in document downloading, and also reduce network traffic?

In this work, we propose and evaluate a storage distribution heuristic. The intuition behind our storage distribution heuristic is that a server with more expected traffic should get a bigger share of the available storage. A heuristic was developed to estimate traffic at each cache of a network.

From this traffic estimation, each cache then receives a corresponding percentage of the total storage capacity of the network. The proposed storage distribution heuristic is presented in Section 3.

Simulation has been the most common method for evaluation of caching algorithms for a proxy cache server [11]. Two possible sources of input data for simulation are traces of requests from a cache server, or synthetic/artificially generated requests. Trace data now available for a single caching server can be used for realistic evaluation of different caching algorithms running at a proxy server. However, simulation study of any data diffusion scheme requires trace data of an entire mesh of caches, but these are not readily available. Hence, for our study we use synthetic data derived from actual server logs. Our simulation model is presented in Section 4. Our observations from the simulation study are presented and discussed in Section 5. Results show that our cache distribution heuristic achieves up to 80% latency improvement in Harvest-type and demand-driven data diffusion algorithms (when compared to equal size cache in all servers). Furthermore, the best improvement was achieved in a cache range that corresponds to practical, real world cache ranges. Section 6 discusses the findings and possible future directions of study.

2. Hierarchical caching

In a hierarchical cache organization, each caching node has a parent, siblings, and children. The term ‘neighbor’ is used to refer to parent or siblings that are one cache-hop away. The protocol that caches in a caching hierarchy use is the Internet Cache Protocol (ICP) [24]. The request resolution protocol in hierarchical caching is as follows: When a cache receives a request, it checks whether it has the requested object. If it has the object, it satisfies the request by serving the object from its cache. If it does not have the object, it multicasts an ICP request to its neighbors. If one of the neighbors has the object, a neighbor hit, then the object is retrieved from that neighbor. If multiple neighbors have the object, the object is retrieved from the neighbor with the lowest measured latency. If none of the neighbors has the object, a neighbor miss, which is determined from a negative ICP response or a timeout, the request is either forwarded to a parent or sent directly to the origin server. The origin server is the initial and permanent repository of the object. Forwarding the request to a parent simply repeats the process. If the request misses at each level of the hierarchy, a top-level parent fetches the object from the origin server, caches it, and passes it down the hierarchy to the leaf node that originated the request.

There are two key objectives in the design of hierarchical caches. The first is to ensure that popular objects are cached at the lower levels of the hierarchy, while less popular objects are cached at the higher levels. The second objective is to ensure that when there is a cache miss at a particular

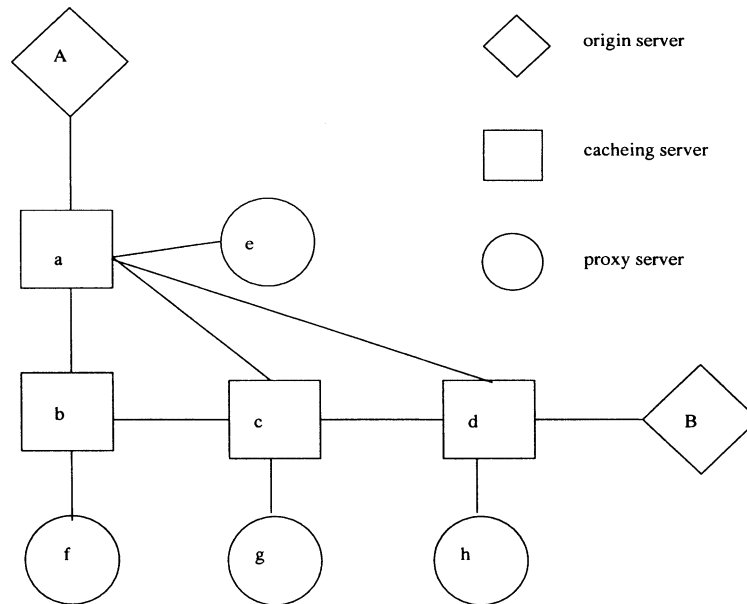


Fig. 1. An example network.

caching server, the request is satisfied by a caching server in the same or higher level, thereby precluding the need for the request object to be served by the origin server. These two objectives reduce the latency perceived by the user and the overall network traffic. Both objectives, especially the first, depend on the data diffusion strategy and the cache replacement strategy. These strategies determine how the objects are placed in the caching hierarchy. In Sections 2.1 and 2.2, we present two such data diffusion strategies.

2.1. Harvest scheme

The data diffusion strategy in the Harvest scheme replicates objects in the caching mesh by caching the objects at every node along the path when it comes down the hierarchy [8]. The motivation for caching the object at all the nodes in the downward path is that subsequent requests for the object would have multiple points in the caching hierarchy from where it could get satisfied, thus decreasing the likelihood that the request would need to be satisfied by the origin server. When the cache space at a node is full, and a new object is to be cached, a previously cached object must be dropped from the cache. This is what is referred to as the cache replacement strategy. Harvest-derived schemes can support different cache replacement strategies, such as LFU and LRU. Finally, in Harvest-derived schemes when an object is dropped from the cache, neither the object itself, nor any meta-information regarding the object, is passed up the hierarchy.

