DISTRIBUTED WEB CACHING

MALITHA NAYANAJITH WIJESUNDARA

(B.Eng.(Hons.), Warwick)

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2004

Abstract

With exponential growth of the World Wide Web, techniques for alleviating the bottlenecks to network performance have gained increased importance. Web caching is one of such techniques which stores frequently used web objects nearer to the end user to reduce unnecessary remote access.

As an evolving area in web caching, co-operative web caching has shown promising results over centralised caching mechanisms. This thesis makes several contributions to the area of co-operative web caching. We introduce a novel co-operative web caching architecture called Distributed Web Caching. Distributed Web Caching introduced here, is a variant of co-operative web caching. The novel feature in Distributed Web Caching is that every client node could act as a cache server and share its cache with neighboring nodes.

A comprehensive set of protocols for access, storage and serving of distributed cached data is developed. We provide analytical models to evaluate and study of Distributed Web Caching. We implement the proposed architecture and measure the performance under different constraints.

Further, we extend our analysis to object replacement strategies in Distributed Web Caching. We introduce simulation models to compare performance of object replacement strategies. Several novel replacement strategies for Distributed Web Caching are introduced and compared with existing stand-alone replacement strategies.

Acknowledgement

I would like to express my heartfelt gratitude and appreciation to Associate Professor Tay Teng Tiow for his guidance, advice and constant encouragement throughout the course of this research.

I am indebted to the Department of Electrical and Computer Engineering of National University of Singapore for awarding me a scholarship for postgraduate studies.

I would like to thank Dr. Bharadwaj Veeravalli, Dong Ligang, Ganesh Kumar, Ajith Ekanayake and Upali Kohomban for providing valuable technical inputs during different stages of my research.

Sincere thanks go to all my friends and colleagues at NUS including, Lesly Ekanayake, Anuruddha Rathninde and Himal Suranga for their support and encouragement during my stay in Singapore.

Finally, I would like to thank Associate Professor G.P.Karunaratne for encouraging me to pursue a postgraduate degree at NUS and to my friend Asanga Gunawansa for encouraging me to upgrade my research programme to a doctoral programme and for proof reading my thesis.

This thesis is dedicated to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The World Wide Web is experiencing exponential growth. The increased use of the Web results in increased network bandwidth usage which in turn strains the capacity of networks. This leads to an increasing number of servers becoming "hot spots", sites where the increasing frequency and volume of requests makes servicing these requests difficult. This combination of overloaded networks and servers result in increased document retrieval latency.

Caching documents throughout the Web can alleviate such problems [1]. Caching refers to the temporary storage of commonly accessed computer information for future reference. Caching is only beneficial when the cost of storing and retrieving information from the cache is less than the cost of retrieving information from the original location. The concept of caching has found its way into many aspects of computing. Computer processors have data and instruction caches, operating systems have buffer caches for disk drives and file systems, Internet routers use

caches for storing recently used routes and Domain Name System (DNS) servers use caches to store hostname-to-address lookups. Similarly, caches are used by Web browsers, proxy servers and reverse proxy servers to store recently used Web objects to reduce both latency and network traffic in accessing the World Wide Web.

Caching is based on a phenomenon called *locality of reference*. This could be divided into *temporal locality* and *spatial locality*. Temporal locality means some pieces of data are more popular than others. Spatial locality means requests for certain pieces of data are likely to appear together [2].

First, Web Caching meant that each client maintained its own cache called the *Browser Cache* to temporarily store frequently accessed web objects. However, since the benefits of caching are more when a number of clients share the same cache, the caching proxy was developed and used [3], [4]. A caching proxy services its clients from its cache whenever possible and retrieving the objects from origin servers if required. Unfortunately, a single *caching proxy* introduces a new set of problems, namely those of scalability and robustness, since a single server is both a bottle neck and a single point of failure [5]. Scalability to a large number of clients is important. This is because when more clients share a single cache, there is a higher probability of getting a cache hit [1].

In certain situations, it is beneficial for caches to communicate with each other. This is called co-operative Web caching in general. The concept of co-operative Web caching aims to minimize certain problems associated with single caching proxy servers.

Even with cache co-operation in place, a proxy cache can become a performance bottleneck due to the limited request service rate. Due to convergence of requests

from many clients in to one network node, the network could experience high congestion. The peak bandwidth demand could be several folds higher than the average bandwidth demand on that network link. Since a proxy server also introduces a single point of failure to the network, and requires to handle peak load bursts, such systems have to be over provisioned by employing costly dedicated servers with high performance and high reliability.

Therefore, it is desirable to explore the possibility of designing a web caching system that does not solely rely on proxy caches for its functionality.

1.2 Research Scope

This study has following objectives:

1. identify the issues and constraints in existing co-operative web caching architectures.

2. propose a possible solution to overcome such issues and improve performance using a distributed systems approach.

3. show its viability and verify its performance both mathematically and experimentally.

1.3 Issues not covered in this thesis

- Issues of security and data privacy arising from cache co-operation and in using the proposed Distributed Web Caching (DWC) protocol.

- Issues arising from dynamic data and methods of caching such data.

- Active object replication and object pre-fetching [6].
- Off-line cache information dissemination.
- Centralised mechanisms of object discovery and delivery.

1.4 Thesis Contributions

By introducing cache co-operation to browser caches, it is possible to provide the functionality of a Distributed Web Cache. Such a system needs no extra hardware or administration and is fault resilient. Work reported in this thesis investigates the possibility of applying co-operative web caching techniques to browser caches to form a distributed cache in a corporate LAN type environment with 100 to 10,000 client nodes and which is confined to a single geographic region. This is a completely novel approach to web caching.

In the proposed scheme, every client node in the network takes on the additional role of a cache server. By becoming a cache server, each client node is able to accept and service incoming requests for web objects. Each client node is therefore able to request web objects from all other clients, if a particular object is not locally available. This is similar to the role of institutional caches in conventional co-operative web caching architectures.

The proposed Distributed Web Caching system could be classified as a pure peer-to-peer web caching system [7].

A comprehensive set of protocols for access, storage, and serving functions is developed. The proposed protocol guarantees data consistency between the original server object and that in the cache. Due to the totally distributed nature of the

design, an increase in number of client nodes corresponds to an increase in the amount of shared resources and therefore an increase in reliability. The system does not rely on centralised servers, improving scalability.

We provide analytical models to evaluate and study the Distributed Web Caching proposed.

A software realization of the proposed system is implemented on the Linux operating system[1], and the performance of the system is studied on a test bed. Further, a simulation model for Distributed Web Caching is developed.

We also explore simulating cache performance under constrains such as limited local storage, slow connection times, varying object sizes and access costs and unreliable client nodes. With limited cache storage on each client node, object replacement schemes play a significant role in determining cache performance.

We extend our analysis to object replacement strategies in Distributed Web Caching both to compare performance of object replacement strategies and to investigate how performance can be improved. This has resulted in several novel replacement strategies of improved performance.

1.5 Publications

Portions of this thesis appear in following papers:

- T.T. Tay, Y. Feng and M.N. Wijesundara "A distributed Internet caching system", in *proceedings of 25th Annual IEEE Conference on Local Computer Networks (LCN 2000) 2000*, pp. 624 -633, 2000.

[1]Subsequently this implementation was ported to Microsoft Windows operating system by Ng Jiah Hui, Department of Electrical and Computer Engineering, National University of Singapore.

- Chapter 4

- T.T.Tay and M.N.Wijesundara "A Replica-Aware extension for replacement algorithms in Distributed Web Caching", in *Communications World*, edited by N. Mastorakis, Athens: WSES, pp. 279-286, 2001.

  - Chapter 6

- M.N.Wijesundara and T.T.Tay "Distributed Web Caching", in *proceedings of the 8th International Conference on Communication Systems (ICCS 2002), 2002*, Volume: 2 , pp. 1142 -1146, 25-28 Nov. 2002.

  - Chapter 5

- M.N.Wijesundara and T.T.Tay "An object replacement strategy for global performance in Distributed Web Caching", in *proceedings of International Conference on Communication Technology (ICCT 2003) 2003*, Volume: 2, pp. 1687 -1690, April 9 - 11, 2003.

  - Chapter 6

- T.T.Tay and M.N.Wijesundara "Distributed Web Caching", submitted to *IEE Communications*, 2004.

  - Chapters 1 to 5

1.6 Thesis Organisation

The thesis is organized as follows: Chapter 2 provides background information on web caching. Commonly used cooperative web caching schemes are introduced and

inter cache communication protocols and techniques are discussed in detail. Issues relevant to web caching and related research work are discussed. Chapter 3 is a case study based on a real client access trace collected at Boston University and University of California, Berkeley. This case study provides the motivation for developing our Distributed Web Caching system. Chapter 4 provides a detailed description of the proposed Distributed Web Caching model and the protocol, followed by the properties of the proposed caching system. Implementation aspects of the proposed system is also discussed. Details of the software realization are presented. Performance of the implemented system is studied. This chapter also includes an explanation of experiment environment, methodology and results.

Chapter 5 is a mathematical study of the Distributed Web Caching system. This includes a mathematical analysis into the movement of an object within the LRU stack. The speedup due to Distributed Web Caching is studied and an upper bound on speed up is derived. A simulation model for the system is also developed. The model is able to simulate different access patterns, object popularity, inter-node popularity correlation, object size, cache capacity, access cost of objects and access delays. Chapter 6 introduces the topic of object replacement and its importance under limited cache capacities. A "replica aware" extension for existing replacements and a novel scheme for "Distributed Web Caching for Global Performance" abbreviated as DWCG is introduced. Chapter 7 concludes the thesis.

Chapter 2

Web Caching

2.1 Introduction to Web Caching

Caching refers to the concept of temporary storage of commonly accessed computer information for possible future reference. This simple concept has proven to be a solution for the scalability issue of the World Wide Web, caused by exponential growth [8], [9], [10], [11]. Web browsers routinely cache recently accessed objects in *Browser Caches* using main memory and local disk storage. Special Cache servers called *Web Caching Proxies* are often used to provide a shared cache to multiple web browsers. A typical institutional proxy cache configuration is shown in Fig 2.1. Browsers first attempt to satisfy the requests from their built-in browser caches. Unresolved requests are forwarded to the institutional cache. Institutional cache then tries to satisfy the requests from its local cache. Unresolved requests are then forwarded to origin servers or to the higher level cache depending on the design of the caching architecture and the configuration of the institutional cache.

For the web browsers, the cache server acts as a web server while for the web

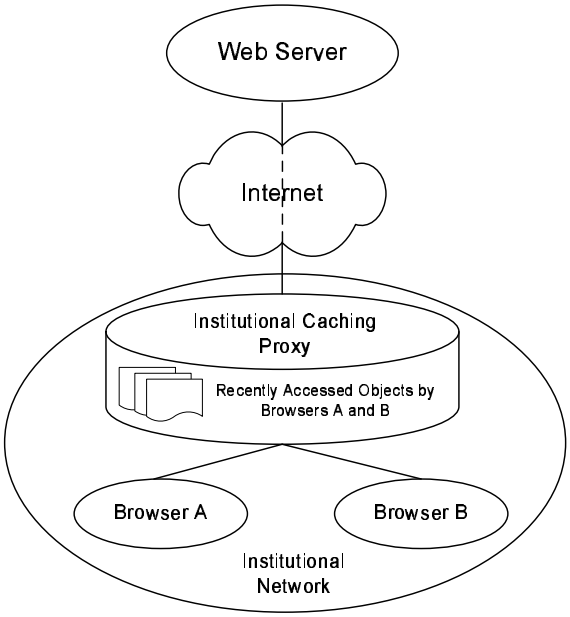

Figure 2.1: An institutional web caching proxy server

server, the cache server is a client when it requests for objects on browsers' behalf. This dual role as a substitute server and client has given rise to the name *'proxy'*. For caching to be effective, the following conditions must be satisfied:

- Client requests must exhibit both spatial and temporal locality of reference.
- The cost of caching must be less than the cost of direct retrieval.

There are three primary benefits of web caching:

- To speed up retrieval of web content by reducing latency.
- To reduce wide area bandwidth usage.
- To reduce load on origin servers.

2.1.1 Retrieval Latency

There are several causes for delays in transmission of data from one point to another. Theoretically, the transmission speed of data over electrical or optical circuits are limited by the speed of light. However, in practice electrical or optical signals will reach only two thirds of the theoretical bound. Transoceanic delays are in the 100ms range.

Network congestion could be another source of latency. When links are close to full utilization, data packets experience queuing delays at routers and switches. There could be a number of points where queuing can occur depending on the length of the link and complexity of the network. When queues are full, the devices are forced to discard data packets. The Hyper Text Transfer Protocol (HTTP) uses the Transmission Control Protocol (TCP), which is able to recover from such a loss by retransmitting lost data. However, even a relatively small amount of packet loss will have a dramatic impact on throughput.

By having web caches closer to web browsers, the transmission delay is reduced due to shorter distances between end points. Shorter communication links require fewer routers and switches between end points. Therefore, congestion and hence packet loss due to queuing is minimized.

If properly designed, a cache miss should not be delayed much longer than a direct request from origin server. Therefore, cache hits reduce the average latency of all requests.

2.1.2 Bandwidth Usage

In a multiuser environment, total HTTP traffic could sum up to a substantial portion of the total bandwidth usage. By locating a cache server at the gateway of the local area network, wide area bandwidth usage could be reduced. Every Cache hit could save wide area bandwidth allowing more bandwidth available for other protocols and applications. In certain countries, bandwidth usage is metered and therefore bandwidth usage reductions will directly result in cost reductions.

Because of this reduction of bandwidth usage, those documents that are not cached, can also be retrieved faster due to less congestion along the path.

2.1.3 Origin Server Load

By intercepting and fulfilling a portion of web requests, proxy cache servers could effectively reduce the load on the origin servers. By using reverse proxy caches, which are located in front of the origin servers to offload some of the HTTP content delivery duties of the origin servers, loads on origin servers could be further reduced and performance could be improved.

2.1.4 Secondary Benefits

If origin server is not available to service a particular request due to some reason, it is possible to obtain a cached copy at the proxy cache. Therefore, robustness of the web service is enhanced [12].

Another advantage of web caching is the ability to analyse institutional web usage patterns by logging requests and responses at the server. A caching proxy server can also be used for content filtering and internet access authentication.

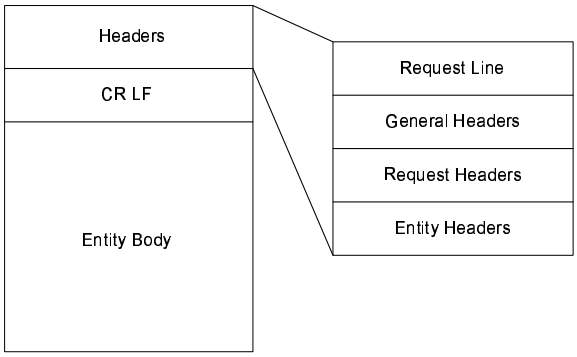

Figure 2.2: The HTTP 1.1 request format

2.2 The HyperText Transfer Protocol (HTTP)

HTTP carried around 70% of the internet traffic in 2002 [13]. HTTP is a simple request-response protocol, which uses URLs (Uniform Resource Locators) as unique identifiers of objects. HTTP 0.9 (1991) [14] was the first widely used version of the protocol which was subsequently replaced by HTTP 1.0 (1996) [15] and HTTP 1.1 (1999) [16].

An HTTP message consists of an HTTP header and an entity body, the two being separated by a carriage return (CR) followed by a line feed (LF). HTTP request is issued by the client and the response is issued by the server. In both request and in response, header field is encoded in clear text (ASCII).

2.2.1 The HTTP Request

The layout of an HTTP 1.1 request is shown in Figure 2.2. The request line is followed by multiple general, request and entity header fields, optionally followed by

an entity body, which contains any data that the user would like to upload to the server. Header fields describe the client capabilities, authorization credentials, and other information that helps in fulfilling the request.

Request line contains the request method, the path part of the URL, and the version of the protocol. Possible request methods are described in section 2.3.1.

A request sent to origin server requesting *http://www.nus.edu.sg/index.html* would be as follows:

```
GET /index.html HTTP/1.1
Host:  www.nus.edu.sg
Accept:  */*
Connection:  Keep-alive
```

An entity body will only be present when POST and PUT methods are used. Otherwise a HTTP request only contains request headers.

2.2.2 The HTTP Response

Similar to the HTTP request, the HTTP response contains a status line, followed by general headers, response headers, entity headers, CR and LF, and an entity body. Response headers are discussed in more detail in Section 2.3.2.

A typical response would look like this:

```
HTTP/1.1 200 OK
Date:  Mon, 12 Mar 2001 19:12:16 GMT
Last-Modified:  Fri, 22 Sep 2000 14:16:18
ETag:  "dd7b6e-d29-39cb69b2"
```

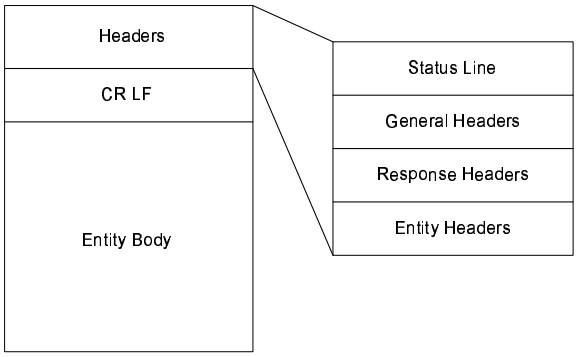

Figure 2.3: The HTTP 1.1 response format

```
Connection:  close
Content-Type:  text/html

<html>
<head>
<title>
Hello World
</title>
</head>
<body>
Hello World
</body>
</html>
```

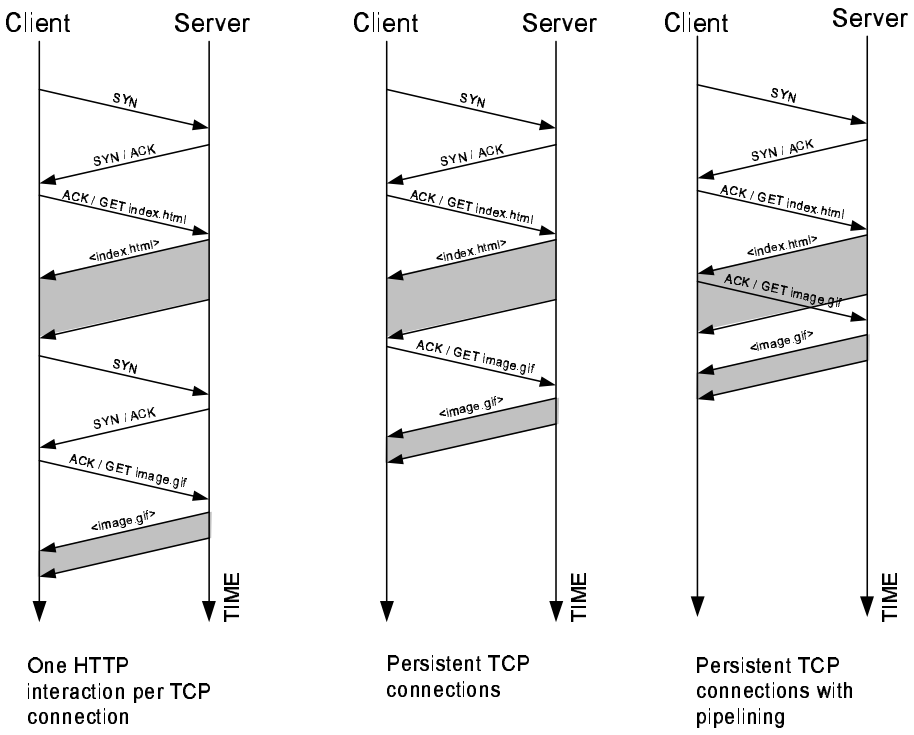

Figure 2.4: The TCP level message exchange in a HTTP transaction (termination not shown)

2.2.3 The HTTP Message Transaction

The HTTP protocol is *stateless* due to its simplicity. Both the server and the client do not require to record any information beyond the boundaries of the simple request-response transaction.

Shown on Figure 2.4 is the TCP message exchange in a simple HTTP interaction. There are 3 possible modes. The first mode is default for HTTP 0.9. The second mode is possible only with HTTP 1.0 and upwards while the third mode was introduced in HTTP 1.1. The first mode establishes and closes the TCP con-

nection for each request, while the second mode has persistent TCP connections so that more than one object can be requested after the TCP connection is established. Connection is only closed when the download is complete. In the third mode where pipelining is allowed, a second request can be made while the server is still responding to an earlier request.

2.3 HTTP Support for Web Caching

The HTTP/1.1 protocol specification (RFC 2616) includes a number of elements intended to provide better support for caching.

The HTTP request, shown in Section 2.2, when requested from a caching proxy server is as follows:

```
GET http://www.nus.edu.sg/index.html HTTP/1.1
Host:  www.nus.edu.sg
Accept:  */*
Proxy-connection:  Keep-alive
```

Here, the first line contains the full address of the resource compared to the relative address in a direct request, while last line contains a `Proxy-connection: Keep-alive` directive. `Proxy-connection` and `Proxy-authorization` directives are used on a hop-by-hop basis to control the persistent TCP connections and to provide credentials for access to the proxy (not the origin server) respectively.

Cache server decides the cachability of responses from the origin server depending on the following components of the request and response.

- Request method

- Response status code
- `Cache-control` directives
- Response validator
- Request authentication

In addition to the above, some heuristic based rules can be set up by the cache administrator to cache or not to cache specific object types and responses.

2.3.1 Request Methods

As shown in Table 2.1, out of seven possible request methods, only three request methods are cachable [17]. Those are namely: GET, HEAD and POST. GET is the most popular request method followed by HEAD and POST. HEAD response could return the new expiration time or it could indicate that the object has expired. POST requests are not cachable by default. This could be overridden by using a `Cache-control` directive or by including a expiration time, both of which are rare in practice.

2.3.2 Response Status Codes

Response status code is an important factor that determines the cachability of the response. Common status codes can be divided into five categories as shown in Table 2.2. Status code 200 (OK) is the most common response, which indicates successful processing of the request. Table 2.3 indicates the cachable HTTP status codes. Responses with status code 304 are not cachable by the proxy, if the object already resides in the browser cache.

Table 2.1: HTTP/1.1 Request Methods and Cachability

Request Method	Cachability
GET	Yes
HEAD	May be used to update a previously cached entry
POST	No unless `Cache-control` headers allow
PUT	No
DELETE	No
OPTIONS	No
TRACE	No

Table 2.2: HTTP/1.1 Server Response Status Code Categories

Status Code	Response Category
1xx	Informational:
	Transaction is being processed.
2xx	Successful:
	Successfully received and processed.
3xx	Redirection:
	Client is redirected by the server to a different location.
4xx	Client error:
	There is an error with client request.
	For example: authentication required or resource does not exist.
5xx	Server error:
	An error occurred on the server while processing the request.

Table 2.3: Cachable HTTP Status Codes

HTTP Status Code	Description
200	OK
203	Non-Authoritative Information
206	Partial Content
300	Multiple Choices
301	Moved Permanently
304	Not Modified
410	Gone

2.3.3 Expiration and Validation

There are two ways caches can maintain consistency of cached objects with origin servers. Those are namely expiration times and validators.

Expiration is based on the concept of *time to live (TTL)*. An HTTP server can provide an explicit time to live value of each object using the `expires` and `max-age` headers. The `expires` header provides the date up to which the cached object may be considered valid. The `max-age` header is a `cache-control` directive which is described in detail in Section 2.3.4. The `expires` header provides the absolute time of expiry while the `max-age` header header expiry is relative to the time the object left the origin server or the time it was last validated with the origin server. Due to difficulties in age determination, especially when the object has to travel through multiple caches, an `age` header is inserted by the proxy that fetched the object from origin server. The subsequent proxies then update the `age` header depending on the time the object was fetched and the time it spent in the cache.

Table 2.4: Possible `cache-control` directives in an HTTP request

Directive	Value	Description
`no-cache`	none	Cached objects cannot be used to satisfy the request.
`no-store`	none	Response to this request cannot be cached.
`max-age`	seconds	Only younger cached objects can be used.
`min-fresh`	seconds	Only cached objects that will not expire for a specified time can be used.
`max-stale`	seconds	Only cached objects that expired up to the specified time ago can be used.
`no-transform`	none	Only the exact response given by the origin server can be used.
`only-if-cached`	none	A proxy should not forward the request on a cache miss.

2.3.4 `cache-control` directives

The `cache-control` header is a feature introduced in HTTP/1.1 that contains multiple directives that control the use of caches along the path from a requesting browser to an origin server. HTTP/1.0 contained `pragma: no-cache` header, which forces the caches to reload a new copy from origin server. However, its functionality was not sufficient to provide the flexibility required. HTTP/1.1 `cache-control` headers could be present in both requests and responses. Possible `cache-control` directives in a HTTP request and a HTTP response are shown in Table 2.4 and Table 2.5 respectively.

Table 2.5: Possible `cache-control` directives in an HTTP response

Directive	Value	Description
`no-cache`	none	The response cannot be cached.
`no-store`	none	The response cannot be stored in any client (proxy or browser).
`max-age`	seconds	A cache must validate this object before serving once the object reaches the specified value.
`s-maxage`	none	Same as `max-age` but applies only to proxies.
`no-transform`	none	Only the exact response given by the origin server can be used.
`private`	seconds	The response can be used only for the client that originally requested the object.
`public`	seconds	The response may be cached and used for any client.
`must-revalidate`	none	A cache must always validate this object before using it.
`proxy-revalidation`	none	Same as `must-revalidate` but only applicable to proxies.

2.3.5 Validation

When a client requests for an object from a cache and if the cached copy is stale, the cache first has to check with the origin server (or possibly an intermediate cache with a fresh response) to see if its cached entry is still usable before choosing to serve that object. This process is called validation [13].

The `last-modified` header in HTTP responses indicate when the resource was last modified at the origin server. An example is shown below.

```
HTTP/1.1 200 OK
Date:  Mon, 10 Nov 2003 03:00:00 GMT
Last-Modified:  Sun, 09 Nov 2003 02:30:45 GMT
```

In order to avoid retransmission of the full response, if the cached object is not stale, the HTTP/1.1 protocol supports the use of conditional methods.

A `If-modified-since` header is used in a conditional GET request in order to validate an object.

```
GET http://www.nus.edu.sg/ HTTP/1.1
If-Modified-Since:  Sun, 09 Nov 2003 02:30:45 GMT
```

An `ETag` header is an alternative to `If-modified-since` header. `ETag`, which stands for "entity tag", is a unique identifier of a specific instant or version of the object. For example origin may respond with:

```
HTTP/1.1 OK
ETag:  "nsdk32-lgt238-kj12knd"
```

When the cache is required to validate a particular object, `If-none-match` header is used. For example a cache could request:

```
GET /index.html HTTP/1.1
If-None-Match:  "nsdk32-lgt238-kj12knd"
```

If the `ETag` of the particular object at the origin server is the same, the server could respond with 304 (Not Modified). If the object has been modified it will respond with a 200 (OK) response followed by full content of the object.

