Web Caching.

Adam Belloum
Dept. of Computer Science,
Computer Architecture and
parallel Systems group
Universiteit Van Amsterdam

Contents

Note: This document has been written to keep track of the different literature studies, discussions and implementation work related to JERA project. It is then subject to continuous upgrades.

Chapter : 1

What is Web Caching

Chapter 1

What's Web Caching

Index Terms– Web Caching, Document Replacement strategy, Document coherency.

1.1 Introduction

The bandwidth demands of the world wide web continue to grow at hyper-exponential rate. Given this rocking growth, the network community proposed the caching mechanism as the most powerful approach to face this problem. A number of web analyses show that the network traffic can be greatly enhanced by using a caching mechanism. Danzig and Hall demonstrated that half of the FTP document transmitted over the NSFNET[1] backbone could be eliminated [4]. A recent analysis of queries to the most popular NCSA Mosaic servers was investigated by Braun. It shows that wide geographic diversity of query sources and popularity of relatively small portion of the server file set present a strong case for deployment of geographically distributed caching mechanisms [8]. This study shows about 40% to 70% of the bytes transfered from the central server could be saved using a cache document timeout[2] as low as 1024 seconds. Netscape, a vendor of web servers claimed in march 1995 that a single web proxy server (proxy server is a web caching mechanism, more details will be given in section 1.5.1) can reduce Internet demands by up to 65%.

In each network transaction, three agents at least are involved: the client, the proxy server, the origin server and the cache.

- The client: An application program that establishes connections for the purpose of sending requests. These are often browsers, editors, or other end user tools.
- The server: An application program that accepts connections in order to service requests by sending back responses.
- The origin server: The server on which a given resource resides or is to be created.
- The proxy server: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of the other clients.

[1]NSFNET: National Scientist Foundation Network

[2]The timeout is the time used for flushing documents from the cache

- The cache: A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and the network bandwidth consumption on future equivalent requests.

The first approaches proposed to solve the traffic jam on WWW were based on client-caching, which means that each client is provided with a local cache that allows it to store the last requests. There are two major drawbacks to this approach:

- This approach is considered to be myopic, which means that two neighbor users who require the same documentation from a remote server perform two distinguished accesses and two distinguished transfer and then store the same information twice at the same site.
- Since the majority of the end-users work on a common PC, the cache size cannot be large enough to support a a vast a mount of data. which forces the users to request again data from the remote server.

The client-caching approach cannot be used alone to solve the hyper-exponential growth of the Internet traffic. New methods have been investigated for the two past years which suggested an approach based on sever caching. The principal is quite simple: some servers receiving requests from the end-users or from other serves send a request on behalf of them to the remote servers. Upon receiving the requested data, these servers store a copy of the data in their own cache, and sent it to the requesters. On the next request to the same data the server will serve data from its local cache. This approach allows to reduce the net traffic, the latency for getting the information, however it introduce new intricacies:

- Since a web document can be updated at the source server the copies which are on the other servers have to be invalidated or updated. This problem is known as the "cache consistency problem".
- To be efficient, the caching servers have to be spread out over the Internet so that the end-user can redirect its request to the nearest one. This issue outlines the problem of the geographical distribution of the cache servers. To answer this question a thorough knowledge of the network topology is required.

1.2 Statistical information on the Internet traffic

In this section we will recall some of the results obtained by James Gwertzman and Margo Seltzer. An important outcome of this analysis is that the access patterns for each document throughout the Internet in not uniform. This means that some sites, and more precisely, some documents at some particular sites are requested more frequently than others, and these sites and documents are spread non uniformly over the Internet. Moreover, even the requesters are distributed non uniformly. In the following subsection, we will discuss some features related to the transferred documents, namely: documents and sites popularity, transfer protocol and document live time.

