Domain Caching: Building Web Services for Live Events

January 25, 2001

Abstract

Web servers providing live coverage of events are hard to design due to a high degree of variation in the load, variation in geographical distribution of users and the frequency with which content is updated. In this paper, we propose *domain caching*, a technique that dynamically distributes the load on the server by instructing selected clients (*domain caches*) to cache server content and service requests from nearby clients.

We evaluate domain caching using a trace driven simulation study of the 1998 Worldcup Soccer logs. Our simulations indicate that domain caching reduces load on the server by as much as 38% while using a few hundred kilobytes of storage at domain caches. The traces also reveal situations where using domain caching instead of content mirroring reduces network traffic from the server.

We built a prototype domain cache to demonstrate the feasibility of some of our ideas. Using this prototype, we show that accessing content from a nearby domain cache is faster than accessing the same content from the origin server. Using our simulator to estimate the load on a domain cache, we find that our prototype is able to handle the load expected to be handled by most domain caches. Accessing the prototype from several clients located geographically near the domain cache suggests that geographical proximity could be used to identify nearby clients.

1 Introduction

An increasing number of events are being "watched live" on the Internet. Such events include the recent US Presidential elections, the 2000 Olympic Games, 1998 Soccer World Cup, 2000 Wimbledon Tennis Championships, and daily stock trading on exchanges such as NASDAQ. The dynamic nature of these events requires the content of web sites to be updated frequently and that clients make frequent calls to fetch new information. The client population for these web sites is also dynamic, characterized by large peaks in access rates during popular events. Finally, there are strong geographic variations in access rates based on the popularity of local sports teams, stocks or other interests.

This variation in access rates makes it difficult to design web servers for live events. Designing these servers for an average load makes it impossible for the server to handle periods of bursty accesses. On the other hand, designing web servers to guarantee service during peak loads is impractical for two reasons. First, it is difficult to estimate peak load a priori. Second, it may be prohibitively expensive to design web servers to satisfy peak load, since they will suffer from poor utilization during non-peak periods.

In this paper, we propose a scheme called *domain caching* that reduces the load on live event web servers by dynamically creating web caches in response to local access patterns. This scheme allows a web server to instruct one of its clients (*domain cache*) to act as a web cache and service requests from other "nearby" clients. When clients that are near the domain cache make requests to the web server, the server refers them to the domain cache. In response, the designated client or domain cache accesses content from the server, changing all HTML links to point to cached content on the client. All subsequent accesses to links on the cached web page are handled by the domain cache, thus reducing the number of accesses that must be serviced by the origin server. A domain cache must be updated with respect to the server frequently enough to satisfy the needs of the its clients.

This paper begins by characterizing live events, using traces from 1998 World Cup soccer matches [3, 9]. Next, we describe the domain caching system. In Section 4, we present the results of trace-driven simulations that show the effectiveness of domain caching, which reduce the number of accesses at a popular web server by up to 38% while using a minimal amount of storage space. Our simulation results also include an anal-

ysis of algorithms used to select a domain cache among several clients in a domain and a comparison of domain caching with a content mirroring scheme.

Section 5 describes our prototype implementation of a domain cache. Using our prototype, we find that depending on the distance of the origin server from the client, accessing content from our prototype is three to twenty times faster compared to accessing content from the origin server. We used a trace-driven simulator to estimate the load seen by a domain cache and used this estimate to show that our prototype is able to handle the load handled by most domain caches with less than a 10% increase in execution time of an application. Finally, in most cases the time taken to access data from our prototype from clients located geographically near the domain cache is either faster than or equal to the time taken to access the same data from the origin server.

This paper concludes with a discussion of several important issues raised by domain caches and a description of related work.

2 Motivation

In this section, we use the logs collected during the 1998 Soccer World cup to illustrate access patterns for coverage of live events.

The 1998 Soccer World Cup was held in France from June 10^{th} through July 12^{th}, 1998. During this tournament, 32 countries played a total of 64 matches while `www.france98.com` provided live coverage of these games to audiences world wide. To distribute the load on `www.france98.com`, a distributed director redirected requests from clients to servers located at Santa Clara, CA, Plano, TX, Herndon, VA and Paris, France. Once a request was directed to one of these locations, one of the servers at that location handled the request. A detailed analysis of these logs can be found in [3]. Table 1, extracted from [3], shows the usage of this web site during the entire duration of this tournament.

In this paper, we extend Arlitt's analysis of the World Cup Soccer logs [3] to consider the distribution of accesses to the four different locations that hosted mirror sites for the Soccer World Cup data.

Figure 1 shows the variation in the number of accesses to the four different servers on one particular day, June 30, 1998. The figure indicates that there is a large variation in the number of accesses to a server during the day. The number of accesses increases for the duration of a soccer game resulting a large difference between the peak load and the average load. Several other popular events also result in a sudden increase in the load on the server. During the recent US Presidential elections, `cnn.com` reported an increase in the number of page views: from an average of 22 million page views daily to 100 million page views immediately after the elections.