`ETag` is considered as a strong validator whereas `last-modified` timestamp is considered as a weak validator since timestamp only provides single-second resolution.

2.3.6 Authentication

`WWW-Authenticate` header and `Authorization` header are used when accessing a protected resource. These requests and responses are not normally cachable. Initially an origin server will respond with a 401 (Unauthorized) status code along with `WWW-Authenticate` header, which contains a challenge such as a user name and password. Browser then resubmits the request along with access credentials using a `Authorization` header. Intermediate caches will not cache the responses unless explicitly specified by the origin server.

2.4 Issues in Web Caching

Benefits in web caching are related to the size of the client population served by the cache. However, increasing the client population could lead to the following scalability issues.

1. Load scalability

   Request Processing Power: When the number of clients served by a cache increases, the rate at which the requests have to be processed also increases. This is limited by the request processing power of the server. Server hardware architecture, hardware resources, operating system and efficiency of caching software determine the maximum request rate it could service.

   Cache Capacity: The amount of data downloaded by a particular client in a typical session could range from a few megabytes to a few gigabytes. When this amount is multiplied by the number of users, the resulting total capacity required is extremely large. Ideally, in order to maximize cache hits, as many documents as possible should be cached, for as long a period as possible. However, in practice, cache storage capacity is limited. In the event that the storage capacity is not sufficient, replacement algorithms are used to determine the documents to be replaced.

   Network Bandwidth: A one-to-multipoint cache system, where a single cache server serves many clients, may lead to a major congestion problem. This is especially so, for a cache server that has a large cache capacity. Since the cache capacity is large, the probability of cache hits is high. All the requests and responses thus concentrate at the cache server node, leading to a high peak

bandwidth demand at the cache server and the attached communication link. When designing a cache system, it is important to factor in the requirement for this peak bandwidth demand at the server node, in order to ensure a non-blocking system, or at least to maintain a certain minimum quality of service. The larger the number of documents a cache server caches and the larger the number of clients a cache server serves, the higher peak bandwidth is required at the server node. To fulfil this peak bandwidth demand, would mean an excess of resources during off-peak period. A compromise has to be found somehow.

2. Geographical scalability

   When the client population is scattered in a large geographical area, due to considerable communication delays between the proxy and the clients, the advantage of fetching from the cache is reduced. Although not directly related to the size of the client population, reliability and cache consistency are issues of concern in web caching.

3. System Reliability

   Usually in a network, all the HTTP requests are sent through the local proxy-cache server. Hence, if the cache server fails, the requests will not be able to bypass the cache. This is because the network architecture is fixed and the requests are not dynamically routed to avoid possible failures. Therefore, the reliability of the cache server becomes a major concern.

4. Cache Consistency

   This is a major issue in any web caching strategy. Once a document is cached,

there is no guarantee that the original document will remain unchanged. In most cases, the document at the original server is updated without the knowledge of the caches that have an older version of the document. In this situation, when the client requests for the document a stale document will be returned. This is a serious problem especially when many web sites today are moving towards delivery of real-time information instead of serving as static archive of information.

The solutions available for cache consistency fall into one of two categories, namely weak consistency and strong consistency. Weak consistency and strong consistency are achieved using the expiration and validation mechanisms that are introduced in section 2.3.3.

Weak consistency is achieved by using a Time-To-Live (TTL) concept employing `cache-control` directives. If the TTL has not lapsed, the cached copy is considered up-to-date and is delivered to the client. Otherwise, the copy is discarded and a new copy is fetched from the origin server.

Strong consistency could be achieved by polling using a `If-modified-since` or a similar HTTP header or by using an invalidation protocol [18]. Polling involves the cache checking the validity of the document from the origin server each time it is requested. Invalidation protocol involves the original server keeping track of all the caches where the document is cached and then sending an invalidation command to the caches, once the document is updated.

2.5 Co-operative Web Caching

Co-operative web caching solves the load scalability issue by sharing the client requests among several caches and geographical scalability issue by servicing requests from locality of each cache server.

If a particular cache does not contain the requested object, co-operative web caching provides the means to query other caches before requesting from origin server.

2.5.1 Co-operative Web Caching Architectures

A caching architecture should provide the paradigm for caches to co-operate efficiently with each other [12]. There are two main caching architectures.

1. Hierarchical caching architecture

   Hierarchical caching was pioneered in the Harvest project [19]. In hierarchical caching, caches are placed at multiple levels of the network. In a hierarchical cache system, a cache locates a missing requested object by issuing a request to the cache at the hierarchy's upper level. The process is iterated until the object is found or the request reaches the root cache. If the object is not found cached at any of the upper levels of the cache hierarchy, the object is fetched from origin server. The object is then cached at all cache servers along the path it traverses from origin to client.

2. Distributed caching architecture

   As mentioned in Chapter 1, in a loosely coupled collection of caches if the topology is flat and ill defined it is called a *mesh*. A mesh configuration where

the caches are placed only at the bottom level of the network (institutional level) where there are no intermediate caches is called a *Distributed Cache*. Therefore, the distributed caching architecture is a sub set of the mesh caching architecture.

The main difference in distributed caching compared to hierarchical caching is that instead of trying to retrieve a missing object from the parent cache, the request is sent to a set of sibling (peer) caches. Since Distributed Web Caching allows to request a missing object from multiple caches, the reliability problem mentioned in section 2.4 could be solved by employing a distributed caching architecture.

2.5.2 Cache co-operation protocols

Protocols employed for co-operative web caching could be categorized as follows [20]:

1. Message based protocols

   Message based protocols use a query / response dialog for exchanging information about cache contents.

   The Harvest project which first proposed cache sharing [19], also designed the Internet Caching Protocol (ICP) [2], [21], [22], which is a message based protocol that supports discovery and retrieval of documents from neighboring caches. Hypertext Caching Protocol (HTCP) is similar to ICP since it also uses a User Datagram Protocol (UDP) based query-response protocol. However, HTCP uses much more information than ICP to decide whether a request would be a hit or a miss and it also supports strong authentication. The downside is that the message format is significantly more complex [2].

Malpani *et al.* proposed a co-operative distributed cache [1], which involves the client randomly choosing a master cache which takes care of locating the object within a group of caches using multicast. Master cache then redirects the client to the cache that is holding the object. If that attempt fails, it retrieves the object from origin server and passes it to the client.

2. Directory based protocols

   Directory based protocols [23],[24],[25],[26],[27] summarize information into frequently exchanged directories. In Summary Cache [24] which is also known as Cache Digests [23], Relais project [25], and in SCOOP [26], caches inter-exchange messages indicating their content, and keep local directories to facilitate finding documents in other caches.

   Povey and Harrison also proposed a Distributed Internet Cache [28]. In their scheme, upper level caches are replaced by directory servers, which contain location hints about the documents stored in every node.

   Tewari et al proposed a similar approach to implement a fully distributed Internet cache where location hints are replicated locally at institutional caches [29].

3. Hash based based protocols

   These protocols [30], [2],[31], [32],[33], [34],[22], [35], [36], [37],[38] employ hash functions to distribute URL space among co-operating web caches. Cache Array Routing Protocol (CARP) [30], [2] is a hash based protocol, which divides the URL-space among an array of loosely coupled caches and lets each node store only the documents whose URLs are hashed to it. In the central

directory approach (CRISP) [33], a mapping service ties together with a certain number of caches.

4. Router based protocols

   Router based protocols [39], [40],[20], [41] intercept web traffic at the IP layer and redirect it to a caching proxy. The Web Cache Coordination Protocol (WCCP) [20] is a router based protocol.

Some researchers have proposed protocols that combine features of multiple categories [42], [43],[44],[45].

2.5.3 Internet Cache Protocol (ICP)

ICP is the original intercache protocol [2]. Its primary purpose is to discover whether any neighbor caches have a fresh copy of a particular object. The neighbor caches answer with either yes (HIT) or no (MISS). Since UDP is used for ICP communications, if there is no response from a cache within a certain period of time it could be concluded that the cache is offline or the network is severely congested. The protocol can be configured to be used in complicated cache hierarchies with multiple parents and sibling caches as neighbors. Despite its shortcomings ICP is in widespread use due to wide area bandwidth savings and because ICP is supported by all major caching products.

Following issues have to be considered before deciding to use ICP for cache cooperation:

- Access Delays

  Let us assume the following:

S_o = Average HTTP service time from origin servers

S_n = Average HTTP service time from neighbor caches

R_n = Average ICP service time from neighbor caches

P = ICP hit ratio

Then we could do the following analysis:

The average service time without ICP is:

$$S_o \tag{2.1}$$

The average service time with ICP is:

$$PS_o + (1 - P)S_o + R_n \tag{2.2}$$

We are interested in the case when:

$$PS_0 + (1 - P)S_o + R_n < S_o \tag{2.3}$$

Which reduces to:

$$P > \frac{R_n}{S_o - S_n} \tag{2.4}$$

When the ICP hit ratio is larger than the ratio of the ratio of the ICP service time to the difference in HTTP service times, then the benefits of the neighbor hits are greater than the costs of ICP queries.

In certain cases, primary focus could be to reduce wide area bandwidth usage, irrespective of service times.

- Bandwidth Usage

Let us assume the following:

L_{ICP} = length of ICP query

L_{URL} = Average length of the URL

n = Number of siblings

Total size of a query and response:

$$(L_{ICP} + L_{URL}) * 2 * n \tag{2.5}$$

If we substitute 24 bytes as the length of ICP query and 55 bytes for the average length of a URL [2], we could obtain the following as the size of the transaction before adding UDP and IP headers:

$$(79) * 2 * n = 158 * n \text{ bytes} \tag{2.6}$$

- Other Issues

  There are several other issues to be considered. Since in an ICP request only contains the URL, and no information on additional headers such as `cache-control` headers, there is a possibility of stale objects in sibling caches returning ICP HIT responses.

  Use of UDP for the protocol could be a drawback, since UDP does not have a built in feedback mechanism when links are congested.

  Since ICP does not indicate the request method in its queries and HTTP GET method is assumed. This usually not an issue since non-GET requests are usually uncachable.

2.5.4 Summary Cache

It is possible to reduce forwarding delays and false hits by having apriori knowledge of cache contents of neighbor caches. By employing an algorithm called Bloom filter [24], [23] the cache contents could be represented in a compact summary. These summaries are then exchanged between neighbors and kept up-to-date using periodic updates.

Given the Bloom filter of a set of X URLs, one can efficiently but imprecisely determine whether a given URL belongs to the set. This is achieved by comparing the Bloom filter for just one URL and the Bloom filter for the set of X URLs. If all unity bits in the former matches with the unity bits in the latter, it is assumed that the URL belongs to set X.

- Bandwidth Usage

  Let us assume the following:

  $S * 2^{30}$ = Cache size in bytes

  $L * 2^{10}$ = Mean object size in bytes

  $T * 3600$ = Digest update period in seconds

  R = Average query rate per peer

  If the summary is sized such that a cached object is represented by b bits, the bandwidth that the summary would use in bytes/second could be calculated as:

$$\frac{b}{8} * \frac{S}{L} * \frac{2^{20}}{3600 * T} \tag{2.7}$$

  The ICP bandwidth requirement would be:

$$2 * 79 * R \tag{2.8}$$

If we substitute $T = 1$, $S = 100$, $L = 10$ and $b = 40$ [13] and solve the two equations for R we get $R = 92.17$. The bandwidth requirements for the two cases considered here are the same if the ICP request rate is 92.17 requests per second. If it exceeds 92.17, then summary cache could save bandwidth while searching for cached objects. Conversely, if the request rate is lower, ICP traffic will be lower. Summary cache requires the same bandwidth irrespective of the request rate.

- False Positives

  Bloom filters are imprecise because there is a remote chance that the test will erroneously indicate that a URL belongs to a certain set, when in fact it is not. This occurrence is called a *false positive*. In order to keep the probability of occurrence of false positives low, size of the filter should be in the order of 40bits times the maximum number of objects in the set [13].

In terms of system resource usage Summary Cache requires more CPU processing to calculate hash functions and requires to store the summaries in the main memory for fast lookups. In contrast, ICP requires much less CPU processing and requires virtually no main memory due to its stateless nature. Also it is impossible to use Summary Cache on slow links as transfer of large summaries could saturate the links.

2.6 Chapter Summary

Web Caching has proven to be a solution for the scalability issue of the World Wide Web caused by its exponential growth. For web caching to be efficient, HTTP has to support, conditional requests and validation. HTTP support for web caching has been improved in HTTP 1.1. There are issues in web caching such as its scalability, reliability and consistency. Some of these could be solved using co-operative web caching. Co-operative web caching is mainly divided into hierarchical and Distributed Web Caching. Previously proposed Distributed Web Caching systems involve proxy cache corporation using centralised control, mapping of objects to be cached using a particular criterion or cache information replication among proxy servers.

Chapter 3

A Case Study of Web Access Patterns

3.1 Introduction

A number of researchers have carried out extensive studies on traces of World Wide Web traffic and their characteristics [46], [47], [48]. In this chapter, characteristics of a few selected traces are studied to determine the benefits of Distributed Web Caching. In particular, temporal and spatial locality of web requests from each browser and the correlation of web requests among all browsers, are key factors that determine performance of Distributed Web Caching.

Out of the three traces that are studied, two are from Boston University (BU) [49] and the other is from University of California, Berkeley (UCB) [50]. The traces are ideal for this study since the client populations in all three traces are more likely to have common interests compared to traces collected at commercial Internet Service Providers. Such common interest groups are likely to produce web requests

with higher correlation.

In order to investigate the applicability of Distributed Web Caching to such a focused group of users, 3 different web caching strategies are simulated using the trace data.

3.2 Nature of Traces

Traces B19 and B272 are collected in the general computing environment at the Computer Science Department of Boston University. The data collection environment consists of 37 *SUN SparcStation 2* workstations. 5 machines are used by the postgraduate students in room 272. The requests originating from room 272 appear in trace B272. The rest of the 32 machines are used by undergraduates in room B19. The requests originating from those machines are recorded in trace B19. B19 contains a broader spectrum of users both students doing homework and less-focused users. Both traces B272 and B19 are unique as they are recorded using a modified version of Mosaic web browser. Since the recording is done at the browser level, these two traces contain the user ID, session ID, machine ID and the browser cache hit status among other standard details of the web requests. One month long segments of each trace were chosen for evaluation purposes in our study.

Trace UCB1 is recorded from the Home IP service offered by UC Berkely to its academic community. We use a 4-hour portion of the trace for evaluation. These client traces were unobtrusively gathered through the use of a packet sniffing machine placed at the head-end of the Home IP modem bank. Therefore, this trace does not contain the user ID, session ID and the browser cache hit status. However, users and machines can be identified using their IP addresses, since fixed IP's

were used. It is possible to assume that each user has a dedicated machine with a dedicated IP. In addition, the trace also provide information about HTTP headers including conditional requests and cache control headers. A summary of the 3 traces is provided in Table 3.1.

Table 3.2 summarises some important characteristics of the traces, with regard to caching and cache sharing.

- Zipf Parameter

  Zipf's law states that the relative probability of a request for the ith most popular page is inversely proportional to i. This can be expressed mathematically as follows:

  $$P = \frac{\Omega}{\rho^{\alpha}} \tag{3.1}$$

  where, P is the frequency of the requests to a particular page, ρ is the popularity rank of the particular page, α is the Zipf parameter [46] and Ω is a positive constant. According to Zipf's law $\alpha = 1$. Cunha *et al.* [49] found that the request probability for a web cache trace, when fitted with a curve of that adheres to Zipf's law, yields a curve with an exponent of $\alpha = 0.982$, which is very close to the value of $\alpha = 1$ in Zipf's law.

  Figure 3.1 illustrates the adherence to the Zipfs law by the traces considered. Once extrapolated, the ratio between the logarithm of the y-intercept and logarithm of the x-intercept gives the value of Zipf parameter for the trace.

- Hot-set

  If 30% of the web requests are made to the hottest 5% of the pages, it is typical to think of this as an instance of Zipf's law, and call it a 30/5 rule [51].

Table 3.1: Summary of Traces

Trace	B272	B19	UCB1
Trace Duration	1 month (Feb 1995)	1 month (Feb 1995)	4 hours (Nov 1997)
Total Sessions	139	3492	NA
Total Distinct Users	29	525	916
Total Machines	5	32	916
Total HTTP Requests	15306	444459	94438

Table 3.2: Other Characteristics of Traces

Trace	B272	B19	UCB1
Hot-set	56/5	79/5	33/5
Zipf parameter	0.98	0.98	0.69

In Chapter 5 Zipf parameter is used to model object popularity.

3.3 Simulation of Web Caching Strategies

3.3.1 Simulation Setup

All traces are first imported into a MySQL relational database. Then selected data sets are imported into MATLAB using SQL and MATLAB database toolbox. Secondary analysis is carried out in MATLAB.

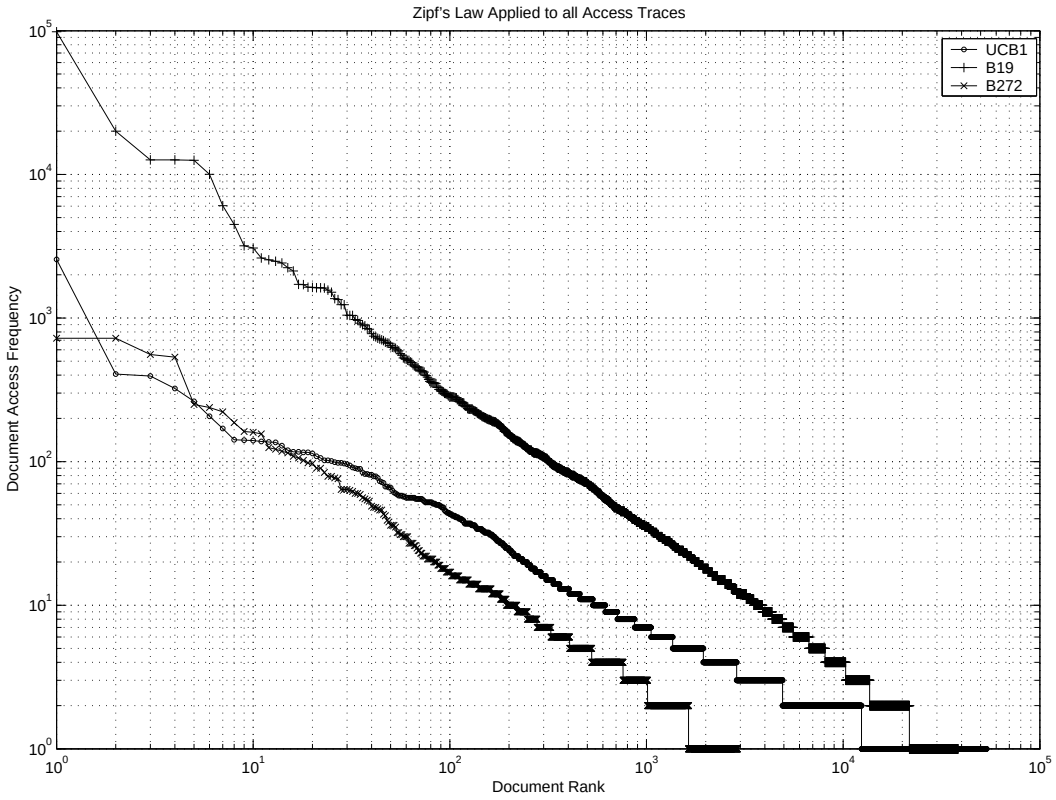

Figure 3.1: Zipf's Law Applied to HTTP Access Traces

Table 3.3: Caching Strategy 1

Trace	B272	B19	UCB1
Total Requests	15306	444459	NA
Sum of distinct requests from each node by each user	3579	80286	NA
Hit Ratio	76.62%	81.94%	NA

3.3.2 Caching Strategy 1

Caching strategy 1 simulates the environment of a student computer center at a university. Each user is allowed to maintain a browser cache in each machine. However, there is no information sharing between those caches. Since UCB1 trace does not distinguish between users and machines, this simulation is not applicable to UCB1 trace. Hit Ratio (HR) for this strategy is expressed in Equation 3.2.

$$\text{Hit Ratio} = \frac{\text{total requests} - \text{sum of distinct requests from each machine by each user}}{\text{total requests}} \tag{3.2}$$

3.3.3 Caching Strategy 2

In this simulation, it is assumed that each user is accessing the web from a dedicated machine. Therefore, in this case, each user has a dedicated browser cache. There is no information sharing between browser caches in different machines. Hit Ratio (HR) for this strategy is expressed in Equation 3.3.

$$\text{Hit Ratio} = \frac{\text{total requests} - \text{sum of distinct requests by each user}}{\text{total requests}} \tag{3.3}$$

Table 3.4: Caching Strategy 2

Trace	B272	B19	UCB1
Total Requests	15306	444459	94438
Sum of distinct requests from each user	3192	64188	70948
Hit Ratio	79.15%	85.56%	24.87%

3.3.4 Caching Strategy 3

In the third simulation, scenario in Simulation 2 is assumed, with cache sharing. This simulates a Distributed Web Caching environment. Hence, only when a user requests for a URL which has not been accessed before by any of the users in the distributed cache, the request will be forwarded to the origin server. Otherwise the request will be fulfilled from within the shared cache. Hit Ratio (HR) for this strategy is expressed in Equation 3.4. Note that the same Hit Ratio could be obtained using a central proxy server with the capacity to hold all the distinct pages. Our aim here, is to provide a service similar to that of a dedicated proxy server, without the problems associated with operating a centralised, dedicated proxy server.

$$\text{Hit Ratio} = \frac{\text{total requests} - \text{sum of distinct requests from distributed cache}}{\text{total requests}} \tag{3.4}$$

Table 3.5: Caching Strategy 3

Trace	B272	B19	UCB1
Total Requests	15306	444459	94438
Total distinct requests from shared cache	2897	38374	53652
Hit Ratio	81.07%	91.37%	43.19%

3.4 Conclusions

By comparing results from Strategy 1 and 2, it could be concluded that, the benefit of caching improves when the overlap of requests is maximum and when the number of distinct requests is minimum. By comparing B272 and B19 we could conclude that, a higher the number of clients leads to a better overlap.

By comparing Strategy 1,2 and 3, it is clear that cache sharing provides the best performance. Between Strategy 2 and 3, improvement in B272, B19 and UCB1 are 1.9%, 5.8% and 18.32% respectively. The relative improvement in sharing is comparatively smaller in B272 and B19, since they have steep hot-sets and Zipf parameter is close to 1. Therefore, the stand-alone caching already provides high hit ratios.

However, since UCB1 only achieves a hit ratio of 24.87% with stand-alone caching, there is a significant improvement, when caches are shared and overlap between users is exploited. This could be attributed to the greater percentage of distinct requests as indicated by the smaller hot-set and the lower Zipf parameter.

3.5 Chapter Summary

In this chapter, a preliminary analysis was carried out on real web access traces to determine the possible benefits in cache sharing using Distributed Web Caching. It is observed that, cache sharing provides a significant improvement in hit ratios over stand-alone caching strategies. This in turn translates into external bandwidth savings and reduced user perceived delays when accessing the web.

Chapter 4

A Novel Distributed Web Caching System

4.1 Introduction

At present, Distributed Web Caching employ proxy cache co-operation at the institutional level[30] as shown in in Figure 4.1.

A closer look at this architecture indicates that, proxy caches in fact are second tier caches. If it is possible to implement cache co-operation at the browser cache level, it is possible to benefit from the the high bandwidth available at the bottom level of any network.

From a processing power point of view, client nodes nowadays are of high speed and have large amounts of resources. These resources are not fully utilised all the time.

This chapter investigates the possibility of implementing a cache co-operation mechanism at the browser cache level of the network hierarchy, which utilises the

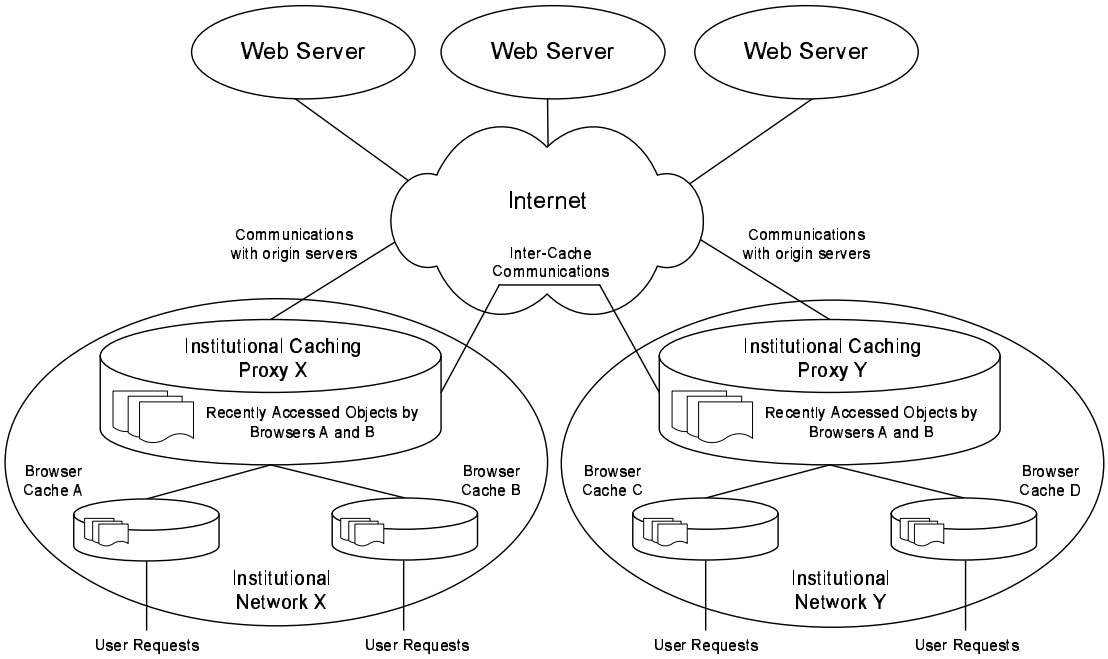

Figure 4.1: Co-operative web caching at institutional level

wide bandwidth and unused computing resources at the bottom level of a network.

4.2 Network Topology

Prior to introducing the proposed distributed caching system, it is essential to define the assumed topology of the network. Figure 4.2 shows the topology of the UK Joint Academic Network (JANET). We assume that the whole of the internet consists of a large number of such networks interconnected at the core routers.

