Self-Organizing Wide-Area Network Caches*

Samrat Bhattacharjee, Kenneth L. Calvert, Ellen W. Zegura
Networking and Telecommunications Group, College of Computing
Georgia Institute of Technology, Atlanta, GA 30332
{bobby,calvert,ewz}@cc.gatech.edu

Abstract

A substantial fraction of all network traffic today comes from applications in which clients retrieve objects from servers. The caching of objects in locations "close" to clients is an important technique for reducing both network traffic and response time for such applications. In this paper we consider the benefits of associating caches with switching nodes throughout the network, rather than in a few locations. We also consider the use of various self-organizing or active cache management strategies for organizing cache content. We evaluate caching techniques using both simulation and a general analytic model for network caching. Our results indicate that in-network caching can make effective use of cache space, and in many cases self-organizing caching schemes yield better average round-trip latencies than traditional approaches, using much smaller per-node caches.

1 Introduction

A substantial fraction of all network traffic today comes from applications in which clients retrieve objects from servers (e.g. the World-Wide Web). The *caching* of objects in locations "close" to clients is an important technique for reducing both network traffic and response time for such applications. Studies have shown [1, 2] that caching can substantially improve performance.

In this paper we consider the benefits of associating caches with switching nodes *throughout* the network, rather than in a few hand-chosen locations. We also consider the use of various *self-organizing* or *active* cache management strategies, in which nodes make globally consistent decisions about whether or not to cache an item in order to reduce overall latency.

We develop the self-organizing algorithms assuming an "active" network platform [3, 10], in which the routing nodes can manipulate packets and even execute code on behalf of users. The per-packet processing required by the self-organizing caching algorithms is a natural application for active networks. Throughout this paper, we use both terms ("self-organizing" and "active") to refer to cache management and organization techniques that make use of active network nodes. The techniques, however, are independent of any particular active network platform.

Our studies involve both analysis and simulation, and use object request distributions that are consistent with studies of actual Web traffic [4]. Our results indicate that in-network caching can make effective use of cache space. In particular, active caching yields round-trip latencies that are about as good, and in certain cases better than more traditional approaches, while requiring much smaller caches per node.

The remainder of this paper is organized as follows. In Section 2 we provide background on wide-area caching and outline our assumptions about the application and network. We also describe related prior work in caching. In Section 3 we describe caching policies that take advantage of the ability to store and process information as packets pass through the nodes of the active network along with the more traditional methods to which we compare them. In Section 4, we describe our simulation study, including the parameters of our model, our simulation methodology and results. In Section 6, we develop an analytic model for network caching, and use it to validate the simulation model. Section 7 concludes the paper.

2 The Problem and Related Work

We assume an application in which clients request objects from servers located throughout the network. Each object is assumed to have a globally unique identifier (e.g. a hash function of a URL or message body), and to fit in a single "message". Each transaction is initiated by a *request message* sent from a client toward a server, containing the ID of one requested object. The request message travels through the network until it reaches a node where the requested object is stored, which may be a cache or the server itself. A *response message* containing the object then travels from the server (or cache) back to the originating client, completing the transaction. For simplicity, we assume messages are never lost. This simple model ignores the problem of end-to-end transfer of objects too large to fit in a single message. However, it is adequate for our purposes, namely comparison of active caching methods with traditional ones.

We assume that a fraction of objects are "popular",

*This work was supported by DARPA. The work of Ellen W. Zegura was supported in part by NSF Careers Award MIP-9502669.

and that nodes can determine whether any particular object is popular; this could be implemented by having each server mark response messages containing popular objects. We measure cache scheme performance by the average number of hops traveled by request and response messages to satisfy a client request. This metric accounts for two beneficial aspects of caching: reduction in network traffic, and reduction of the latency perceived by the requesting user.

The Harvest Cache [5] project, initiated at the University of Colorado, is probably the largest wide area cache implementation in the Internet. Harvest caches are usually arranged in a hierarchy, and Web clients are manually configured to access a particular cache in the hierarchy designated as the client's proxy. If the client request can be satisfied at the initial proxy cache, it is served by the proxy. In case of a miss, the parent and sibling caches in the hierarchy are contacted using the Internet Cache Protocol (ICP [6] discussed below). If the requested object is not available at the sibling caches or at the parent, then the client's proxy cache generates another HTTP query with its parent cache as the target using the HTTP proxy protocol or ICP. This process is recursively repeated until the request is served or the object is retrieved from its origin (by the root of the hierarchy). Once the item is retrieved, it is cached at each node on its way down to the leaf. This scheme is effective in reducing the wide-area bandwidth requirements, and in accessing "hot-spots" only once per hierarchy, as the "hot" item is then cached within the hierarchy, and subsequent requests to the same object can be satisfied by a Harvest cache hit.