Duration	May 1 to July 23, 1998
Total Requests	1,352,804,107
Avg Requests/Minute	10,796
Total Bytes Transferred (GB)	4,991
Avg Bytes Transferred/Minute (MB)	40.8

Table 1: Summary of Accesses to `www.france98.com`

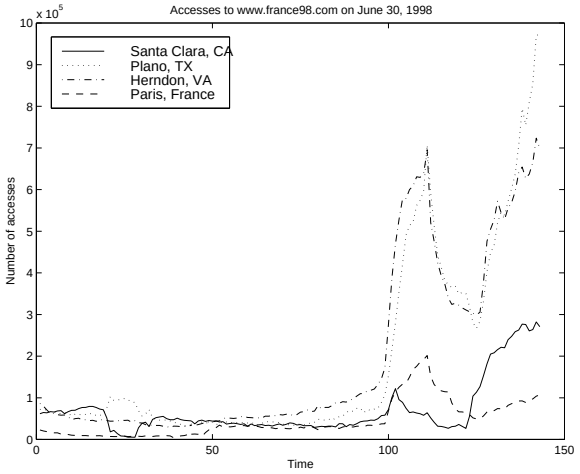

Figure 1: Variation in the number of accesses to the four different locations

This variation in the number of accesses makes it difficult to design web servers for such services. Designing the web server for an average load makes it impossible for the server to handle periods of bursty accesses. On the other hand designing web servers assuming peak load is impractical due to the difficulty in estimating the peak load a priori. Further, web servers designed for peak load result in very low utilization due to the relatively long idle periods.

For the rest of this study, we focus on two fifteen minute periods from different games played as part of this tournament. Table 2 shows the usage of the web site during these fifteen minute periods. The `Eng-Arg` trace represents the fifteen minute period

England Argentina Qualifying game (**Eng-Arg**)	
Duration	11:30 to 11:45 pm June 30, 1998
Total Requests	3,135,993
Avg Requests/Minute	209,066.2
Total Bytes Transferred (MB)	8,658.60
Avg Bytes Transferred/Minute (MB)	577.24

Italy France Quarter Final (**Ita-Fra**)	
Duration	6:00 to 6:15 pm July 3, 1998
Total Requests	1,678,768
Avg Requests/Minute	111,917.866
Total Bytes Transferred (MB)	3708.88
Avg Bytes Transferred/Minute (MB)	247.26

Table 2: Summary of accesses to `www.france98.com` during two fifteen minute periods corresponding to two different games

with the maximum number of accesses to the website. The `Ita-Fra` trace is used as a representative of the access pattern during a soccer game.

Table 3 shows the percentage of accesses to each of the four locations for the two traces considered. Assuming that the distributed director directs clients to a location based on network proximity, Table 3 represents the distribution of the clients in the network. The table indicates that the distribution of the clients accessing the website depends on the nature of the game. The server located in Paris, France handles 17.54% of the requests for the `Ita-Fra` trace; while for the `Eng-Arg` trace it handles only 5.03% of the requests. During international events like the 2000 Olympics, we expect to see a larger variation in the distribution of requests depending on the nature of the event. This variation in the distribution of requests makes it hard to statically configure a caching infrastructure, since it is hard to determine in advance the number and location of the caches that will be needed.

Another common technique used to improve user perceived latencies while browsing the web is to mirror content. Akamai [8] is one example of a content mirroring service that is being used by web

Location	**Percentage of Accesses**	
	Eng-Arg	**Ita-Fra**
Santa Clara, CA	13.36	3.55
Plano, TX	46.99	33.94
Herndon, VA	34.61	44.96
Paris, France	5.03	17.54

Table 3: Distribution of requests at the four locations

Description	**Reload time**
Wimbledon Tennis Scoreboard	60 seconds
Cricket Scoreboard	70 seconds
Nasdaq Stock Quotes	180 seconds

Table 4: Time between successive reload operations

servers. Content mirroring services typically work by executing a script that modifies content at popular web servers to redirect some of the requests to a nearby mirror.

However, the content at these servers catering to "live" events is modified very frequently (once in 1-2 minutes), thus making the process of mirroring content difficult. In Table 4, we estimate the rate at which some servers modify content with the rate at which their web pages are designed to be reloaded.

3 Domain Caching

In this paper we propose and evaluate *domain-caching*, a technique that is intended to reduce the number of requests seen by the servers and improve client response times. The architecture of the domain caching system is illustrated in Figure 2. The domain caching scheme can best be explained using the below algorithm:

1. The server maintains a table of all clients currently receiving service from the server.

2. The server clusters all the clients in this table into groups based on network "nearness" of the clients. One mechanism for clustering is to group clients based on the administrative domains to which they belong. A recent paper [10] describes a technique that uses information present in BGP routers to identify the administrative domain to which a client

belongs. In Figure 2, the server groups clients into three different clusters.

3. When the number of clients in a cluster exceeds a threshold parameter, the *cluster threshold*, the server identifies one client from among the cluster to represent the entire cluster and makes it a *domain cache*. Future requests from all clients in this cluster are re-directed to this domain cache. In Figure 2, the arrival of client X resulted in cluster C exceeding the cluster threshold of three. The formation of the domain cache and the process of re-directing requests to the domain cache is illustrated using dashed lines.

   Our prototype re-directs requests to the domain cache by sending a html page. The browser responds to this page by contacting the domain cache. Alternatively, the server could use redirex [14], a perl program that responds with "redirect" (HTTP status code of 301) replies indicating the new server's name.

4. The server identifies a domain cache from among several alternatives by solving the server selection problem [7]. In Section 4.2, we discuss strategies that the server can use to identify a domain cache. This mechanism of server initiated forwarding results in the ability of clients to dynamically detect "nearby" clients interested in the same content. This is in contrast to the existing system of hierarchical web caches [5] where caches need to know the existence of nearby caches.

5. Once a client is re-directed to a domain cache, it continues to request content from the domain cache. If the content requested by a client is not present at the domain cache (a miss at the domain cache), the domain cache fetches content from the origin server (or parent cache) on behalf of the client. In Section 6, we discuss techniques that a domain cache can use when the number of clients in the cluster drops below the threshold.

6. Finally, the above algorithm is used by the domain cache to create a hierarchy of domain caches. In Figure 2, the domain cache in cluster A uses the above algorithm to create another domain cache in cluster D. The server can use this hierarchy of domain caches to distribute content by using a push based model or a pull based model. For a server using a push based model, this hierarchy of domain caches represents a multicast tree along which content has to be distributed.