At the core of JANET topology are the Core Point of Presence (CPOP) routers inter-connected by high speed Dense Wavelength Division Multiplexing (DWDM) fibre optic backbones. Some of these CPOP routers are connected to external net-

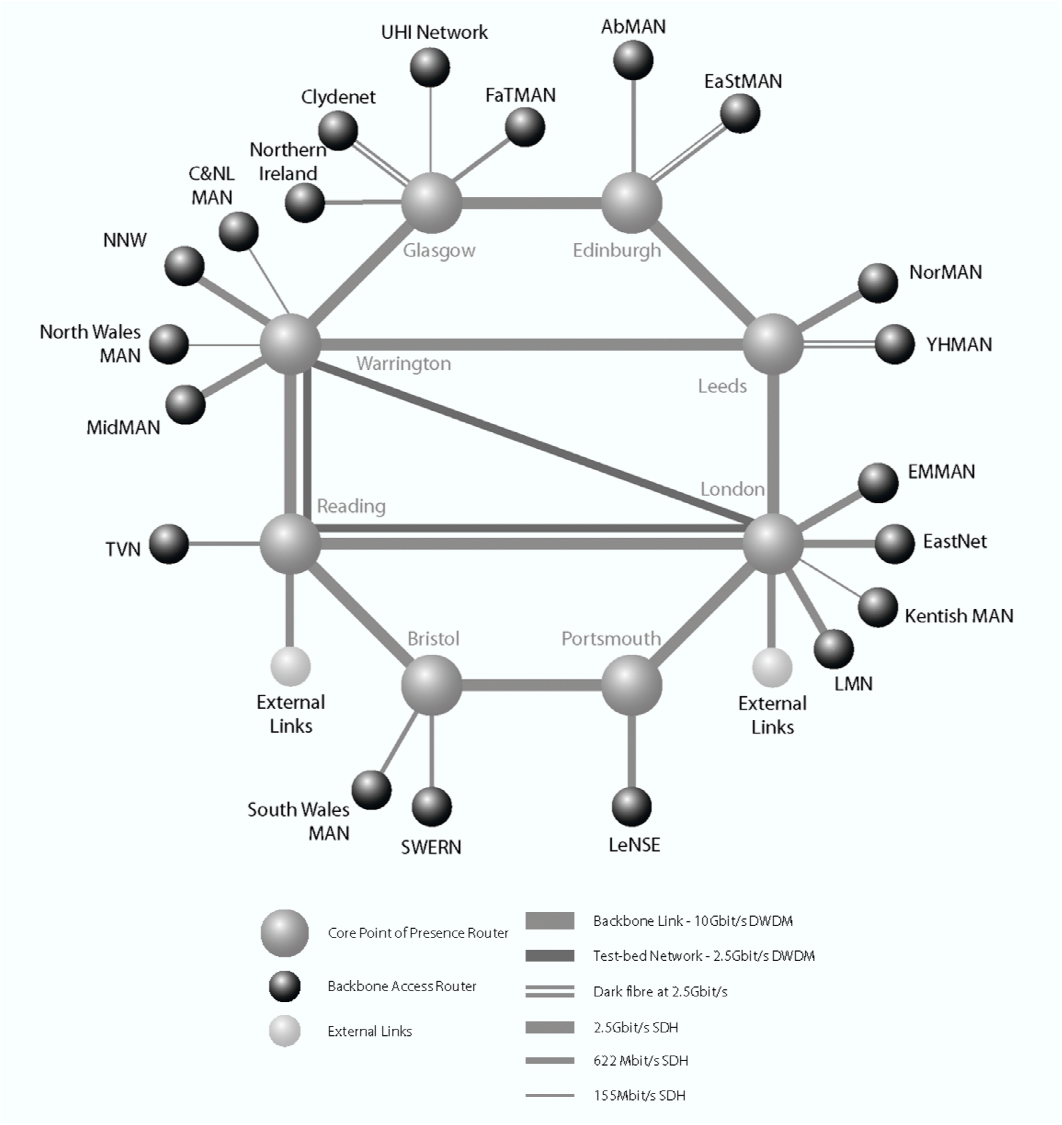

Figure 4.2: Janet topology and link capacity - ©JNT Association 2003

works providing connectivity to other networks in the UK and global connectivity to USA, Europe, Japan, etc., mostly via submarine fibre optic cables. A second tier of Backbone Access (BA) routers are connected to the core backbone network at the CPOP routers via slower access links. It is possible that all the routers are not fully interconnected. Therefore, it may be necessary for a message to pass through more than one router to get from one location to another.

Institutional networks such as campus networks are connected to these BA routers. For example, the campus computer network of University of Warwick is connected to MidMan backbone access router via a 2.4Gbps link.

Gateway computers at the border of institutional networks function as exit and entry points to those networks. Within an institutional network, computers are organized into groups by their locations. Each of these groups forms a local area network. These local area networks are again interconnected by lower level routers. Within a local area network, there are computers that function as servers and client workstations.

4.3 The Proposed Model

Typically, server nodes are powered by powerful computers with a high degree of reliability whereas client nodes are usually served by less powerful personal computers. However, the computer hardware landscape appears to have changed in recent years. The gap in terms of computing power between server class computers and client class computers has narrowed dramatically. Personal computers running clients applications are becoming as powerful as low-end server computers. In terms of reliability, there is still a significant difference. However, there is a steady im-

provement in reliability as client operating systems are becoming more stable. In the case of server computers, the workload is very much dependent on the number of clients they serve. Demand does vary throughout the day but in general tends to be much more uniform compared to client computer systems. Client computers today are in general structured to be personal computers with the main task being to service a machine-to-man interface. The system demands tend to vary a lot more and, in fact, for a good part of the time the client computers are in an idle state.

Our proposal is that the client computer should be assigned to perform some server functions on an on-demand-basis when the demand for local computing power is low. One possible server function that can be assigned is a cache serving function. In our proposal, a client computer will have as its first priority performing locally initiated tasks. When there is no locally initiated task listed in the queue, the operating system (OS) can assign the computing resources to perform network cache functions. In the event that a locally initiated task is created while the client computer is performing an OS initiated server function, the client computer could abandon the server task to attend to the locally initiated task. It could also lower the priority of the cache serving function so that locally initiated tasks would always get attended to first by allocating all the necessary resources available on the computer for the successful completion of locally initiated tasks. Under this arrangement, users should not experience a degradation of performance of their client computers. However, one implication of putting the server functions as low priority tasks is that a definite quality of service with respect to the server functions cannot be guaranteed. In fact, this makes the client computer unreliable from the server function perspective. The key issues that have to be addressed in this proposal are

then the methods to utilize the additional power of the client nodes to increase the performance of the server functions while ensuring that the inherent unreliability characteristic does not jeopardize the overall cache service performance.

The proposed Distributed Web Cache model is as follows:

The operating system in each client computer will be extended to equip each client computer with a Cache Server Service (CSS). The extended OS will enable a client computer to locally store objects that it requests and receives from the web. The storage capacity allocated for this caching purpose is a parameter that can be set by the user. These cached objects, will be served locally should the same objects be requested again by the local browser or they will be served to any other users in the vicinity that requests any of the cached objects. This latter service is a new feature, as a result of the extended operating system. To support the objective of serving the cached objects that are available locally to other user in the vicinity, the following transactional protocol is adopted for CSS. Note that CSS intercepts all web requests from client applications and processes them according to the following protocol.

4.3.1 Distributed Web Caching (DWC) Protocol

There are two consistency options in the protocol.

- Strong Consistency (Mode A)
- Weak Consistency (Mode B)

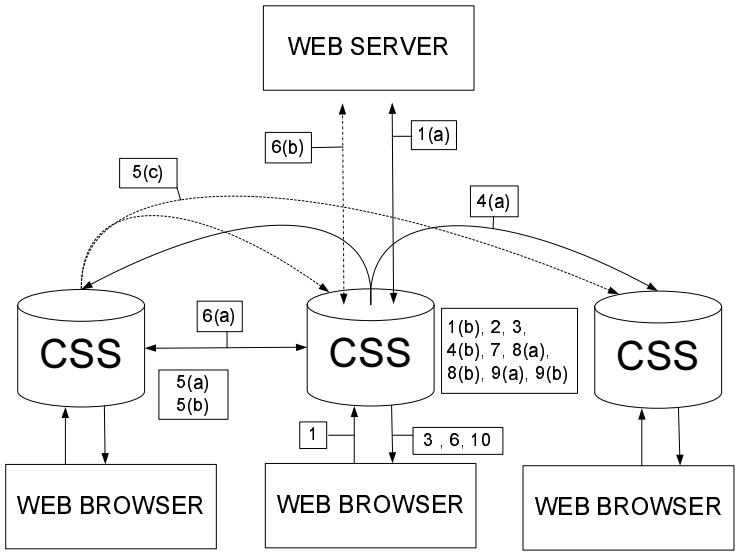

Figure 4.3: Proposed Distributed Web Cache Protocol

Step 1: When the web browser on a client node A, generates a web request for a particular web object, this request is first forwarded to the CSS module. On receiving the request, the CSS module performs the following tasks:

(a)
- Mode A

  it opens a network connection with the origin server of the object and requests for the header information of the object using an HTTP HEAD request. Sets a timeout t_1 for successful delivery of the header form origin.

- Mode B

  no action is taken.

(b) It checks to determine if the object requested is available in its local cache.

Step 2: If the object is not available in its local cache, it proceeds to **Step 4**.

If the object is available locally;

- Mode A

  it waits for the arrival of the HTTP header information from the origin server. If there is no response from origin before the timeout t_1, the CSS module aborts servicing the request and moves to the initial state. This is an *error state*.

  If the object is fresh the module moves to **Step 3**. If the object is stale, the cached object is flushed off, and the module proceeds to **Step 4.**

- Mode B
  - if the web browser has specified the freshness of the object in a `cache-control` directive or in a conditional GET request, time stamp

of the cached object is checked against the request. If the object is deemed fresh the module proceeds to proceeds to **Step 3**.

- If the browser request does not specify the freshness and if the object is still fresh according to its header information, the module proceeds to **Step 3**.
- If the browser request does not specify the freshness and there is no freshness information available in the header of the cached object, the module assumes the object to be fresh and moves to **Step 3**.
- For all other cases, local copy of the object is assumed to be stale and it is flushed off to move to **Step 4**.

Step 3: For both Mode A and Mode B, if the locally cached object is fresh, the object is delivered to the client application from the local cache and the request is fulfilled. This is called a *local cache hit*. The CSS module exits.

Step 4: For both Mode A and Mode B, the CSS module performs the following two tasks:

(a) It multicasts the request to the neighbors who are listening to a particular multicast address using a modified ICP-like query. If the freshness requirement is specified in **Step 1** it is attached to the query.

If no such information is available (as in the case where the header information from the origin server is not completely received yet), then only the name or identifier of the object is included in the query.

Note that the number of multicast hops is set to 1 by default. The maximum number of hops should reach the whole of the corporate network.

Setting it higher to reach networks beyond the corporate network may not be feasible due to lack of support for multicast and the excessive delays in accessing remote networks. This issue will be further discussed in Section 5.1.2.

(b) The CSS module starts a timer to implement a deadline t_2, for the receipt of responses from potential cache servers.

Step 5: For both Mode A and Mode B, all nodes monitoring the particular multicast address will receive the multicast query from Node A. If a particular node computer, say Node B, has the requested object, it will do the following:

(a) It checks if the copy of the object in its cache is fresh. This is possible if the freshness requirement is already available in the query. If the freshness information is not included in the query and the header of the cached object does not indicate that it is stale, Node B will assume that the object is fresh.

(b) If the object cached in Node B is deemed stale, Node B will flush the object off its cache. It then takes no further action.

(c) If Node B is able to serve the object and if the object is deemed fresh, it will respond to Node A via the same multicast address.

By responding on the same multicast address, other nodes which have the object in their local caches but are slower than Node B to respond due to local processing loads would clear the request from their queues and not respond to Node A. This will avoid a flood of responses to Node A. Note that, the ability to serve or not is a parameter that can be set

within Node B. This parameter is dependent on the computation load of the node at the time of receiving the request.

Step 6: For both Mode A and Mode B,

(a) On receiving a response, Node A immediately opens a TCP channel with Node B to receive the object into its own local cache. A first-come-first-accept algorithm is adopted for the selection of cache servers. Later responses will be ignored by Node A. A first-come-first-accept algorithm is most acceptable since a fast response is a good indication that Computer B has the resources currently to perform the cache serving function well.

(b) If there is no response from any computers by the deadline t_2 for a response, Node A can go back to **Step 4** and re-multicast the request with perhaps a higher hop count. Alternatively, it can directly open a communication channel with the origin server to bring the object in from the origin server. In this case, it is similar to existing web request protocol to the original server. The choice between the two options is a parameter to be set during system implementation. The recommended configuration is to cover the corporate LAN with one multicast with the appropriate hop count in Step 4 and, after the multicast timeout, to request the object directly from the origin.

Step 7:

- Mode A

Upon receiving the header information of the object from Node B, Node A compares the information with the information from origin to determine freshness. If the origin has failed to deliver the header by this stage and if

the timeout t_1 has elapsed, the protocol closes all connections and moves to the initial state. This is an *error state*.

- Mode B

  Protocol moves to **Step 8** assuming that the incoming object is fresh.

Step 8: For both Mode A and Mode B, If the incoming object from Node B is verified to be fresh,

(a) A deadline t_3 for the successful delivery of the requested object from Node B is computed and set. This deadline is related to the size of the object and the expected rate of data delivery in the network.

(b) Reception of the object into the local cache of Node A continues. Once the reception is complete, proceed to **Step 10**.

Step 9:

- Mode A

  If the cached object from Node B is verified to be not fresh, the following is done.

  (a) The session with Node B is aborted.

  (b) The CSS module returns to **Step 4**. Note that returning to **Step 4** after this point would allow the freshness information ro be attached to the multicast query. This repetition of **Step 4** onwards serves to invalidate the stale copies of the object in the shared caches.

- Mode B

  No action taken.

Step 10: For both Mode A and Mode B, if the object requested is successfully received from Node B into the local cache of Node A, the CSS module then invokes the local serve function to deliver the object to the client application that requested the object. This is called a *remote cache hit*. If by the deadline t_3, the object is still not received from Node B, the communication channel with Node B is aborted. The CSS module returns to **Step 4** and restarts the request, with a block on replies from Node B.

At any stage, if the protocol fetches the object from the origin, it is a *cache miss*. The protocol will impose a deadline similar to t_3 for origin to successfully deliver the object. If that fails, the protocol will close all connections and move to initial state. This is an *error state*.

4.3.2 Design of the CSS Module

Multicast Timeout t_2

As explained in [52], it is desirable to have a dynamic value for the multicast timeout t_2, depending on the size of the requested object. Let h_{shared} be the hit ratio of the shared cache. The two possible ways to receive an object O of size s, are either from the origin which has a time cost of s/r_o or from the shared cache which has a time cost of s/r_s where, r_o and r_s are the average transmission rates from origin and from shared cache respectively. The extra waiting time is expected to be less than $\frac{s}{r_o} - \frac{s}{r_s}$. Therefore, the longest acceptable waiting time for a multicast response t_2 can be expressed as

$$t_2 = h_{shared}(\frac{s}{r_o} - \frac{s}{r_s}) \tag{4.1}$$

The analysis in section 5.1.2 on multicast timeout, arrives at the same result. The parameters r_o, r_s and h_{shared} are based on known values, if available. Otherwise, default values are used. The parameters r_o and r_s are not maintained for each origin server or each shared cache node. Instead average transmission rates over a period of time are used. At the onset, size of the object s is unknown. The initial timeout period, $t_{2(initial)}$ is selected based on some assumed probability for a "cache-hit" for the requested object. A more accurate deadline t_2, is subsequently calculated when the cached copy is received.

CSS Task Queue Length L

The CSS task queue length is an important parameter to be determined. Ideally, the length of the queue should be set such that queries are processed within the multicast timeout t_2 at the requesting node. It is possible to model the multicast query and response messages at a node as a Poisson process [52]. Let $\lambda(t)$ be the arrival rate of the multicast messages. Probability that k "cache-query" messages arriving from time t to $t + \Delta t$, $P(k \text{ in } [t, t + \Delta t])$, could be calculated as

$$P(k \text{ in } [t, t + \Delta t]) = \frac{1}{k!}[\int_t^{t+\Delta t} \lambda(\varsigma) d\varsigma]^k exp[-\int_t^{t+\Delta t} \lambda(\varsigma) d\varsigma] \tag{4.2}$$

If Δt is assumed to be very small, Equation 4.2 can be simplified to:

$$P(k \text{ in } [t, t + \Delta t]) = \frac{1}{k!}[\lambda(t)\Delta t]^k exp[-\lambda(t)\Delta t] \tag{4.3}$$

The average number of messages arriving from time t to $t + \Delta t$, $\eta(t)$, is then

$$\eta(t) = \lambda(t)\Delta(t) \tag{4.4}$$

The CSS module only processes messages received in the past t_2' of time, where t_2' is an estimate of t_2 of the requesting node. Therefore, the queue length L, can be set to

$$L = \overline{\lambda(t)}.\overline{t_2'} \tag{4.5}$$

where, $\overline{\lambda(t)}$ is estimated based on the average incoming message rate in the past and $\overline{t_2'}$ is calculated by averaging t_2' in the past.

Processing Multicast Requests

The CSS module receives two types of messages from neighboring CSS modules. It adheres to following guidelines.

1. The local web requests have highest priority.

2. The probability of a "cache-hit" flood is minimized. Note that since a "cache-query" from a requesting CSS reaches every CSS module within the muticast range, it is possible for each node to respond to each query.

3. The probability of a responding CSS module issuing a "cache-hit" while the requesting node has given up waiting for a response is minimised.

A task queue is maintained within the CSS module for incoming "cache-hit" and "cache-query" messages. Upon receipt of a muticast message, it is processed as shown in Table 4.1.

Table 4.1: Inserting multicast messages to CSS task queue

Case 1: The message is a "cache-query"	Case 2: The message is a "cache-hit"
The message is inserted to the rear of the task queue together with the time, $t_{receive}$.	**If** the "cache-hit" was a response for a local request, the message is forwarded to the function handling local web requests. **Otherwise** the message is inserted to the rear of the task queue together with the time $t_{receive}$.

When there are no higher priority tasks in the system, CSS processes the multicast messages picked up from the task queue at random and processed as shown in Table 4.2.

Guideline 1

The foreground task of CSS module only services incoming "cache-hit" responses that match local queries.

Guideline 2

The "cache-hit" responses are used to search and remove from the queue those queries that have been attended to by other nodes. However, during and after processing a "cache-query" in the queue, multiple "cache-hit" responses for the same "cache-query" could arrive from other nodes.

Let query q exist in more than one CSS task queue. Let T_1 be the time when a node picks the query q for processing for the first time in the entire system and let T_2 be the time when that node finishes processing query q and issues a "cache-hit"

Table 4.2: Processing of tasks from CSS task queue

Case 1: The message is a "cache-hit"	Case 2: The message is a "cache-query"
Search the queue for a matching "cache-query" **If** (a match is found) remove the matching "cache-query" and the "cache-hit" message. **Othewise** remove the "cache-hit" message.	**If** (a cached copy is available and deemed fresh) estimate the timeout t_2 at the peer as shown below: $t_2' = h'_{shared}(\frac{s}{r_o} - \frac{s}{r_s})$ where, r_o, r_s and h'_{shared} are values from earlier transactions. **If**$(t_{receive} + t_2' < t_{now}$ that is, if the requesting client has given up) the "cache-query" is removed. **Otherwise** multicast a "cache-hit" response on the same multicast address as the "cache-query". **If** (a cached copy is available but stale) the cached copy is flushed off and the "cache-query" is removed. **If** (a cached copy is not available) the "cache-query" is removed.

response.

If another node picks q between T_1 and T_2, there is a possibility of another "cache-hit" response being issued for the same query q.

Let p_0 be the probability that a node is free to process the CSS task queue, p_1 be the probability that a node holds a cached copy of the object being requested in q and p_2 be the probability that a node picks up q to be processed next. Under the assumption that all nodes take the same time to process a "cache-query" then in a system with N nodes, the probability of a burst of m "cache-hit" responses for the same query q is

$$P\{\text{burst of } m\} = C_{N-1}^{m-1}(p_0p_1p_2)^{m-1}(1-p_0p_1p_2)^{N-m} \tag{4.6}$$

Let N_m be the number of multicast messages including the "cache-query" and the "cache-hit" due to a request. Its average, $\overline{N_m}$ is

$$\begin{aligned} \overline{N_m} &= 1 + \sum_{m=1}^{N} mC_{N-1}^{m-1}(p_0p_1p_2)^{m-1}(1-p_0p_1p_2)^{N-m} \\ &= 2 + (N-1)(p_0p_1p_2) \end{aligned} \tag{4.7}$$

A smaller $p_0p_1p_3$ results in a smaller $P\{\text{burst of } m\}$. However, high values for p_0 and p_1 are required to maintain an acceptable caching performance. As for p_0, if queries are picked in any regular order (e.g. FIFO or LIFO), the probability of multiple nodes picking up the same query is high. This could lead to a flood of "cache-hit" messages for the same query. Therefore, to counter that, we randomly pick messages from CSS task queue.

Guideline 3

In order to achieve this, CSS makes use of t'_2 which is an estimate of t_2. If the query is processed before the time $t_{receive} + min(t'_2, t_2)$, the accuracy of estimation does not matter. This eliminates the possibility of responding nodes processing requests after the requesting node has given up. In addition, the requesting node will not wait for responses that will never be issued. The situation is the same for the case when $t_2 = t'_2$. If the query is processed after $t_{receive} + max(t'_2, t_2)$, the accuracy of the estimate does not matter since, the requesting node has given up and the responding nodes ignore the query.

In the case of $t'_2 > t_2$ and when cache query is processed after $t_{receive} + t_2$ and before $t_{receive} + t'_2$, a "cache-hit" response will be sent when the requesting node has given up waiting. The response will be ignored by the requesting node and the message will flush off matching queries in CSS task queue of other nodes.

Similarly, in the case when $t'_2 < t_2$ and when the query is processed after $t_{receive} + t'_2$ and before $t_{receive} + t_2$, no response will be sent although the requesting node is still waiting. But it is possible that another which has larger t'_2 will subsequently issue a "cache-hit".

4.3.3 Cache Maintenance

As the storage capacity at each client computer is limited, inevitably at some point in time, the storage capacity at each node computer will become fully utilised. To cache an incoming object, it is then necessary to replace some existing objects in the system. This issue comes under placement and replacement strategies which are discussed in detail in Section 6.

4.3.4 Properties of the Proposed System

In designing the DWC protocol, following assumptions were made regarding the client nodes.

- *Ad-hoc Nature*

  Client nodes join and leave the Distributed Web Caching system without direct control of any entity. Therefore, the number and location of active and the network topology interconnecting them are highly dynamic. Due to the ad-hoc nature, the Distributed Web Caching system has to be self organizing.

- *Limited Capacity and Reliability of Client Nodes*

  Peers do not have the same capacity or the same reliability when compared to servers. They fail more often. Therefore, the Distributed Web Caching system has to be fault tolerant and adaptive.

- *Rationality of Client Nodes*

  Rational client nodes make decisions to maximize their own benefit. These decisions may not always be inline with the objectives of the caching system as a whole.

The proposed Distributed Web Caching system has following properties:

1. **Guaranteed Cache Consistency**

   If the system is operating in the Strong Consistency Mode (Mode A), In Step 1 of protocol, the header information of the object, which includes the time stamp of the object, is requested from the origin server following a request for

the object. This procedure is repeated for every request. The requesting node compares the time stamp received from the origin server to time stamp of the cached object. Only when the two time stamps are the same, is the object obtained from the cache delivered to the web browser. In the event that the time stamp differs, cached object is discarded and a more up-to-date object is obtained from another cache server or from the origin server. If the object comes from another cache server, the same time stamp criterion is imposed. If the object is eventually obtained from the origin server after the search procedure, then consistency is no longer an issue. It shows that none of the cached versions in the system are up to date.

Note that, in Strong Consistency Mode (Mode A), HTTP header is requested from the origin server, to enforce the consistency requirement for each request. This has following implications [53].

(a) Additional wide area network traffic.

(b) Additional origin server load.

(c) Additional access latency .

As an alternative, protocol has a Weak Consistency Mode (Mode B) which will rely on validity of objects, based on their header information. In this mode, origin server will not be contacted to check the time stamp of the original object. Strong cache consistency is always achieved with an added cost in the form of increased access delay, additional wide area bandwidth usage and additional work at the origin server [53]. Therefore, a choice has to be mode, which mode is most appropriate to the individual node or to the network as a

whole. In a situation where, wide area bandwidth is scarce and costly, Mode B is more appropriate. Note that, the end user has the liberty to send out requests by passing all caches using the web browser when deemed necessary.

2. **Communications Overhead**

Let us assume the following:

L_{DWC} = length of DWC query (size is comparable to ICP)

L_{URL} = Average length of the URL

n = Number of siblings

From Section 4.3.2, total size of a multicast query and a multicast response:

$$(L_{DWC} + L_{URL}) * (2 + (n-1)(p_0 p_1 p_2)) \tag{4.8}$$

It is expected that,

$$(L_{DWC} + L_{URL}) * (2 + (n-1)(p_0 p_1 p_2)) \ll (L_{DWC} + L_{URL}) * (2 * n) \tag{4.9}$$

where, the latter is similar to the unicast version of ICP. As mentioned in Section 4.3.2, a multicast response would stop other nodes form multicasting a response for the same query, the probability of a "cache-hit" flood is minimal.

Therefore the best case would be,

$$(L_{DWC} + L_{URL}) * 2 \tag{4.10}$$

If the request rate is R, the minimum amount of data transfer for one client node is,

$$(L_{DWC} + L_{URL}) * 2 * R \tag{4.11}$$

Since there are n nodes and multicast messages will reach all participating nodes, the minimum total data transfer is,

$$(L_{DWC} + L_{URL}) * 2 * R * n \tag{4.12}$$

Since the request rate is about 10 to 20 requests per hour [54] (R is $2.77{*}10^{-3}/s$ to $5.5510^{-3}/s$), if we substitute $R = 4.16 * 10^{-3}/s$, and $n = 100$, we get the minimum communications overhead on the ethernet port of a client node to be,

$$79 * 2 * 4.16 * 10^{-3} * 100 = 65.7 bytes/s = 526 bps \tag{4.13}$$

If the corporate network has 10,000 nodes the minimum communications overhead will be $51 Kbps$ per network link. We believe that, on a 10 to 100 $Mbps$ network link, this is acceptable.

In comparison, if the clients are served by a central proxy server, the minimum communications overhead due to queries on the internal network link of the proxy server will be,

$$L_{URL} * R * n \tag{4.14}$$

However, it must be noted that all object transfers will take place via the same single network link, unlike in the Distributed Web Caching system.

3. **In-built System Reliability**

A concern when employing client nodes to perform the functions of a cache server is the reliability. The client node can fail in many more ways compared to a dedicated cache server. In the first instance, the client node could be shut

down by the user without prior notice, and this could happen in the middle of a cache serving transaction, thereby terminating the transaction pre-maturely.

In addition, without prior notice, a client could be loaded with a computationally intensive local job so that the amount of resources available for the cache serving purpose may suddenly diminish. Note that the caching serving function is structured as a low priority task on the client nodes. Any locally initiated task will be serviced with high priority even though it is created later. As such, it may then not be possible to deliver the promised object to the requesting computer within an acceptable time limit.