2.2. A demand-driven data diffusion algorithm (D4)

We shall refer to this algorithm as the D4 algorithm. The D4 algorithm uses a reference counting scheme as part of its cache replacement strategy [21]. Each cache in the mesh

maintains a reference count for all the objects for which it has received a request over some time interval. This meta-information is maintained even when the object is not in the cache. When a node receives an object request, it increases the reference count for that object. If the object is in the node's cache, it serves the request, otherwise it forwards the request to the next node along the path to the origin server, which is then handled recursively. When the object comes down, after being served at some higher level, each node along the path uses an LFU replacement strategy to determine whether or not to cache the object when the cache space is full. The reference count is used as the frequency measure. When an object is dropped from a cache, its reference count measured at that node is passed up to the next node (parent node) along the path to the origin server. The parent node adds this reference count to its own reference count for the object. If the parent cache has the object, it does nothing. If it does not have the object, it determines what the result would be of trying to cache the object at that instant using the new reference count. If the decision was that the object would be cached, nothing is done. However, if the decision was that the object would be dropped, the parent sends up the reference count it received from the child to its own parent, which is then handled recursively. The motivation here is to propagate the meta-information to the right point in the hierarchy, i.e. a node where an object is already cached or a node which determines it would be able to cache the object the next time it comes around. Furthermore, in this scheme, when a node receives a request for an object that already has an unsatisfied request, the reference count for the object is increased. The node then determines whether to generate meta-information regarding the request and pass it up the hierarchy. The procedure here is analogous to the one before. The motivation is to propagate the

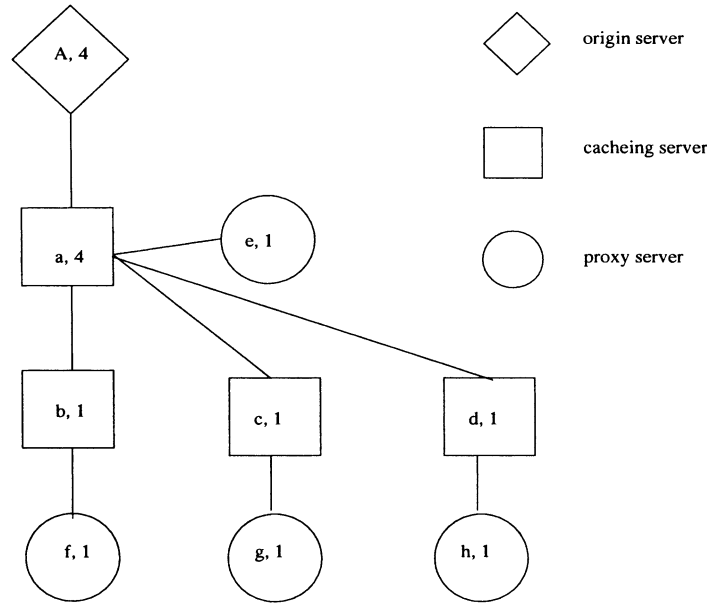


Fig. 2. Enumerated layered network from origin server A.

meta-information to the point where the object is cached or ‘expected’ to be cached.

3. A cache distribution algorithm

The motivation behind distributing the cache is to maximize the benefits of caching. Given a limited amount of cache in the network, placing more cache in the nodes which experience more traffic, while keeping the average cache per node over the whole network the same, may improve the overall latency of the network. Since the cache size of the whole network is fixed, placing more cache at one node means reducing cache at another node. The cache distribution algorithm determines how much cache to place at each node by giving an estimate of how much traffic each particular node may experience. The following definitions are introduced to explain the algorithm.

3.1. Definitions

Definition 1. A network topology is a graph $NT(V; E)$, where $V = \{v_1, v_2, \dots, v_n\}$ is the nonempty set of n nodes of the network, and $E = e_1, e_2, \dots, e_m$ is the set of m edges, such that $e_i = (v_{i_1}, v_{i_2})$, and $v_{i_1}, v_{i_2} \in V$. The set V is the union of the set of origin servers (OS), the set of caching servers (CS), and the set of proxy servers (PS). Thus, $V = OS \cup CS \cup PS$. Also, in our study $OS \cap CS = \emptyset$, $CS \cap PS = \emptyset$, and $OS \cap PS = \emptyset$.

Example. Fig. 1 is a sample network to illustrate the definitions and the cache distribution algorithm. The set V of

nodes in this network are $\{A, B, a, b, c, d, e, f, g, h\}$ and the set E of edges are $\{(A, a), (a, b), (a, c), (a, d), (a, e), (B, d), (b, c), (b, f), (c, d), (c, g), (d, h)\}$. Furthermore, the set of origin servers (OS) is $\{A, B\}$, the set of caching servers (CS) is $\{a, b, c, d\}$; and the set of proxy servers (PS) is $\{e, f, g, h\}$. It is readily seen that the union of OS, CS, and PS is V , and that the intersection of OS, CS, and PS is null.

Definition 2. A layered network $LN(V', FL, E')$ is a subgraph of network topology $NT(V, E)$, where $V' \subseteq V$, $E' \subseteq E$, and $FL \subseteq V'$. The set of nodes V' of $LN(V', FL, E')$ is partitioned into subsets L_1, L_2, \dots, L_l , where $L_1 = FL$ is the first layer of the layered network. There is no edge between two nodes of a layer. The first layer L_1 is connected to only the nodes of the layer L_2 . The nodes of a layer L_i can only be connected to the nodes of the layer L_{i-1} and L_{i+1} . Naturally, the nodes of layer L_1 are only connected to the nodes of layer L_{l-1} .

Example. From the network in Fig. 1, we create two layered networks as shown in Figs. 2 and 3. The numbers in the figures will be explained in Definition 3. In Fig. 2, $FL = \{A\}$ so $L_1 = \{A\}$, $L_2 = \{a\}$, $L_3 = \{b, c, d, e\}$ and $L_4 = \{f, g, h\}$. Also, $E' = \{(A, a), (a, b), (a, c), (a, d), (a, e), (b, f), (c, g), (d, h)\}$. In Fig. 3, $FL = \{B\}$ so $L_1 = \{B\}$, $L_2 = \{d\}$, $L_3 = \{a, c, h\}$, $L_4 = \{b, e, g\}$, and $L_5 = \{f\}$. Finally, $E' = \{(B, d), (d, a), (d, h), (d, c), (a, e), (a, b), (c, b), (c, g), (b, f)\}$. Note: For our purposes, nodes of OS and CS are included in the layered network only when they are the