1.2.1 Popular documents

It has been outlined that only a small percentage of the files available on a given server is responsible for a disproportionate share of requests from that server [5]. For example, the ten most popular files of Harvard servers [3]were responsible for respectively 49%, 50%, 13%, 25% of the total request for each one of the four servers, and the 100 most popular files were responsible for 79%, 96%, 68% and 58%. Bestavros confirms this result in [5] adding: "the more popular the server, the smaller the fraction of the documents that account for most of its accesses". Besides, he shows that the most popular files are the less frequently changed files [6]. Furthermore, Braun indicates in the analysis of the NCSA[4] web that the most popular files are under 10 kbytes [7]. From these previous studies, it is possible to identify a class of documents that greatly affect the Network traffic. These documents can be characterized by the following points:

1. They represent a small percentage of the files at a given server.
2. They are responsible for a large percentage of the accesses at the server.
3. They have a relatively small size.
4. They are the less frequently updated.

This class of files seems to be worth to be considered in the Internet traffic problem. The characteristics listed above showed that they are really suitable for a caching mechanism.

1.2.2 The transfer protocol

The study proposed by Danzig in [4] showed that the Internet file transfer protocol is the source of about half of the documents that traverse the NSFNET. In a previous study the same author showed that many large essentially read-only files are frequently FTP'd across the Internet many time each day. The study his presented deals with the improvement of the FTP protocol to reduce network traffic. Three points have been discussed: server independent file naming, whole file caching and automatic file compression.

1.2.3 The document life time

The document life time is the parameter that affects the choice of the frequency used to update or to check the cached copies. A short period of checking is inefficient for static documents, on the other hand, a long period generates the possibility of cache inconsistencies, which leads to transfer stale data. Nevertheless, in the Internet domain the so called "weak consistency protocol" are tolerated, which is not the case for the traditional computer architecture caches. On the base of the document life time, the web documents were classified in two categories:

[3]These servers include the Harvard Computer Science Department (das.harvard.edu), the Harvard Campus Computer Organization server (fas.harvard.edu) and the www.ncsa.uiuc.edu, home of the mosaic browser

[4]NCSA: National Center Supercomputing Application

Dynamic documents are defined to be the frequently modified by the source server. An example of such a document could be an electronic newspaper which is updated every day.

Static documents are less frequently updated, often they represent the oldest documents stored in the server. According to Bistavros results they include the most popular files [].

Some of the documents that travel through the Internet include headers containing information to help the server to decide either or not to cache the document. However the server can also use statistical information to decide if the document is cacheable or not. For example, dynamic documents should be classified as a non cacheable as much as possible.

1.3 Important task involved in web caching

The basic idea behind caching is to keep copies of the documents which have recently been fetched. On the web, caching typically involves the browser itself remembering the documents which were returned when the URLs [5] were looked up, by storing them on the disk, and possibly sending its request via a *"cache server"* rather than using the stored URL to directly access the remote server. The *"cache server"* is a particularly useful concept, because it allows large numbers of users to shares a pool of cached documents. The *"cache server"* takes the URL required by the web browser and examines a database of a local copies of resource that it has already cached. If the resource with the given URL is held in its database, the *"cache server"* returns its local copy of the resource straight back to the browser. If not, it uses the URL to retrieve the resource from the remote servers, passes one copy of the resource to the end-user's browser and enters another copy of the resource into its database.

The fact that the Internet is composed of a large number of servers and only few of them can be used for caching the information, and the fact that not all of the documents have to be cached introduce a number of new problems that have to be solved. We summarize some of them:

- Cache consistency: this is a classic problem of all caching techniques used at all levels of modern computer systems.
- Cache replacement strategies: the traditional cache replacement policy seems to be less efficient for web caching, since they do not take into account the new parameters related to the web environment such as: the expected time to live, the time it takes to retrieve a document etc.
- Selecting cache servers (where caching?): this task allows to identify the geographical position of the caching servers.
- When caching? This task is involved in the "server-initiate-caching", it allows the server to identify the appropriate document to be cached and in the appropriate

[5] URL: Uniform Resource Locators

time. The identification process has to be adaptive in order to follow the network traffic evolution.