In fact, under the above described protocol, there is no guarantee of any minimum quality of service by any particular client computer. The philosophy of the protocol proposed in this chapter is very similar to that of the Internet Protocol (IP) of the TCP/IP protocol. Here IP is inherently unreliable and works on a best effort basis only. TCP (Transport Control Protocol) is then the upper layer mechanism to ensure reliable end-to-end delivery of the data packets. In our proposal, serving is also based on a best effort basis, and is subjected to local availability of resources at each node computer. In our case, response to requests may be delayed, transactions may be terminated before completion or data delivery rate may deteriorate to an unacceptable level. In view of this inherent unreliability nature of each client node in performing the cache function, reliability is built into the upper layer protocol. This comes in the form of a timeout and a retry mechanism. The requesting node is tasked with the responsibility of monitoring the timeouts and performing retries.

4. **Redundancy and Reduction of Hot-spots**

One of the problems associated with popular web objects is a hot spot phenomenon. This phenomenon could happen to the origin server or even a dedicated cache server. In such a situation, the origin server or dedicated cache server that has the object cached has to fulfil a large number of requests within a short time span. The result is a bottleneck in computing resources of the server as well as a huge demand on the bandwidth of the communication medium linking the server to the rest of the network. Our proposed structure and protocol effectively avoids such a hot spot phenomenon. Popular pages requested by many clients would end up being cached by all the clients that requested those pages. Further requests by other clients would attract cache serve responses by the clients that hold copies of the page. If a particular client is already engaged in serving the object, it would not have the resources to respond to further requests quickly (even it is a fast computer), giving rise to opportunities for, perhaps, other slower clients to have a chance at fulfilling the requests. Overall, with many requests of the same object occurring at the same time, the serving load as well as bandwidth demand of the communication medium will be spread over many nodes and communication links respectively, effectively making the hot spot phenomenon non-existent.

5. **Zero Administrative Overhead**

   There is zero administrative overhead when client nodes join and leave the Distributed Caching System. There is only a graceful degradation in cache hit ratio when a client node leaves the system, since highly popular objects are bound to have more than one copy in the system.

6. **Low Memory Usage**

Since client nodes do not retain any information about the network topology or the cache contents of the rest of the nodes, there very little memory usage on client nodes. However, memory is required to maintain the CSS task queue.

4.4 Software Implementation

This section includes implementation aspects of the proposed Distributed Web Caching system.

The cache server service (CSS) in section 4.3.1 is divided in to five functional modules. Flowcharts 4.4 to 4.8 illustrate the five modules *CSS Module 1* to *CSS Module 5* operating in Strong Consistency Mode (Mode A). In this mode, origin server is consulted to validate the freshness of objects for each request. The system has the ability to operate on Weak Consistency Mode (Mode B), using a subset of features from Mode A, which relies on HTTP header information of objects to determine freshness.

CSS Module 1 is the core functional module in the CSS. It is the only module which communicates between CSS and external client nodes and servers. It initiates and controls the search mechanism of the Distributed Web Caching System.

CSS Module 2 and *CSS Module 3* are sub-functions of *CSS Module 1.* The function of *CSS Module 2* is searching for the requested object in the local cache. The function of *CSS Module 3* is to receive the object from a remote node of the Distributed Web Caching System.

CSS Module 4 maintains a task queue on each node to keep track of all requests from the local browser and from remote nodes. This queue holds all the cache service requests in the system, and the content of the queue is updated whenever a task

is fulfilled locally or by a remote node. As explained earlier, the CSS task queue length L is a tuneable parameter that controls the service capability of each node. If the queue saturates, it will discard all remote requests and only attends to local requests, as local requests are given priority over remote requests. *CSS Module 5* handles serving of cached objects from the local cache to remote nodes.

Upon receiving a web request, the CSS service will enter one of following four states from the idle state:

- Local Cache Hit
- Remote Cache Hit
- Cache Miss
- Error State

4.4.1 Error State

This state indicates that, neither the Distributed Web Cache nor the origin has been able to fulfil the request. The protocol closes all connections and moves to the initial state. In Mode A, this could occur due to failure by the origin to deliver the HTTP header information during an acceptable time. In both modes, if the origin fails to deliver the object, if and when requested, it will move to an error state, if the origin fails to deliver the object on time.

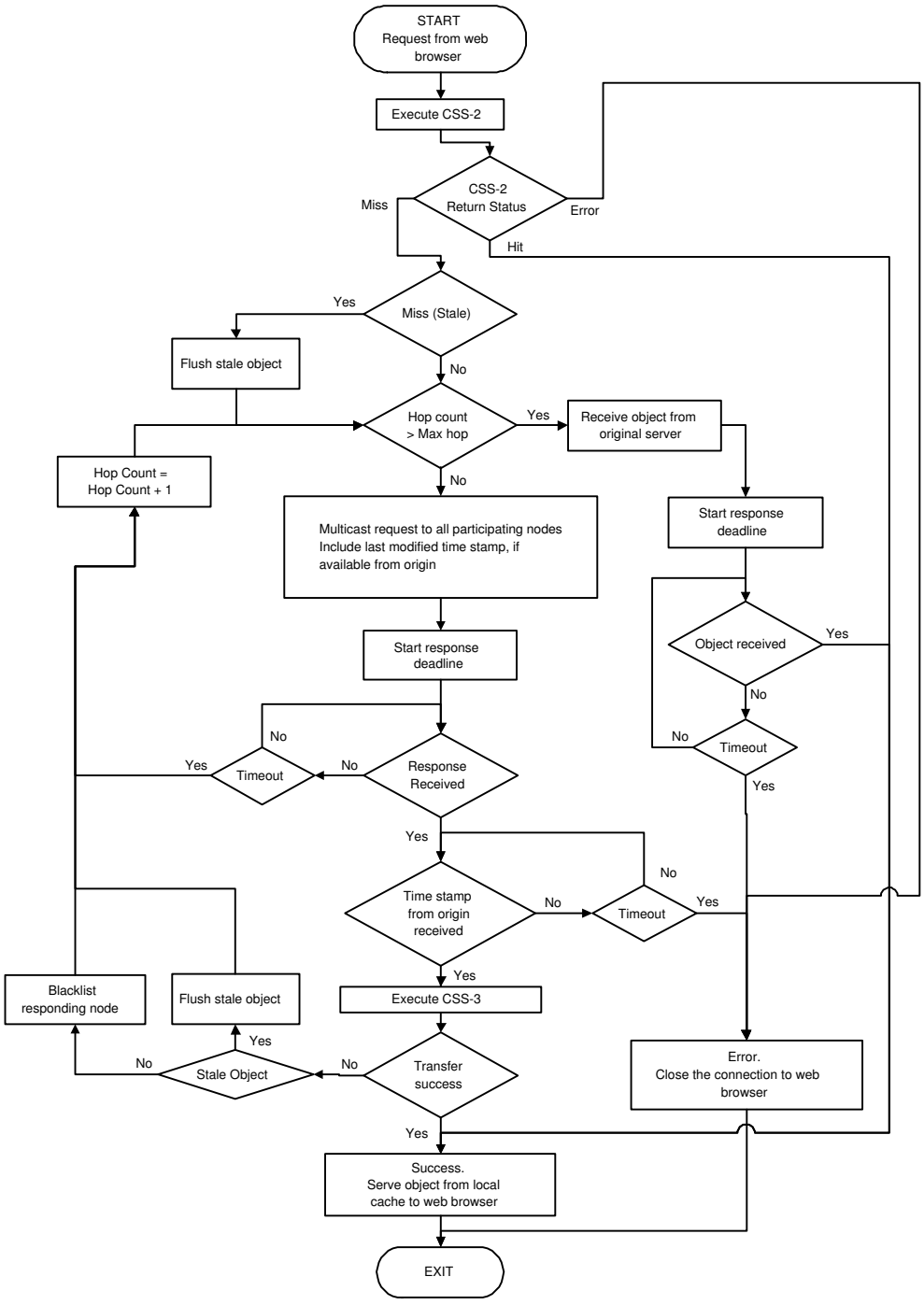

Figure 4.4: CSS Module 1

Figure 4.5: CSS Module 2

Figure 4.6: CSS Module 3

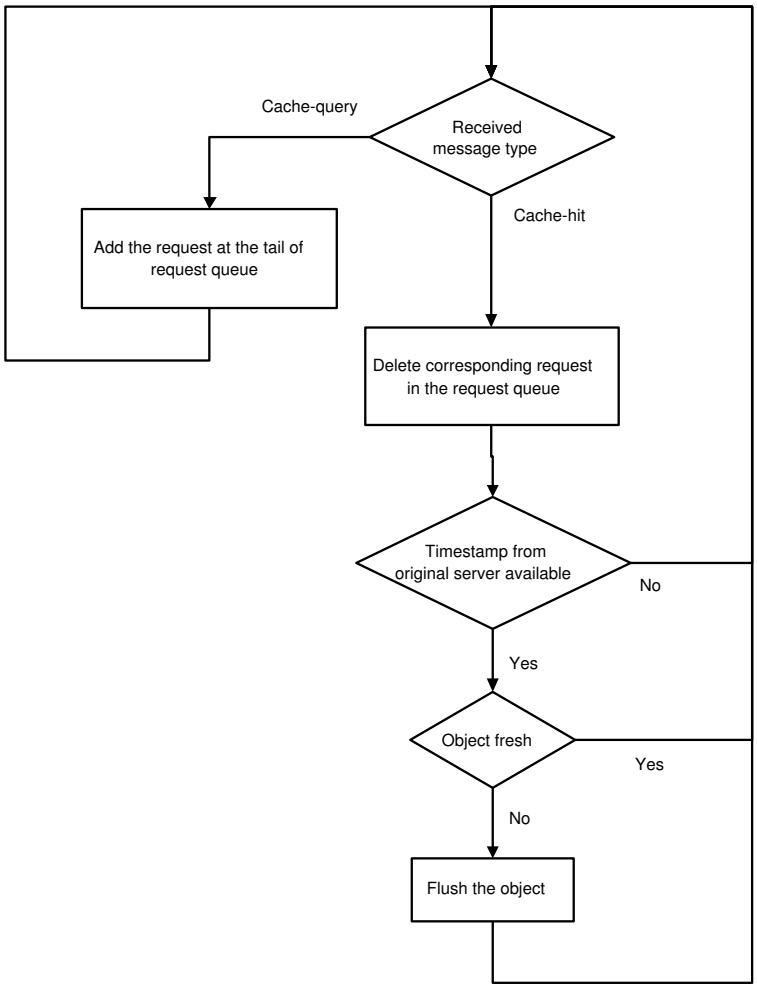

Figure 4.7: CSS Module 4

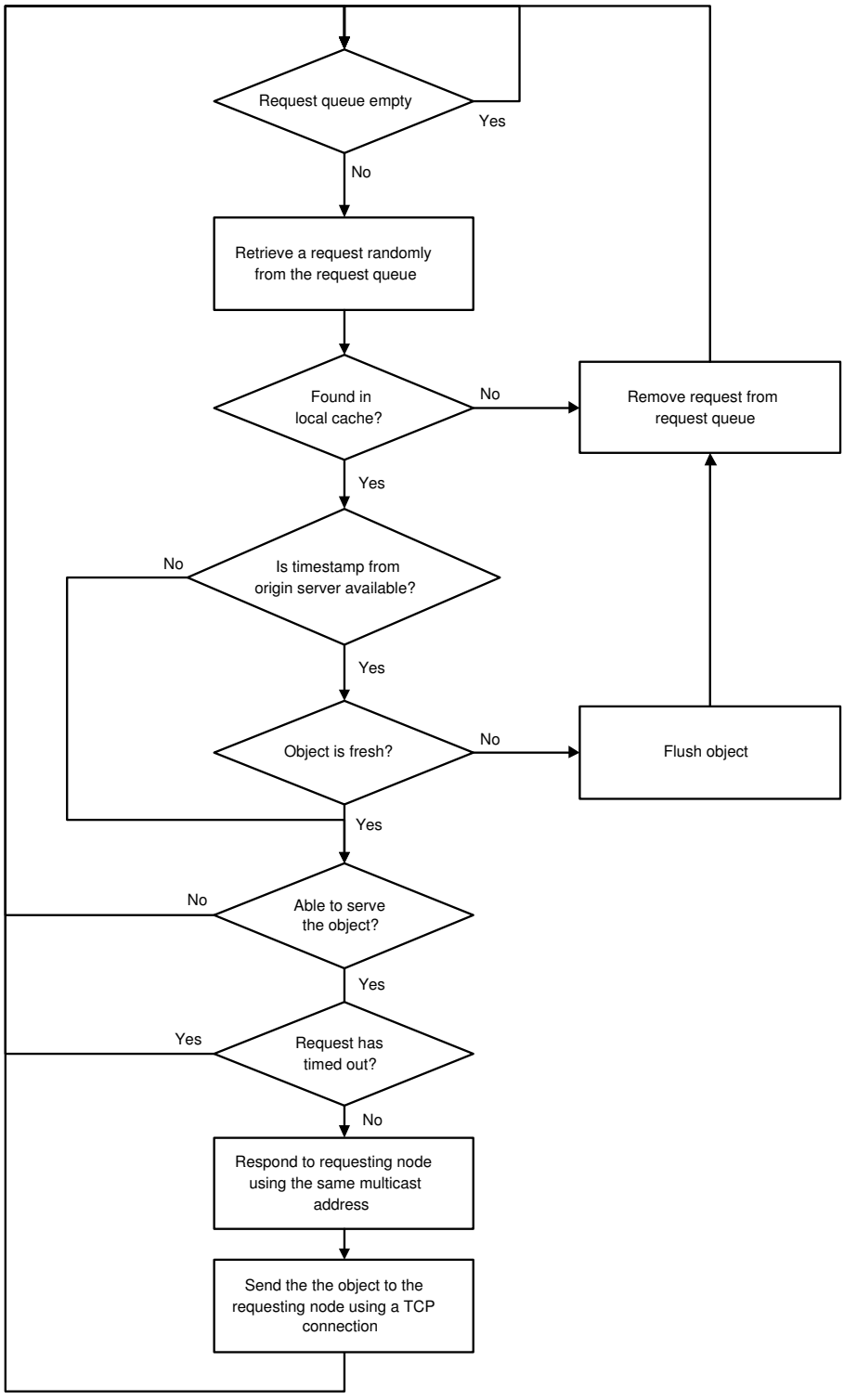

Figure 4.8: CSS Module 5

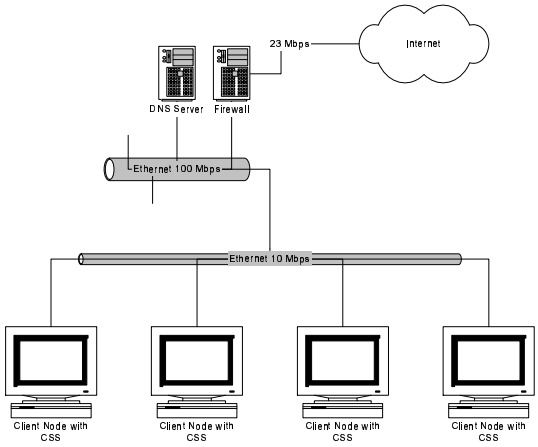

Figure 4.9: Experimental Setup 1

4.5 Experimental Performance Evaluation

In order to evaluate the performance of the proposed system, an experimental system was implemented on a Linux platform. This section describes the experiments carried out and their results.

First, the communication delays of the protocol in different states and the average response time of web requests are investigated. Then, the average response time is compared with that of a centralized cache system under similar conditions.

From results of the experiments, it was possible to determine the average response time of our Distributed Web Caching system.

4.5.1 Experimental Setup

Figures 4.9 and 4.10 illustrate the 2 arrangements used for the experiments. The first setup is a Distributed Web Caching system consisting of 4 client nodes running

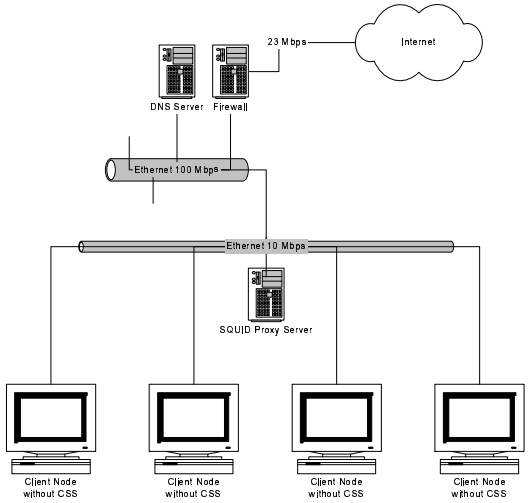

Figure 4.10: Experimental Setup 2

the CSS. The clients used in these experiments are personal computers with PIII 500 CPUs and 64MB of memory. The network is an Ethernet with a bandwidth of 10 Mbps within the subnet and 100 Mbps between the subnet and the campus firewall. The firewall is the Internet access point of the whole network. Web requests on each client are made using a web access simulator.

The second setup emulates a conventional centralized caching system. In this setup, client nodes are not equipped with CSS. The SQUID cache server software [21] is used to configure a LAN level proxy cache server. Hardware used for the cache server are identical to those on client nodes. It is connected to the Internet through the campus firewall. Each client node is equipped with the web access simulator mentioned above.

Both strong and weak consistency schemes were tested on both setups. In both cases, browser caches are bypassed, eliminating their effects on measurements.

4.5.2 Methodology

The web access simulator running on client nodes generates the web requests by reading in a list of URLs from a file. The time required to fulfill every request is logged. This time duration includes the following:

- time taken for searching in the Distributed Web Cache
- time taken for validation from origin server (for Mode A)
- time taken for retrieval of the requested object (from a remote node or from origin server)
- time taken for serving the object to the client simulator.

The *tcpdump* program [55] running on each client is used to record the TCP layer transaction information while timing on the application layer is captured by the applications' transaction log.

URLs used in these experiments are extracted from log files of the IRCache project [1] [56]. The IRCache Project provides a global hierarchical caching service. The caching system has six top root cache sites located in USA, serving 5 to 14 GB of web contents every day to 125 to 200 client caches throughout the world. The latest 7 days' sanitized access logs are available on the web [56]. The log files are in SQUID [57] format and each line in a log file includes the requested URL. Experiments are carried out from 6AM to 9PM in order to avoid fluctuations in available bandwidth throughout the day.

[1] IRCACHE is the NLANR Web Caching project primarily funded by the National Science Foundation of USA.

Session Type	Local search	Validation	Multicast delays	Remote search	Multicast timeout	Transfer	Total effective
Local Hit	0.020 s	1.050 s	NA	NA	NA	NA	1.050 s
Remote Hit	0.020 s	1.050 s	0.010 s	0.020 s	NA	0.024 s	1.050 s
Cache Miss	Object not found						
	0.020 s	NA	0.010 s	0.020 s	0.150 s	2.440 s	2.610 s
Cache Miss	Object found stale after retrieval from remote node						
	0.020 s	1.050	0.010 s	0.020 s	0.150 s	1.964 s	2.990 s

Table 4.3: Performance Evaluation - Average Delays - Experiment 1
Strong Consistency Mode (Mode A)

4.5.3 Experiments and Results

In the first experiment, a typical object of size 16.17 KB is requested and the average delays in different layers of the protocol are recorded. The experiment was carried out for both Mode A and Mode B and the results are shown in Table 4.3 and Table 4.4 respectively.

In both modes, if the object is fetched or validated from the origin, first TCP connection with the origin includes the DNS resolution delay of around 500ms. Once the DNS entry is cached at the campus DNS server, subsequent DNS requests are fulfilled in less than 1ms.

As mentioned in Section 4.3.4, it is evident from Table 4.3, that Strong Consistency Mode (Mode A) has a considerable cost in terms of access delay.

In the second experiment, average access times are compared for the two situations. They are namely, cache miss and remote cache hit.

For this experiment, 10000 unique URLs with objects of varying size were used.

Session Type	Local search	Validation	Multicast delays	Remote search	Multicast timeout	Transfer	Total effective
Local Hit	0.020 s	NA	NA	NA	NA	NA	0.020 s
Remote Hit	0.020 s	NA	0.010 s	0.020 s	NA	0.024 s	0.074 s
Cache Miss	0.020 s	NA	0.010 s	0.020 s	0.150 s	2.440 s	2.610 s

Table 4.4: Performance Evaluation - Average Delays - Experiment 1

Weak Consistency Mode (Mode B)

Number of requests	Top level domains
7500	.com
1113	.net
167	.uk
161	.org
116	.edu

Table 4.5: Performance Evaluation - Experiment 2 - Most visited *Top Level Domains* (TLDs)

First, the client caches were emptied so that all requests have to be forwarded to the origin. Once the access delays are measured, the caches were filled so that all unique objects are available within the distributed cache. For both cases, cache hits are controlled to exclude local hits.

There were 2063 unique sites in the access trace and total object size requested was 27.7 MB. Table 4.5 shows top five domains visited and Table 4.6 shows the top ten sites accessed.

The performance advantage in terms of access times were compared for 4 cat-

Number of requests	Top level domains
179	m.doubleclick.net
105	servedby.advertising.com
104	www.panasonic.com
93	www.mercedes.com
91	ad.doubleclick.net
89	www.bmw.com
82	www.weather.com
78	ads.aol.com
72	www.cartoonnetwork.com
71	www.mingpao.com

Table 4.6: Performance Evaluation - Experiment 2 - Most visited sites

File Size	*Number of files*	*Average Response Time (sec)*		*Improvement*
		Cache Miss	*Remote Cache Hit*	
$\leq 1kB$	4199	0.414 sec	0.055 sec	87%
$1kB \leq 5kB$	3565	0.702 sec	0.057 sec	92%
$5kB \leq 10kB$	847	1.278 sec	0.063 sec	95%
$> 10kB$	1389	3.168 sec	0.082 sec	97%

Table 4.7: Performance Evaluation - Experiment 2 - Effect of file size - Mode B

Cache Architecture	*Cache Miss*	*Cache Hit*
Distributed Web Caching	2.610 sec	0.064 sec
Squid proxy on LAN	2.460 sec	0.050 sec
Campus proxy	2.465 sec	0.056 sec

Table 4.8: Performance Comparison - Experiment 3 - Average Delays - Mode B

egories of file sizes. Results from the first experiments are as shown in Table 4.7. It was observed that with increasing file size the performance advantage improves. This is due to the higher speeds of file transfer within the distributed cache.

The third experiment involves comparing performance of the distributed caching system with that of a centralised caching system using the Experimental Setup 2 described in Section 4.5.1 and that of the campus proxy server. Average access delays for to access the whole 10,000 requests were measured for the three cases considered. Local hits are excluded in all measurements.

Results are shown in Table 4.8. It was observed that, for cache misses, Squid server on LAN was 5.7% faster than Distributed Web Caching while campus proxy was 5.5% faster. This is due to the multicast timeout delays in Distributed Web

Session Type	Local search	Validation	Multicast delays	Remote search	Multicast timeout	Transfer	Total effective
Local Hit	0.020 s	NA	NA	NA	NA	NA	0.020 s
Remote Hit	0.020 s	NA	0.026 s	0.020 s	NA	2.150 s	2.216 s
Cache Miss	0.020 s	NA	0.026 s	0.020 s	0.150 s	2.540 s	2.710 s

Table 4.9: Performance Evaluation - Average Delays over WAN - Experiment 4 Weak Consistency Mode (Mode B)

Caching.

For the case of cache hits, the two cases were 21% and 12% faster than Distributed Web Caching respectively. This is due to lower communication delays for the Squid server while and faster and efficient cache search performance of the campus proxy, partially compensates for the slightly higher communication delay compared to the Squid server which was on the LAN.

4.5.4 Experimental Setup for Performance over WAN

A special experiment was setup to measure the performance of Distributed Web Caching over Wide Area Networks, considering possible constraints in collaborating with remote LANs using IP multicast. A similar setup to Experimental Setup 1 described in Section 4.5.1 was setup at a remote location connected to the internet via a 128 kbps leased line.

A VPN tunnel was established using `poptop` VPN server, so that the remote network would see the local network to be in the same subnet. Encryption was disabled to keep the tunnel encapsulation overhead to a minimum. Experiment 1 was repeated with this setup.

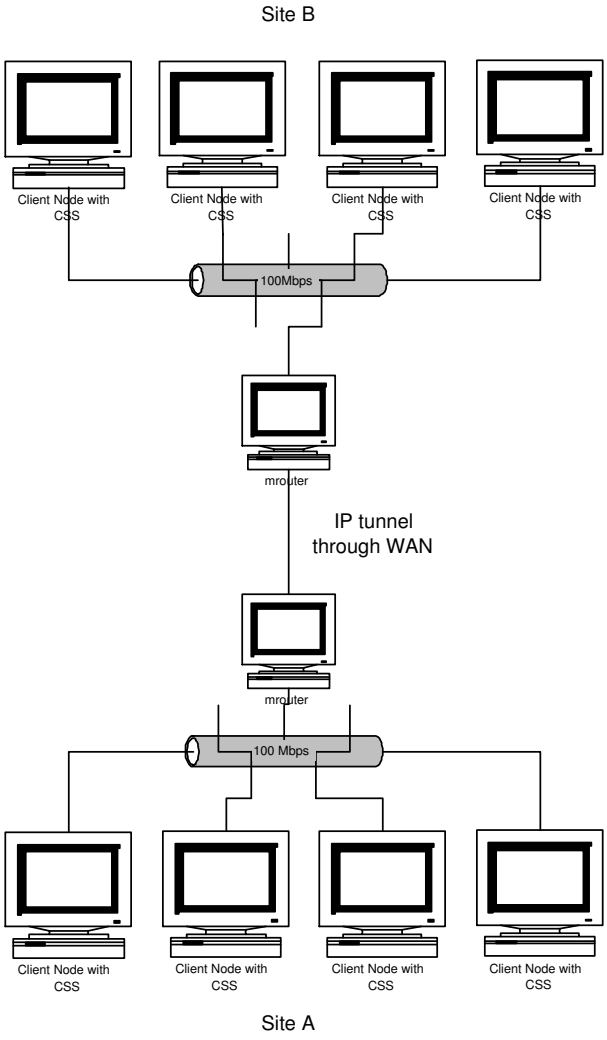

Figure 4.11: Experimental Setup 4

As shown in Table 4.9, the multicast overhead was slightly higher while file transfer delay was significantly higher and the transfer speeds from local network to the remote network were only marginally better than those from the local network to the origin servers. Therefore, the performance significantly deteriorated compared to Experiment 1.

For the system to have a significant speedup, the communication latency between any two client nodes should be at least an order of magnitude smaller than the latency to access external servers, and they should be connected by a network of bandwidth at least and order of magnitude more than the external bandwidth. Given a high local area bandwidth, if the distributed cache could produce more cache hits than the centralised cache, it is possible to obtain an overall speedup in access latency. This is expected due to the large number of clients participating in the system.

It must be noted that the proposed system does not have the disadvantages of a centalised network component such as a proxy server, which requires a highly powerful and reliable server grade hardware with a high maintained cost. A central node also could become a single point of failure and a hot-spot.

4.6 Recent Developments in Distributed Web Caching

The work reported in this thesis was first published in year 2000 [58]. It is believed to be the first publication that proposed a web caching system which exploits client side caches.