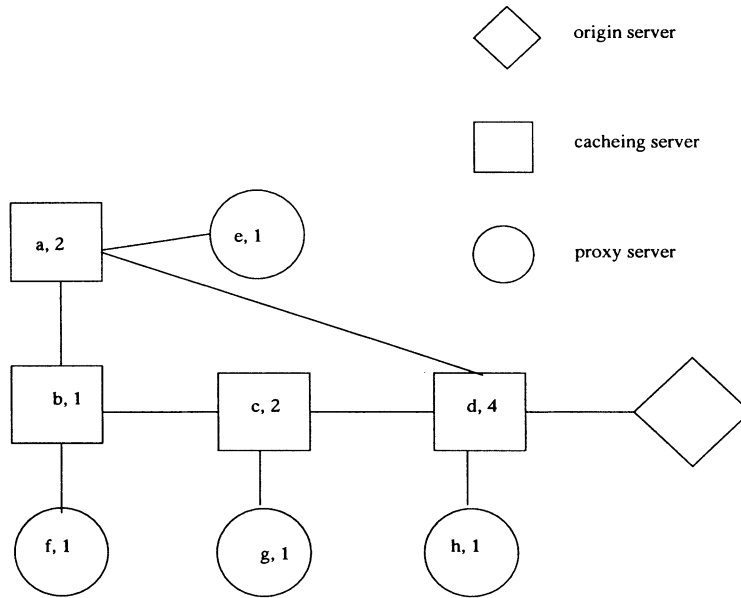


Fig. 3. Enumerated layered network from origin server B.

FL or when they have connections to layers above and below.

Definition 3. An enumerated layered network, $ELN(VNP', FL, E')$, where $VNP' = \{(v_1, N_1, P_1), (v_2, N_2, P_2), \dots, (v_n, N_n, P_n)\}$, is obtained from a $LN(V', FL, E')$ such that $P_i \subseteq V'$ and N_i is the number of elements in P_i .

An enumerated layered network is obtained from a layered network as follows: If a node $v_j \in PS$ in layer L_i has no connection to a node, the set P_j is $\{v_j\}$ and the enumerated value, N_j for that node is one, resulting in the

triple $(v_j, 1, \{v_j\})$. If $v_j \in OS$ or $v_j \in OS$ then the set P_j is empty and N_j is 0, resulting in the triple $(v_j, 0, \{\})$. A node's set P can be calculated only when all the nodes of its higher layer have obtained their sets P . Let v_i be a node in layer j . The set of nodes in the $(j + 1)$ th layer that are connected to v_i are $v_{i_1}, v_{i_2}, \dots, v_{i_{s_i}}$, the corresponding enumerated values are $n_{i_1}, n_{i_2}, \dots, n_{i_{s_i}}$, and the corresponding sets P are $p_{i_1}, p_{i_2}, \dots, p_{i_{s_i}}$. The set p_i is given by

$$p_i = \bigcup_{k=1}^{s_i} p_{i_k}$$

and n_i is the number of elements of p_i .

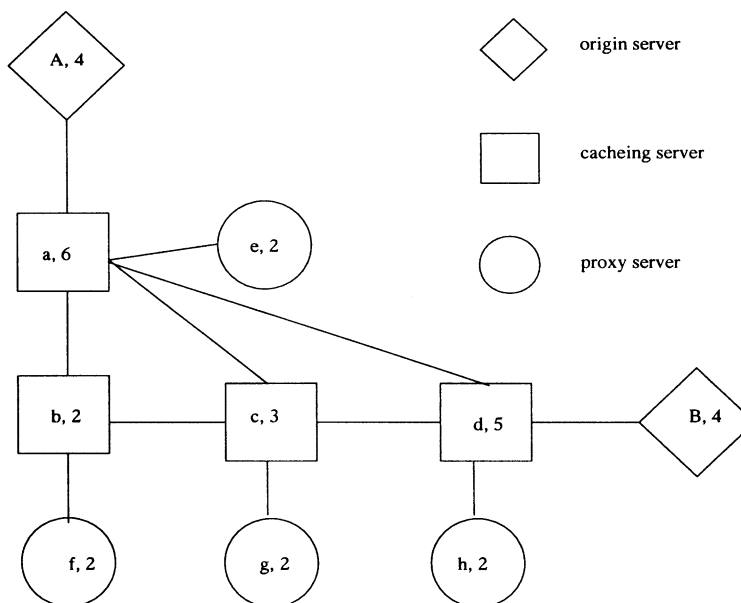


Fig. 4. Union of enumerated layered networks.

Example. Figs. 2 and 3 are the enumerated layered networks derived from the sample network in Fig. 1.

Note: For our purposes the sets P need not be included in the figures. For the ELN in Fig. 2 VNP' is $\{(A, 4, \{e, f, g, h\}), (a, 4, \{e, f, g, h\}), (b, 1, \{f\}), (c, 1, \{g\}), (d, 1, \{h\}), (e, 1, \{e\}), (f, 1, \{f\}), (g, 1, \{g\}), (h, 1, \{h\})\}$ and the FL and E' are the same as in the corresponding LN. For the ELN in Fig. 3 VNP' is $\{(B, 4, \{e, f, g, h\}), (d, 4, \{e, f, g, h\}), (a, 2, \{e, f\}), (c, 2, \{f, g\}), (h, 1, \{h\}), (e, 1, \{e\}), (b, 1, \{b\}), (c, 2, \{f, g\}), (f, 1, \{f\}), (g, 1, \{g\})\}$ and once again the FL and E' are the same as in the corresponding LN.

Definition 4. Let $\{ELN_1(VNP_1, E_1), ELN_2(VNP_2, E_2), \dots, ELN_k(VNP_k, E_k)\}$, be a set of k ELNs. The union of this set is an enumerated graph $UELN(VN, E'')$, such that

$$E'' = \bigcup_{i=1}^k E_i \quad \text{and} \quad VN = \{(v_i, N_i)\}$$

where v_i is a node in one or more ELNs, and N_i is the sum of all the enumerated values corresponding to the node v_i .