The geographical push caching scheme [7] of Seltzer and Gwertzman at Harvard uses a friends-of-friends algorithm in which servers selectively push their content to *friend* caches that reside in client domains. A similar push caching approach in which servers disseminate popular pages has been proposed by Oritz and German at the University of Waterloo [8]. In both schemes, the server initiates the caching of an object (that it deems popular) at a remote site.

The Internet Cache Protocol (ICP) [6] defined by the Network Working Group of the IETF is a message format used for communicating among Web caches. Harvest and its successor (Squid) both use ICP to exchange information about objects stored at neighboring caches.

3 Network Caching Schemes

Traditional approaches to network caching place large caches at specific points in the network. In contrast, we consider networks in which relatively small caches are placed at every node. As a response message moves through the network, each node decides whether or not to store an object. Effective use of a large number of small caches is a non-trivial problem: unless they are effectively organized, only a small number of unique items will be cached (as the caches are much smaller), but these cached items will be replicated at many locations throughout the network. Thus, accesses to the few cached objects will exhibit low latencies, but overall average latency will not decrease appreciably. However, if objects are cached too sparsely, then the latency again does not decrease and caching provides little benefit.

In this section, we describe caching policies governing where an object is cached, along with mechanisms that allow the policies to be implemented locally at each node.

3.1 Self-Organizing Caches

Our self-organizing schemes are described here as if every node of the network caches objects, but all that is required is that caches be uniformly distributed. Thus, these schemes obviate the need to decide *where* to place caches—which can be a critical decision for traditional caching mechanisms.

In what follows, the network is considered to be a collection of *domains*, each of which is represented as a graph of switching nodes connected by links. Domains are of two types, *transit*, which (as their name implies) carry transit traffic, and *stub*, through which only packets addressed to or from some node in the domain are carried. The graph models used in our simulations are constructed using the GT-ITM Internet topology modeling package [9]. These graph models ensure that the paths along which packets travel in the simulations have the characteristics that (i) the path connecting two nodes in the same domain stays entirely within that domain, and (ii) the shortest path connecting node u in stub domain U to node v in another stub domain V goes from U through one or more transit domains to V, and does not pass through any other stub domains. Note that "nodes" in these models represent routers, and end systems are not explicitly modeled. Thus references to "servers" or "server nodes" should be interpreted as meaning nodes to which one or more servers are connected.

Our goal is to have nodes make local decisions about which objects they place in their (small) caches, in such a way that resources are used effectively overall. In particular, we wish to avoid having the same (few) objects cached at most of the nodes of the network. We describe two related approaches.

Modulo Caching. To ensure that cached objects are distributed through the network, we introduce a distance measure called cache *radius*, measured in transmission hops. The caching policy uses the radius measure as follows: on the path from the server (or cache) to the requesting client, an item is cached at nodes that are the cache radius apart. Thus, an object ends up being distributed in concentric "rings" centered on the server where it resides; the rings are separated by a number of hops equal to the cache radius. The radius is a parameter of the policy; it might be set globally, on a per-object basis, or even locally in

different parts of the network. (Our simulation results assume a common global cache radius, equal to 3.)

The mechanism used to implement this policy locally at each node is a simple modulus. The response message contains a hop count that is initially set to the object identifier modulo the radius; the count is incremented by each node through which the packet passes. When the incremented count modulo the cache radius equals 0, the object is cached at that node.

Our results show that the performance of this *modulo caching* mechanism is extremely robust across a wide range of access patterns and client–server distributions.

Lookaround. Network caches store relatively large objects compared to the amount of space required to store the location of an item within the network. For example, an object in a network cache can be several thousand bytes, while its location could be an IP address (4 bytes). This fact can be exploited by having a self-organizing cache dedicate some of its cache space to store *locations* of (nearby) items.

Caching nodes keep a periodically-updated list of items cached at neighbors. Logically, each node's cache is divided into "levels": level zero contains objects cached locally. Level one contains the locations of objects cached at nodes one hop away, level two contains locations of objects cached at nodes two hops away, etc. When a request message is processed, the levels are searched in sequence beginning with 0; if a hit is detected in a nearby cache, the request is simply re-routed to that node (source and destination addresses are not changed). If the information about the neighbor's cache turns out to be incorrect, the neighbor simply forwards the datagram toward the destination. Thus, the mechanism is fail-safe and backward compatible: a mix of active and non-active nodes may exist in the network, and the active cache functions may fail at any time and fall back on the regular forwarding functions. (In our simulations, we constrain the lookaround to nodes in the same domain.)

The number of levels of adjacent caching maintained and checked in this *lookaround* algorithm becomes a parameter of the policy. With this approach, even very small caches can look like "virtual" large caches. We refer to this extension of the modulo caching scheme as **Modulo Caching with Lookaround**.

3.2 Traditional Mechanisms

Our simulations compare the above self-organizing caching schemes to "traditional" caching schemes, in which each cache attempts to store each (popular) object passing through it, without any global coordination beyond the placement of the cache within the network. We consider the following placement strategies:

Cache at Transit Nodes ("Transit-Only"). Transit nodes have to be traversed for every non-local stub domain access; a large fraction of paths in the network have to go through transit routers. This ubiquity of transit nodes in network paths make them prime candidates for caches.