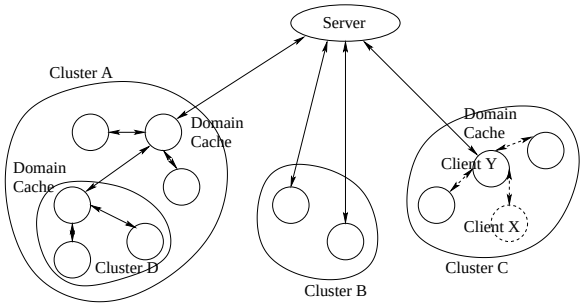

Figure 2: Domain Caching Architecture. Figure shows the formation of the domain cache, client Y, in Cluster C on the arrival of a new client X.

Item	Description
Client ID	Integer identifying the client that made the request
Location	The server to which the distributed director routed the request
Size	Number of bytes transferred by the server to the client
Timestamp	The time at which the server received the request from the client
Status Code	HTTP status code of the response sent by the server

Table 5: A portion of the information present in the traces used

4 Trace-Driven Evaluation of Domain Caching

We evaluated domain caching using logs collected at `www.france98.com` during the 1998 Soccer World Cup. We analyzed portions of these logs (described in Table 2) using a trace driven simulator. We evaluated domain caching by considering reduction in the number of requests seen by the server, storage space required at the domain caches, and the algorithms used to select a domain cache. We extended the simulator to explore the advantages of domain caching in comparison with content mirroring. In this section, we describe our evaluation methodology and analyze the simulation results.

Table 5 summarizes a portion of the information present in the traces [9]. Additionally, we obtained a mapping from clients to domains that associated every client with a *Domain ID* based on the client's IP address[1]. We used this mapping to group clients into clusters based on the domain to which they belonged. The *Domain ID* was identified by extracting the network portion of the client's IP address. For Class A networks, we used the first 8 bits; for Class B networks the first 16 bits; and the first 24 bits for Class C networks. This network portion is then anonymized by mapping it to a unique integer representing the domain. We realize that there can be multiple domains using a single network address ([10] identifies limitations of using this scheme to map client IP addresses to domains). However, we continue to use this scheme since we do not have the actual IP addresses of the clients.

We implemented a trace driven simulator that uses the above logs to simulate domain caching. The simulator takes as input a trace record containing the information in Table 5. Due to the lack of the actual IP addresses of the clients, we use a very simplistic clustering algorithm in which all clients belonging to the same domain are placed in one cluster. When the size of a cluster exceeds the cluster threshold, the simulator identifies a domain cache by using a server selection algorithm.

For each client, the simulator keeps track of the URLs accessed by the client and the size of the cache at the client. When a client requests a page that is not present at the domain cache, the simulator handles this domain cache miss by sending a request to the server or parent cache. By maintaining detailed information about the contents of caches at clients, we were able to experiment with different server selection algorithms. In Section 4.2, we study different server selection algorithms.

4.1 Performance of Domain Caching

Figure 3 shows the number of requests seen by the servers at Plano, TX with and without domain caching for the `Eng-Arg` trace. While domain caches are being created, there is very little benefit due to domain caching. After a sufficient number of domain caches have been created, the number of requests seen by the server at Plano, TX reduces even for large values of the threshold. Table 6 shows the percentage reduction in the number of requests seen by the Plano server during the last seven minutes of the simulation. For

[1] This mapping was made available to us by Martin Arlitt, HP Labs

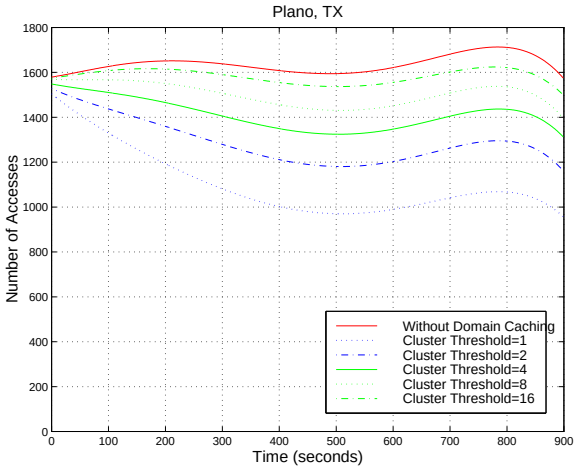

Figure 3: Number of accesses to the server at Plano, TX, for the trace of `Eng-Arg` without domain caching and with domain caching at a variety of cluster threshold values

a threshold of one, the number of requests seen by the Plano server reduce by up to 38%. Figure 4 indicates that for the `Eng-Arg` trace 76% of the requests come from 30% of the popular domains, explaining the reduction in the number of requests due to domain caching.

Next, we look at the average size of the domain caches. Figure 5 shows that size of a domain cache is fairly small, typically in the range of hundreds of kilobytes, indicating that a client with a small amount of storage space can act as a domain cache. We believe that the small amount of storage space required by a domain cache is due to the small number of pages that are accessed repeatedly.

Recall that when the number of clients accessing the server from a particular domain exceeds the cluster threshold, a new domain cache is created for that group of clients. As might be predicted, Figure 5 shows that the size of a typical domain cache increases with the cluster threshold. When the cluster threshold is high, a domain cache is created for a larger number of clients. The working set of the URLs accessed by these clients is bigger. Hence, for larger values of the threshold, the domain cache requires a larger amount of storage space to store all the requested content.