Example. Fig. 4 shows the union of the two enumerated layered networks from Figs. 2 and 3. This figure is essentially Fig. 1 with the corresponding enumerated values included for each node. The sum of all the enumerated values corresponding to each node is readily obtained from Figs. 2 and 3, $VN = \{(A, 4 + 0 = 4), (B, 0 + 4 = 4), (a, 4 + 2 = 6), (b, 1 + 1 = 2), (c, 1 + 2 = 3), (d, 1 + 4 = 5), (e, 1 + 1 = 2), (f, 1 + 1 = 2), (g, 1 + 1 = 2), (h, 1 + 1 = 2)\}$.

3.2. The algorithm

A pseudo code of the algorithm is presented in Fig. 5, and an informal description follows. When a proxy server makes a request, the shortest path to the origin server is taken to satisfy the request. Along the way, intermediate caching servers try to satisfy the request and if unable to do so then continue to send the request towards the origin server. Each caching server then ‘sees’ requests coming from a specific set of proxy servers. To determine the traffic at a particular node, the numbers of proxy server, which may potentially send requests through the node are determined by creating an enumerated layered network (Definition 3) from each origin server. Each node retains a count of how many proxy servers may send requests through it on the way to each origin server.

(Note: The algorithm considers all potential shortest paths. It is possible that a specific shortest path is chosen by the routers out of many potential shortest paths, yet this

```

let CT be the total amount of cache to be distributed;
let OS be the set of origin servers;
for each  $os_i \in OS$  {
construct layered networks,  $LN(V_i, os_i, E_i)$ ;
construct enumerated networks,  $ELN(VNP_i, os_i, E_i)$ ;
}
construct union of enumerated layered networks,  $UELN$ , by
taking the union of the enumerated layered networks,  $ELNs$ ;
let  $N = \sum_{i=1}^k N_i$ ;
let  $c_i$  be the available cache at each node  $v_i$ ;
for each node  $v_i, c_i \cong ((N_i/N) * CT)$ ;

```

Fig. 5. Pseudo code for the cache distribution algorithm.

path may not be readily known, so the algorithm must take them all into account. If specific shortest paths are known, the algorithm can be easily modified to account for this.) The union of these enumerated layered networks (Definition 4) then gives us the total counts assigned to each node in the network. The percentage of the total count is determined for each node, which correlates to the percentage of cache each node should receive. Since we are using whole numbers to represent cache size, some rounding is introduced, so further manipulation of the numbers may be required to keep the total cache size of the network equivalent to the original size before redistribution. Section 3 illustrates an example of the algorithm.

3.3. An example

We revisit Figs. 1–4 one more time to give a complete example of the cache distribution algorithm. From the example network in Fig. 1 we create two layered networks, one from server A and one from server B, as shown in Figs. 2 and 3, respectively, and ignoring the numbers. Note: Since caching is only performed by the caching and proxy servers, the origin servers are included in the layered networks only when they serve as the first layer in the network. Furthermore, caching servers are included in the network only when they have a connection to a layer above and a layer below. The next step is to create an enumerated layered network from each of the layered networks which are also shown in Figs. 2 and 3. Finally, the union of these enumerated layer networks is determined as shown in Fig. 4. Once the union of the enumerated layered networks is determined, we have the total enumeration value for each node. This number provides the basis for distributing the cache.

The cache distribution results are shown in Table 1. In this example, we assume the total available cache in the network is 40, so if we were to distribute the cache evenly among the proxy servers and caching servers, each would receive a cache size of 5. The first step in distributing the cache is determining each node’s percentage of enumeration over the whole network. Note: Since the origin servers do

Table 1
Results of cache distribution

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	Total
<i>A</i>	4	1	1	1	1	1	1	1	11
<i>B</i>	2	1	2	4	1	1	1	1	13
Total	6	2	3	5	2	2	2	2	24
%	25.00	8.33	12.50	20.83	8.33	8.33	8.33	8.33	99.98
Calculated cache size	10	3	5	8	3	3	3	3	38
Final cache size	11	3	5	9	3	3	3	3	40

not perform caching, their enumeration values are not considered in these calculations. To distribute the cache, the node's count percentage is multiplied by the total cache in the network. As figure shows, the calculated cache size for each node does not add up to 40 simply because a rounding error is introduced. To compensate for this, the cache sizes of individual nodes are adjusted to make the total cache size of the redistributed cache an even 40.

4. Simulation model

To study the effects of the cache distribution using the novel algorithm proposed here, the operation of a mesh of caches is simulated. Both the data diffusion algorithms described in Section 2 were implemented in the simulated environment. The simulator is a C++ program, which makes extensive use of LEDA [17] object libraries, and is an extended version of the simulator used in the study reported in Ref. [21]. The extended version is discussed in Section 4.1. The inputs to the simulator make up the network topology and the request profile for each object. The network topology is specified in Graph Meta Language (GML) format, which also specifies the cache size of each node and the bandwidths of individual links.

4.1. Simulator extension

The simulator was extended to accommodate different cache sizes for the nodes. For all the nodes, the original simulator only allowed to have the same cache size. The GML format provides the means to assign the different cache sizes, so for each topology at each average cache size two GML files were created, one with cache distribution and one without. The original simulator would only need two GML files (one for each topology) to run the simulations. Our simulations required a total of 40 GML files. Each topology was simulated using 10 different average cache sizes, so each topology requires 20 GML files, 10 with cache distribution and 10 without cache distribution. Lastly, the program code was modified in order to read in this new cache attribute in the GML files.

4.2. Network model

The network consists of three types of nodes, namely

proxy servers, origin servers and intermediate caching servers. Proxy servers are the leaf nodes in the network where requests are generated. Origin servers are the initial storing place of the requested objects. Intermediate caching servers reside somewhere along the path from the proxy servers to the origin servers. Note that only the proxy servers and intermediate caching servers perform caching. Origin servers contain the same set of objects throughout the entire run of the simulation. The nodes are connected by bi-directional links with a specified bandwidth. Each node also performs the function of forwarding packets to the next hop.