1.3.1 World wide web cache consistency

Several naive mechanisms are in wide use by caches today. "Check Every-time" is a common naive mechanism, where a request is made at each time a document is requested. The caching server always sends a "conditional GET"[6] to the next higher cache or server to verify if the cached copy is still up to date. Such a mechanism would result in a message to the document's home server for each one of the copied documents. Thus the home servers of popular documents would be overwhelmed with "conditional GET" request. "Never-Check" is another mechanism supported by Netscape's cache. A cache would never verify the validity of its copy, consequently, a fresh copy can only be retrieved when explicitly requested by using the "reload button" in the browser, resulting in a "pragma"[7] request. The third method uses an expiration parameter to keep documents up to date. Each cached document is assigned an expiration time, for which it will be assumed valid. The cache will verify the validity of the cached documents when the expiration time elapses. This mechanism do not guarantee strong consistency of the cached data.

In the following subsections, we will discuss the most important cache consistency protocols used in current web caching implementations, namely: time-to-live, client polling, and invalidations.

1.3.1.1 Time-to-live fields: TTL

The TTL is a priori estimate of an document's lifetime that is used to determine how long cached data remain valid. Each document is assigned a TTL, when it elapses, the data is considered invalid [6]. TTL's are simple to implement in HTTP using the optional "expires" header field. The challenge in supporting TTL lies in selecting the appropriate time out value. If the latter is too small, data may be reloaded unnecessarily, if it's too long stale data may be returned. The study of Lopez [10] tries to automatically place a value for this time-out using some vital statistics of each file (modification rate, access frequency...). He proposed the following formula for the expire date:

$$Expire\_Date = (Today) + (Frequency\_access) + (Time\_since\_last\_change) + (size) \quad (1.1)$$

Where each term might be weighted in an appropriate form.

The staleness of each document is defined by Dingle as the time elapsed since its last-good-time, indicating the last time at which the document's content on its home server were exactly equal to the present contents of the cached copy [19].

1.3.1.2 Client polling

Client polling is a technique were clients periodically check back with the server to determine if cached documents are still valid. Again, the challenge in supporting client

[6] Conditional GET is a request header provided by HTTP protocol (discussed in section 1.4.1)

[7] pragma is another header provided by HTTP protocol (discussed in section 1.4.1)

polling lies in selecting the frequency of polling. The Alex FTP cache represents one of the implementations of the client polling approach [6]. It uses an update threshold to determine how frequently to poll the server. The update threshold is expressed as a percentage of the document's age. A document is invalidated when the time since the last modification exceeds the update threshold. There are two important things to note with the respect of client polling.

- it's possible that the cache could return stale documents.
- it's possible that the cache invalidates documents that still is valid.

  Often an assumption is used in this technique: young files are modified more frequently than the old files. Bestavros found that on many given server only few files change rapidly. Furthermore, he observed that globally popular files are the least likely to change.

1.3.1.3 Invalidation protocol

An invalidation protocol is required when weak consistency is not sufficient. Invalidation protocols depend on the server keeping track of the cached documents. Each time an item changes the server notifies caches that their copies is no longer valid. One problem with invalidation protocols is that they are often expensive in terms of network traffic.

- Servers must keep track of where copies of their documents are cached, which introduce a scalability problem.
- Invalidation protocols must also deal with invalid clients as a special case.
- Invalidation protocols require modifications to the server.

The simulations performed by Gwertzman and Seltzer [7] compare both TTL and client polling (Alex File System: ALE) to an invalidation protocol. According to TTL value or the threshold defined in ALE approach, the TTL and Alex File can respectively save more or less bandwidth compared to the invalidation protocol.