Finally, we found that doubling cluster threshold halves the number of domain caches created. We also found the average hit rate at domain caches (75%) to be significantly greater than the hit rate at traditional web proxy caches [4, 15]. We believe that this increase in the hit rate of the domain cache is due to small size of the working set of the

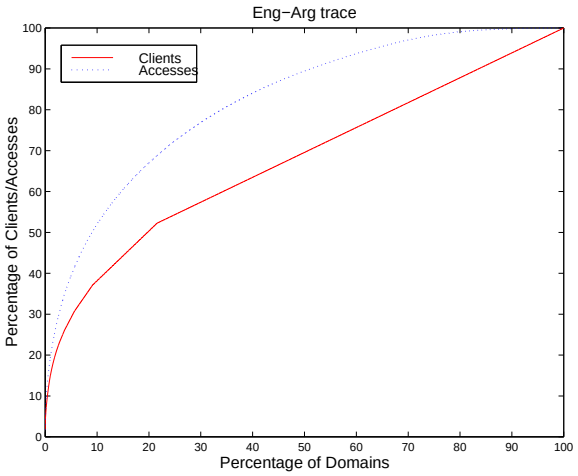

Figure 4: Distribution of clients and accesses from various domains. A point (x, y) in the solid line indicates that y percent of the clients belong to x percent of the popular domains

Cluster Threshold	**Total Number of Accesses**	**Reduction in Accesses**
1	425985	38.44 %
2	517330	25.24 %
4	577291	16.57 %
8	619803	10.43 %
16	660271	4.58 %
∞	692000	0 %

Table 6: Reduction in the number of requests seen by the server at Plano, TX with domain caching

domain cache compared to that of a traditional web proxy cache. Due to the lack of space, we do not present these results here.

4.2 Algorithm Used to Select a Domain Cache

When the cluster threshold is reached, the domain caching system must designate one of the clients in the cluster as the new domain cache that will service requests for the cluster. We experimented with a variety of algorithms for selecting a client to act as a domain cache:

- `Max Cache` The client with the largest cache is picked to act as the domain cache. For the results presented so far, we have used `Max Cache` to select a domain cache.
- `Min Cache` The client with the smallest cache is assigned the domain cache.

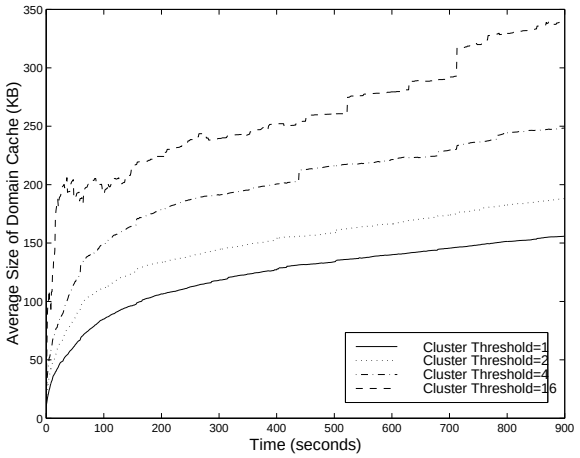

Figure 5: Average size of the domain cache created for different values of cluster threshold

- `Random` A client is picked at random to act as the domain cache.

Our results indicate that there is little difference in domain cache hit rates based on domain cache selection algorithms, as illustrated in Figure 6. The `Max Cache` performs slightly better, with a hit rate approximately 5.2% better than for `Random` and 8.3% better than for `Min Cache`. This small difference in the hit rates between `Max Cache` and `Random` indicates that other parameters such as computational power, network bandwidth and storage space at the client could be used by the server selection algorithm. In Section 5.4, we show that in some situations CPU load is an important parameter in selecting a domain cache.

4.3 Comparison with Mirroring

Mirroring the contents of a server at several mirror sites is an effective technique used in reducing load on the server. However, making exact copies of server content at several mirror sites could result in unnecessary network traffic.

Consider a situation in which multiple events occur simultaneously (as in the case of Olympics 2000). Lets suppose that all clients accessing a mirror site are interested only in a small subset of these events, *i.e.* these clients share restricted interests. In this situation, only a portion of the content (the portion corresponding to the subset of events) at the mirror site is accessed. Since content at the mirror site is a true replica of the content at the server, content that is never accessed from the mirror site is also replicated at the mirror site. This results in unnecessary network traffic from the server to this mirror site.

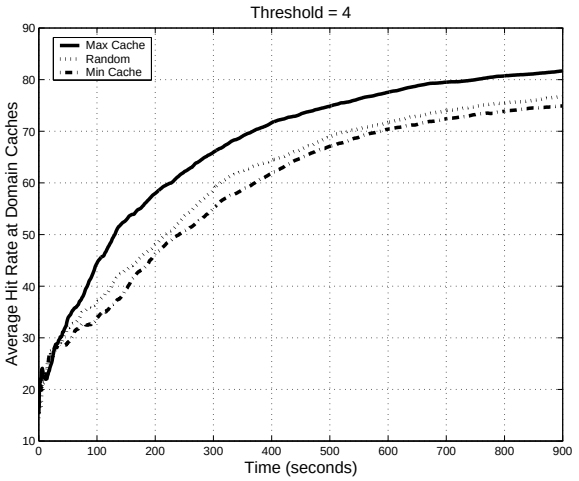

Figure 6: Variation in the average hit rate at all the domain caches for different algorithms used to select domain caches for a cluster threshold of four

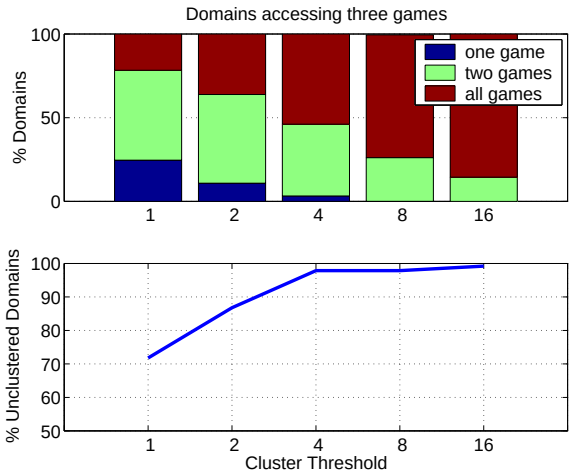

Figure 7: Percentage of domains accessing one, two and three simultaneous games

On the other hand, a domain cache fetches the content only when it is requested by one of its clients. When a domain cache's clients are not interested in a particular event, content corresponding to this event is not transferred to the domain cache - thus reducing the traffic from the server.