When an object is to be transmitted through the network it is first split into packets. These packets are then sent to their destination where they arrived in order and reassembled there. The path taken through the network is always the shortest path between the sender and the receiver. The link and node models were carefully constructed to ensure fairness among multiple connections. Consider a node that simultaneously starts transmitting a large file and a small file to the same node. If all the packets of the large file are sent first, then the large file would reach the destination prior to any of the packets of the small file. The large file ends up monopolizing the link, which is unfair.

To handle the above scenario properly, in the node, arriving packets are queued on a per connection basis, so that packets belonging to different connections occupy different queues. In this way, packets from different connections do not affect each other. A node can transmit multiple packets to a link at the same time as long as the packets belong to different connections. Packets within a queue are transmitted to the appropriate link in order, so a packet cannot be placed on a link until the previous one packet has been completely transmitted.

4.3. Request generation

The request profile for an object is a piece-wise linear graph that plots the request rate for the object over time. The plotted request rate for the object is its request rate in the aggregate, i.e. the sum total of the rates at which requests are generated for the object by all the proxy servers in the network. Once the aggregate request profile for an object is determined, it is broken up into per proxy server request profiles. This is done by taking a fixed set of points from the piece-wise linear graph and then for each point,

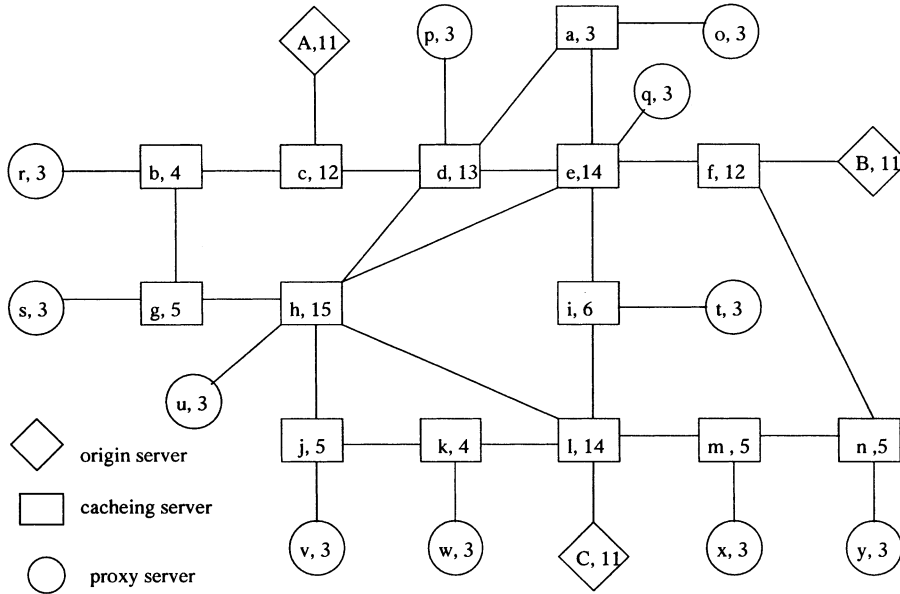


Fig. 6. Topology I displayed with the enumeration values obtained after performing the union of the three layered networks obtained from this network.

distributing the corresponding request rate among the proxy servers. This distribution is pseudo-uniform, i.e. each proxy server’s share of the aggregate request rate lies within a bounded interval of the uniform distribution value. Once the proxy server request profiles are determined, the actual request generation in the simulator takes place as follows. To generate a request for an object O at proxy server P at time T, the request rate for O at P at instant T is taken. If this rate is λ , then the inter-arrival time is obtained by modeling the incoming stream of requests for O at P as a Poisson process with mean λ .

Finally, the 90/10 rule [3] was used to assign the request rates to the objects. Consequently, 10% of the objects are marked as popular and the aggregate request rates for these

popular objects are scaled up by a factor of 81 times the corresponding rate for the unpopular objects. Both popular and unpopular objects are uniformly distributed across the origin servers of the particular network.

4.4. Network topologies

In our simulation, we considered two different mesh like topologies, which we refer to as Topology I and Topology II, and shown in Figs. 6 and 7, respectively. Topology I is looked at in depth for the purposes of demonstrating the cache distribution algorithm.

Topology I was obtained by taking the backbone mesh of a major carrier services provider and attaching proxy servers

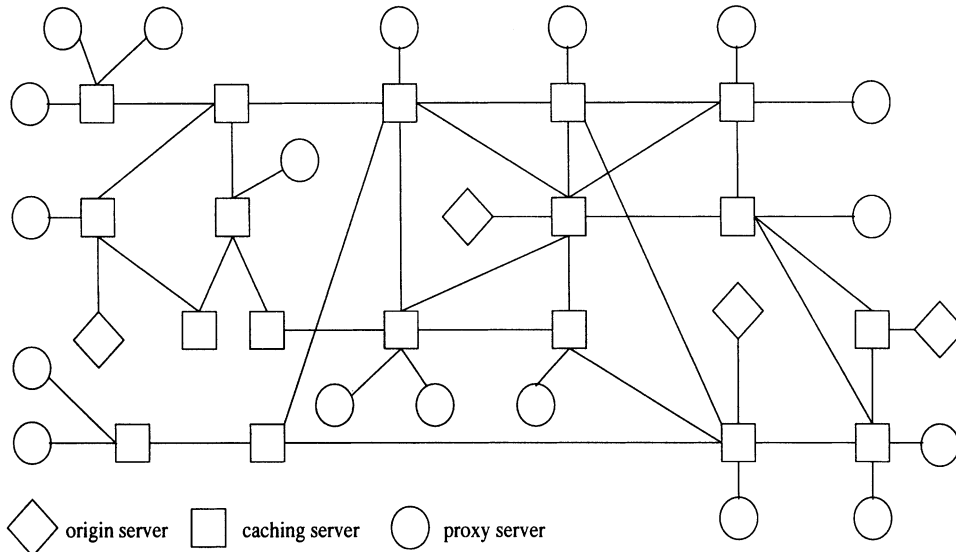


Fig. 7. Topology II.