Cache at Stub Nodes Connected to Transit Nodes ("SCT"). Stub nodes connected to transit nodes have to be traversed in order to access the transit network. Thus, these stub nodes form good locations for network caches. The Harvest [5] scheme recommends that caches be placed at such locations.

Cache at Every Node ("No AN"). We also consider an approach in which caches are located in every node (like the self-organizing schemes), but without any coordinating mechanisms enabled. This case corresponds to a "null" active function in an active network. We refer to it as **"No AN"**.

3.3 Discussion

Many traditional caching schemes do *not* explicitly minimize latency, but rather minimize bandwidth consumption on transit links. The classic example is the Harvest cache, in which several different queries may be initiated within caches in a domain before either the request is satisfied, or the query is sent to the original server. In many cases, this results in higher latency, but no bandwidth consumption in the transit network. In contrast, our schemes are designed to reduce latency—the bandwidth savings are secondary to reducing pure latency. Both traditional and the self-organizing schemes can be implemented in a network. For example, within a stub network, caches could be set up in a hierarchy to save bandwidth, while in the wide area, the self-organizing schemes could be used to reduce latency.

4 Simulation Methodology

We compare the performance of traditional and self-organizing techniques for wide-area caching using a locally developed discrete event network simulator called AN-Sim. AN-Sim simulates an active network as described in [10], and allows for realistic models of the network topology. This section discusses the various parameters of our simulations.

Network Topologies. We simulated many networks that differ in number of nodes, diameter, average node degree, ratio of transit nodes to stub nodes, etc. Table 1 summarizes the properties of the topologies for which we present results. All of these graphs have 1500 nodes, of which 60 are transit nodes. In Table 1, SN, SD, TN and TD stand for "stub node", "stub domain", "transit node" and "transit domain", respectively. Thus the `Base` graphs average four stub domains per transit node, and six stub nodes per stub domain.

Graph	Avg. Deg.	Avg. SD/TN	Avg. SN/SD
`Base`	3.71	4	6
`More Stub Domains`	3.14	6	4
`Less Stub Domains`	4.11	2	12
`Higher Degree`	4.53	4	6
`Lower Degree`	3.06	4	6

Table 1: Simulated Topologies

Servers and Objects. Each stub node is assumed to connect to one server, thus each graph has 1440 servers. A subset of the servers, chosen uniformly at random, are designated to be *popular servers.* The number of popular servers is nominally 300. (One experiment explores the effet of varying the number of popular servers.) There are 4 billion (2^{32}) unique objects in each simulation, the vast majority of which are not accessed. Each object is associated with a particular server, thus each server's content is unique. A subset of objects at each popular server is designated to be popular. (Unpopular servers have only unpopular items.) The number of popular objects is fixed at 48 per popular server (for a nominal total number of popular objects of 14400.) To decide upon a query, first a client is chosen. The client picks a server, then picks an object at the server. The access patterns governing the choice of client, server and object at a server are described below.

Access Patterns Every stub node is assumed to connect to one client. We simulate several different access patterns. In the *uniform access pattern*, a client is chosen at random. Then, a server is chosen using a Zipf distribution: the popular servers are chosen with a probability $1 - \epsilon$. All other client requests go to an unpopular server, chosen at random. If a popular server is selected, then a popular object at that server is selected 95% of the time. The remaining 5% of the accesses to a popular server select from the unpopular objects. If an unpopular server is selected, then a random object is selected. (Recall that all objects at an unpopular server are unpopular.)

We also model a *correlated access pattern*, in which accesses are not independent, but rather may involve the same client and server pairs. In the correlated access pattern, there are two types of accesses: initial accesses and dependent accesses. An initial access is generated using the uniform access pattern described above. Initial accesses are separated in simulation time using an initial access interarrival time distribution. With a fixed correlation probability, an initial access is the "anchor" for a train of future dependent accesses that are interspersed in time with the initial access generation process. A train has an average length of 16 accesses; the time of each future dependent access is determined by an offset distribution from the time of the anchor access. A dependent access has the same client and server as the anchor access. With a fixed probability, the item selected in a dependent access is the same as the previous item in the train. Otherwise, the item is selected according to the uniform access pattern described above.

In the third access pattern simulated, a set of servers is associated with each node. A fraction of all accesses from the node is then directed towards servers in this (per node) set. This models the case in which server popularity is not uniform throughout the network.

These access patterns are consistent with what is known about access to objects in the World Wide Web. Among the "invariants" found by Arlitt and Williamson in their study of server logs [4] was that the number of distinct objects requested is between .3% and 2.1% of the total number of requests. They also found that the vast majority of all requests are for objects that would fit inside a single IP datagram and that at least 75% of all requests to each server they studied are nonlocal.