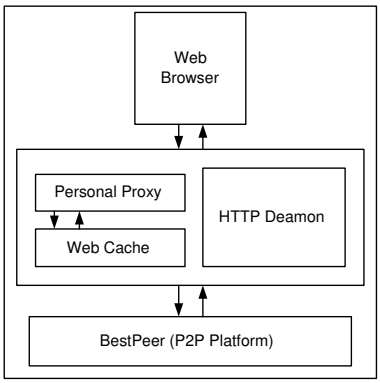

Figure 4.12: A BuddyWeb Client Node

From year 2002 onwards, a few other researchers have come up with improved methods of exploiting client caches [59], [60], [54], [61], [62].

4.6.1 BuddyWeb

BuddyWeb[62] is a peer-to-peer (p2p) based collaborative web caching system. It is implemented on top of BestPeer[63], an agent based p2p system. The BestPeer network consists of two types of entities: a large number of computers (nodes) and relatively fewer number of *Location Independent Global Name Lookup* (LIGLO) servers. Each participant node must register with a LIGLO and can communicate or share resources with other nodes (i.e. peers) in the network. A LIGLO server, which has a fixed IP address, provides two main functions. It generates a BestPeer Global Identity (BPID) for a so that, nodes with varying IP addresses can be identified as a single unique entity. It also maintains peer's current status, such as whether it is online or offline. Agents are extensively used in searching and to perform tasks at remote peer site.

Figure 4.12 depicts the architecture of an autonomous peer in BuddyWeb. Whenever the web browser submits a URL query, the local proxy will receive and rewrite the query into input format of the BestPeer platform. Based on the requirement, BestPeer generates a mobile agent and dispatches it to the BuddyWeb network to search for matching documents. Upon receiving a match, BestPeer passes the information, i.e., document location back to the personal proxy. Then the personal proxy will issue a direct HTTP request to the peer that has the documents. The peer, upon receiving the HTTP request, will process it by the HTTP deamon, and sends the requested documents to the requester.

- LIGLO servers maintain *Peer-Tendencies* as word lists of meta data of the requested objects by each node. These *Peer-Tendencies*are then transformed to a corresponding vector in a vector space. These vectors are used for dynamic reconfiguration of the BuddyWeb network. The vectors assist the nodes to keep the k peers with the highest similarity, directly connected, where k is a system parameter that can be set by participating peers.

- When a node forwards a query, it will look at the neighbor with the highest similarity value, instead of forwarding the agent to all its directly connected neighbors. When a peer initiates a query, it sets a TTL value for the maximum number of hops for the agent. Each time the agent is forwarded, the hop count it updated, and forwarding is stopped, if it exceeds.

4.6.2 Squirrel

Squirrel[54] is a decentralised p2p web cache, built on a self organising p2p routing substrate called Pastry[64] as its object location service, to identify and route to

nodes that cache copies of a requested object. Pastry assigns random, uniformly distributed *nodeIds* to participating nodes (say N in number), form a circular 128-bit namespace. Similarly, application objects are assigned uniform random *objectIds* in the same same namespace. Objects are then mapped to the live node whose nodeId is numerically closest to the objectId. To support object insertion and lookup, Pastry routes a message toward the live node whose nodeId is numerically closest to a given objectId, whithin an expected $log_{2^b}N$ routing steps. In a network of 10,000 nodes with $b = 4$, an average message would route through three intermediate nodes. Despite the possibility of concurrent failures, eventual message delivery is guaranteed unless $l/2$ nodes with adjacent nodeIds fail simultaneously, where, l has typical value $8 * log_{16}N$. Node additions and abrupt node failures are efficiently handled, and Pastry invariants are quickly restored.

Pastry also provides applications with a leaf set, consisting of l nodes with nodeIds numerically closest to and centered around the local nodeId. Applications can use the leaf set to identify their neighbours in the nodeId space, say for replicating objects onto them.

In Squirrel, it is assumed that all nodes can access the internet either directly or through a firewall. Each participating node runs an instance of Squirrel. On each node, browser cache is disabled and Squirrel instance is set as the proxy server. Web browsers issue their requests to Squirrel proxy running on the same node. If the proxy knows that the object is uncacheable, it forwards the request directly to the origin server. Otherwise, it checks the local cache. If a fresh copy of the object is not found in the local cache, then Squirrel essentially tries to locate a copy on some other node. It starts by mapping the object URL (the key) to a node in the

network using Pastry, as follows. It computes a SHA-1 hash of the URL to obtain a 128-bit objectId, and invokes Pastry routing procedure to forward the request to the node with nodeId numerically closest to this objectId. It designates the recipient as *home node* for this object.

Squirrel has two modes of operation.

1. Home-store:

   In this model, Squirrel stores objects both at the client caches and at its home node. If the local search fails, Squirrel node issues a cGET or a GET request to the home node. If the home node has a fresh copy, it directly connects back to the client node and responds with the object or a not-modified message as appropriate.

   If the home node instead finds a stale copy in its cache, or if it incurs a cache miss, it issues a cGET or a GET request to the origin server respectively. If the origin responds with a not-modified message or a cacheable object, then the home node revalidates the local copy or stores the object as appropriate, and forwards a suitable response to the initiating client.

   All external requests to an object are routed through its home node. So home node normally maintains the mode up-to-date copy of the object in the Squirrel network, and does not have to search among other nodes before responding. On each home node, the replacement policy treats all objects are treated equally, regardless of whether the object was stored there because the node was its home, or because the node accessed it recently (or both).

2. Directory:

In this model, a node that recently accessed an object can be asked to serve it to subsequent clients. In this approach, the home node for an object remembers a small directory directory of K (e.g., 4) pointers to nodes that have most recently accessed the object. The key idea is to redirect subsequent requests to a randomly chosen node among these (called the delegate), which would be expected to have a copy of the object locally cached. This protocol maintains the invariant that these copies stored at the delegate nodes are the same *version* of the object. Squirrel synthesises and ETag for internal use by hashing the object contents to keep track of the version.

4.6.3 Other Approaches

A Churn-Resistant Peer-to-Peer Web Caching System [61] by Linga *et al.* built on Kelips peer-to-peer routing substrate focuses on maintaining a constant load on participants and is able to reorganise continuously under churn.

There is another interesting approach to cache co-operation among client nodes which uses proxy servers as directory servers that keep track of browser cache contents [59], [60].

In the approach introduced by Xiao *et al.*[59], for a given browser request, browser cache is first searched, attempting to satisfy the request. If the object is not found in the local cache, the request is forwarded to the proxy server, where the proxy cache is searched for the same purpose. If the search fails to produce a hit, the browser index file is searched where, the URL is matched in another client machine. The proxy server informs the latter client to forward the object to the requesting node, or fetches the cached object to the proxy cache and forwards it to the requesting

node.

In the system introduced by Zhu *et al.* [60], the proxy and the client caches try to optimize the cache contents using a hierarchical greedy-dual (Hier_GD) replacement algorithm. Here, the client caches are organised in a p2p cache using the Pastry overlay introduced in Section 4.6.2. When the proxy has to evict an object, it then stores the evicted object in a node in the p2p cache using a distributed hash table (DHT) functionality.

Experimental results on both Browsers-Aware systems indicate that, hit rates and response times are much better when there is co-ordination between the proxy and the client caches, than when there is no co-ordination between the two levels. The obvious drawback in this approach is the presence of a centralised component, leading to hot-spot phenomenon, reliability and scalability issues, and high operating costs.

4.7 Chapter Summary

Since local area networks nowadays have ample bandwidth and the client nodes are becoming increasingly powerful and resourceful in terms of memory and storage, it could be possible that the clients could take part in web caching. These client nodes could be combined to form a distributed caching system. In this chapter, a protocol for such a Distributed Web Caching system is proposed. The system is then implemented and tested. The performance is measured and compared with a conventional centralised caching system. Although the access latencies were higher in the case of Distributed Web Caching compared to centralised caching, the aim here is to provide a similar service at no extra cost and without the disadvantages

of having a central component in the network. In [59], Xiao *et al.* have shown, that a co-operative browser cache such as the one introduced here, has better hit rates and byte hit rates compared to having isolated browser caches. It is also shown that this performance could be further enhanced using a proxy server to co-ordinate the co-operative browser caches.

Several researchers have proposed co-operative browser caches using a p2p approach. A general problem with the use of p2p routing substrates for web caching is that they are too "heavy-weight" because individual clients may not participate in the p2p network for very long necessitating constant updates of the distributed data structures[65].

However, it must be noted that, the communication overhead in locating an object is considerably lower in structured p2p systems compared to unstructured p2p systems such as the one introduced in this thesis. The same guarantee cannot be given to access latency, as it depends on the number of hops, the p2p substrate requires to locate an object. When client nodes join and leave the distributed cache, there is no overhead in the unstructured approach, while the in structured approach there is a considerable time lag and a computational overhead for the system to fully adjust to such changes.

Chapter 5

An analysis of Distributed Web Caching

5.1 Introduction

In this chapter a study on object movement in the LRU (least recently used) stack is carried out to provide a novel perspective to behavior of a Distributed Web Cache. This study shows that Distributed Web Caching increases *the maximum inter-arrival time between two adjacent requests for a given document without a miss.*

A mathematical study on speedup of web caching, shows that Distributed Web Caching always produces a speedup of greater than unity. This analysis provides an upper bound for speedup and a maximum for multicast timeout, if a speedup is to be greater than unity.

5.1.1 Miss Rate in Distributed Web Caching

Let there be N documents of equal size in the sample space. Let there be M caches of capacity C_k $(k = 1, 2, 3, ..., M)$ measured in unit document size in the Distributed Caching System. Let us assume that Least Recently Used (LRU) algorithm is being used in all M caches for object replacement.

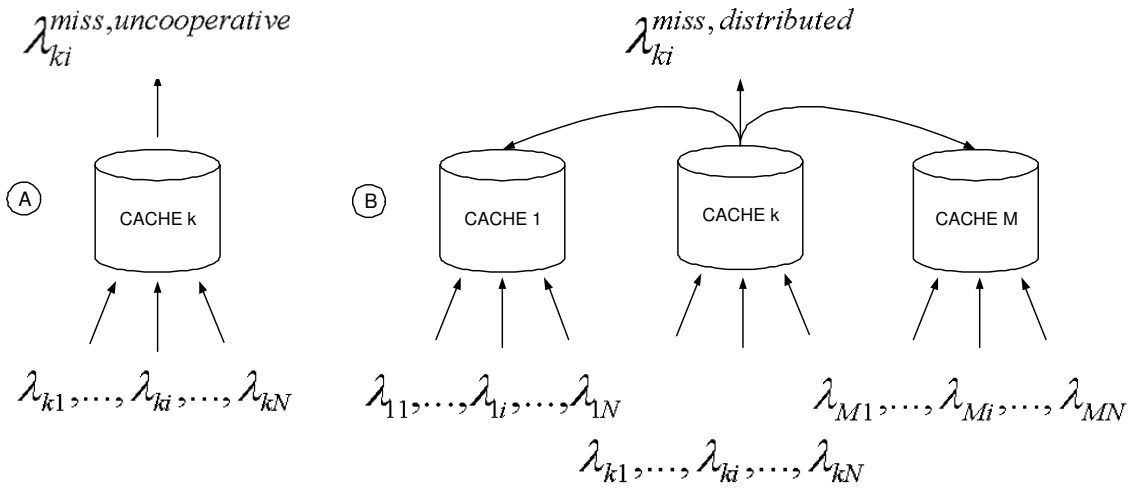

Figure 5.1: A:Unco-operative and B:Distributed Web Caching

As shown in Figure 5.1A, let the aggregate request arrival process at cache $k(k = 1, 2, 3, ..., M)$ be Poisson with mean arrival rate λ_k. Let the arrivals of the request for individual document $i(i = 1, 2, 3, ..., N)$ at cache k be independently sampled from the aggregate arrival process based on the probability set p_{ki} , where p_{ki} is the access probability for the document i at cache k. Here, p_{ki} is modelled using Zipf-like distributions.

$$p_{ki} = K_k \frac{1}{R_k(i)^{Z_k}} \tag{5.1}$$

where $(k = 1, 2, 3, ..., M)$,$(i = 1, 2, 3, ..., N)$ and

$$\sum_{k=1}^{N} p_{ki} = 1 \tag{5.2}$$

$$\lambda_{ki} = p_{ki}\lambda_k \tag{5.3}$$

where λ_{ki} is the mean arrival rate for document i in node k, K_k is the normalization factor, $R_k(i)$ is the popularity rank of the document i at cache k and Z_k is the hot-set parameter taking values such that they correspond to hotsets of 30/5, 20/5, 10/5 etc. Here, hot-set 30/5 refers to the case where 30% of all requests are for the hottest 5% of the pages [1].

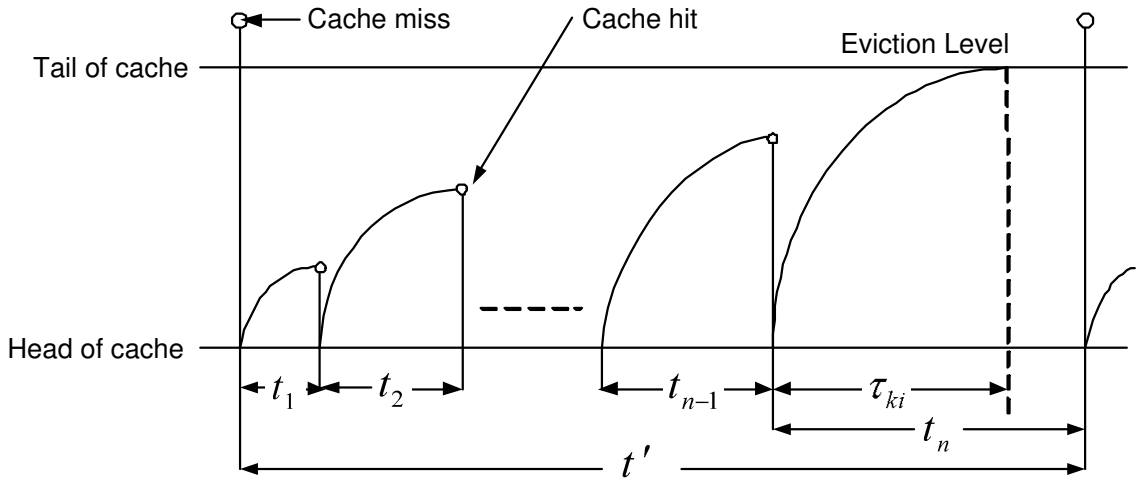

Figure 5.2: General LRU stack movement of a document in a node

General movement of an object in the LRU stack is illustrated in Figure 5.2. After a document is accessed, it occupies position at the head of the stack. The document starts moving towards the tail of the cache due to other documents being accessed. If there are no repeat accesses the document gets evicted once it reaches

the tail of the cache. The number of documents the cache can hold is determined by cache capacity. Immediately after a repeat access the object moves to the head of the cache. The vertical lines indicate repeat accesses. The time epochs t_1,t_2 to t_{n-1} are random variables and are not necessarily in increasing order. The behavior with respect to time is unique to LRU since, it only considers time since last access as the key for replacement.

As illustrated in Figure 5.2 and explained in [66], let us consider the inter-arrival time between two successive cache misses for document i at cache k. This is composed of $n-1$ *independent identically distributed* (i.i.d.) random variables $\{t_1, t_2, t_3, \ldots, t_{n-1}\}$ and an independent random variable t_n. Here the $n-1$ random variables correspond to the time period between two successive hits and the time period t_n refers to the time interval between the last cache hit and the next cache miss.

$$t' = \sum_{i=1}^{n-1} t_i + t_n \tag{5.4}$$

The effect of $n-1$ constituent Poisson processes is represented by a mean field τ_{ki} where, $t_j \leq \tau_{ki}$ $(j = 1, 2, \ldots, n-1)$ and $t_n > \tau_{ki}$. Here, τ_{ki} is defined as the maximum inter-arrival time between two consecutive requests for a document without a cache miss.

As shown in Figures 5.1A and 5.1B respectively, let $\lambda_{ki}^{miss,uncooperative}$ be the miss rate for uncooperative caching and $\lambda_{ki}^{miss,distributed}$ be the miss rate of distributed caching. Using the mean field approximation approach, it has been shown in [66] that for uncooperative caching the miss rate is as follows.

$$\lambda_{ki}^{miss,uncooperative} = \lambda_{ki} e^{-\lambda_{ki} \tau_{ki}^{uncooperative}} \tag{5.5}$$

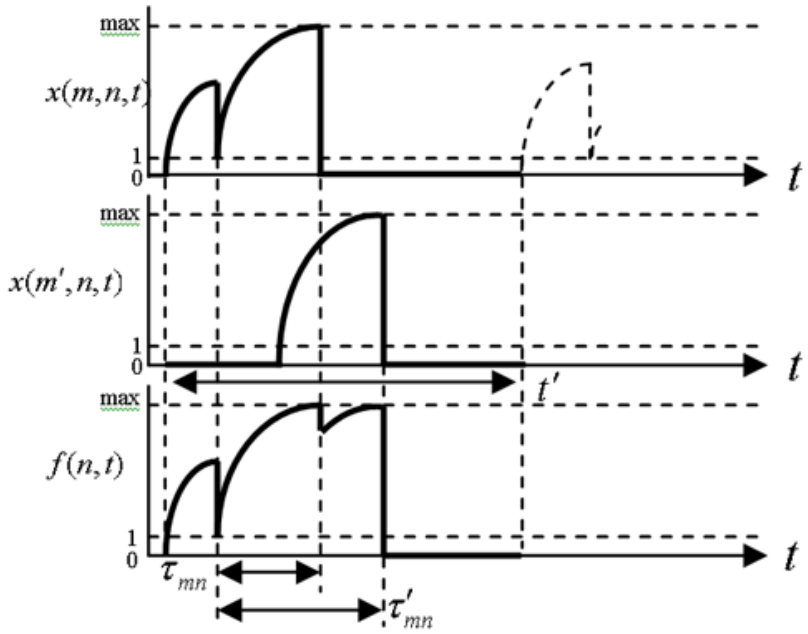

Figure 5.3: Resultant LRU stack movement

This result could be further extended to Distributed Web Caching. In a Distributed Web Caching system, the local misses are forwarded to co-operating caches in the group as shown in Figure 5.1B. A portion of these requests will receive hits from the co-operating caches. Therefore, only those requested objects that are not found in all co-operative caches are considered as *distributed cache misses* and requested from the origin servers.

Similar to $\tau_{ki}^{uncooperative}$,$\tau_{ki}^{distributed}$ can be defined as the maximum inter-arrival time between two adjacent requests for a given document i without a miss from the distributed cache.

As illustrated in Figure 5.3, if we consider the time period t' between two cache

misses for page m in node n, the maximum inter-arrival time between two consecutive requests without a miss for page m in node n is τ_{mn}. If we take the resultant movement of the same page n in node m and m', the maximum inter-arrival time between two consecutive requests without a miss becomes τ'_{mn}.

Let us consider the LRU stack movement of a particular page i in node k and let $x(k,i,t)$ represent the stack position as a function of time t for the duration t'. Let $x(k,i,t) = 0$ if the page is not cached. Then the resultant stack movement with function $f(i,t)$ is derived as follows,

$$f(i,t) = max\{x(k,i,t)\} \tag{5.6}$$

where $k = 1, 2, \ldots, M$

In function $x(k,i,t)$ and the resultant function $f(i,t)$, let $\tau_{ki}^{uncooperative}$ and $\tau_{ki}^{distributed}$ be the maximum inter-arrival time between two consective requests without a miss respectively. Here, $\tau_{ki}^{distributed} > \tau_{ki}^{uncooperative}$ if any of the other nodes has the same object cached when node k has to evict the object.

Otherwise, $\tau_{ki}^{uncooperative} = \tau_{ki}^{distributed}$ if node k was the only node which had the object i cached at the time of eviction. Therefore,

$$\tau_{ki}^{distributed} \geq \tau_{ki}^{uncooperative} \tag{5.7}$$

Since $\lambda_{ki}^{miss,uncooperative} = \lambda_{ki} e^{-\lambda_{ki}\tau_{ki}^{uncooperative}}$ and $\lambda_{ki}^{miss,distributed} = \lambda_{ki} e^{-\lambda_{ki}\tau_{ki}^{distributed}}$ it could be concluded that,

$$\lambda_{ki}^{miss,uncooperative} \geq \lambda_{ki}^{miss,distributed} \tag{5.8}$$

This means the cache miss rate decreases due to co-operative caching other than in the special case where $\lambda_{ki}^{miss,uncooperative} = \lambda_{ki}^{miss,distributed}$. This is possible when

Table 5.1: Symbols in mathematical expressions

Definition	Symbol
Hit ratio at local cache	h_{local}
Hit ratio at neighboring shared caches	h_{shared}
Average time to retrieve an object from local cache	T_{local}
Average time to retrieve an object from neighboring shared caches	T_{shared}
Average time to retrieve an object from origin server	T_{origin}
Average time overhead for cache discovery for a hit	$T_{o,hit}$
Average time overhead for a cache discovery for a miss	$T_{o,miss}$

object i is the most popular in node k or when LRU stack movement of object i on other nodes are not time overlapping with node k in such a way that could increase $\tau_{ki}^{distributed}$.

5.1.2 Speedup Due to Distributed Web Caching

The decrease in distributed cache miss rate does not necessarily translate into a decrease in average user response. In order to quantify the reduction in user response time, access delays and overheads have to be incorporated into the analysis.

Consider a Distributed Caching System as shown in Figure 4.3. Consider a subnet where client caches are shared with a high-speed local network and are connected to the origin servers via a wide area network. There are no intermediate caches.

- Assumption 1: The average time to retrieve an object from the local cache is

significantly less than the average time required to retrieve the object from a neighboring cache or origin server. ($T_{local} \ll T_{shared}, T_{origin}$)

On average, local disk I/O is faster than network file transfers especially when there is congestion.

- Assumption 2: The average response time ratio T_{shared}/T_{origin} does not change as more caches join the Distributed Cache.

  In reality, increasing cache co-operation shifts the load of the web servers to Distributed Cache and affects the network congestion patterns.

- Assumption 3: Shared client caches are randomly distributed across the network.

  In reality, certain client caches may have faster access to the origin servers than others. However, by assuming a random distribution, we can use an average response time (T_{shared}), and hit ratio (h_{shared}) [67].

For Distributed Web Caching to be viable, the speedup that can be achieved using Distributed Web Caching has to be greater than unity. Speedup S, can be defined as the ratio of task time without Distributed Web Caching to task time with Distributed Web Caching.

$$S = \frac{T_{without}}{T_{with}} \tag{5.9}$$

Without Distributed Web Caching, all the local cache misses are forwarded to the origin servers.

$$T_{without} = h_{local}T_{local} + (1 - h_{local})T_{origin} \tag{5.10}$$

$$T_{with} = h_{local}T_{local} \quad (5.11)$$
$$+ (1-h_{local})[h_{shared}(T_{shared}+T_{o,hit}) + (1-h_{shared})(T_{origin}+T_{o,miss})]$$

Therefore,

$$S = \frac{A+B}{A+B[h_{shared}(\frac{T_{shared}+T_{o,hit}}{T_{origin}})+(1-h_{shared})(1+\frac{T_{o,miss}}{T_{origin}})]} \quad (5.12)$$

where, $A = h_{local}\frac{T_{local}}{T_{origin}}$ and $B = 1 - h_{local}$.
Since $A \geq 0$, for all h_{local} where $(1-h_{local}) > 0$, for S to be greater than or equal to unity ($S \geq 1$),

$$[h_{shared}(\frac{T_{shared}+T_{o,hit}}{T_{origin}})+(1-h_{shared})(1+\frac{T_{o,miss}}{T_{origin}})] \leq 1 \quad (5.13)$$

If we assume the overheads $T_{o,hit}$ and $T_{o,miss}$ are negligible this simplifies to $h_{shared}(T_{origin} - T_{shared}) \geq 0$.

Since $0 \leq h_{shared} \leq 1$ and $T_{origin} \gg T_{shared}$, the above condition is satisfied. Hence the conditions for ($S \geq 1$) are valid.

Let us consider the conditions for a speedup of less than unity ($S < 1$). For S to be less than 1,
$h_{shared}(T_{origin} - T_{shared}) \leq 0$.

Since $0 \leq h_{shared} \leq 1$ and $T_{origin} \gg T_{shared}$, above condition is not valid. Therefore, speedup is always greater than or equal to unity ($S > 1$), provided that $h_{shared} > 0$.

We could also derive an upper bound for Speedup S. Let us consider the case where the local cache access time is much faster than the shared cache access time and shared cache access time is much faster than origin server access time ($T_{local} \ll$

$T_{shared} \ll T_{origin}$) and overheads are negligible ($T_{o,hit} = T_{o,miss} = 0$) . This provides an expression for the maximum speedup S_{max}.

$$S_{max} = \frac{1}{1 - h_{shared}} \tag{5.14}$$

If we assume $T_{o,hit} \neq 0$ and $T_{o,miss} \neq 0$ we could express Equation 5.13 as:

$$h_{shared}[(T_{orign} + T_{o,miss}) - (T_{shared} + T_{o,hit})] \geq T_{o,miss} \tag{5.15}$$

Here, $[(T_{orign} + T_{o,miss}) - (T_{shared} + T_{o,hit})]$ is the time saving due to a shared cache hit. Therefore, the multicast timeout $T_{o,miss}$ has the following upper bound.

$$T_{o,miss} \leq h_{shared} * \text{time saving due to a shared cache hit} \tag{5.16}$$

5.2 Simulation Experiments

5.2.1 Assumptions

For simplicity, the assumptions of our simulation model of Distributed Web Caching are as follows:

- User requests are generated using Zipf-like access probability distribution p_{ki} in (1).
- All nodes have the same activity.
- All documents are of the same type and size
- Replacement policy is restricted to LRU on all nodes

- The issues of dynamic contents and non-cacheable contents are not considered here.
- All cached documents are assumed to remain fresh.

5.2.2 Objectives

The objectives of this experiment are as follows:

- To investigate possible increase in maximum inter-arrival time (τ_{ki}), for two successive requests for a particular document without a cache miss.
- To determine possible speedup due to Distributed Web Caching using experimental values derived in [58].

5.2.3 Experiment 1

Experiment Design

In modelling the cases where nodes have different hot-set curves, we keep z_k unchanged, and vary the pages comprising the hot-sets. It is also important to represent the correlation between hot-sets with a single parameter (ρ). The correlation factor ρ [51] takes on integer values from 1 to N. Let us assume that for Node 1, pages are sorted in decreasing popularity and given page indices 1 to N. Let us then consider Page 1, the most frequently accessed page in Node 1. Page 1 occupies position 1 in the hot-set curve for Node 1. For all other nodes, Page 1 occupies a randomly chosen position between 1 and ρ. More generally, page i in the original curve is placed in a randomly chosen position from 1 to $min(\rho + i - 1, N)$, except that the position occupied by any of the previous $(i - 1)$ pages is not allowed.