Ideally, we would like to characterize the reduction in network traffic when using domain caches compared to mirroring. Unfortunately, our traces do not contain information about when content was modified on the server. So, we are only able to demonstrate that clients in a domain share restricted interests.

To demonstrate this, we simulated a situation in which three soccer games are played simultaneously. (We used 1998 Soccer World cup traces for three games that were actually played on different days.) Figure 7 shows the percentage of domains that accessed one, two and all three games. For a threshold of one (a new domain cache is created whenever two clients from the same domain access the server), about 20% of the domain caches access a single game, while approximately 60% access two games. This results in significantly less network data transfer than if all domain caches must cache data from all three games.

The lower part of Figure 7 shows the limits of our current clustering algorithm, which is based solely on domain names. This graph shows the percentage of domains for which there is no domain cache, because the number of clients in these domains did not exceed the specified cluster threshold. For cluster thresholds above two, there were very few domains serviced by domain caches. Improving these algorithms to identify nearby clients across domains is essential to further reduce the number of accesses to the server. In Section 5.5, we describe one way of improving this algorithm.

5 Domain Cache Prototype

To demonstrate the feasibility of creating a domain cache, we developed a prototype implementation of a domain cache by modifying the CERN http daemon. In this section, we describe the domain cache prototype and evaluate its performance.

Figure 8 shows the design of our prototype implementation. We implemented the prototype by adding two additional processes, the `SecMgr` and the `loader`, to the CERN http daemon. The http daemon forks off the SecMgr, which in turn forks off the loader. The SecMgr provides the following security model to prevent access to users' web accessible private data via the domain cache. The SecMgr maintains a table of valid URLs to which it allows access. This table is initialized using a set of URLs taken as input. All embedded images and links reachable from a valid URL are also considered valid.

The SecMgr forks another process, the `loader`, that fetches URLs and stores them in the filesystem after re-writing all links in the URLs. Additionally, the loader transfers the URLs of all embedded images and links in a URL to the SecMgr. The SecMgr adds these to the table of valid URLs. For valid URLs fetched by the loader, the SecMgr keeps track of its location in the filesystem. Valid URLs that have not been fetched by the loader are

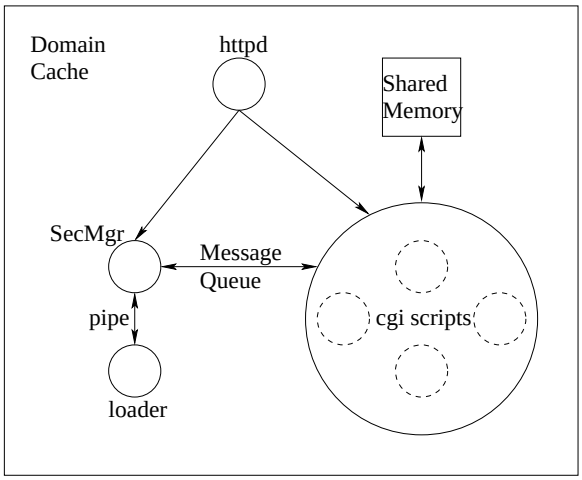

Figure 8: Prototype implementation of a domain cache

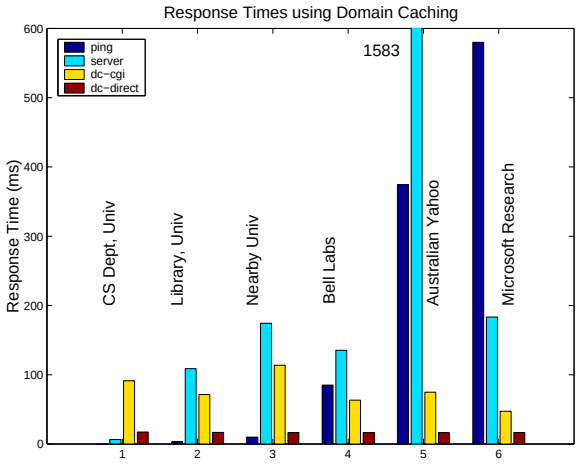

Figure 9: Comparison of the response times observed while accessing both the origin servers directly and a nearby domain cache for a request rate of 8 requests per second

considered domain cache misses and are identified by the absence of their location in the filesystem. The SecMgr handles domain cache misses by requesting the loader to fetch the URL.

URLs are accessed from the domain cache via CGI scripts that take the URL as an argument. The CGI script contacts the SecMgr both to validate the URL and to determine the location of the URL in the filesystem. Once the CGI script knows the location of the URL in the filesystem (for invalid URLs, the SecMgr returns the location of an error file), it reads the file and transfers content to the requesting client.

Additionally, CGI scripts use shared memory to maintain clustering information. For the prototype, we divided machines in our department into four different clusters based on their IP addresses. When the size of a cluster becomes larger than the threshold, the CGI scripts pick the client that requested the maximum number of accesses as the domain cache. Finally, requests are re-directed to the domain cache by sending a special html page to the client. On interpreting this html page, the browser contacts the domain cache.