Cache Sizes In our simulations, the size of the cache at each caching node is proportional to the node degree. This is equivalent to having a fixed amount of caching per edge (interface) incident upon the node. For each caching simulation, the total number of cache slots in the network is held constant. Thus, in cases where the total number of caching nodes is small (as in Transit-Only caching, or caching only in stub nodes connected to transit nodes), the caches are relatively large (compared to cache size when each node has a cache).

Limitations of Experiments. Our experimental setup has some known limitations. The simulator does not enforce link rates, and thus lost datagrams due to full buffers are not represented in the results. The lookaround algorithm does not generate actual packet traffic in the simulation. Also, the space required for storing the list of cached items at neighbors is not accounted for. We do not believe that any of these is particularly serious.

Performance Metrics We use the following metrics to evaluate the performance of network caches.

- **Round trip length (RTL).** We measure the number of hops traversed by the packets involved in a transaction. This is perhaps the simplest and most "true" measure of network cache performance.

- **Fraction of queries that generate cache hits** After an initial startup period, the cache performance stabilizes. We measure the fraction of queries that are serviced by cache hits. Note that

queries served by caches not only reduce access latencies and conserve bandwidth, but also reduce server load.

5 Simulation Results

We have simulated cache performance for the topologies specified in Table 1 across a wide range of cache sizes, server distributions, and access parameters. We first summarize the general results, and then describe how the results are affected by variations in the certain simulation parameters.

5.1 Summary

For non-correlated accesses, the Transit-Only caching scheme performs best (in terms of average RTL). One explanation for this is that for the base graph, there are 4.92 stub nodes, and 3.32 transit nodes on the average client-server path. In this graph, there are 60 transit nodes, and 1440 stub nodes. Thus a much larger fraction of all transit nodes (and thus cache slots) are encountered on each access in the Transit-Only scheme than in the other schemes. Also, for Transit-Only caching, in the base graph, the average cache is 25 times larger than the average cache under active caching. Thus, the large caches lead to large gains due to multiplexing at the transit nodes—especially if the accesses are not correlated.

The active schemes are always within 10% of the Transit-Only RTL, even for uncorrelated accesses. However, Transit-Only is not able to adapt to correlated accesses from particular stubs. Thus the active mechanisms generally outperform all other methods (including Transit-Only caching) for correlated accesses.

In the following subsections we present details of a small cross section of our experiments. Except where otherwise noted, our results are for the base graph topology, and the LRU cache replacement policy.

5.2 Variation in Cache Size

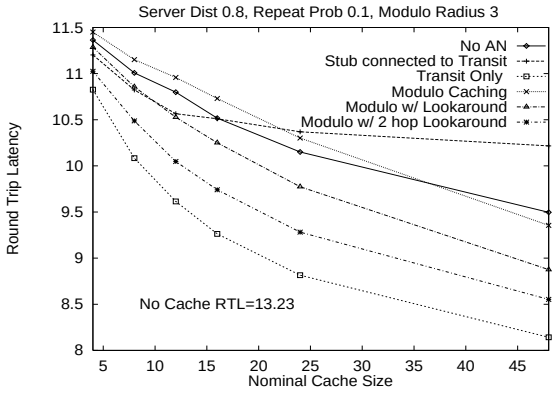

Figure 1: Latency with Low Access Correlation

We varied the nominal cache size from 4 to 48 cache slots per interface. The Modulo and No-AN methods use nominal cache size, as these methods cache at all nodes. The corresponding numbers for Transit-Only caching are 33 to 397 slots per interface, and for Stubs-connected-to-Transit (SCT) caching, 15 to 188 slots/interface. All of the caching mechanisms, except SCT, show a smooth decrease in number of hops traversed per round trip as the cache size is increased.

In Figure 1, the probability of repeating an accesses within a set of correlated accesses is 0.1. The Modulo cache radius is fixed at three. The Lookaround schemes perform better than all but the Transit-Only caching scheme, and the performance of the two-level Lookaround schemeis within 10% of the Transit-Only scheme in all cases. It should be noted that the number of cache slots per interface for the two-level Lookaround scheme is an order of magnitude smaller than for the Transit-Only scheme. Also, the average degree of the transit nodes is much greater than the average degree of the graph. Thus the transit node caches are 25 times larger than the Modulo caches. Comparatively, the SCT caches average 4.25 times larger than the Modulo caches.

As accesses become more correlated, as shown in Figure 2 (repeat probability 0.5), the Modulo with Lookaround scheme outperform all others. Also significant in Figures 1 and 2 is the behavior of the caches in the SCT scheme. Their performance improvements are negligible beyond 12 cache slots per interface, and as such this method does not scale well with increase in cache size.