Table 5.2: System parameters for the simulation experiment 1

System Parameters	Symbol	Value(s)
Total number of documents	N	1000
Number of nodes (caches)	M	3,6
Cache capacity on each node	C_k	120 documents
Hot-set parameter (hot-set of 30/5)	Z_k	0.65
Cache capacity for no eviction	C_{no_ev}	665
Correlation factor	ρ	1000 (1:max, 1000:min)
Number of requests from each node	R	1000 (1 request/sec)

Cache capacity was chosen to be 30% of the total capacity needed for no eviction. A sharp hot-set of 30/5 was chosen for the access probability distribution so that the chosen cache capacity provides a local hit ratio of about 0.2. Cache warm up was performed by sending the same request stream through the caching system before measurement. Hence any transient effects during cache warm up have been eliminated.

As illustrated on Figure 5.2, τ_{ki} is the total time the object takes from a hit to eviction. Figure 5.4 and Figure 5.5 show the movement of a randomly selected page within a particular node (node 2). After a document is accessed, it occupies position 1 in the LRU stack. The document starts moving towards the tail of the cache due to other documents being accessed. If there are no repeat accesses the document gets evicted once it reaches the tail of the cache. The number of documents the cache can hold is determined by cache capacity.

Here, a vertical line from eviction level to level 0 indicates eviction of the page. Shown in a dashed line is the maximum stack position the same document has on all other nodes. The vertical fall of the dashed line shows the time when all other nodes have evicted the same document. The improvement in τ_{ki} is measured in terms of the time all other nodes are able to retain the document after the local node has evicted the document due to limitation of cache space.

The number of co-operating caches affects the shared cache hit ratio and hence the overall miss rate. Therefore, two cases were simulated where number of nodes is 3 and 6. In both cases, total number of hits, total number of misses, hit rate, miss rate, hit rate and miss rate were recorded.

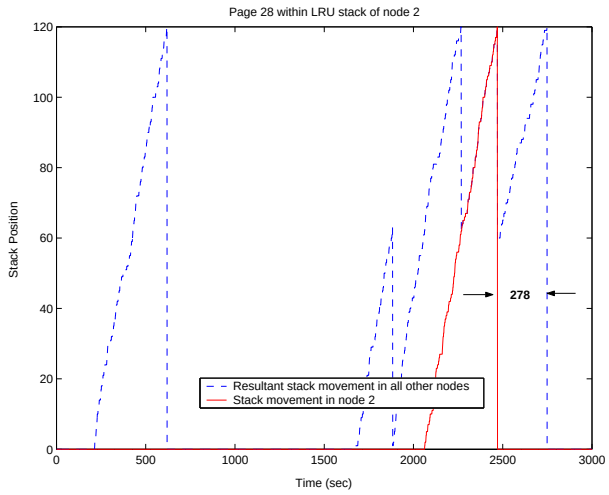

Figure 5.4: Improvement in $\tau_{2,28}$ in node 2 (number of nodes = 3)

Experimental Results

Figure 5.5 illustrates a classic example of the benefit of increased τ_{ki}. Here, Node 2 receives a cache hit within the distributed cache. This is because other nodes have been able to retain the particular document within the distributed cache until Node 2 requests for the same document again. In the long run increased τ_{ki} would cause this phenomenon to repeat reducing the number of distributed cache misses.

Table 5.3 summarises the hit and miss rates due to uncooperative caching and cooperative caching for the two cases with 3 nodes and 6 nodes in the system. From Table 5.3 it could be clearly seen that miss rate decreases due to Distributed Web Caching ($\lambda^{miss,uncooperative} > \lambda^{miss,distributed}$). It is also seen that when the number of co-operating caches increases, distributed cache miss rate is further reduced ($\lambda^{miss,uncooperative}_{M=3} > \lambda^{miss,distributed}_{M=6}$), due to the increase in the effective cache size.

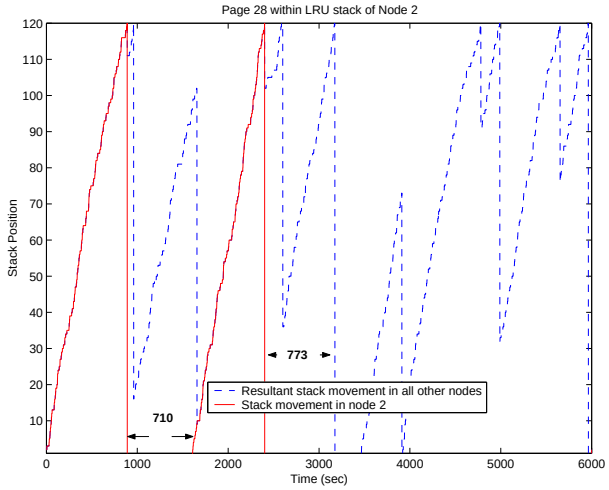

Figure 5.5: Improvement in $\tau_{2,28}$ in node 2 (number of nodes = 6)

Table 5.3: Comparison of hit rates and miss rates in simulation experiment 1

		Unco-operative		Distributed	
No of nodes		Total	Rate (λ)	Total	Rate (λ)
3	Cache hits	627	0.2090/sec	1189	0.3963/sec
3	Cache misses	2373	0.7910/sec	1811	0.6037/sec
6	Cache hits	1286	0.2143/sec	3584	0.5973/sec
6	Cache misses	4714	0.7857/sec	2416	0.4027/sec

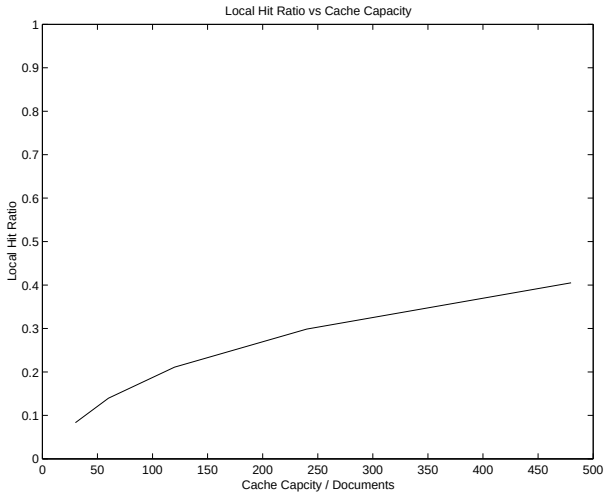

Figure 5.6: h_{local} vs. Cache Capacity

5.2.4 Experiment 2

Experiment Design

The objective of this experiment is to determine the speedup due to Distributed Web Caching in terms of access time. Shown in Table 5.4 are the additional system parameters used in this experiment. Data used here are gathered from a real implementation of the Distributed Web Caching System [58]. This provides realistic values for the delays involved in the protocol.

Experimental Results

The experiment was carried out for 5 cache sizes representing the range 5% to 70% of cache size required for no eviction. Cache co-operation provided a lower access time compared to non-co-operative caching as illustrated in Figure 5.9 and Figure 5.10 respectively. It is clear from Figure 5.11, that there is a speed up of greater

Table 5.4: System parameters for the simulation experiment 2

System Parameters	Symbol	Value(s)
Total hit ratio of the Distributed Web Caching system	h_{total}	refer to Figure 5.8
Shared level hit ratio of the Distributed Web Caching system	h_{shared}	refer to Figure 5.7
Local hit ratio	h_{local}	refer to Figure 5.6
Average time to retrieve an object from local cache	T_{local}	0.100s
Average time to retrieve an object from a shared cache	T_{shared}	0.200s
Average time to retrieve an object from origin server	T_{origin}	2.600s
Average time overhead for cache discovery of a hit	$T_{o,hit}$	0.050s
Average time overhead for cache discovery of a miss	$T_{o,miss}$	0.150s
Cache capacity (documents)	C_k	30,60,. . . ,480
Hot-set parameter (hot-set of 30/5)	Z_k	0.65
Cache capacity for no eviction	C_{no_ev}	665
Number of nodes	M	2,4,. . . ,12

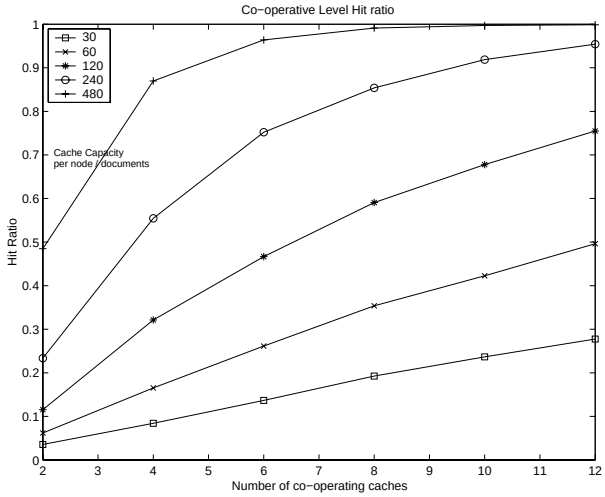

Figure 5.7: h_{shared} vs. Cache Capacity and Number of Nodes in Distributed Web Caching

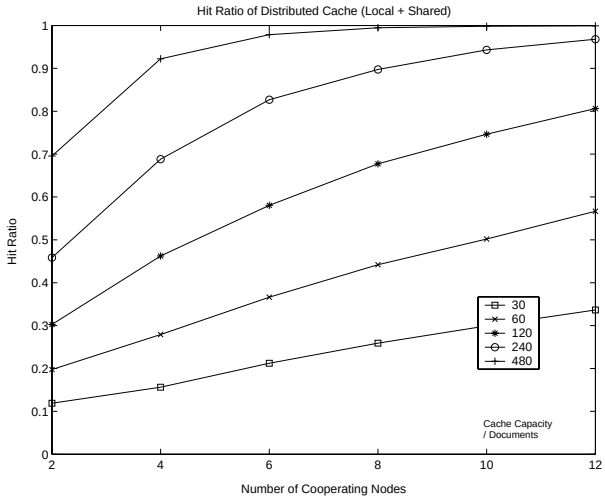

Figure 5.8: h_{total} vs. Cache Capacity and Number of Nodes in Distributed Web Caching

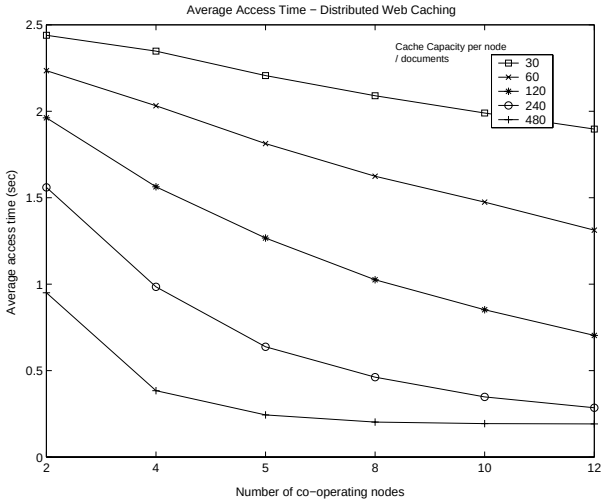

Figure 5.9: Average access time vs. Cache Capacity and Number of Nodes in Distributed Web Caching

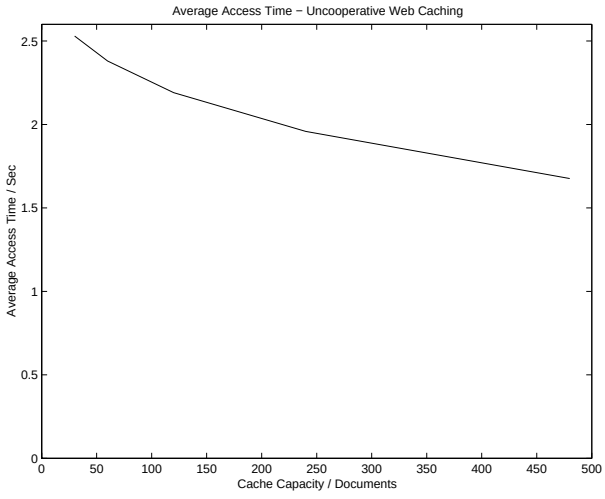

Figure 5.10: Average access time vs. Cache Capacity in uncooperative web caching

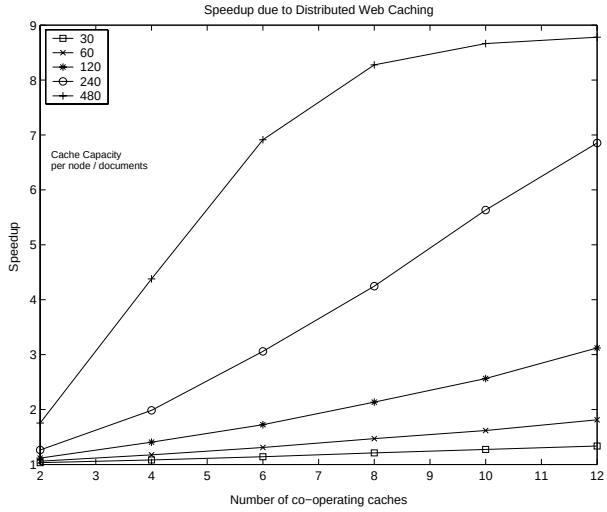

Figure 5.11: Speedup due to Distributed Web Caching vs. cache capacity and number of nodes

than unity throughout the range considered.

However, a reduction of speed up with higher cache sizes (beyond the region considered here) was noticed. This could be attributed to reduction of shared cache hits when the local cache size is closer to the size required for no eviction. Therefore the benefit of external caching reduces.

In reality, cooperative caching systems operate with h_{shared} around 0.3 [67] and h_{local} from 0.2 to 0.7 [47],[67], [68] representing cache sizes much lower than the maximum required for no eviction. There is a definite improvement in access time in regions with those hit ratios (e.g. 4 nodes with capacity 120).

5.3 Chapter Summary

In Section 5.1.1, the effect of Distributed Web Caching to maximum inter-arrival time between two adjacent requests τ was studied. We showed that Distributed Web Caching should increase the value of τ and thereby redue the miss rate of the Distributed Cache. In our simulation study in Section 5.2.3, we investigated the behavior of objects within the LRU stack. It was clearly observed that the value of τ increases due to other nodes caching the same object during the same time period. It was further observed that when the number of participating nodes increases the value of τ further and thereby further reducing the miss rate of the Distributed Cache. In Section 5.1.2, the speedup due Distributed Web Caching was investigated. Conditions for speedup were derived along with an expression for maximum speedup S_{max}. We further derived an upper bound for the multicast timeout $T_{o,miss}$. In our simulation study in Section 5.2.4, we investigated the average access times and the speedup due to Distributed Web Caching. It was observed that the system provides a speedup of more than unity through out the practical range of parameter values.

Chapter 6

Object Replacement in Distributed Web Caching

6.1 Introduction

A complete cache management system requires the following.

- A location strategy
- An admission control strategy
- A replacement strategy
- A placement strategy
- A consistency strategy

In this section we address the problem of object replacement in Distributed Web Caches. The problem is a version of the knapsack problem which is known to be

NP-hard. We propose a heuristic based approach to the problem termed Distributed Web Caching for Global Performance (DWCG). Using event driven simulations, we show that in a Distributed Web Caching environment, the proposed strategy can outperform conventional uncooperative caching strategies such as Least Frequently Used (LFU), Least Recently Used (LRU) and Greedy Dual Size Frequency (GDSF) in terms of Latency Savings Ratio (LSR), Hit Ratio (HR) and Byte Hit Ratio (BHR).

6.2 Object Replacement Strategy

Object replacement algorithms for web caching has gained increased attention among researchers during recent times [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80]. In general, a caching strategy A, takes a request sequence ρ, and a cache of size S in initial state S_0, produces a state sequence S_1,S_2,...,S_m and incurs a cost $W(A, \rho, S)$. The normalized cost is defined as,

$$C(A, \rho, S) = \frac{W(A, \rho, S)}{W_{max}(\rho)} \tag{6.1}$$

where, *maximum cost* $W_{max}(\rho)$ is the cost incurred when no caching is done and $0 \leq C \leq 1$. Given a specific access sequence, an optimal strategy should minimize the cost incurred while accessing uncached objects. Hence it should minimize Equation 6.1.

Object replacement strategies such as LRU [81] and LFU [81] aim to maximize the hit ratio. SIZE [82] aims to maximize the byte-hit ratio. HYBRID [83] and LRV [84] are aimed at maximizing both hit ratio and byte-hit ratio while minimizing cost of access. However, these two strategies are heavily parameterized and require tuning of parameters [85]. GD-SIZE [86] uses access cost along with an aging mechanism

and outperforms LRU, SIZE, HYBRID and LUV [87]. However, it does not take access frequencies into account. A modified version of GD-Size called Greedy-Dual-Size-Frequency [88] takes cost, size and frequency into account, further improving performance of the original GD-Size.

Our primary objective is to minimize client request response time. Clearly, when comparing two objects with the same popularity and size, the document that took a longer time to fetch should be retained at the expense of the other. The access time saving by caching a document can be formally expressed as the latency savings ratio (LSR) which can be expressed mathematically as below.

$$LSR_{A,\rho,S} = \frac{D_{\rho,0} - (D_{A,\rho,S} - D_{\rho,\infty})}{D_{\rho,0}} \tag{6.2}$$

where,

$$D_{\rho,0} = \sum_{i=1}^{N} d_i^{origin}.r_i$$

$$D_{\rho,\infty} = \sum_{i=1}^{N} \delta_i(d_i^{origin}) + d_i^{local}.(r_i - \delta_i))$$

$\delta_i = 1$ if $r_i \geq 1$ and $\delta_i = 0$ if $r_i = 0$

$$D_{A,\rho,S} = \sum_{i=1}^{N} d_i^{origin}.f_i^{origin} + \sum_{i=1}^{N} d_i^{shared}.f_i^{shared} + \sum_{i=1}^{N} d_i^{local}.f_i^{local}$$

$D_{\rho,0}$: Time taken in accessing the request sequence ρ without caching.

$D_{\rho,\infty}$: Time taken in accessing the request sequence ρ with an infinite client cache capacity.

$D_{A,\rho,S}$: Time taken in accessing the request sequence ρ with a cache capacity S on each client node and when replacement algorithm A is applied.

d_i^{origin}**:** Time taken to fetch document i from origin.

d_i^{shared}**:** Time taken to fetch document i from distributed cache.

d_i^{local}**:** Time taken to fetch document i from local cache.

f_i^{origin}**:** total number of origin fetches for document i for capacity S and replacement algorithm A with Distributed Web Caching.

f_i^{shared}**:** total number of fetches from distributed cache for document i for capacity S and replacement algorithm A with Distributed Web Caching.

f_i^{local}**:** total number of local fetches for document i for capacity S and replacement algorithm A with Distributed Web Caching.

r_i**:** total number of references for document i.

Here, $D_{\rho,\infty} \leq D_{A,\rho,S} \leq D_{\rho,0}$. Therefore the range of $LSR_{A,\rho,S}$ is , $\frac{D_{\rho,\infty}}{D_{\rho,0}} \leq LSR_{A,\rho,S} \leq 1$. Therefore, it is clear that the limits of the matrix is dependent on the properties of access sequence ρ and not the replacement strategy A or the cache capacity S.

Latency savings ratio (LSR) introduced here, is derived from Delay Savings Ratio (DSR) used in [89],[90],[82] and [90]. However, this has been modified to remove the effect of *cache warm up* during measurement of latency savings. Therefore, for a given sequence only the steady state performance is compared.

Since the time to satisfy a request from local or distributed cache is smaller than the time to satisfy a request from the origin server ($d_i^{local} < d_i^{shared} < d_i^{origin}$), maximal latency savings ratio guarantees minimal client request response time.

In order to maximize LSR we can combine 3 key parameters of a cached document i (probability of access p_i, size of document $size_i$ and fetch latency t_{access_i}) into one

performance metric called *local benefit*, defined as;

$$local\ benefit_i = \frac{p_i * t_{access_i}}{size_i} \tag{6.3}$$

Here, $local\ benefit_i$ gives the expected latency reduction obtained from caching i per unit size. This is the normalized contribution of document i to LSR. Object replacement is carried out by evicting documents starting from the least beneficial, in the ascending order of benefit until sufficient space is created for the incoming object. In Equation 6.3, probability of access p_i can only be estimated by observing the past reference pattern. This could be done using LRU, LRU-K [91] or LFU.

However, these methods alone do not perform satisfactorily since LRU and LFU only take recency and frequency of access respectively. LRU-K takes in to account last K references. However, sometimes it is not possible to consider K references if fewer than K references have been made. Therefore we propose to combine LRU and LFU as shown in Equation 6.4.

$$p_i \propto \frac{nref_i}{tref_i} \tag{6.4}$$

where, $nref_i$ is the number of references for object i after being cached, $tref_i$ is the time since last reference to object i since caching. This could be incorporated into $local\ benefit_i$ as follows.

$$local\ benefit_i = \frac{nref_i * t_{access_i}}{tref_i * size_i} \tag{6.5}$$

6.3 Replica Awareness in Object Replacement in Distributed Web Caching

Distributed Web Caching assumes there is a considerable correlation between the access streams of different nodes (or users). Due to this correlation and due to independent admission and replacement policies, multiple copies of the same object (replicas) could co-exist in the Distributed Cache.

6.3.1 Detection of Replicas

Detection of replicas could either be done using a centralised server which monitors multiple copies or by using a regular information exchange mechanism between the client caches.

Both these solutions were not feasible since a centralised server totally contradicts the distributed systems approach to overall design and regular information exchange adds extra communications and processing overheads to the system.

However, a multicast request could provide an indication of a possible fetching of that object in the near future into another node. In each cache, a counter is maintained for each object in the cache. The counter starts at 1 indicating the local copy. The counter is then incremented for each multicast request that is detected in the system. When the object is evicted the counter information discarded. Although this approach does not provide the exact number of replicas in the system, it provides an indication of global popularity of the object while the object remains in the cache.

There exist two different approaches to make use of the replica count in object replacement.

1. **To give higher importance to objects requested by other nodes.** This *less greedy* approach expects that if a page is requested by another node, the probability of more nodes requesting for the same page is high. Therefore, the page is given more importance for the benefit of future remote requests. In the following sections, we investigate the performance of this approach.

2. **To give less importance to objects requested by other nodes.** This approach expects the requesting node to cache the object so that other nodes can access that particular copy if there is a local miss. We have introduced this scheme in [92] as a replica-aware extension to replacement policies in Distributed Web Caching. This is discussed in section 6.7.

6.4 Distributed Web Caching for Global Performance (DWCG)

The benefit to the whole Distributed Web Caching system can be named as the global benefit. We express the global benefit as follows.

$$global\ benefit_i = \frac{nref_i * t_{access_i}}{tref_i * size_i} * remote\ benefit\ factor_i \tag{6.6}$$

Here, the *remote benefit factor* has to be dynamically calculated depending on the demand for a particular page by the neighboring nodes. Using event driven simulations we found out that $e^{nremote_i}$, where, $nremote_i$ is the number of remote requests for a particular page, gives a considerable improvement in performance.

Therefore, the proposed priority key for DWCG algorithm is as follows.

$$Priority\ Key_i = \frac{nref_i * t_{access_i}}{tref_i * size_i} * e^{nremote_i} \tag{6.7}$$

In the following section we develop a simulation model for the Distributed Web Caching system and simulate key replacement algorithms such as LRU, LFU, GD-SIZE-FREQUENCY [83] along with the proposed DWC-G algorithm. The performance is then compared according to latency savings ratio (LSR) defined above.

6.5 Simulation Model

First, a simulation model for Distributed Web Caching is developed. It is then simulated using empirically derived analytical workloads. Researchers [93],[48],[6] have frequently observed a number of invariant characteristics of web caching by doing trace analysis. Such information is also used in these simulations.

6.5.1 Assumptions

For simplicity, the assumptions of our simulation model of Distributed Web Caching are as follows [94]:

- All nodes have the same activity.
- The issues of dynamic contents and non-cacheable contents are not considered here.
- The all cached documents are assumed to remain fresh since the protocol has a built in method to guarantee freshness of objects using "polling every time" technique.

6.5.2 Objectives

- Performance comparison of commonly used replacement strategies with the proposed strategy DWCG.
- To understand the effect of varying parameters on performance of caching strategies.

6.5.3 Object Popularity

Assuming that there are N distinct objects and M nodes, probability of access of a document i in node k can be expressed using Zipf-like distributions as follows.

$$p_{ki} = K_k * \frac{1}{R_k(i)^{Z_i}} \tag{6.8}$$

for $k = 1, 2, .., M$ and $i = 1, 2, .., N$ such that $\sum_{i=1}^{N} p_{ki} = 1$, where K_k is a normalisation constant, $R_k(i)$ is the rank of document i in node k and Z_k is the hot-set parameter. Z_k can be adjusted so that the hot-set is 20/5, 10/5, etc. "20/5" corresponds to the case where 20% of all requests are made to the hottest 5% of the objects. In our experiments we simulated hot-sets 20/5 and 10/5.

6.5.4 Correlation of object popularity

In modelling the cases where nodes have different hot-set curves, we keep Z_k unchanged. Instead similar to Experiment 1 on Section 5.2.3, we vary the pages comprising the hot-sets. It is important to represent the correlation between hot-sets with a single parameter (ρ). The correlation factor ρ [51] takes on integer values from 1 to N. Let us consider Page 1, the most frequently accessed page in Node 1.

Page 1 occupies position 1 in the hot-set curve for Node 1 ($R_1(1) = 1$). For all other nodes, Page 1 occupies a randomly chosen position between 1 and N. Any page i in the original curve is placed in a randomly chosen position from 1 to $\min(\rho+i-1, N)$ except that the positions occupied by any of the previous $i-1$ pages are not allowed. When $\rho = 1$, we have perfect correlation between nodes and when $\rho = N$, we have random correlation. In our simulations we use $\rho = \frac{N}{5}, \frac{4N}{5}$ to represent both extremes.

6.5.5 Access Cost of Objects

In modelling the cost of access form origin, we use the model suggested in [23]. Access cost of object can be expressed as follows.

$$taccess_i = \omega.s_i + 2.(1-\omega).A.random(0,1) \tag{6.9}$$

where, s_i is size of object, A is the average size of an object and ω is the cost correlation parameter which lies between 0 and 1. In our simulations, we use $\omega = 0.5$.

6.5.6 Object Size Distribution

We assume that there are N different objects with size s_i. Sizes are uniformly distributed between 1 and N. We simulate three different size distributions [95].

1. First we sort the N objects with descending global popularity $p_i = \sum_{j=1}^{M} p_{ij}$ and assign sizes 1 to N (popularity and size negatively correlated).

2. Next we sort the objects in the ascending order of global popularity and assign sizes 1 to N (popularity and size positively correlated).

3. Finally we shuffle the sizes randomly (popularity and size randomly correlated).

6.5.7 Cache Capacity

We have chosen the cache capacities to be 0.10 to 0.50 of minimum per node capacity required for no replacement S_{NR}. i.e. $0.10 * \frac{\sum_{i=1}^{M}}{M}$ to $0.50 * \frac{\sum_{i=1}^{M}}{M}$

6.6 Simulation Experiments

6.6.1 Experiment 1

Experimental Setup

In our first simulation experiment we compare the LSR for LRU, LFU, GD-SIZE-FREQUENCY algorithms with the proposed DWCG algorithm. Table 6.1 shows the parameters used in the experiment.