5.1 Experimental Setup

We evaluated the performance of the prototype using `httperf` [13]. For all experiments described below, we used `httperf` to request one URL from a http daemon at a specified request rate for a duration of three minutes. Typically, a small number of pages are accessed repeatedly from live event web servers. To model this access pattern, we designed our experiments to access one URL from the server. We used `httperf` to contact the same page directly from the origin server and also via the domain cache prototype.

The CERN http daemon that we modified forks off a process for every request and another additional process for every CGI script that is executed. To obtain an estimate of the overhead of our implementation of the domain cache, we accessed the file corresponding to the URL directly from the domain cache. For the rest of the paper, *dc-direct* refers to results obtained by accessing the domain cache directly.

5.2 Response Times

To determine the reduction in response time while accessing a nearby domain cache as opposed to the origin server, we measured the response time of accessing pages from six different servers. As expected, we found considerable reduction in the response time while accessing a nearby domain cache.

Figure 9 shows the observed response times for accessing the six different web servers at a rate of 8 requests per second. For each server, the figure plots bars corresponding to the time taken to ping the server (*ping*), the response time while contacting the origin server (*server*), the response time while contacting the domain cache using CGI scripts (*dc-cgi*) and the response time while contacting the domain cache directly (*dc-direct*).

Figure 9 shows that accessing a nearby domain cache results in a reduction in response time when

Trace	Max	Mean	Std Dev	95 %ile
Eng-Arg	5.5528	0.2090	0.3661	0.7111
Ger-Mex	5.1758	0.1858	0.3354	0.6913
Rom-Hrv	8.6852	0.2153	0.4074	0.7549
Ita-Fra	9.8760	0.1629	0.3844	0.5161
Bra-Hol	9.4939	0.2223	0.4381	0.7799
Fra-Hrv	5.6652	0.1852	0.3209	0.6804

Table 7: Requests handled by the domain caches at the end of six different fifteen minute periods from the Worldcup Soccer traces

Server	Capacity (reqs per sec)	Size (KB)
Microsoft Research	18.2	8.3
US Open Tennis	15.4	11.9
Bell Labs	11.9	13.4
Australian Yahoo	12.2	18.3
Library, Univ	12.8	19.0
CS Dept, Univ	10	23.0
Nearby Univ	8.5	29.9

Table 8: Capacity of the domain cache for seven different web servers

compared to accessing the origin server directly. (The only exception is for the server labeled `CS Dept, Univ`, which is a faster machine on the same local area network as the domain cache.) Depending on the distance of the origin server from the client, the reduction in response times varies from a few milliseconds to hundreds of milliseconds.

5.3 Requirements of the Client Machine

Recall that in Section 4.1, we measured the average amount of storage space used by the domain caches for the `Eng-Arg` trace and found it to be a few hundred kilobytes. In this section, we estimate the CPU requirements of the client machine. We estimate the load on the domain cache by using the simulator to determine the request rate that a domain cache is expected to handle. We then measure the capacity of the prototype and find that it exceeds the required request rate at the domain cache.

Table 7 summarizes the request rates handled by all the domain caches (for a cluster threshold of four) at the end of fifteen minute periods from six different games played during the Soccer Worldcup, 1998. The 95 percentile for all the traces is less than one, indicating that most domain caches handle less than one request per second. However, the maximum request rate handled by the domain caches is as high as 9 requests per second for some traces. This indicates that although most of the domain caches handle very few requests per second, some domain caches handle higher load.

Table 8 shows the measured capacity of the prototype for various web sites. For each of the web sites considered, the prototype fetches content from the web site and re-writes the URLs. The column labeled `Size` refers to the size of the page accessed from the web site. For this experiment, we ran the prototype on a uniprocessor SUN Ultra running at 360 MHz with 128 MB of memory. The table indicates that the capacity of the prototype is inversely proportional to the size of the URL. For most web pages, the capacity of the prototype exceeds the maximum number of requests expected to be handled by the domain cache. This indicates that the client machine we used to run our prototype can handle the request rates expected to be seen by most domain caches.

5.4 Effect on Client Machine

Next, we studied the effect of running a domain cache on the client machine by measuring the increase in execution time of an application. Our results indicate that when the domain cache handles one request per second, the increase in application execution is less than 10%. However, the application execution time increases substantially for higher request rates. This suggests that the CPU load plays an important role in selecting the client machine for the domain cache when the request rate at a domain cache is expected to be high.

To determine the effect of running a domain cache on a client machine, we measured the time taken by an application on a machine running the prototype. Our application deleted the executables, libraries and the object files used by the http daemon and then built the http daemon by compiling the source code. By subjecting the prototype to various request rates, we determined the increase in execution time of the compilation for varying load on the domain cache. Figure 10 shows the average increase in execution time for five successive compilations of the http daemon. (Each compilation is followed by deleting all the executables and object files).

Figure 10 shows that when the request rate at the domain cache is less than one, the increase in application execution time is less than 10%. This

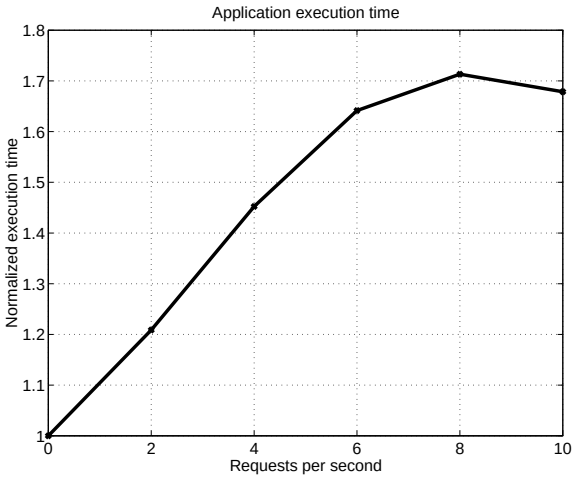

Figure 10: Increase in application execution time for varying request rates handled by the domain cache. The y-axis represents the fractional increase in application execution time