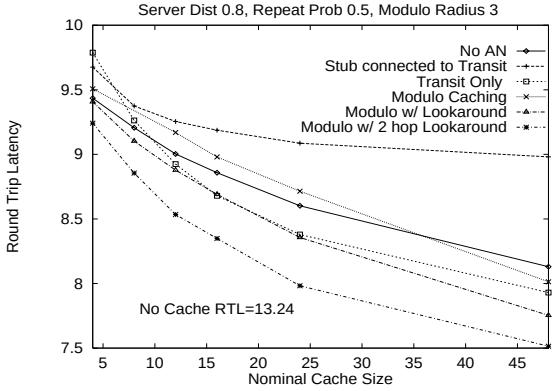

Figure 2: Latency with High Access Correlation

For the same parameters as in Figure 2, Figure 3 shows the fraction of queries that generate cache hits. Once again, all the methods except SCT show improvement with increase in cache size. The SCT method actually results in a proportionately large fraction of hits—but the large number of hops in the round trip suggests that more hits occur in the stub node to which the *server* is connected, and not the client. This is not unexpected—the *gateway* stub node connected to the transit domain for a busy server will experience a lot of traffic due to the busy server, and as such, will cache a large part of that data as well.

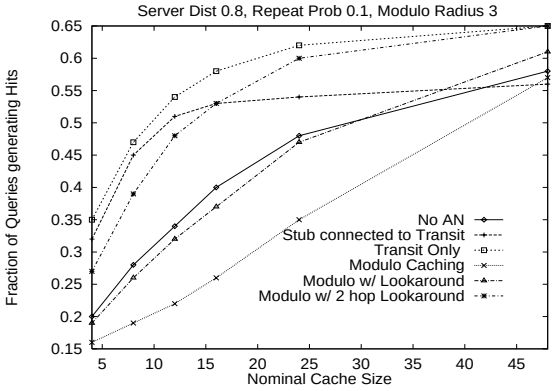

Figure 3: Cache Hits with High Access Correlation

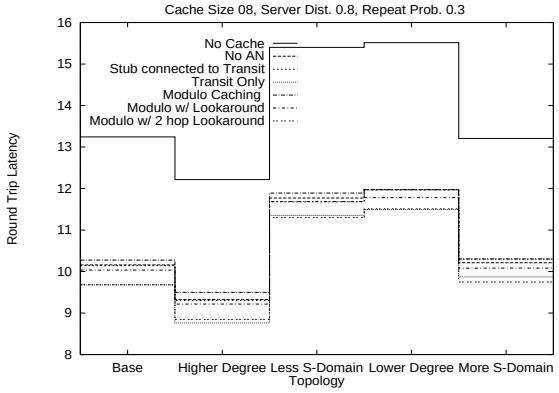

Figure 5: Topology Effect under Moderate Access Correlation

5.3 Variation in Server Distribution

We have used a Zipf distribution to model server popularity (i.e. a fraction $1-\epsilon$ of the accesses are to the fraction ϵ of the nodes). However, it is not clear exactly what fraction of the nodes should be considered to be *servers*. In this experiment (Figure 4), we vary the fraction of all nodes that are servers from 0.01 to 0.5. Even when a large fraction of nodes are servers, the cache performances are not significantly affected. Thus, wide-area caching seems robust in face of widely varying server locations and distributions. It is interesting to note that round trip latencies for Transit-Only cache schemes do not improve much when the server distributions are extremely skewed, e.g. when less than 10% of the nodes are servers. The other schemes improve as the number of popular objects (which is a multiple of the number of servers) decrease, but in case of Transit-Only caches, even if all the objects are cached everywhere, the query has to reach the transit nodes before it is serviced.

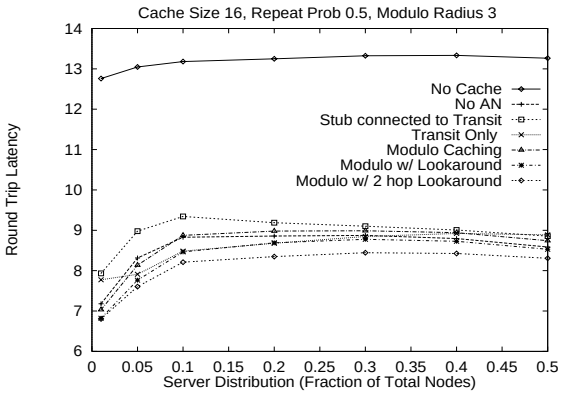

Figure 4: Latency with Varying Server Distributions

5.4 Variation in Topology

In Figure 5 we consider the effectiveness of the various cache policies as the underlying topologies are varied. The round trip latency is inversely proportional to the graph degree, and directly proportional to the average size of the stub domains. The two level lookaround algorithm has similar performance to the Transit-Only cache method for the base graph for this moderately correlated access pattern. As topologies are varied, the performances are comparable—with the active methods performing better as the number of stub domains increase. Although it is not presented graphically here, our experiments showed that as repeat probabilities increase, the AN mechanisms perform better than all other mechanisms in all topologies.