Experimental Results

Figures 6.1 to 6.4 illustrate the experimental results for Experiment 1. Four sets of results were obtained for combinations of moderate (20/5) and flatter (10/5) hot-sets with high (N/5) and low ($4N$/5) object popularity correlation settings between nodes.

Table 6.1: System parameters for the simulation experiments

System Parameters	Symbol	Value(s)
Total number of documents	N	5000
Number of nodes (caches)	M	6
Cache capacity on each node	C_k	$0.10 * \frac{\sum_{i=1}^{M} S_{NR}}{M}$ to $0.50 * \frac{\sum_{i=1}^{M} S_{NR}}{M}$
Hot-set parameter	Z_k	0.53 (hot-set of 20/5)
		0.35 (hot-set of 10/5)
Object Size - Frequency Correlation	γ	3:Random (1:Positive, 2:Negative)
Object Size - Cost Correlation Factor	ω	0.5
Object Popularity correlation factor	ρ	$\frac{N}{5}$ to $\frac{4N}{5}$
Number of requests	N	500 per node
Total length of request sequence	L	3000 requests
Local cache access time	d^{local}	$0.04 * d^{origin}$
Shared cache access time	d^{shared}	$0.10 * d^{origin}$

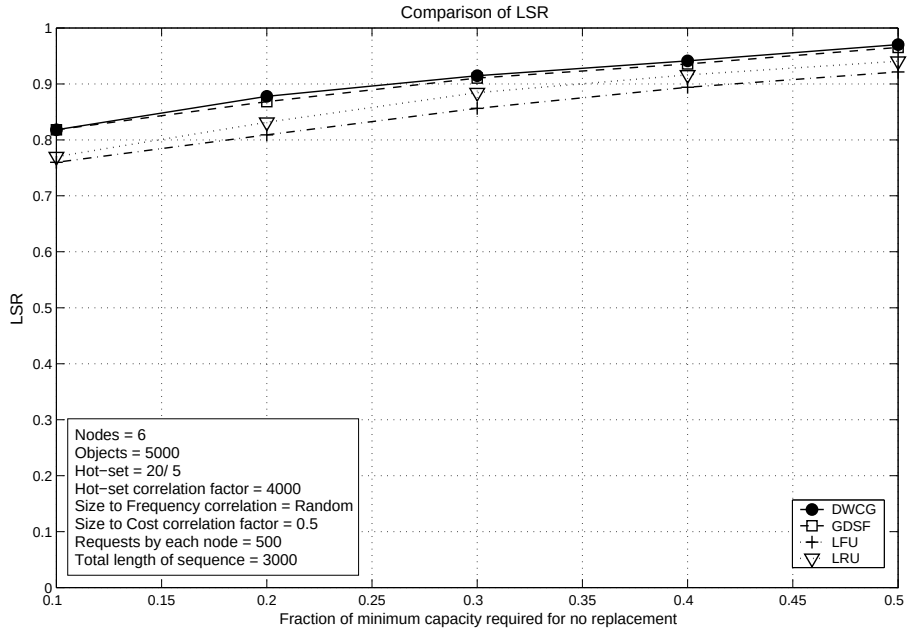

Figure 6.1: LSR: hot-set=20/5 (moderate), popularity correlation (ρ)=$\frac{4N}{5}$ (low)

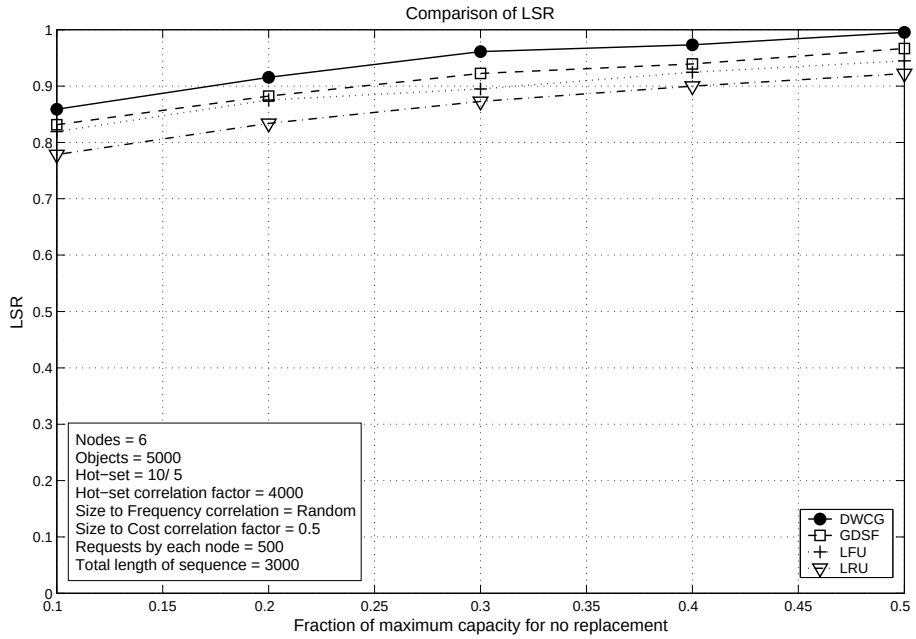

Figure 6.2: LSR: hot-set=10/5 (flatter), popularity correlation (ρ)=$\frac{4N}{5}$ (low)

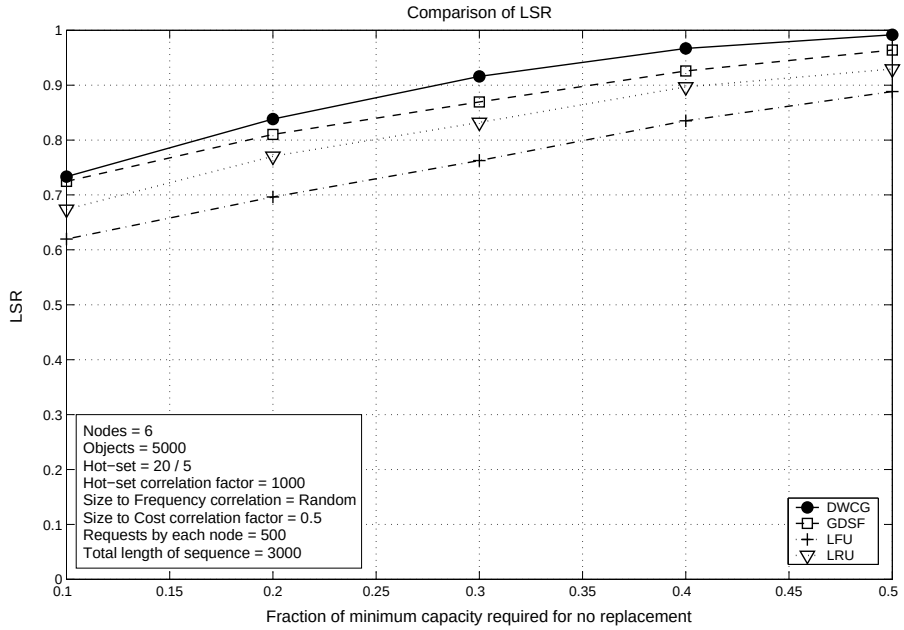

Figure 6.3: LSR: hot-set=20/5 (moderate), popularity correlation (ρ)=$\frac{N}{5}$ (high)

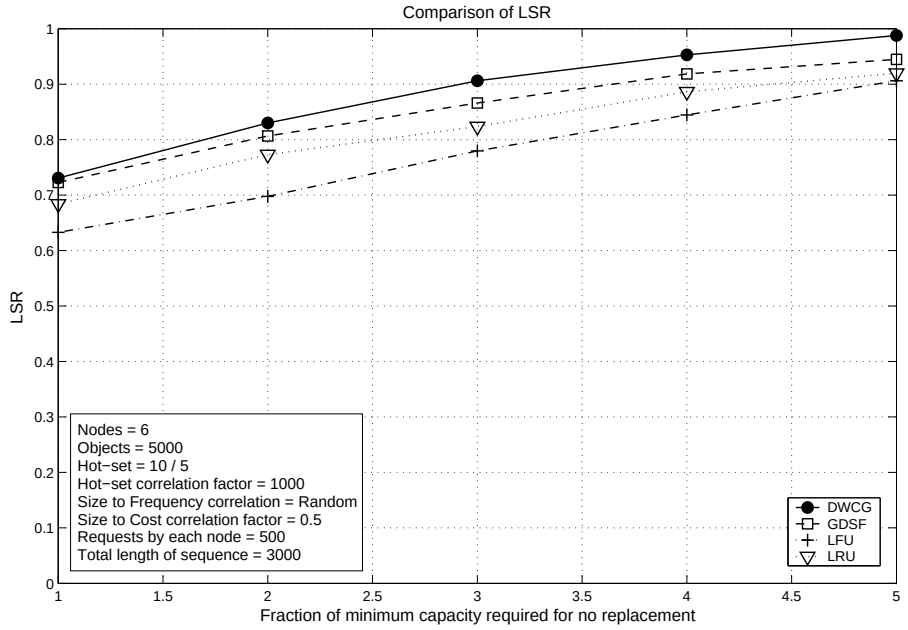

Figure 6.4: LSR: hot-set=10/5 (flatter), popularity correlation (ρ)=$\frac{N}{5}$ (high)

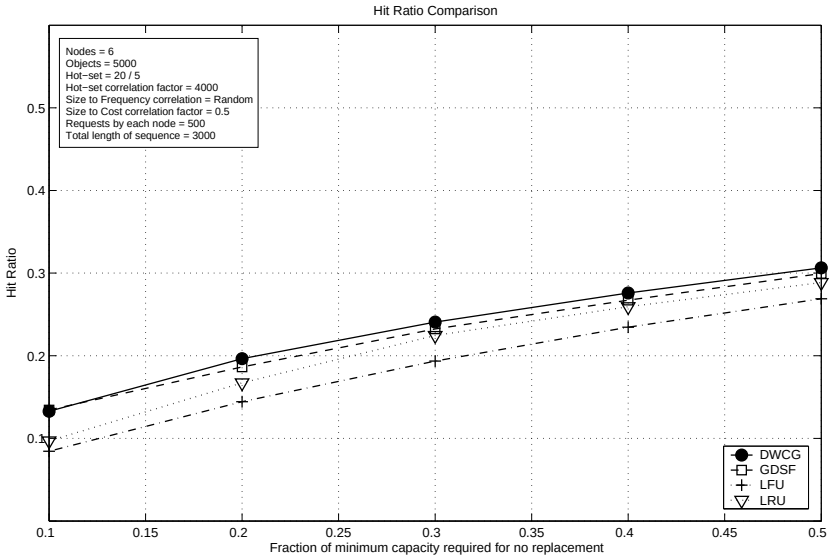

Figure 6.5: Distributed Cache Hit Ratio : hot-set=20/5 (moderate), popularity correlation (ρ)=$\frac{4N}{5}$ (low)

6.6.2 Experiment 2

Experimental Setup

In our second simulation experiment we compare the hit ratio for LRU, LFU, GD-SIZE-FREQUENCY algorithms with the proposed DWCG algorithm. Hit ratio can be considered as a secondary performance matrix. The same simulation parameters used for Experiment 1 were used for this experiment. In calculating the hit ratio, both local and shared cache hits are considered as cache hits.

Experimental Results

Figures 6.5 to 6.8 illustrate the experimental results for Experiment 2.

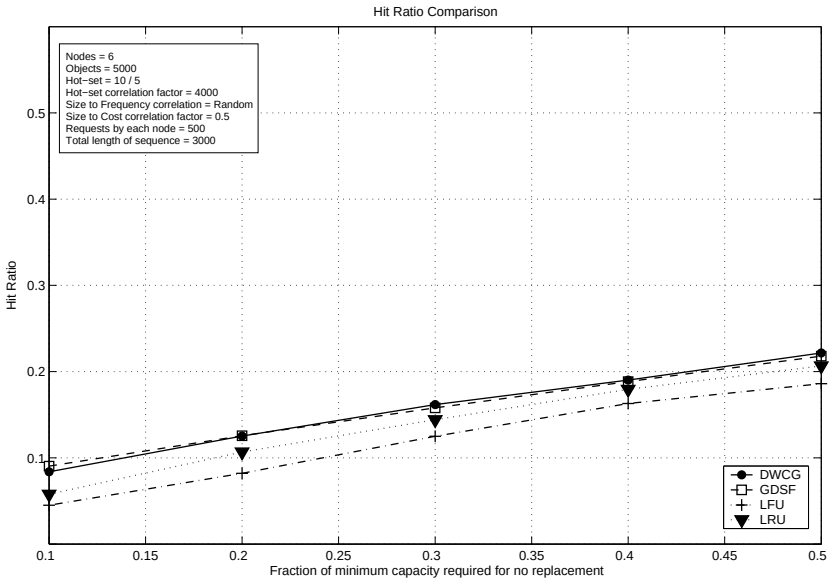

Figure 6.6: Distributed Cache Hit Ratio : hot-set=10/5 (flatter), popularity correlation (ρ)=$\frac{4N}{5}$ (low)

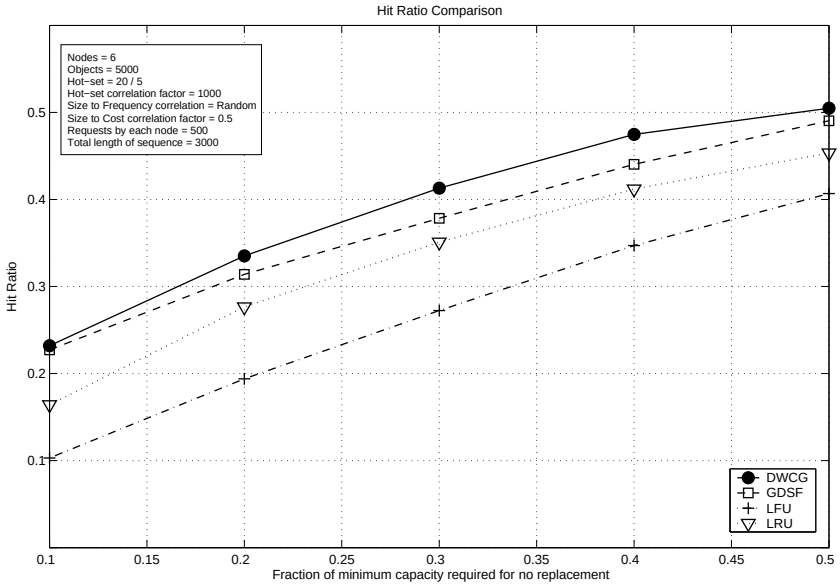

Figure 6.7: Distributed Cache Hit Ratio : hot-set=20/5 (moderate), popularity correlation (ρ)=$\frac{N}{5}$ (high)

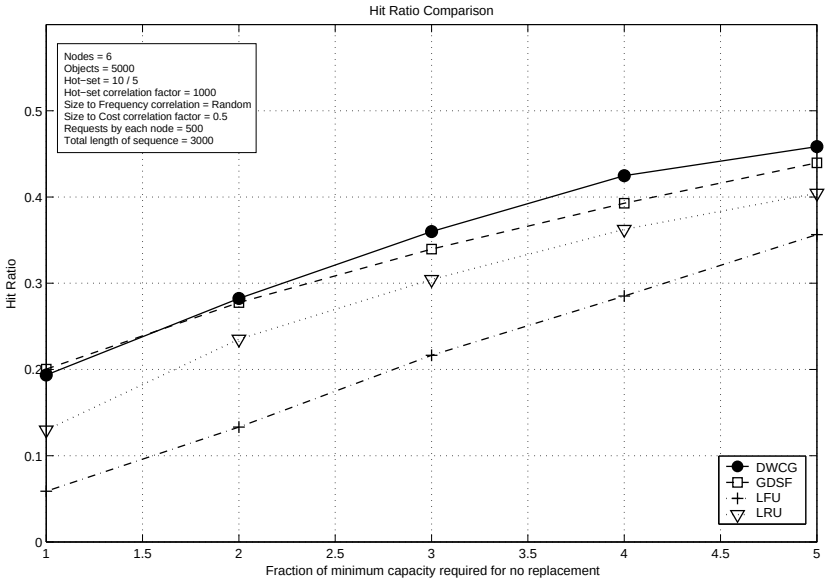

Figure 6.8: Distributed Cache Hit Ratio : hot-set=10/5 (flatter), popularity correlation (ρ)=$\frac{N}{5}$ (high)

6.6.3 Experiment 3

In our third simulation experiment we compare the byte hit ratio for LRU, LFU, GD-SIZE-FREQUENCY algorithms with the proposed DWCG algorithm. Similar to hit ratio, byte hit ratio can be considered as a secondary performance matrix. Significance of byte hit ratio is that the object size is taken into account in measuring the performance compared with hit ratio. In calculating the byte hit ratio, both local and shared cache hits are considered as cache hits.

Experimental Results

Figures 6.9 to 6.12 illustrate the experimental results for Experiment 3.

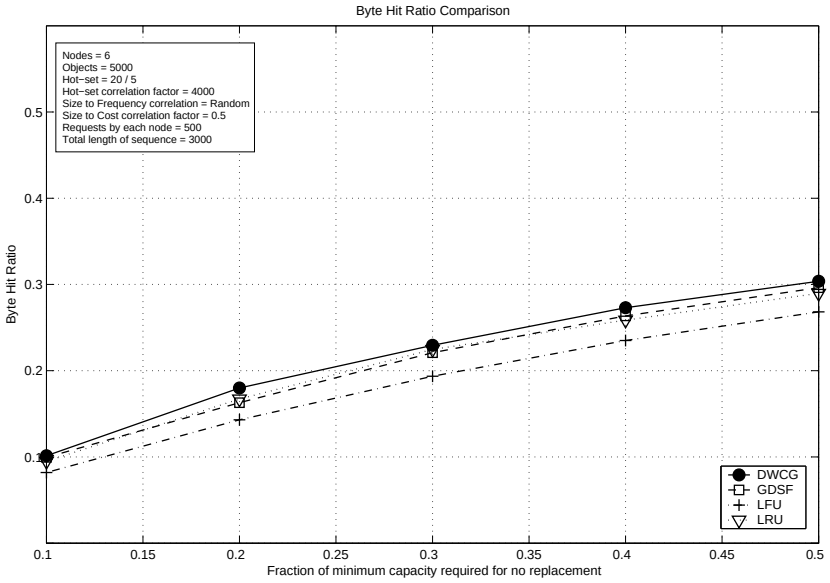

Figure 6.9: Distributed Cache Byte Hit Ratio : hot-set=20/5 (moderate), popularity correlation (ρ)=$\frac{4N}{5}$ (low)

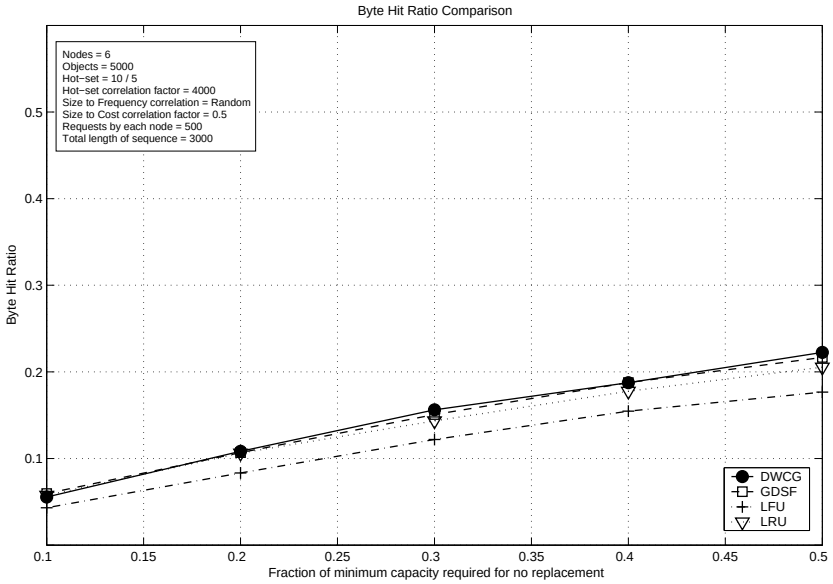

Figure 6.10: Distributed Cache Byte Hit Ratio : hot-set=10/5 (flatter), popularity correlation (ρ)=$\frac{4N}{5}$ (low)

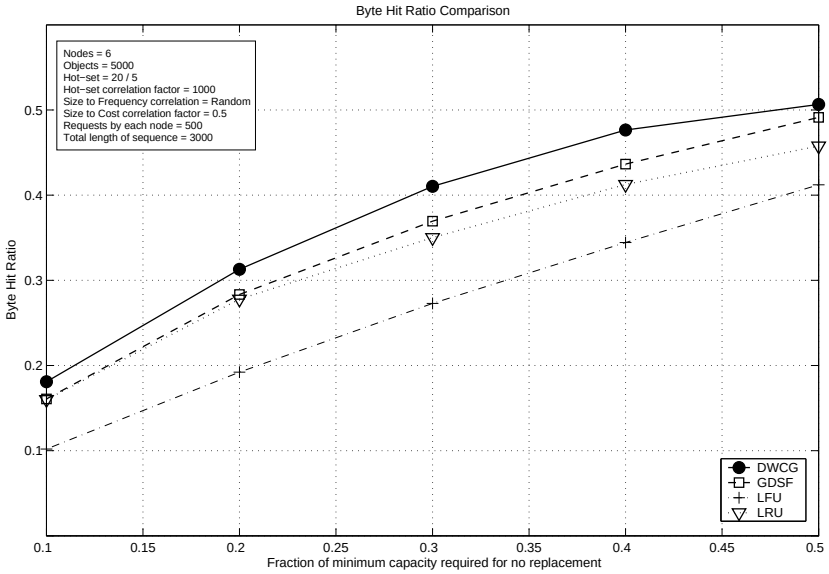

Figure 6.11: Distributed Cache Byte Hit Ratio : hot-set=20/5 (moderate), popularity correlation (ρ)=$\frac{N}{5}$ (high)

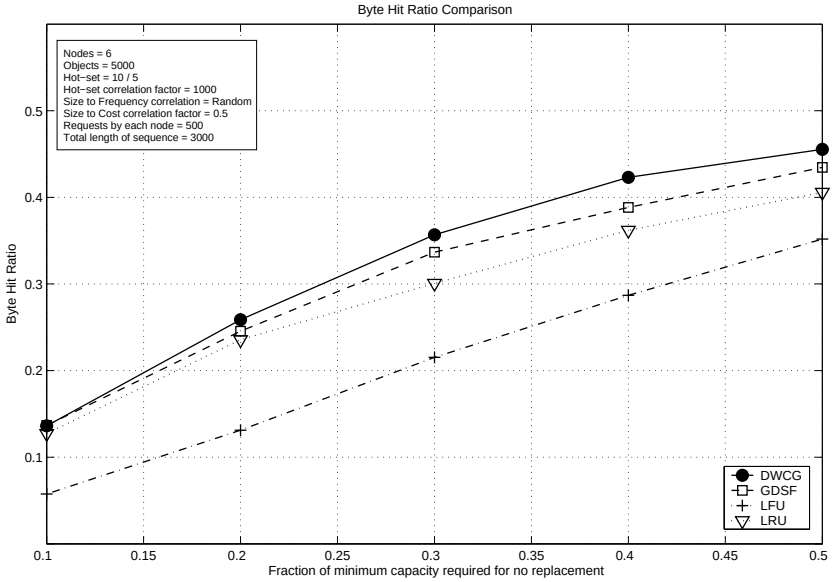

Figure 6.12: Distributed Cache Byte Hit Ratio : hot-set=10/5 (flatter), popularity correlation (ρ)=$\frac{N}{5}$ (high)

Discussion on Experimental Results

It is clear from the results that DWCG out performs LRU, LFU and GDSF in terms of LSR. Gain in LSR is from 4.5% to 17.2%, over LFU, 1.2% to 8.4% over LRU and 1% to 3.4% over GDSF depending on the hot-set and inter-site popularity correlation factor.

In Experiments 2 and 3 the results show that DWCG performs better in terms of HR and BHR. The gains are similar to those in Experiment 1.

An interesting observation is that when there is a close object popularity correlation between nodes, a sharp hot-set yields better performance in terms of LSR, HR and BHR. In contrast, when the object popularity correlation between nodes is low, a flatter hot-set yields better results in terms of LSR, HR and BHR. This is due to the amount of overlap between popular objects in the two cases, if the overlapped objected are popular it yields better performance. In a less correlated case, a sharp local hot-set further reduces the usefulness of cache cooperation. The performance of DWCG is further improved when number of nodes are increased and when the request sequence is longer.

Adding more nodes to the system will improve the overlap of popular objects and hence improve the usefulness of Distributed Web Caching. In addition, length of the request sequence and number of objects in the system affect the performance. However, the key parameters determining the performance of a given scenario are the local hot-set parameter and the inter-site object popularity parameter.

6.7 A Replica-Aware extension for object replacement algorithms in Distributed Web Caching

When inter-node object popularity correlation is high, replica awareness could be used to reduce the local importance of a particular object, when that object is popular elsewhere. This way more importance could be given to objects that are popular only locally.

Some of the commonly used object replacement algorithms could be modified to incorporate this feature as follows:

$$Modified\ Priority\ Key_i = Priority\ Key_i * \frac{1}{Remote\ Dependency\ Factor_i} \quad (6.10)$$

$Remote\ Dependency\ Factor_i$ indicates the dependency on remote nodes for having this object cached. Two Remote Dependency Factors are considered.

RDF-1: $Remote\ Dependency\ Factor_i = nremote_i$

RDF-2: $Remote\ Dependency\ Factor_i = e^{nremote_i}$

6.8 Simulation Experiment

LSR, BHR and HR performances of original LRU algorithm is compared with RDF-1 and RDF-2 versions for the 2 cases when hot sets are 30/5 and 20/5 with inter-node object popularity correlation factor fixed at $\rho = 1000$ (high correlation).