A new extension of the validation check method was proposed by Dingle and Partl in [19] to maintain cache coherency. It is based on expiration time mechanism and provides a stream of document to versions in response to each request. It supposes that it is better to have an old version of the document rather than sitting staring at a blank screen. As a fresher version of the document arrive, the document should be updated in the user's web browser (option may be used to specify whether the user want to see the newly-arrived documents version). The user should always have a clear indication of the staleness of the documents being viewed (expiration time, the last time the document is known to be good...).

1.3.2 Cache replacement strategies

On a cache miss, if the cache is full, one of the entries of the cache must be replaced to store the missing document. Several algorithms have been described in the literature, the more notable of which is the "Least Recently Used" (LRU) algorithm and its many derivatives. The LRU algorithm does not take into account the delay introduced to retrieve the discarded documents. The time required to retrieve documents from different sites can be considerably different. A new replacement algorithm addressing this problem, has been proposed by Bolot in [15]. This approach assigns a weight to each item in the cache and replaces the document with the lowest weight. The weight is a function of the last-reference, the time it takes to retrieve the item, the expected TTL etc. A mathematical model for defining the weight has been proposed

$$w(t_i, s_i, rtt_i, ttl_i) = \frac{w_1 * rttl_i + w_2 s_i}{ttl_i} + \frac{w_3 * + w_4 * s_i}{t_i} \tag{1.2}$$

Where
t_i is the time since the document i was last referred
s_i is the document size
rtt_i is the time it takes to retrieve the document
ttl_i is the time to live of the document
w_4, w_3, w_2, w_1 are constants. In the equation, the second term present the temporal locality, while the first one presents the cost of retrieving the document.

1.3.3 How to select cache servers?

Most caching schemes in wide area distributed systems are "client initiated". Typically, the decision of when and where to cache information are made without the benefit of the server's global knowledge on the usage pattern. In this approaches clients request a document from a proxy server, which on its turn requests it from the remote server. Another approach, called "server- -initiate-caching" consists in relegating caching decisions to the server, which has the ability to monitor the documents that should be cached [5]. The server initiates caching by determining when to distribute documents and where they will be distribute to. These two decisions are based on:

- The knowledge of the network topology.
- The access history of the document in question.

Server-initiate-caching is inherently cooperative; servers must accept and serve documents sent to them by other servers. To do so, a server should be modified to accept files for replication using a modified "POST"[8] request. A server which receives a thousand of requests per hour should obviously have server-initiated caching priority. To address the "where to push" question; the servers use access histories to calculate the optimal location of the replicas. Moreover, the "When caching" question is addressed using a "push-threshold". Each server maintains a load factor based on how many requests it serviced per hour. Every time the server satisfied a request for a document it increments

[8]POST is a communication header provided by HTTP protocol

its load counter. Every hour, the server computes: *Loadnew = loadold x decay*, where decay= 0.5. When a server exceeded the push-threshold, it replicates its popular files to the optimal location.

1.3.4 Cache organization

Since each implementation supports highly concurrent streams of requests. It has been shown in [1] that the hierarchical organization of the cache is one of the most powerful solutions. Caches resolve misses through other caches higher in the hierarchy. In addition to the parent-child relationship, some hierarchical caches support a notion of "siblings caches". The "siblings caches" are situated at the same hierarchical level, and are allowed to exchange copies of data among each other. The principal drawbacks of the hierarchical organization are:

- As the hierarchy deepens, the root cache becomes responsible for more and more clients, It is therefore recommended that the hierarchy terminates at the first place in the regional network or backbone where the bandwidth is plentiful [1].
- If the document is not already cached in a leaf node it typically isn't cached in a higher level node either. Furthermore, the leaf node caches are typically limited in size (10 MB, 100MB for proxy cache) [5].