Table 2
Cache distribution results of Topology I

Nodes	Count	%	Avg-2	Final	Avg-4	Final	Avg-6	Final	Avg-10	Final	Avg-20	Final
<i>a</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>b</i>	4	2.7	1	1	3	3	4	4	7	7	13	13
<i>c</i>	12	8.0	4	4	8	8	12	12	20	20	40	40
<i>d</i>	13	8.7	4	4	9	9	13	13	22	22	43	43
<i>e</i>	14	9.3	5	5	9	9	14	14	23	23	47	47
<i>f</i>	12	8.0	4	4	8	8	12	12	20	20	40	40
<i>g</i>	5	3.3	2	2	3	3	5	5	8	8	17	17
<i>h</i>	15	10.0	5	5	10	11	15	15	25	26	50	49
<i>i</i>	6	4.0	2	2	4	4	6	6	10	10	20	20
<i>j</i>	5	3.3	2	2	3	3	5	5	8	8	17	17
<i>k</i>	4	2.7	1	1	3	3	4	4	7	7	13	13
<i>l</i>	14	9.3	5	4	9	9	14	14	23	23	47	47
<i>m</i>	5	3.3	2	2	3	3	5	5	8	8	17	17
<i>n</i>	5	3.3	2	2	3	3	5	5	8	8	17	17
<i>o</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>p</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>q</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>r</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>s</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>t</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>u</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>v</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>w</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>x</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
<i>y</i>	3	2.0	1	1	2	2	3	3	5	5	10	10
Total	150		51	50	99	100	150	150	249	250	501	500

to it. Topology I has 3 origin servers, 15 caching servers, and 11 proxy servers. Fig. 6 shows the network topology as well as the enumeration values obtained after performing the union of the three layered networks obtained from this network. Table 2 shows the results after performing the cache distribution algorithm given the enumeration values of Topology I. As the table shows, the cache distribution algorithm is applied using various cache sizes. These cache sizes correspond to the first 6 of the 10 cache sizes that were used to run the simulation. Since there are a total of 25 caching nodes in the network the total available cache for each average cache size is obtained by multiplying the average cache size by 25. The table shows two columns of numbers for each cache size. The first column of numbers corresponds to the result of multiplying the percentage of enumeration for each node times the total cache size for that particular average cache size. These numbers do not always add up to the total original cache size, so the second column of numbers are the final cache configurations after fine tuning some of the numbers to get the total to come out correctly.

As Fig. 7 shows Topology II has 4 origin servers, 18 caching servers, and 18 proxy servers. It was derived from Mapnet [20]. The raw data obtained from Mapnet's web site contains topology information on federal educational/research networks and commercial networks. To get an experimental topology of reasonable size networks, which had about 20 nodes each were identified. These topologies were then analyzed in groups of three to find the combina-

tion that maximizes the number of nodes shared by these networks. Finally, since Topology II contains 4 origin servers, 4 enumerated layered networks are obtained and the union of these then yields the final enumeration values for Topology II. We do not show a table of cache distribution for Topology II for saving space.

4.5. Other simulation parameters

The caching capacity of the caching nodes is in units of number of objects. The object size is fixed at 10 KB. The size of the request packets and the meta-information in the case of the D4 Algorithm is 100 bytes. Finally, the total cache size of the network was varied from very low to very high. The cache was then distributed throughout the network using the cache distribution algorithm described in Section 3.

5. Performance comparison

In this section, we present the simulation results of the two caching schemes (see Section 2) with and without cache distribution. When the cache is not distributed, all the nodes, i.e. proxy and caching servers, have the same cache size. The simulations were run on the two network topologies presented in Section 4.4. To keep the simulation time reasonable, a total of 400 objects were served by the origin servers and three different request rates — moderate to very high — were used. The selection of cache sizes between 2

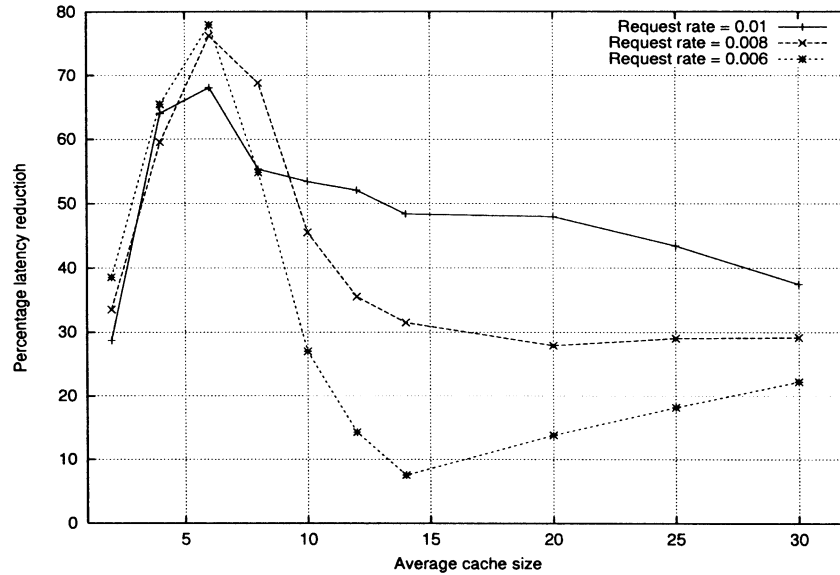


Fig. 8. Percent reduction in download time as a function of cache size for Topology I using Scheme 1 for three request rates.

(or 0.5%) to 30 (7.5% of all documents served) were chosen to reflect what has been used in other studies [6]. Finally, the link bandwidths to the origin servers were set at 80 kb/s while all others were set at 800 kb/s. These bandwidths were carefully chosen, after some experimentation with the simulator, to ensure the creation of the server-side bottleneck scenario.