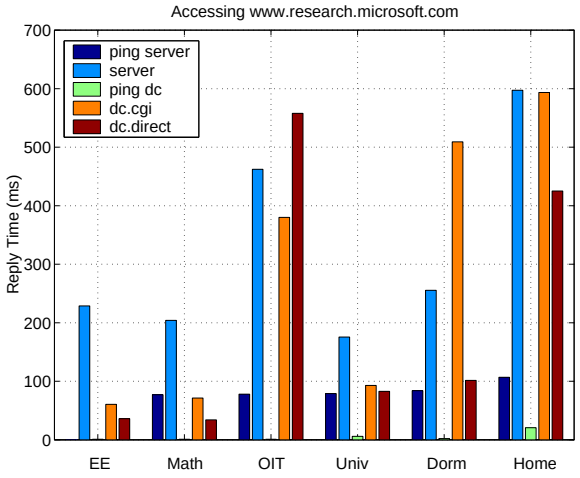

Figure 11: Time taken to access the Microsoft Research web page from six clients located geographically near the domain cache

suggests that for low request rates, the client machine can handle the additional load due to domain caching without a significant increase in the execution times of applications.

However, for higher request rates, the application execution time increases by as much as 70%. Thus when a domain cache is expected to handle high request rates, CPU load on the client machine is an important parameter in identifying a domain cache.

5.5 Network Nearness

Finally, we designed experiments to better understand the notion of clients that are near each other on the network. We accessed a web site via the prototype from six clients that are located geographically near the domain cache. Our results indicate that in most of the cases the time taken to transfer the URL from the domain cache is either faster than or equal to the time taken to access the same URL directly from the origin server.

Figure 11 shows the time taken to access the Microsoft Research web page from six clients located near the prototype. The bars in the figure correspond to:

- *EE*, *Math* and *OIT* represent accesses from clients in three deparments from our University. The domain cache ran on a client machine in our University, but in a fourth department.
- *Univ* are from a client machine in another University in the same city.
- *Dorm* are from a client machine in the residential dorms from the same University.
- *Home* represent accesses from a nearby home using DSL.

For each client machine, the figure has five bars corresponding to the time taken to ping the origin server (*ping server*), the time taken to access the URL from the origin server (*server*), the time taken to ping the domain cache (*ping dc*), the time taken to access the prototype using CGI scripts (*dc.cgi*) and the time taken to access the prototype directly (*dc.direct*).

Figure 11 indicates that for the three clients from the same University, accessing the page via the prototype (*dc.cgi*) is faster than accessing the page from the origin server. Machines in our University have Class B network addresses. Recall that the clustering algorithm described in Section 4 uses the first two bytes of a Class B network address to determine the domain address. This algorithm places the three clients *EE*, *Math* and *OIT* belong to the same domain.

For clients *Univ* and *Home*, the figure indicates that the time taken to access the page via the prototype using CGI scripts (*dc.cgi*) is either faster than or equal to the time taken to access the page from the origin server. However, the above clustering algorithm fails to detect that these two clients are near our prototype on the network.

In general, we believe that it is very hard for the server to identify clients that are near each other on

the network. Current research on drawing maps of the internet [6] will be useful in identifying clients that are near each other on the network. Until such maps are available, the simple clustering algorithm described above can be used as a heuristic for network nearness.

In Section 4.3 we saw that for the `Eng-Arg` trace the above algorithm resulted in more 70% of the domains having only one client. We believe that using well-known information about geographical proximity about domains will prove to be a useful strategy in clustering clients from different domains.

6 Outstanding Issues for Domain Caches

Domain caching relies on several client machines acting as servers. This raises several important concerns relating to the willingness of clients to act as servers, authenticity of the content provided by the domain caches, privacy of users' private content and failure of clients acting as domain caches. In this section, we address some of these concerns.

Volunteering to be a Domain Cache We believe that the success of peer-to-peer systems like Napster [2] and Gnutella [1] indicates the willingness of clients to participate as servers.

Clients willing to act as a domain cache send additional information to the http daemon. The information provided by clients includes the CPU load on the machine, the network bandwidth and storage capacity. Clients volunteering as domain caches send this information in addition to the file requested from the server. An example of information sent by existing clients is `User-Agent: Mozilla/4.61 [en] (X11; U; SunOs 5.7 sun4u)`.

Clients leaving the System Domain caches could monitor the number of clients accessing content by keeping track of clients entering and leaving the system. In a pull based model, domain caches could monitor the time since the last access from clients. Using appropriate timeouts, this information can be used to detect clients leaving the system. When the number of clients accessing the domain cache drops below the cluster threshold, the domain cache re-directs requests back to the origin server.

When users bookmark content at domain caches, this mechanism of re-directing requests back to the origin server provides a way of handling the "bookmarking problem".

Hit Metering Several web sites collect information about their clients. These servers typically keep a record of the client address and the number of accesses from a client.

Using domain caching, servers re-direct clients to nearby domain caches. Since all clients first access the origin server, these servers are aware of the addresses of clients accessing them. However, information regarding the number of accesses from a client is lost since all future accesses are from the domain cache. To address this concern, the domain cache could maintain the number of accesses from each client and periodically transfer this information to the origin server.

Privacy of Users' Private Content The presence of users' private content on the web (For example, email and calendar) makes it necessary for the domain cache to protect the privacy of this data. In the prototype, we used CGI scripts to provide access to a limited number of URLs. This provides privacy to users' private content by denying access to sites that contain such information. Another approach would be to use the access rights provided by the file system to protect privacy of users' private content.