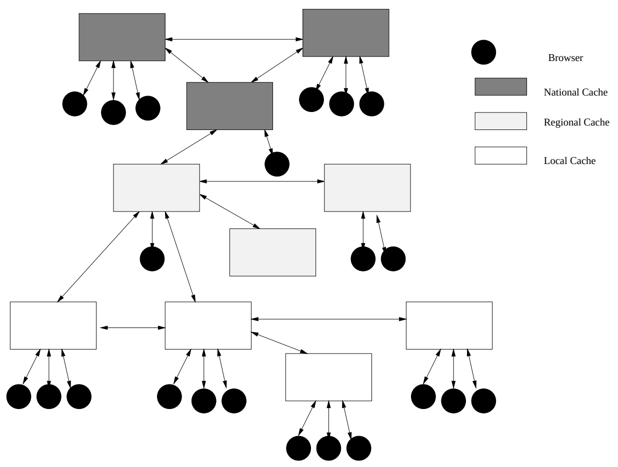

Figure 1.1: Example of a hierarchical topology with sibling relationship

1.4 Parameters that affect cache operations

The problem faced in implementing a cache in a web environment, here have been solved a decade ago in the OS community. However, since the working frame has changed, new parameters interfere now in cache operations . These parameters are related to different domains such as: transfer protocol, network topology, etc. In the next subsection, we will discuss a few parameters that have been used in web caching mechanisms.

1.4.1 Network transfer protocols and web caching

It has been outlined in [4] that the network protocols could be used to classify data transactions on the net, which allows to identify the most frequently used protocols in the network transactions (FTP is the source of half of the documents that traverse the NFSNET backbone). In addition to this, the transfer protocol such as HTTP provides mechanisms that have been widely used to solve some of the problems involved in caching through the Internet. We have listed bellow some of the mechanisms provided by the HTTP protocol and which have been successfully used in a number of web cache implementations.

"If-Last-Modified: date" Some caching servers take advantage of the CERN supported "If-Last-Modified' request which allows to identify the last update time of a document. It allows the server only forwarding clients to caching servers which hold a fresh copies, as well as for the caching server to check out the validity of their content. The use of this information requires the definition of an "expire time" used as a threshold to start executing actions onto the documents.

"Pragma: no-cache" This header can be appended to a GET message and indicate that the cache may not return a cached document; it must return an authorative and definitely fresh version of the document retrieved from the document home's server. A "pragma: no-cache" request therefore passes up through all caches in the hierarchy to the document's home server.

"Last-Modified: date" this header is attached to every document returned from a GET message, and indicates when the document was modified to create the specific version being returned. A document's last-Modified time acts as a unique version identifier for the document [19].

"Date: date" is returned in response to every GET request. "Date" represents the last date at which a document was known to be good. This is not the same as a document's Last-Modified header. A detailed example is given in [19] which explain the difference between the "Date" and Last-Modified header.

"Expire: date" When this header is sent in response to a request, it means that the document definitely will not change before the given date.

Many approaches were proposed as it will be discussed in the following sections. It has been outlined also in many papers, that the cache strategies are designed according to statistical results, derived from collected traces of the Internet traffic. For those tasks the use of network protocol parameters is useful to identify the network traffic (Danzig uses IP network numbers to collect the traces through the network).

1.5 Current web caching implementations

1.5.1 Proxy Caches servers

The proxy mechanism was introduced into the CERN HTTP server in 1994. One of the primary uses of the proxy cache was to allow access to the web from within a firewall figure 1.2. The proxy server waits for a request from inside the firewall reads the response and sent it back to the client. Current proxy methodology is based on the earlier gateway code written by Tim Berners-lee as a part of libwww [9].

Besides the ability of caching data the proxy server improves the communication between a user inside the firewall and a remote sever (more security, easy to install, new abstraction level for the user...). The forking process used by the proxy server decreases the performance of the system, since it forked for each connection it received. The incoming connection could not be accepted fast enough and users were turning away. The Netscape proxy server (beginning 1995) which relies on a non-forking process model allows to significantly reduce strain on the hardware [16].