5.5 Spatial Access Patterns

In these experiments, we consider a different access pattern. Associated with each stub node is a set of *preferred* servers. A fraction of queries generated by the node is always directed to the preferred set. Figure 6 shows the round trip latencies as the number of servers in the preferred set is varied from 2 to 12. The probability of accessing a server in the preferred set is 0.25, and the probability of a repeat access was 0.3. The two level Lookaround and the Transit-Only cache schemes perform the best, with the Lookaround schemes being better if the number of preferred servers is small. In fact, the round trip latency for Transit-Only caching is nearly constant for any number of servers in the preferred groups since the accesses are nearly uniform at the transit domains. The performance of the Lookaround schemes deteriorate slowly as the number of servers increase because the locality in accesses are lost.

We also considered spatial access patterns in cases when the preferred set is common to the entire stub domain. As expected, the round trip latencies increase with increase in the number of preferred servers. Also, caching at stub nodes connected to transit node scheme works quite well: it is within 1 hop per round trip of optimal in all cases.

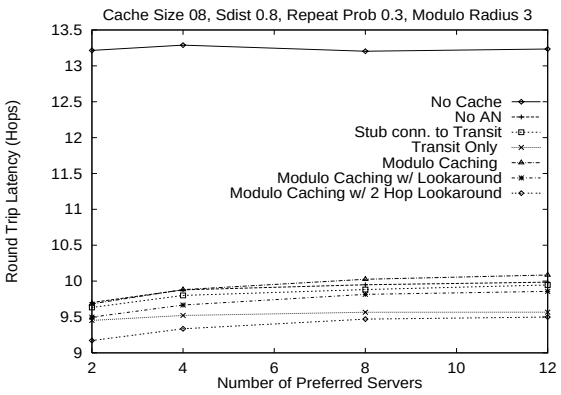

Figure 6: Preferred Node Servers

6 Analysis of Cache Performance

In this section, we present a simple analytic model for the expected one-way latency in accessing an item in a network that has intra-network caching. The basic model makes the following assumptions: (1) the set of items is partitioned into "popular" items and "unpopular" items, and (2) only popular items are cached, (3) every cache is full, and (4) the cached items encountered from one client access to the next are random and independent. This independence assumption keeps the model tractable, though it clearly departs from reality.

6.1 Network Caching Performance Model

Consider an access from a client to an item. If the item is unpopular, the one-way latency is simply the average client-server path length. If the item is popular, it may be encountered at a cache in the network. Let L denote the average one-way latency, measured in hop count, when a client accesses an item. Let q denote the probability that the item is popular. Let d denote the average length of a client-server path and c denote the average length of a path from a client to a popular item (possibly encountered at a cache). Then:

$$L = (1-q)d + qc$$

Let N_s denote the average path length from a stub node to the transit node for a stub domain. Let N_t denote the average path length across transit domains. Then $d = 2N_s + N_t$, since a client-server path traverses two stub domains (client and server domains) and a path through the transit domains.

We focus the rest of the discussion on deriving an expression for c. Let H_i denote the event "a hit occurred at distance i from the client"; let M_i denote the event "a miss occurred at *every* distance up to and including i from the client". Then

$$\begin{aligned} c &= \sum_{i=1}^{d} iPr\{H_i \cap M_{i-1}\} \\ &= \sum_{i=1}^{d} iPr\{H_i \mid M_{i-1}\}Pr\{M_{i-1}\} \end{aligned}$$

The probabilities in this expression are affected by details of the locations of the caches, the cache policy and the client access pattern. We develop variations on the basic model to handle different instances of these details.

Let us assume the best possible conditions regarding duplication in items stored in caches along the path. Namely, we assume that the items stored in different caches are *different*. Though this is difficult to achieve in a distributed environment, it provides a lower bound on average round-trip length. We also assume that each cache is equally likely to store any popular item.

Since all items are different, the probability of a hit when accessing a popular item and examining S cache locations (potentially distributed over multiple nodes) is S/P. If we know that a miss has occurred at locations at every distance up to and including $i-1$, then we know that the item we are looking for is in a reduced set of popular items, namely reduced by the number of cache locations we have already checked. Let T_{i-1} denote the total number of items checked at distances up to and including $i-1$. Then[1] $T_{i-1} = \sum_{j=1}^{i-1} S_j$ and:

$$\begin{aligned} c &= \sum_{i=1}^{d} iPr\{H_i \mid M_{i-1}\}Pr\{M_{i-1}\} \\ &= d\left(1 - \frac{T_{d-1}}{P}\right) + \sum_{i=1}^{d-1} i\frac{S_i}{P - T_{i-1}}\left(1 - \frac{T_{i-1}}{P}\right) \\ &= d\left(1 - \frac{T_{d-1}}{P}\right) + \sum_{i} i\frac{S_i}{P} \end{aligned}$$

The first term is the boundary condition for the request reaching the server.