Trusting Domain Caches Servers identify some of their clients to act as domain caches. To prevent malicious clients from becoming domain caches, we propose the use of third party services to authenticate clients. For example, a third party service could decide that the machine hosting our University's web page is trustworthy whereas a client from the University dorms is untrustworthy.

Authenticity of Server Content To prevent domain caches from tampering with server content, the server could compute a digital signature of the content and transfer the signature along with the content. Since domain caches re-write URLs within a html page, the server would have to compute this signature after excluding html tags corresponding to embedded images and links within the URL.

Failure of Domain Caches The possibility of the failure of domain caches introduces multiple points of failure for the service provided by the origin server. In a pull based model, the server could use appropriate timeouts to decide on the

failure of a domain cache. On detecting the failure of a domain cache, the server resets the information it maintains for the cluster to which the domain cache belongs. This enables clients from the cluster to make another connection to the origin server and obtain service.

Protocol Extensions Although we implemented our prototype using http, we believe that domain caches need to understand only a subset of the http protocol. We envision domain caches that can be incorporated as part of a web browser.

7 Related Work

To our knowledge there is very little work on designing web servers for frequently updated content.

Napster [2] is a music sharing tool that uses a peer-to-peer architecture. Napster users can query a napster index for music files. The napster index identifies peers willing to share their music files. Clients can then transfer music files directly from their peers. In the domain caching architecture, the servers are similar to napster indices: the servers identify peers that are near a client.

Li and Cheriton's work [11] on using reliable multicast to distribute frequently updated objects examines several multicast and unicast strategies to distribute content to proxy caches. They propose a multicast based protocol that groups objects into volumes, each of which maps to a different IP multicast group. These multicast groups are used to send invalidation messages and to maintain strong cache consistency. Another multicast based file transfer protocol is then used to distribute content. The technique described in this paper can be used to build a hierarchy of domain caches that can be as a multicast tree.

Michel et al. [12] propose the development of a large number of multicast groups that can be used to cache and distribute content from popular web servers. Requests for data are multicast within these groups: any client that has a cached copy of the data multicasts the data to all members of the group. The request is multicast in another group in the direction of the server, if no client within the group has a cached copy. This process of multicasting content to all members of a group ensures that popular content is cached at most locations and hence can be retrieved very quickly. However, this technique does not work well for frequently updated data.

Arlitt and Jin [3] presented a very detailed analysis of the server logs collected at `www.france98.com`. They studied the amount of data transferred, variation in the number of requests with time and location, distribution of HTTP response codes and an analysis of user sessions. We extended their analysis by clustering clients based on the domain to which they belonged.

8 Conclusions

In this paper we present domain caching, a technique to reduce load on web servers for live event servers and improve client response times. Domain caching clusters clients into groups based on network proximity. Once a cluster becomes "large", the server identifies a representative client (or *domain cache*) and handles requests only from this representative. Requests from all other clients in this cluster are redirected to the domain cache. The domain cache accesses content from the server periodically, changes all embedded links to refer to itself and stores them in its cache. Thus all future requests from other clients in the cluster are handled by the domain cache.

We evaluated domain caching using a trace driven simulation of the logs collected during the 1998 Soccer World Cup. These simulations show that the number of accesses to the World Cup Soccer web server reduce by up to 38% while using domain caching. Further, the storage requirements at a domain cache are minimal - hundreds of kilobytes. Our simulations also revealed that the choice of a client as the domain cache has negligible impact on the hit rate at domain caches.

Finally, we describe the implementation of a prototype domain cache. Using the prototype, we found that accessing a nearby domain cache can provide up to an order of magnitude reduction in response time in comparison to an access from the origin server. We also found that for most domain caches, our prototype is able to handle the request rate expected to be handled by a domain cache. Further, in most cases running our prototype increased execution time only by 10%. By accessing our prototype from several clients located geographically near the domain cache, we found that for most clients the time taken to access content from the domain cache is either faster than or equal to the time taken to access the same content from the origin server.

References

[1] Gnutella. http://gnutella.wego.com.

[2] Napster. http://www.napster.com.

[3] Martin Arlitt and Tai Jin. Workload characterization of the 1998 world cup web site. Technical report, HP Labs, 1999.

[4] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies*, Dec 1997.

[5] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 USENIX Technical Conference*, Jan 1996.

[6] Bill Cheswick, Hal Burch, and Steve Branigan. Mapping and visualizing the internet. In *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000.

[7] James D. Guyton and Michael F. Schwartz. Locating nearby copies of replicated internet servers. In *Proceedings of ACM SIGCOMM 1995*, August 1995.

[8] Akamai Technologies Inc. Akamai: Delivering a Better Internet. http://www.akamai.com/.

[9] ITA. Internet Traffic Archive. http://www.acm.org/sigcomm/ITA/.

[10] Balachander Krishnamurthy and Jia Wang. On Network-Aware Clustering of Web Clients. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[11] Dan Li and David Cheriton. Scalable Web Caching of Frequently Updated Objects using Reliable Multicast. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, October 1999.

[12] Michel, Nguyen, Rosenstein Zhang, Floyd, and Jacobson. Adaptive Web Caching: Towards a New Global Caching Architecture. In *Proceedings of the 3rd International WWW Caching Workshop*, June 1998.

[13] David Mosberger and Tai Jin. httperf—a tool for measuring web server performance. In *Proceedings of the 1998 Workshop on Internet Server Performance*, 1998.

[14] John Walker. Redirex: Redirect HTTP requests to new Web server. http://www.fourmilab.ch/webtools/redirex/.

[15] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of ACM SIGCOMM'96*, Aug 1996.