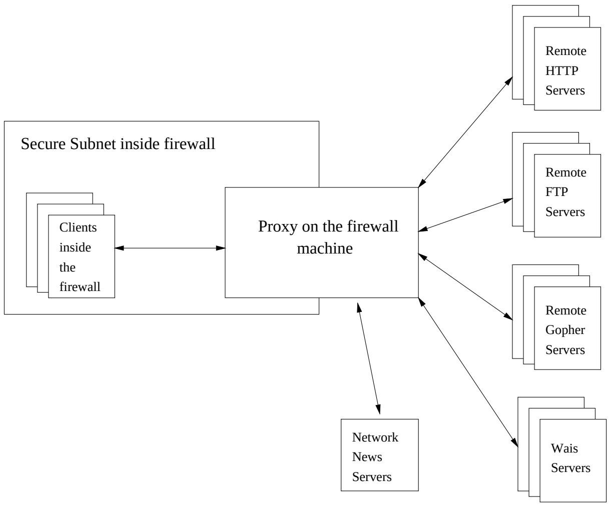

Figure 1.2: Firewall Topology

1.5.1.1 Technical details

- When a client sends a request to a proxy server, it always uses the HTTP protocol, even when accessing a resource served by a remote server using another protocol like gopher or FTP.
- However, instead of specifying only the pathname and possibly search keywords to the proxy server, the client specifies the full URL. This way the proxy server has all

[9] The libwww is the WWW common library URL: http://info.cern.ch/hypertext/www/daemon/status.html

the information necessary to make the actual request to the remote server specified in the requested URL, using the protocol specified in the URL.

- Most www clients are built on top of libwww, which handle the different communication protocols used in the web. The entire proxy support is handled automatically for client using libwww. Environment variable are used to control the library. The is an individual environment variable for each access protocol: http_proxy, ftp_proxy, gopher_proxy and wais_proxy (these variables are set to the URL pointing to the proxy server).
- When environment variables are set to connect the user to a defined proxy server, the libwww code causes a connection to be always made to the proxy rather than directly to the remote server. The latest version (version2.15) of libwww also supports an exception list so client don't have to always go through the proxy server [3].
- Proxy server support is implemented only for HTTP/1.0.
- To check the validity of the cached data, proxy servers are able to do a conditional GET ("If-Modified-Since") requests. The GET request is otherwise the same except that this header contain the date and the time that the document currently in the client (proxy server) was last modified. If the document has not been modified since the date and time specified, it will not sent back the documents, instead only the relevant object meta-information header such as expire date will be returned.

1.5.2 The Harvest servers

The next generation of the proxy servers was not only able to act as a simple proxy with a cache of documents to improve performance, but it also supports interaction with neighbor and parent caches in a co-operating hierarchy. The Harvest server was the most notable of this caching server generation. This model improves performance in the case of a cache miss by allowing other close-by caches to say whether they have the requested document. This means that any cache that is a part of a large cooperative hierarchy benefits from the documents stored in all the other caches. The harvest cache implementation is optimized to support:

- A highly concurrent stream of requests with minimal queuing for the OS-level resource, using non-blocking I/O, application-level threading and virtual memory management, and a Domain Naming System (DNS) cache.
- The Harvest cache can also be paired with the existing HTTP servers (httpd's) to increase document server throughput by an order of magnitude. It is then used as an http-accelerator, by running this http-accelerator in front of the httpd, once a document is in the http-accelerators cache all future hits will go to the cache and the misses will go to the native httpd.
- To reduce the cost of repeated failures (e.g., from erroneously looping clients) two forms of negative cache were implemented. First when a DNS lookup failure occurs, the negative result is cache for five minutes. Second when a document retrieval failure occurs, the negative result is also cached for a parameterized period of time.

- The Harvest cache keeps all meta-data for caches documents (URL, TTL, reference counts, disk file reference, and various flag) in the virtual memory. It will also keep exceptionally hot document loaded in virtual memory.