6.2 Optimal cache partitioning

As discussed in Section 3, it may be beneficial to use some of the network cache's memory to store locations of *nearby* objects. In this section, we analyze the optimal partition of the network caches into object and location caches.

Assume that each object is the same size, and let the maximum number of objects in each cache be S. Let the ratio of the size of an object and the size of a location be α, i.e. α locations can be stored in the cache instead of one object. Let the average one-way length of server–client paths be d. We assume paths are symmetric, and as such, the average client–server round trip latency is $2d$. Starting from a given cache, as the level of lookaround is increased, the number of

[1] We assume that $T_i < P$ for all i. If this assumption does not hold, then at some point in the path, all popular items will have been encountered, and the hit probability will go to 1.0.

S	–	Max. number of objects in cache
α	–	Number of object locations that can be
	–	stored instead of one object
δ	–	Average degree of the graph
d	–	Average one-way length of server–client path
η	–	Frac. of cache used for storing objects

Table 2: Definitions and Notation

caches searched increases. For most graphs, this increase is exponential. As a simplification, we assume that the number of caches searched at a lookaround level of x is exponential in $\delta - 1$, where δ is the average degree of the topology graph[2]. Let η denote the fraction of the cache memory is used for caching objects; the rest is used for caching location information.

We now derive the value of fraction η such that the round trip latency is minimized. The precise fraction that minimizes the latency depends on the size of the network, the size of the caches, and the number of locations that can be stored instead of each object. Using the basic caching model developed before, the expected latency is $c = \sum_i iPr\{H_i \mid M_{i-1}\}Pr\{M_{i-1}\}$.

Using our model of memory usage, the expected latency can be calculated as follows. For the local objects (in the object cache), the latency after traveling i hops is i. If at node i, there is a hit in the location cache, then a detour has to be made. We can approximate the one-way length of the detour as follows.

We know that if η fraction of the memory is used for caching objects, then memory for $(1-\eta)S$ objects is used for storing locations. This is equal to $(1-\eta)S\alpha$ locations. As each cache stores ηS objects, the $(1-\eta)S\alpha$ locations correspond to complete location information from $\frac{(1-\eta)\alpha}{\eta}$ nodes. Under the assumption that the number of nodes increases exponentially as the level of lookaround increases, $\frac{(1-\eta)\alpha}{\eta}$ nodes imply a maximum detour of length $\log_{\delta-1}\frac{(1-\eta)\alpha}{\eta}$. Let f be the expected value of the length of this detour, and let $F = \log_{\delta-1}\frac{(1-\eta)\alpha}{\eta}$ be the maximum length of the detour. At each level x of lookaround, $(\delta-1)^x\eta S$ items are cached. Thus, the expected one way detour length is:

$$f = \sum_{x=1}^{F} x\frac{(\delta-1)^x\eta S}{(1-\eta)S\alpha}$$

If we assume that the lookaround memory is arranged such that the content of nearest caches are searched first (i.e. caches at level 1 are searched before caches at level 2, and so on), then we can derive a simpler expression for the expected value of the detour. Under this case, on average, half of the cache will be checked before a hit (if any) occurs in the lookaround cache. Thus, the expected length of a detour in this case is:

$$f = \log_{\delta-1}\frac{(1-\eta)S\alpha}{2\eta S} = \log_{\delta-1}\frac{(1-\eta)\alpha}{2\eta}$$

As previously defined, let P be the total number of popular items, and let T_{i-1} be the total number of items checked till distance $i-1$ from the source of the request. In our case, as the maximum number of objects in a cache assumed to be equal, $S_i = S$ (for all i), and $T_{d-1} = (d-1)S$.

Thus, the expected value of the one way latency is:

$$\begin{aligned} c &= \sum_i^d iPr\{H_i \mid M_{i-1}\}Pr\{M_{i-1}\} \\ &= d\left(1-(d-1)\frac{S}{P}\right) + \sum_i^d i\frac{S_i}{P} \\ &= d\left(1-(d-1)\frac{\eta S+(1-\eta)S\alpha}{P}\right) + \\ &\quad \left(\frac{\eta S}{P}+\frac{(1-\eta)S\alpha}{P}\right)\frac{d(d+1)}{2} + \frac{(1-\eta)S\alpha}{P}f \end{aligned}$$

Minimizing this expression with respect to η gives the optimal amount of cache to be used for storing location information such that the expected latency is minimized. In Figure 7, we compare the result of