5.1. Improvement in latency

Figs. 8 and 9 show the percentage reduction in latency for Topology I using a Harvest-type data diffusion algorithm (Scheme 1) and a demand-driven data diffusion algorithm (Scheme 2), respectively. Figs. 10 and 11 the same for

Topology II. The request rates were 0.01, 0.008, and 0.006 requests/sec for unpopular objects. It may be recalled that the request rates for popular objects were 81 times higher than that for unpopular objects.

As all the figures show, distributing the cache reduces the latency significantly in both topologies, using both data diffusion algorithms, over all the request rates. In particular, the highest gains are achieved when the average cache size is up to 2% of the number of documents, as observed in all the figures. This observation is of particular importance since this scenario corresponds to the network being considerably limited in terms of storage space, which is the situation in the real world. Furthermore, all the plots exhibit the same pattern when considering a specific request rate. The

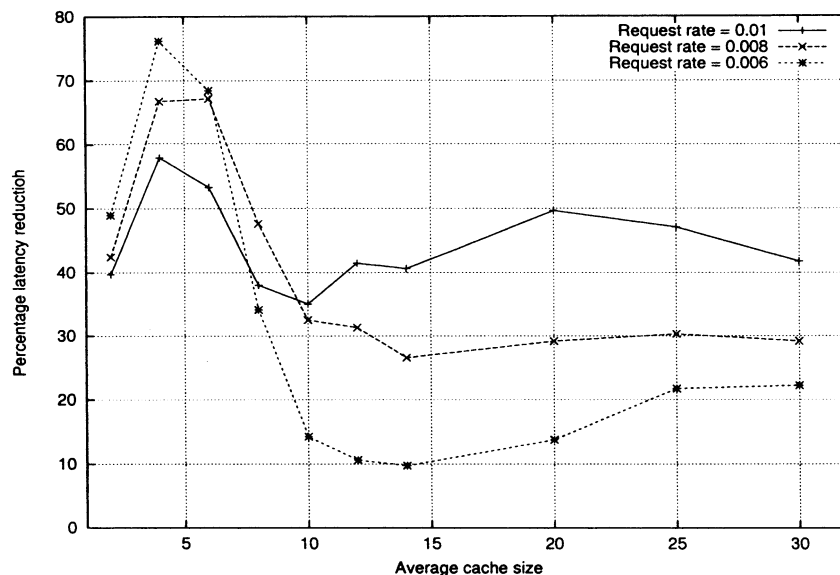


Fig. 9. Percent reduction in download time as a function of cache size for Topology I using Scheme 2 for three request rates.

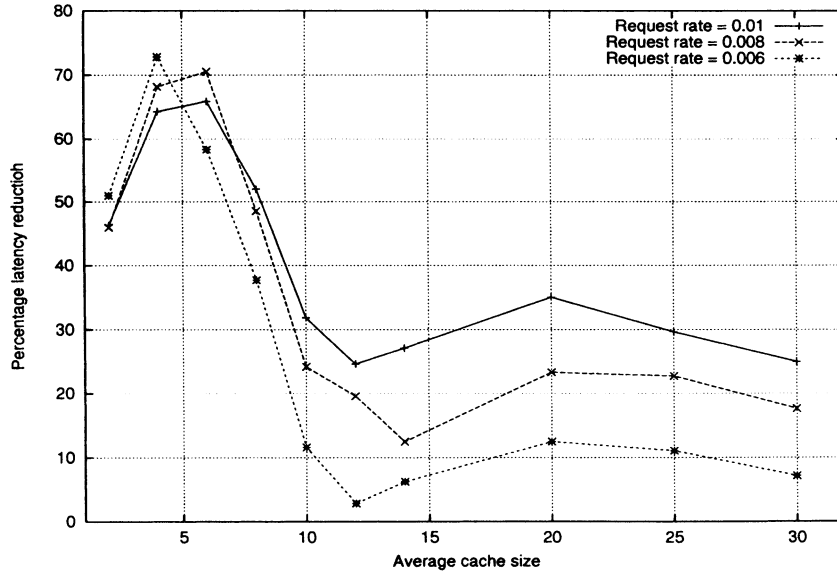


Fig. 10. Percent reduction in download time as a function of cache size for Topology II using Scheme 1 for three request rates.

highest request rate (0.01) shows the best overall improvement in latency. This again is of particular importance since this corresponds to very high network traffic, which is the situation under which better caching schemes are of greatest importance. In a network with low congestion, better caching schemes will naturally benefit less because the lower traffic gives way to lower latency regardless of the caching scheme. This trend is observed in all the figures. As the request rate decreases, the overall gains in latency also decrease.

6. Conclusions

In this work, we looked at a specific facet of caching as it

applies to the Web. Specifically, we looked at how to improve document download time, by distributing a fixed amount of total storage in a mesh of caching servers. We studied the effect of our cache distribution by simulating two data diffusion schemes for hierarchical caches. The intuition behind this approach is to give more storage to the caching nodes in the network which experience more traffic, in the hopes that this will reduce the overall latency of the network. A heuristic was developed to estimate traffic at each cache of a network. From this traffic estimation, each cache then receives a corresponding percentage of the total storage capacity of the network.

The results show a significant improvement in overall network latency after application of this cache distribution algorithm, specifically at very modest storage capacities.