Both versions of RDF has outperformed the original version of LRU. RDF-2 performs marginally better than RDF-1. Since RDF relies on other nodes to cache some

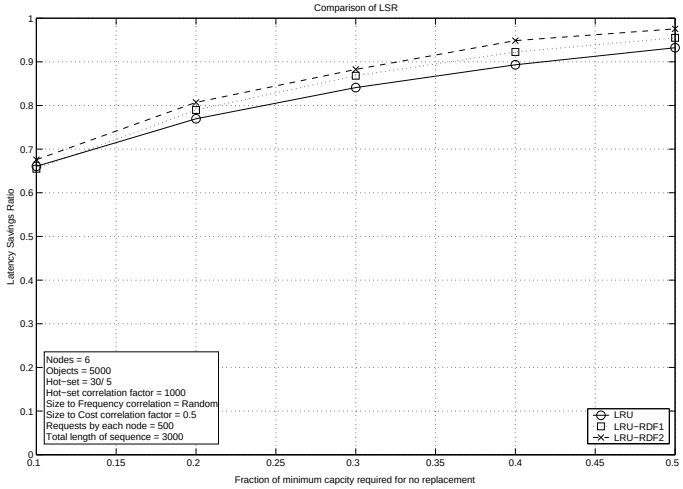

Figure 6.13: Latency Savings Ratio : LRU with hot-set=30/5 (moderate), popularity correlation (ρ)=$\frac{N}{5}$ (high)

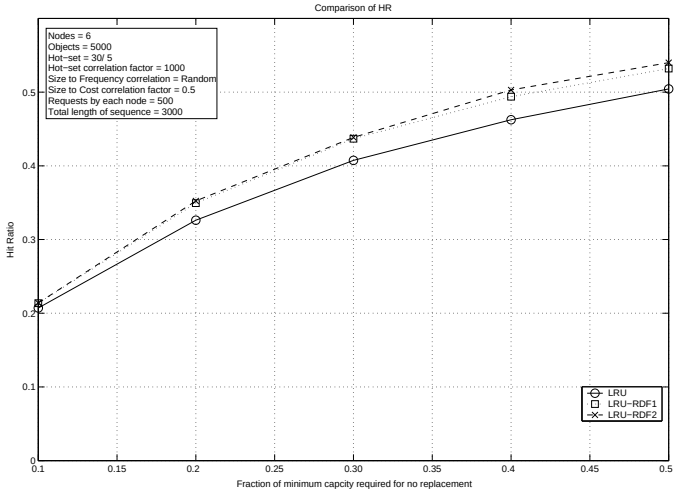

Figure 6.14: Distributed Cache Hit Ratio : LRU with hot-set=30/5 (moderate), popularity correlation (ρ)=$\frac{N}{5}$ (high)

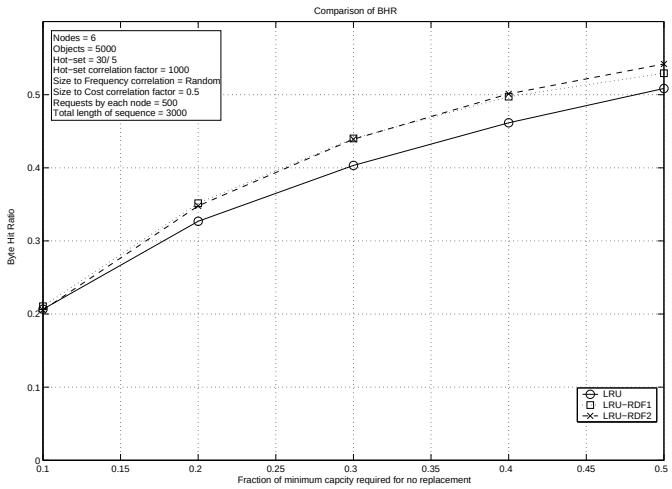

Figure 6.15: Distributed Cache Byte Hit Ratio : LRU with hot-set=30/5 (moderate), popularity correlation (ρ)=$\frac{N}{5}$ (high)

of the locally popular objects, the gains are only visible when there is a significant correlation of hot-sets.

6.9 Chapter Summary

In this chapter, a novel object replacement strategy the DWCG was introduced. The strategy aims at improving the global performance using a co-operative replacement scheme. In order to achieve this, the remote benefit factor was incorporated in to the priority key. The objective of this factor is to reflect the importance of caching the page for the benefit of cooperating nodes by taking in to account the number of remote references so far. We have further introduced a modified performance matrix called Latency Savings Ratio (LSR) to suit Distributed Web Caching. The measurement removes any effect of cache warm up and measures the steady state latency saving.

It was noted that the performance can be further fine tuned by introducing the parameters a and b as shown below.

$$Priority\ Key_i = \frac{nref_i * taccess_i}{tref_i^a * size_i} * e^{\frac{nremote_i}{b}} \tag{6.11}$$

$a = 0.2$ and $b = 4$ yielded marginally better results in the considered scenario. These parameters have to be determined according to the total length of the request stream and the number of nodes in the system respectively. However, for a general case with an infinite request stream and a large number of nodes these two parameters are not required. Using simulation experiments, the performance of the proposed scheme was compared with LRU, LFU and GDSF. It was observed that DWCG out performs LRU, LFU and GDSF in terms of LSR. Gain in LSR is from 4.5% to 17.2%, over LFU, 1.2% to 8.4% over LRU and 1% to 3.4% over GDSF depending on the hot-set and inter-site popularity correlation factor.

Chapter 7

Conclusions

This thesis focuses on providing an alternative to centralised web caching, due to numerous problems associated with centralised network components.

In Chapter 1, a brief introduction to the work reported is provided, along with the scope of the research, areas not covered in this thesis, and the organization of the thesis.

In Chapter 2, a detailed on web caching is provided. Benefits of web caching and HTTP support for web caching are discussed. These are followed by issues in web caching and how co-operative web caching could solve some of those issues is discussed. Co-operative web caching architectures and cache co-operation protocols are also introduced. Properties of two widely used cache co-operation protocols namely, ICP and Summary Cache are investigated and their properties are compared, including the query overhead.

Provided in Chapter 3, is a case study using user access traces collected at Boston University (BU) and University of California, Berkeley (UCB). BU trace logs are unique due to the fact that they have been collected using a modified browser,

therefore, the logs contain, all user requests, which are not filtered by the browser caches. Traces collected at the proxy caches tend to loose some of the requests due to local fulfilment of some of the requests by the browser caches. UCB trace log is collected through the use of a packet sniffing machine placed at the head-end of the Home IP modem bank. All three traces represent academic users, who are most likely to have similar web access patterns. Simulations carried out clearly indicate that there is a considerable benefit in cache sharing compared to isolated browser caching. Our findings have been further verified by results in [59].

In Chapter 4, the concept of a novel Distributed Web Caching system is presented. The software implementation along with modular break down and flow charts are provided. Performance from the experimental implementation is evaluated and compared with a centralised caching system. The access latencies were slightly higher compared to centralised caching systems. However, it is clear that a Distributed Web Cache could provide an alternative to centralised caching at no extra cost. The proposed system has resilience to node failures and uses very little state information, therefore considerably reducing memory usage. The system also has no extra overheads or delays when nodes enter and exit the distributed cache. We see that the system can easily scale up to 10,000 client nodes with acceptable levels of query overhead. The recent developments in the area of client cache cooperation include the use of p2p substrates and coordinating proxy servers. The advantages in both those approaches compared to the one proposed here, is the reduced, object location overhead. The main disadvantage of those systems is that, those systems require the client nodes to be online for a considerable period of time for full functionality [65].

In Chapter 5, results from a study on object behavior in the LRU stack is presented. A mathematical expression is derived for the speedup. An upper bound for the speed up and the maximum multicast timeout for a speedup of greater than unity are also derived. A simulation model is developed and the system is simulated to verify the findings. It was observed that the increasing number of nodes in the Distributed Web Cache, increase the hit rates and decrease access latencies.

In Chapter 6, focus is given to replacement algorithms and their performance in a Distributed Web Caching environment. A novel replacement algorithm called DWCG is developed and its performance is compared with standard replacement algorithms. It was observed that DWCG out performs LRU, LFU and GDSF in terms of LSR. Gain in LSR is from 4.5% to 17.2%, over LFU, 1.2% to 8.4% over LRU and 1% to 3.4% over GDSF depending on the hot-set and inter-site popularity correlation factor.

7.1 Thesis outcome

The work reported in this thesis proposes a novel approach to web caching. The proposed distributed approach to exploit client caches is reliable and reduces hot-spots in the network.

The system can scale well in corporate networks with up to 10,000 client nodes. A rule of thumb for acceptable performance with the proposed system would be that, the communication latency between any two client nodes should be at least an order of magnitude smaller than the latency to access external servers, and they should be connected by a network of bandwidth at least and order of magnitude more than the external bandwidth. This rule is valid for all attempts to exploit

client caches [54].

The main advantage of the system is that it could be implemented with no additional cost and eliminates the need to maintain a costly, highly reliable and powerful caching infrastructure for the same purpose. The system tolerate the transient behavior of client nodes. For example, when a client node leaves the distributed cache, nothing needs to be done, since there is no specific structure to restore or to maintain. The multicast query method provides a robust and comprehensive search mechanism.

The main disadvantage in the proposed system is the considerable query overhead, which is proportional to the number of participating nodes. If we consider a network of 1000 nodes, the overhead per peer is about twice that of the p2p cache by Linga *et al.* [61]. However, it has been made clear that the overhead will stay within acceptable levels for corporate networks with 100 to 10,000 client nodes.

Conditions for speedup and expressions for maximum speedup and maximum multicast delay are derived. Object replacement plays a major role in any caching system.

Due to the independent replacement carried out by each node, the cache capacity is not optimally utilized. A special object replacement algorithm called DWCG is developed specially for Distributed Web Caching environment. This novel algorithm is able to outperform conventional algorithms such as LRU, LFU and GDSF. It requires no additional communications for coordination between caches compared to the tightly coordinated method suggested in [60].

The proposed system can easily be integrated to the operating system or could be executed as a daemon or a service on client nodes and it can co-habit with legacy

infrastructure, requires no changes to the network beyond the corporate network.

7.2 Future Work

- SuperNodes

The proposed system has following disadvantages:

1. expensive searching process and therefore, more network traffic and less scalability and,

2. lack of guarantee on locating a requested file exists in the system.

Since the requested object could be anywhere in the distributed cache, intuitively, more work is needed to locate it compared to the case of structured p2p systems. The scope of the search is restricted to a certain number of hops (or to the corporate network) to limit the overhead. This means that, if the file happened to be on a client outside the range of the search, it will not be located. Another problem is many internet service providers don't route UDP multicast messages for various reasons.

A natural solution would be to employ some Supernodes, such as in the case of Morpheus. In Morpheus, peers are automatically elected to become SuperNodes if they have sufficient bandwidth and processing power (a configuration parameter allows users to opt out of running their peer in this mode). Once a Morpheus peer receives its list of SuperNodes from a central server, little communication with the server is required.

A similar approach could be used to make the proposed system to scale beyond the corporate networks. The SuperNodes, which could connect two corporate net-

works together via TCP connections, could maintain information about the files available within the clients on that network. This information is periodically exchanged between SuperNodes. Therefore, only if the remote network seem to contain the object, the request will be relayed to the remote SuperNode and then to the client in that network. The file transfer may take place directly between client nodes. The method used in Summary Cache[24],[23] using Bloom Filters to obtain compact representations of the cache contents, may be employed. Such a mechanism, could also be used to partition a corporate network, so that multicast range could be further reduced.

Another possible approach is to let clients query only the SuperNodes, which will maintain summary cache like information of all clients connected to them. Client requests are made through the SuperNodes similar to a proxy server. SuperNodes could be located via multicast or using a set of servers with static IPs, that can also act as authenticating servers. This method provides a more accurate location strategy at the expense of higher resource usage at SuperNodes and bandwidth usage for exchanging summaries.

- Issues of information security and privacy

The proposed Distributed Web Caching system requires a remarkable amount of trust from the participating client nodes. A node must trust that other nodes implement the same protocols and will respect the goals of the system.

It is possible to decoy files to have the correct length, but not the original content. Similar issues have traditionally hurt search engines, where any page with a given search term inside it had an equal chance of appearing highly on the search results. The best solution to the search engine problem, as used by Googles PageRank

technology, has been to form a notion of popularity. For Google, pages that are linked from "popular" pages are themselves more popular. An interesting issue is how to add such a notion of popularity into a p2p storage system.

It might be possible to extend the idea to the proposed system.

Bibliography

[1] R. Malpani, J. Lorch, and D. Berger, "Making world wide web caching servers cooperate," in *Proceedings of 4th International WWW Conference*, (Boston), pp. 107–117, December 1995.

[2] D. Wessels, *Web Caching*. USA: O'Reilly & Associates, 2001.

[3] M. Baentsch, L.Baun, G. Molter, S. Rothkugel, and P. Sturn, "World wide web caching: the application-level view of the internet," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 170 – 178, June 1997.

[4] G. Barish and K. Obraczke, "World wide web caching: trends and techniques," *IEEE Communications Magazine*, vol. 38, no. 5, pp. 178 – 184, May 2000.

[5] J. C. Mogul, "Squeezing more bits out of http caches," *IEEE Network*, vol. 14, no. 3, pp. 6 – 14, May-June 2000.

[6] S. G. Dykes, C. L. Jeffery, and S. Das, "Taxonomy and design analysis for distributed web caching," in *Proceedings of the IEEE Hawaii International Conference on System Sciences HICSS'99*, 1999.

[7] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *IEEE Conference on Peer-to-Peer Computing (P2P'01)*, (Linkping, Sweden), 2001.

[8] W. Chou, "Building an infrastructure for a powerful web presence," *IT Professional*, vol. 3, no. 6, pp. 54 – 60, Nov.-Dec. 2001.

[9] B. D. Davison, "A web caching primer," *IEEE Internet Computing*, vol. 5, no. 4, pp. 38 – 45, July-Aug. 2001.

[10] C. Kenyon, "The evolution of web-caching markets," *IEEE Computer*, vol. 34, no. 11, pp. 128 – 130, Nov. 2001.

[11] M. Liu, F.-Y. Wang, D. Zeng, and L. Yang, "An overview of world wide web caching," in *IEEE International Conference on Systems, Man, and Cybernetics, 2001*, (Tucson, AZ USA), pp. 3045 – 3050, Oct. 2001.

[12] J. Wang, "A survey of Web caching schemes for the Internet," *ACM Computer Communication Review*, vol. 25, no. 9, pp. 36–46, 1999.

[13] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. New York: Addison-Wesley, 2002.

[14] T. Berners-Lee, "World wide web consortium (W3C): The original HTTP as defined in 1991," *http://www.w3.org/Protocols/HTTP/AsImplemented.html*, 1991.

[15] T. Berners-Lee, R. Fielding, and H. Frystyk, "Network working group: Request for comments (RFC): 1945," *http://www.faqs.org/rfcs/rfc1945.html*, May 1996.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Network working group: Request for comments (RFC): 2616," *http://www.faqs.org/rfcs/rfc2616.html*, June 1999.

[17] X. Zhang, "Cachability of web objects," *Technical Report - Department of Computer Science, University of Boston.*

[18] D. Li, P. Cao, and M. Dahlin, "WCIP: Web cache invalidation protocol," *IETF Internet draft, work in progress*, 3 2001.

[19] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *USENIX Annual Technical Conference*, pp. 153–164, 1996.

[20] C. Lindemann and O. P. Waldhorst, "Evaluating cooperative web caching protocols for emerging network technologies," in *Proc. Workshop on Caching, Coherence and Consistency*, (Sorrento, Italy), 2001.

[21] D. Wessels and K. Claffy, "Icp and the squid web cache," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, pp. 345 – 357, April 1998.

[22] S. Selvakumar and P. Prabhakar, "Implementation and comparison of distributed caching schemes," in *Proceedings. IEEE International Conference on Networks, 2000. (ICON 2000).*, (Singapore), p. 491, Sept. 2000.

[23] A. Rousskov and D. Wessels, "Cache digests," *Computer Networks and ISDN Systems*, vol. 30, no. 22–23, pp. 2155–2168, 1998.

[24] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[25] The Relais Group, "Relais: Cooperative caches for the world-wide web," *http://www-sor.inria.fr/projects/relais/*, 1998.

[26] J. M. Menaud, V. Issarny, and M. Banatre, "A scalable and efficient cooperative system for web caches," *IEEE, Concurrency*, vol. 8, no. 3, pp. 56 – 62, July-Sept. 2000.

[27] M. Makpangou, G. Pierre, C. Khoury, and N. Dorta, "Replicated directory service for weakly consistent distributed caches," in *19th IEEE International Conference on Distributed Computing Systems*, (Le Chesnay, France), pp. 92 – 100, June 1999.

[28] V. Valloppillil and K. W. Ross, "Cache array routing protocol v1.0.," *Internet Draft*, 1995.

[29] R. Tewari, M. Dahlin, H. Vin, and J. Kay, "Beyond hierarchies: Design considerations for distributed caching on the Internet," *Technical Report*, no. CS98-04, 1998.

[30] P. Rodriguez, C. Spanner, and E. W. Biersack, "Web caching architectures: Hierarchical and distributed caching," in *Proceedings of the 4th International Web Caching Workshop*, 1999.

[31] K. W. Ross, "Hash routing for collections of shared web caches," *IEEE Network*, vol. 11, no. 6, pp. 37 – 44, Dec. 1997.

[32] K.-L. Wu and P. S. Yu, "Load balancing and hot spot relief for hash routing among a collection of proxy caches," in *19th IEEE International Conference on Distributed Computing Systems, 1999.*, (Austin, TX USA), pp. 536 – 543, June 1999.

[33] S. Gadde, M. Rabinovich, and J. S. Chase, "Reduce, reuse, recycle: An approach to building large internet caches," in *Workshop on Hot Topics in Operating Systems*, pp. 93–98, 1997.

[34] T. Asaka, H. Miwa, and Y. Tanaka, "Distributed web caching using hash-based query caching method," in *1999 IEEE International Conference on Control Applications*, (Kohala Coast, HI USA), pp. 1620 – 1625, Aug. 1999.

[35] X. Tang and S. Chanson, "Optimal hash routing for web proxies," pp. 191–198.

[36] L. Brunie, J. M. Pierson, and D. Coquil, "Semantic collaborative web caching," in *Proceedings of the Third International Conference on Web Information Systems Engineering, 2002. WISE 2002.*, pp. 30 – 39, Dec. 2002.

[37] A. Tanaka and K. Tatsukawa, "Interreference interval for purged objects: a new metric for design and analysis of web caching algorithms," in *Proceedings of the 2003 IEEE International Performance, Computing, and Communications Conference*, pp. 549 – 554, April 2003.

[38] S. Inohara, Y. Masuoka, J. Min, and F. Noda, "Self-organizing cooperative www caching," in *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, p. 74, IEEE Computer Society, 1998.

[39] Z. Liang, H. Hassanein, and P. Martin, "Transparent distributed web caching," in *26th Annual IEEE Conference on Local Computer Networks, 2001. (LCN 2001)*, (Tampa, FL USA), pp. 225 – 233, Nov. 2001.

[40] Z. Qing, P. Martin, and H. Hassanein, "Transparent distributed web caching with minimum expected response time," in *Proceedings of the 2003 IEEE International Performance, Computing, and Communications Conference*, pp. 379 – 386, April 2003.

[41] H. Hassanein, Z. Liang, and P. Martin, "Performance comparison of alternative web caching techniques," in *Seventh International Symposium on Computers and Communications, 2002 (ISCC 2002)*, pp. 213 – 218, July 2002.

[42] A. Santoro, B. Ciciani, M. Colajanni, and F. Quaglia, "Two-tier cooperation: a scalable protocol for web cache sharing," in *IEEE International Symposium on Network Computing and Applications, 2001.*, (Cambridge, MA USA), pp. 186 – 193, Oct. 2001.

[43] R. Lancellotti, B. Ciciani, and M. Colajanni, "Distributed cooperation schemes for document lookup in multiple cache servers," in *Second IEEE International Symposium on Network Computing and Applications, 2003.*, pp. 43 – 50, April 2003.

[44] C. Y. Chiang, M. T. Liu, and M. E. Muller, "Caching neighborhood protocol: A foundation for building dynamic web caching hierarchies with proxy servers," in *International Conference on Parallel Processing, 1999.*, (Aizu-Wakamatsu City Japan), pp. 516 – 523, 1999.

[45] C.-Y. Chiang, Y. Li, M. T. Liu, and M. E. Muller, "On request forwarding for dynamic web caching hierarchies," in *20th International Conference on Distributed Computing Systems, 2000.*, (Taipei Taiwan), pp. 262 – 269, April 2000.

[46] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM (1)*, pp. 126–134, 1999.

[47] B. Duska, D. Marwood, and M. J. Feeley, "The measured access characteristics of world wide web client proxy caches," in *1st USENIX Symp. on Internet Technologies and Systems*, 1997.

[48] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing reference locality in the WWW," in *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, (Miami Beach, FL), 1996.

[49] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of World Wide Web Client-based Traces," Tech. Rep. BUCS-TR-1995-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[50] S. D. Gribble, "UC Berkeley Home IP HTTP Traces," Tech. Rep. http://www.acm.org/sigcomm/ITA/, University of California, Berkely, July 1997.

[51] A. Leff, J. L. Wolf, and P. S. Yu, "Replication algorithms in a remote caching architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 4, pp. 1185–1204, 1993.

[52] T. T. Tay and Y. Zhang, "Peer distributed web caching with incremental update scheme." IEE Communications, accepted for publication, Nov 2004.

[53] J. Dilley, "The effect of consistency on cache response time," *IEEE Network*, vol. 14, no. 3, pp. 24 – 28, May-June 2000.

[54] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *21th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, 2002.

[55] tcpdump *http://www.tcpdump.org/*.

[56] IRCache Project *http://www.ircache.net/*, Since 1997.

[57] Squid Web Proxy Cache *http://www.squid-cache.org/*.

[58] T.T.Tay, Y.Feng, and M.N.Wijesundara, "A distributed internet caching system," in *Proceedings of 25th Annual IEEE Conference on Local Computer Networks (LCN'00)*, (Tampa, Florida), p. 642, 2000.

[59] L. Xiao, X. Zhang, and Z. Xu, "On reliable and scalable peer-to-peer web document sharing," in *International Parallel and Distributed Processing Symposium, IPDPS 2002*, (Ft. Lauderdale, FL USA), pp. 15–19, April 2002.

[60] Y. Zhu and Y. Hu, "Exploiting client caches: an approach to building large web caches," in *International Conference on Parallel Processing, 2003*, pp. 536 – 543, Oct. 2003.

[61] P. Linga, I. Gupta, and K. Birman, "A churn-resistant peer-to-peer web caching system," in *ACM Workshop on Survivable and Self-Regenerative Systems, 2003*, Oct. 2003.

[62] X. Wang, W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou, "Buddyweb: A p2p-based collaborative web caching system," in *Revised Papers from the NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, pp. 247–251, Springer-Verlag, 2002.

[63] "Bestpeer: A self-configurable peer-to-peer system," in *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, p. 272, IEEE Computer Society, 2002.

[64] A. Rowstron, P. Druschel, L. Fan, G. Phillips, and S. Shenker, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Middleware'2001*, November 2001.

[65] V. N. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *Peer-to-Peer Systems: First International Workshop, IPTPS 2002*, (Cambridge, MA, USA), pp. 178–190, 2002.

[66] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching syetems," in *Proceedings of IEEE INFOCOM 2001*, (Anchorage, Alaska), April 2001.

[67] S. G. Dykes and K. A. Robbins, "A viability analysis of cooperative proxy caching," in *INFOCOM*, pp. 1205–1214, 2001.

[68] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Symposium on Operating Systems Principles*, pp. 16–31, 1999.

[69] A. Belloum and L. O. Hertzberger, "Document replacement policies dedicated to web caching," in *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS)*, pp. 576–581, 1998.

[70] K. Cheng and Y. Kambayashi, "LRU-SP: a size-adjusted and popularity-aware LRU replacement algorithm for web caching," in *Computer Software and Applications Conference, 2000*, pp. 48 – 53, 2000.

[71] J. Dilley and M. Arlitt, "Improving proxy cache performance: analysis of three replacement policies," *IEEE Internet Computing*, vol. 3, no. 6, pp. 44 – 50, Nov.-Dec. 1999.

[72] N. L. Fonseca and R. M. Oliveira, "Role of download time as a key in web cache management policies," in *Global Telecommunications Conference, 2001*, pp. 2031 – 2035, 2001.

[73] A. P. Foong, Y. H. Hu, and D. M. Heisey, "Logistic regression in an adaptive web cache," *IEEE Internet Computing*, vol. 3, no. 5, pp. 27 – 36, September 1999.

[74] K. Kyungbaek and P. Daeyeon, "Least popularity-per-byte replacement algorithm for a proxy cache," in *Eighth International Conference on Parallel and Distributed Systems*, pp. 780–788, 2001.

[75] C. Lindemann and O. P. Waldhorst, "Evaluating the impact of different document types on the performance of web cache replacement schemes," in *International Conference on dependable Systems and Networks, 2002*, pp. 717 – 726, 2002.

[76] C. D. Murta and V. A. F. Almeida, "Using performance maps to understand the behavior of web caching policies," in *Second IEEE Workshop on Internet Applications (WIAPP '01)*, 2001.

[77] K. Psounis and B. Prabhakar, "Efficient randomized web-cache replacement schemes using samples from past eviction times," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 441 – 454, Aug. 2002.

[78] X. Tang and S. T. Chanson, "Coordinated en-route web caching," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595 – 607, June 2002.

[79] H. Wang, J. Peng, Y. Wu, and H. Feng, "SzLFU(k) web cache replacement algorithm," in *IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, pp. 754 – 758, 2002.

[80] A. P. Foong, Y. H. Hu, and D. M. Heisey, "Essence of an effective web caching algorithm," in *Proceedings of the International Conference on Internet Computing*, pp. 269–276, 2000.

[81] A. Silberschatz and P. Galvin, *Operating Systems Concepts*. Reading, MA: Addison Wesley, 4 ed., 1994.

[82] R. P. Wooster and M. Abrams, "Proxy caching that estimates page load delays," in *Proceedings of International WWW Conference*, April 1997.

[83] 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*, 1996.

[84] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions On Networking*, vol. 8, no. 2, April 2000.

[85] S. Jin and A. Bestavros, "Popularity-aware greedydual-size web proxy caching algorithms," *In Proceedings of ICDCS*, 4, 2000.

[86] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, (Monterey, CA), 1997.

[87] P. Cao and S. Irani, "Greedydual-size: A cost-aware WWW proxy caching algorithm," in *Proceedings of* 2^{nd} *Web Caching Workshop*, (Boulder, Colorado), 1997.

[88] L. Cherkasova, "Improving WWW proxies performance with greedy-dual-size-frequency caching policy," *HP Laboratories Report*, no. HPL-98-69R1, 1998.

[89] J.-C. Bolot and P. Hoschka, "Performance engineering of the World Wide Web: Application to dimensioning and cache design," *Computer Networks and ISDN Systems*, vol. 28, no. 7–11, pp. 1397–1405, 1996.

[90] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy cache algorithms: Design, implementation, and performance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 4, pp. 549–562, 1999.

[91] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," in *Proceedings of the 1993 ACM SIGMOD Conference*, pp. 297–306, 1993.

[92] T.T.Tay and M.N.Wijesundara, *Communications World*, ch. A Replica-Aware extension for replacement algorithms in Distributed Web Caching, pp. 279–286. WSES Press, 2001.

[93] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: the search for invariants," *SIGMETRICS Perform. Eval. Rev.*, vol. 24, no. 1, pp. 126–137, 1996.

[94] M.N.Wijesundara and T.T.Tay, "Distributed web caching," in *Proceedings of 8th IEEE International Conference on Communications Systems (ICCS 2002)*, 2002.

[95] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 94–107, 1999.