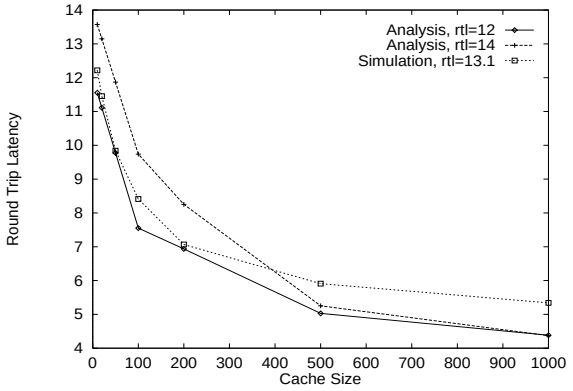

Figure 7: Comparison of analytic and simulation results — Lookaround Caches

the analysis of lookaround caching with a simulation with similar parameters. The base graph (see Table 1) was used in the simulation, with α equal to 50. The lookaround was fixed to 2 levels, and the radius of caching at 3. The two analytic curves were generated by setting the one-way client–server path length to 6 and 7, respectively. This corresponds to the average length of client–server round trip lengths being 12 and 14, respectively. Thus, in general, we would expect the curve generated by simulation to be bounded (on

[2]The exponent is $\delta - 1$ because after the first level of lookaround, at least one node has already been searched at the previous lookaround depth

either side) by the two curves generated by the analysis. For small cache sizes ($S < 500$), this is the case. The refinements to the basic model generate an extremely accurate measure of the cache latency. When the cache sizes are larger, the analysis overestimates the number of *unique* items that are cached.

In Figure 8, we use the expression for expected round trip latency to evaluate the benefits of lookaround caching. The x-axis shows the size of each cache, the y-axis denotes the amount of the cache memory devoted to caching local objects. The round trip latency is shown on the z-axis. We assume that nearer caches are checked earlier to evaluate the length of the detour. In this plot, we set the number of popular objects in the network to 80,000, and again, α is set to 50. The average client–server round trip path length was set to 14.

The plot clearly shows the benefit of lookaround caching — the *valley* in the plot is directly due to the benefits of lookaround caching. In the plot, the expected round trip latency decreases as the amount of memory devoted to the lookaround cache is increased ($.99 < \eta < 0.5$). However, after a point, as more memory is dedicated to the lookaround cache, the number of objects cached in the network decreases. The round trip latency increases sharply ($0 < \eta < 0.3$) as there are not enough objects cached in the network and each lookaround hit causes a large detour. It is interesting to note that at very small values of η, the round trip latency can increase to greater than the average latency without caching—this is again expected, as there are not enough objects cached, and each detour is greater than the remaining path length to the server. Of course, a trivial optimization that forbids detours greater than the remaining path length can easily be incorporated.

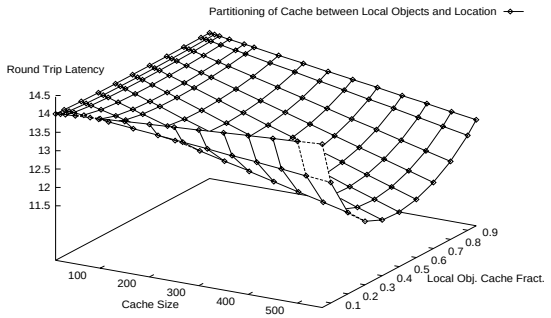

Figure 8: Optimal partitioning of cache space for local objects and location information

7 Concluding Remarks

We have developed and evaluated mechanisms for caching objects within the network. Our active caching mechanisms allow transparent, self-organizing, location of objects where they can be of benefit to reduce access latency. These methods contrast with traditional wide-area network caching that relies on fixed and limited locations for caches.

We have used both simulation and an analytic model to evaluate performance. The simulation uses access patterns that are consistent with studies of Web access; the analytic models are simplified for tractability, though still provide reasonable estimates of access latency. The models could be extended using information about uniqueness of items and/or more complex mathematical modeling to remove simplifying assumptions.

Our results show that active caching is beneficial across a range of network topologies and access patterns, and is especially effective when access patterns exhibit significant locality characteristics.

References

[1] H. Braun and K. Claffy, "Web traffic characterization: An assessment of the impact of caching documents from NCSA's web server," in *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, October 1994.

[2] P. Danzig, M. Schwatrz, and R. Hall, "A case for caching file objects inside internetworks," in *Proceedings of ACM SIGCOMM '93*, September 1993.

[3] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, 1997.

[4] M. Arlitt and C. Williamson, "Web server workload characterization: The search for invariants," in *Proceedings of ACM SIGMETRICS '95*, May 1995.

[5] C. M. Bowman, P. Danzig, D. Hardy, U. Manber, M. Schwartz, and D. Wessels, "Harvest: A scalable, customizable discovery and access system," Tech. Rep. CU-CS-732-94, University of Colorado - Boulder, 1995.

[6] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), version 2," tech. rep., IETF Network Working Group, May 27 May 1997. draft-wessels-icp-v2-03.txt.

[7] J. Gwertzman and M. Seltzer, "The case for geographical push caching," in *Hot Operating Systems*, 1995.

[8] A. Lopez-Ortiz and D. M. German, "A multicollaborative push-caching http protocol for the WWW," in *World Wide Web Conference (WWW5)*, 1995. Poster Presentation.

[9] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, June 1997.

[10] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "An Architecture for Active Networking," in *Proceedings of High Performance Networking 97*, 1997.