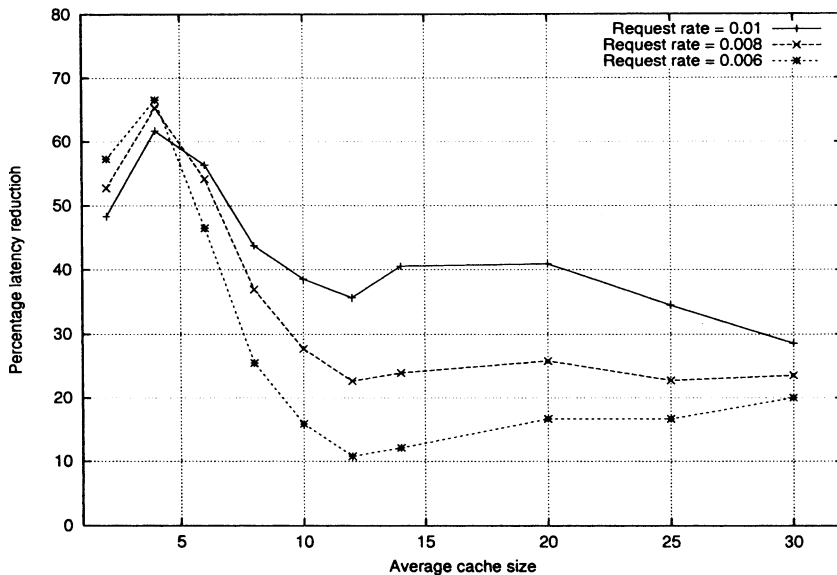


Fig. 11. Percent reduction in download time as a function of cache size for Topology II using Scheme 2 for three request rates.

This observation is of particular importance since these modest storage capacities correspond to practical, real world cache ranges. We compared the performance of this algorithm versus a caching scheme in which all nodes of a network receive an equal amount of storage. The cache distribution algorithm by no means produces an optimal distribution of the cache, yet the results indicate that clever cache placement is an area of caching which can significantly improve performance of the network.

A recent study has found a similar problem, called the cache location problem, to be NP-hard for general network [27]. The results directly do not apply to the hierarchical caching networks we have studied. Thus, an open problem is finding a polynomial-time optimal algorithm for cache distribution, if it exists; or proving that our problem is also NP-hard. Another problem is developing an analytical method for evaluation of the proposed heuristic. A desirable extension of the work reported here is to implement it over a real-world mesh of caching servers, and then study the performance of our algorithm. However, this would require collaboration with a major Internet Service Provider (ISP). We plan to explore the possibility of such a study in the near future. In this study, we not only plan to measure percentile improvement of average latency, but also the actual latency in seconds. The objective is to find the threshold value of latency that makes a perceptual difference to the end users.

References

- [1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, E.A. Fox, Caching proxies: limitations and potentials, Proceedings Fourth International WWW Conference, Boston, December 1995.
- [2] R. Alonso, M. Blaze, Dynamic hierarchical caching for large-scale distributed file systems, Proceedings of the Twelfth International Conference on Distributed Computing Systems, June 1992.
- [3] M.F. Arlitt, C.L. Williamson, Web server workload characterization: the search for invariants, 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Philadelphia, PA, USA, 24(1) (1996) 126–37.
- [4] A. Bestavros, Demand-based document dissemination to reduce traffic and balance load in distributed information systems, Proceedings of the 1995 Seventh IEEE Symposium on Parallel and Distributed Processing, San Antonio, TX, October 1995.
- [5] A. Bestavros, R.L. Carter, M.E. Crovella, Application-Level Document Caching in the Internet, Technical Report BU-CS-95-002, Boston U., CS Department, Boston, MA 02215, March 1995.
- [6] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and zipf-like distributions: evidence and implications. Proceedings of IEEE INFOCOM 99, 1999.
- [8] C.M. Bowman, P.B. Danzig, D.R. Hardy, U. Manber, M.F. Schwartz, Harvest: A scalable, customizable discovery and access system, Technical Report CU-CS-732-94, University of Colorado, Boulder, 1994.
- [9] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, K. Worrel, A hierarchical internet object cache, Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, Jan 1996.
- [10] P.B. Danzig, M.F. Schwartz, R.S. Hall, A case for caching file objects inside internetworks, Proceedings ACM SIGCOMM 92 Conference, August 1992, pp. 281–292.
- [11] B.D. Davidson, A survey of proxy cache evaluation techniques, Fourth International Caching Workshop, San Jose, CA, 1999.
- [12] S. Gadde, J. Chase, M. Rabinovich, A taste of crispy squid, Proceedings of the Workshop on Internet Service Performance (WISP'98), June 1998.
- [13] J. Gwertzman, M. Seltzer, The case for geographical push-caching, Proceedings of the 1995 Workshop on Hot Operating systems, 1995.
- [14] The Harvest Home Page URL: <http://harvest.transarc.com>.
- [15] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy, Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, STOC 97, El Paso, TX, 1997.
- [16] T.P. Kelly, Y.M. Chan, S. Jamin, J.K. MacKie-Mason, Biased replacement policies for web caches: differential quality-of-service and aggregate user value, Fourth International Web Caching Workshop, San Diego, CA, March 31–April 2, 1999.
- [17] The LEDA Home Page URL: <http://www.mpi-sb.pg.de/LEDA/leda.html>.
- [18] A. Luotonen, K. Altis, World-wide web proxies, Computer Networks and ISDN Systems 27 (1994).
- [19] A. Luotonen, Web Proxy Servers, Prentice Hall, New York, 1997.
- [20] The Mapnet Web Page URL: <http://www.mapnet.com>.
- [21] R. Mukhopadhyay, A New Demand-driven Data Diffusion Algorithm for Hierarchical Caching, Master's Thesis, Department of Computer Science, University of California at Davis, Davis, CA 95616, June 1999.
- [22] National Laboratory for Applied Network Research, Anonymized access logs, <ftp://ftp.ircache.net/Traces>.
- [23] P. Rodriguez, C. Spanner, E.W. Biersack, Web caching architectures: Hierarchical and distributed caching, Proceedings of the Fourth International Web Caching Workshop, San Diego, April 1999.
- [24] RFC2186,2187, URL: <http://ircache.nlanr.net/Cache/reading.html>.
- [25] R. Tewari, M. Dahlin, H. Vin, J. Kay, Design Considerations for Distributed Caching on the Internet, The 19th IEEE International Conference on Distributed Computing Systems (ICDCS), May 1999.
- [27] P. Krishnan, D. Raz, Y. Shavitt, The cache location problem, IEEE/ACM Transactions on Networking 8 (5) (2000) 568–581.