The Harvest server provides a very efficient means of gathering and distributing indexing information, supports the easy construction of many different types of indexes customized to suit the peculiarities of execution information collected, and provides caching and replication support to alleviate bottlenecks. Each Harvest server consists of the following subsystems:

- Gatherer: provides an efficient and customizable way to collect indexing information. It scans periodically, maintains a cache of indexing information, and allows information to be retrieved in a single stream. It addresses network transmission inefficiencies by providing remote access to the pre-filtered, incrementally updatable indexing information in a compressed way.
- Broker: provides an indexed querying interface to gathered information. Broker retrieve information from one or more Gatherers or other Brokers, and incrementally update their indexes. It records the records unique identifiers and time-to-live's for each indexed document, garbage collects old information and invokes the Index/search subsystem when it receives a query.
- Index/Search subsystem: to accommodate diverse indexing and searching needs, the Harvest server defines a general broker-indexer interface that can accommodate a variety of search engines. The principal requirements are that the back-end support boolean combinations of attribute-based queries and it support incremental updates.
- Replicator: The harvest server provides a weakly consistent, replicated wide-area file system called mirror-d. Each mirror-d instance in a replication group occasionally floods complete state information to its immediate neighbors.

  Object-cache: The cache sends query datagram to each neighbor and parent, plus an "ICMP??"echo to document's home site, and chooses the fastest responding server from which-to retrieve the data. It runs as a single, event-driven process. I/O to the disk and to the cache clients are non-blocking.
- Harvest Object subsystem: allows users to type data and associate method with the data, which are automatically invoked when a user selects a hypertext link to the data.

1.5.3 Cache goes Replication: CgR

By merging the administratively simple passive caching with the more work-intensive active mirroring of data into an active scheme, a new concept of caching mechanism was proposed by Baentsch, Molter and Sturn in [18]. The idea here is to create a hierarchy of replicated servers, such that only few of the "Replicate Servers" are known to the "Primary Server".

These replicas of the first level may themselves have "Replicate Servers" and forward the data they obtain from the "Primary Server" on to their "Replicated Servers". Each server in the hierarchy has the ability to replicate the cached documents using an appropriate heuristics-based algorithm.

The problem of locating the "Replicate Servers" remain to be solved. To address this problem a new service called WLIS for "Web Location and Information Service" was developed. Its primary purpose is to determine what servers are grouped together in a logical server groups of primary and replicate servers.

1.5.4 Main Memory Caching

This mechanism allows storing frequently accessed documents in the main memory of the server instead of storing all documents on disk. The server accesses its own cache directly without any help from the file system and therefore without the overhead. The experimental results in [21] show the efficiency of this mechanism using traces gathered from variety of Web servers. It has been shown that a few Mbytes reserved in the main memory for cache operations are enough to hold the most frequently accessed document of the servers, resulting in a hit rate between 33% and 70 %. To make the best use of this limited main memory cache, a caching policy that prefers caching of small documents is used. This caching policy dynamically adjusts itself to the incoming requests and the available cache size, so that it always achieves (almost) "optimal" hit rates [21].

1.6 The Future of web caching

In terms of efficiency, web caching has a lot of potential. We summarize some of the main ideas for future development in web caching:

- Cache co-operation: As far as co-operation is concerned, Harvest is leading the field. Unfortunately this co-operation is limited to networks of Harvest caches as no other server currently understands the inter-cache communication protocol used [16].
- Client resilience in the event of cache or a network failure.
- Proxy auto-configuration: the client is able to determine which proxy server(s) it should use.
- Networks for caches: it would make sense to provide dedicated international network links for cache traffic alone. It is hoped that the old, now redundant, links will become dedicated to the caching operations.
- New HTTP protocol: the current used HTTP protocol shows weak performance when retrieving long files across a congestioned network. It should be possible for a particularly busy cache to hold open connection to a popular server permanently. This would completely eliminates the cost associated with asking for a new connection for every requested URL.
- The use of more speculative techniques: these methods try to update a cache document before the document has been explicitly requested. If we do have enough network bandwidth to spend some of it downloading documents which might never be used, we might use two sorts of speculative mechanisms: