In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

Web Object-Based Policies for Managing Proxy Caches

by

Abdolreza Abhari

B.Sc., M.Sc.

A thesis submitted to
The Faculty of Graduate Studies and Research
In partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
July 2003

Copyright 2003, Abdolreza Abhari

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisisitons et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88708-1

Our file Notre référence

ISBN: 0-612-88708-1

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
Acceptance of the thesis,
Web Object-Based Policies for Managing Proxy Caches
Submitted by
Abdolreza Abhari
In partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Director, School of Computer Science

Supervisor

Supervisor

External Examiner

Carleton University, July 2003

Abstract

The continued growth of the World Wide Web necessitates the use of proxy caches to reduce latency, network traffic and Web server loads. These caches utilize a traditional file-based replacement policy such as Least Recently Used (LRU). A file is commonly used as a unit of fetching and replacement. In this research, we propose two new cache management strategies for proxies. Our strategies exploit Web document structure to improve cache performance. The strategies discussed in the thesis, WOLRU and WOLFU, use a Web object that consists of a Web page source file and all of its embedded files, as a unit of fetching and replacement.

Since our proposed strategies use a Web object, we first present our study on Web object characterization. This study is based on collected data concerning the size and structure of the Web objects. Then, through a trace-driven simulation based on a number of access logs, we show that WOLRU and WOLFU perform significantly better than their file-based counterparts: LRU and LFU. We also show that WOLRU and WOLFU are simpler in comparison to a number of existing cache management policies available in the literature, and can be implemented on a real system. Insights are presented into system behaviour that includes the effect of cache size on performance and the cost associated with the policies.

We also propose a new Web object-based policy to manage the storage system of a proxy cache. We propose two techniques to improve the storage system performance.

In the first technique, a Web object is used as a unit for replacement and prefetching between the disk and main memory. The prefetching part improves performance, as most of the files can be provided from the main memory instead of a proxy disk. The second technique stores the Web object members on contiguous disk blocks in order to reduce the disk access time. This in turn reduces the disk response time. We have used trace-driven simulations together with a disk simulator to study the performance improvements that can be obtained with these two techniques. Our results show that a significant performance improvement can be achieved by using these techniques. Most of this performance improvement accrues from the first (i.e., Web object-based prefetching and replacement between disk and main memory). The performance improvement produced by the second (i.e., contiguous allocation of Web object members on disk) is small in comparison.

Acknowledgments

Thanks to my supervisors Prof. Sivarama Dandamudi and Prof. Shikharesh Majumdar for providing years of invaluable support to my research. I would like to thank them for their guidance, encouragement and valuable suggestions throughout the completion of my thesis.

Trace data used in our simulations were collected by the National Laboratory for Applied Network Research under National Science Foundation grants NCR-9616602 and NCR-951745. Thanks to NLANR for making the traces available.

The disk simulator used for evaluating disk performance was DiskSim. DiskSim has been developed by G. R. Ganger at the University of Michigan to support research work on various aspects of storage subsystem architecture. Thanks to Dr. Ganger for making Disksim available for research work.

I am grateful to Nortel Networks, Carleton University and the Natural Sciences and Engineering Research Council (NSERC) of Canada for providing financial support for this research.

Finally I would like to give my heartfelt gratitude to my parents for all of the encouragement and support that they provided throughout my life.

Table of Contents

1- Introduction.....	1
1.1 Web Performance Improvement	2
1.2 Web Object Characterization.....	3
1.3 Using Web Objects to Improve the Performance of Proxy Caching.....	4
1.4 Outline of the Thesis	6
2- Background Information and Previous Work.....	8
2.1 Background Information.....	8
2.1.1 Web Document Structure.....	8
2.1.2 Characteristics of Web Workload.....	11
2.1.3 Impact of Embedded Objects on Web Performance	14
2.1.4 Caching Dynamic Files.....	17
2.1.5 Proxy Caching.....	21
2.1.6 Cache Replacement Policies.....	26
2.1.7 Latency in the Web.....	35
2.2 Previous Work.....	38
2.2.1 Measurements on Web Documents.....	38
2.2.2 Performance Improvement through Prefetching	42
2.2.2.1 Prefetching-Based Policies.....	43
2.2.2.3 Comparison with our Work.....	49
2.2.3 Proxy Cache Storage Management.....	51

3- Characterization of Web Objects.....	55
3.1 Introduction.....	55
3.2 Data Collection and Methodology.....	57
3.3 Data Analysis.....	60
3.3.1 Web Object Sizes.....	60
3.3.2 Number of Embedded Objects	73
3.3.3 Size of Embedded Objects.....	81
3.3.4 Types of Embedded Objects	86
3.3.5 Other Web File Types.....	87
3.4 Implications of Characteristics on Web Performance.....	88
3.4.1 Improving Web/Proxy Server Cache Management.....	88
3.4.2 Reducing Disk Latency in Web/Proxy Servers.....	91
3.4.3 Implication of Results for Dynamic Web Pages.....	92
3.4.4 Efficient Web Page Design.....	92
3.5 Summary.....	94
 4- Using Web Objects to Improve the Performance of Proxy Caching.....	 97
4.1 Introduction.....	97
4.2 Proxy Log Characteristics.....	98
4.2.1 One-time Referencing.....	100
4.2.2 Popularity Characterization.....	102
4.2.2.1 Popularity Analysis for Files.....	103
4.2.2.1 Popularity Analysis for Web objects.....	107

4.2.3 Analysis of the Request Arrivals.....	108
4.3 The Proposed Cache Management Strategy.....	112
4.3.1 Web Object-Based Grouping.....	112
4.3.2 Web Object-based Cache Management Policy.....	115
4.3.3 Implementation Implications.....	118
4.4 Simulation.....	122
4.4.1 Simulation Parameters.....	123
4.4.2 Performance Metrics.....	125
4.4.3 Simulation Warm-up Period.....	126
4.5 Results.....	132
4.5.1 Performance of the Web Object-Based Policy.....	132
4.5.2 Cost of the WOLRU Policy.....	143
4.6 Discussion.....	145
4.6.1 Web Object-Based Prefetching.....	145
4.6.2 Miss Analysis.....	148
4.7 Summary.....	152
5- Web Object-Based Storage Management for Proxy Caches	155
5.1 Introduction.....	155
5.2 Web Object-Based Storage Management Policy	156
5.2.1 Web Object-Based Replacement and Prefetching.....	158
5.2.2 Simulation	159

5.2.3 Experimental Results.....	162
5.2.4 Disk Traffic.....	165
5.3 Web Object-Based File Placement Policy.....	168
5.3.1 Introduction.....	168
5.3.1.1 Overview of the Storage of Files.....	168
5.3.1.2 Using File level Clustering by Proxy Applications.....	170
5.3.1.3 Overview of Disk Drive Performance Evaluation.....	171
5.3.2 Contiguous Allocation of Web Object Members.....	171
5.3.3 Simulation.....	173
5.3.4 Results.....	177
5.4 Discussion.....	185
5.5 Impact on Proxy Cache Hit Latency	189
5.6 Summary.....	192
6-Conclusions.....	194
6.1 Web Object-Based Proxy Cache Management.....	194
6.2 Web Object-Based Storage Management.....	196
6.3 Thesis Contributions.....	197
6.4 Future Work.....	199
References.....	202

List of Tables

3.1: Statistics of Web object sizes for popular data sets	61
3.2: K-S statistics for popular Web object sizes.....	67
3.3: Estimates of α for popular Web object sizes.....	69
3.4: Estimates of α of Web object sizes for log data.....	72
3.5: Statistics of number of embedded objects for popular data sets.....	74
3.6: K-S statistics for number of embedded objects in popular data sets.....	75
3.7: Estimates of α for number of embedded objects in popular data sets.....	76
3.8: Estimates of α of number of embedded objects for log data.....	80
3.9: Statistics of Web file sizes for popular data sets.....	81
3.10: K-S statistics for popular embedded object sizes.....	83
3.11: Estimates of α for size of embedded objects in popular data sets.....	85
3.12: Statistics on popular Audio/Video files.....	87
3.13: Summary of Web objects characteristics	95
4.1: Statistics on used NLANR traces.....	99
4.2: Statistics on one-timers for NLANR traces.....	102
4.3: Estimated slopes for popularity characteristics.....	106
4.4: Characteristics of Boeing access log.....	140
4.5: Hit ratio for the Boeing trace when using Hine's algorithm with a window time of 120 seconds	142

4.6: Data transfer characteristics for LRU and WOLRU with a 64MB cache size.....	144
4.7: Hit ratio for the traces with infinite cache.....	146
4.8: Miss analysis for UC3 (steady-state).....	152
5.1: Basic parameters for simulated disk drives.....	175
5.2: Queuing characteristics observed with LRU policy and PB trace for HP2490 disk.....	181
5.3: Queuing characteristics observed with WOLRU policy (consecutive allocation) and PB trace for HP2490 disk.....	181
5.4: Comparison of file placement policies for PB trace.....	185
5.5: Performance comparison of policies with 256MB of main memory.....	191

List of Figures

2.1: A typical Web page.....	9
2.2: ON/OFF model	12
3.1: CDF of Web object sizes for popular data sets.....	61
3.2: pdf of Web object sizes for popular data sets.....	62
3.3: CDF comparisons for March 1999 popular data set.....	64
3.4: CDF comparisons for November 1999 popular data set.....	64
3.5: CDF comparisons for February 2000 popular data set.....	65
3.6: LLCD of Web object sizes for popular data sets.....	68
3.7: CDF of Web object sizes for log data sets.....	70
3.8: pdf of Web object sizes for log data.....	70
3.9: LLCD of Web object sizes for log data.....	72
3.10: CDF of Number of embedded objects for popular sets.....	74
3.11: pdf of the number of embedded objects for popular data sets.....	75
3.12: LLCD of the number of embedded objects for popular data sets.....	76
3.13: CDF for number of embedded objects for log data.....	77
3.14: pdf for number of embedded objects for log data.....	79
3.15: LLCD for number of embedded objects for log data.....	80
3.16: CDF of file sizes for popular data sets.....	82
3.17: pdf of file sizes for popular data sets.....	82
3.18: LLCD of file sizes for popular data sets.....	84

4.1: Popularity patterns for (a) UC1, (b) UC2, (c) UC3 and (d) PB traces.....	105
4.2: Web object popularity pattern for UC3.....	107
4.3: Request per interval time series for UC3 (interval: 1 minute).....	110
4.4: Request per interval time series for UC3 (interval: 30 minutes).....	111
4.5: Periodic hit ratio of 1 GB cache size for UC3 (interval: 30 minutes).....	111
4.6: Web object-based LRU algorithm used by the simulation model.....	117
4.7: Typical scenario on a real system employing a Web object-based cache strategy.....	119
4.8: Transient identification for UC1.....	129
4.9: Transient identification for UC2.....	129
4.10: Transient identification for UC3.....	131
4.11: The variation of cumulative hit ratio with time for UC3.....	131
4.12: Performance comparison of cache management policies for UC1.....	133
4.13: Performance comparison of cache management policies for UC2.....	134
4.14: Performance comparison of cache management policies for UC3.....	136
4.15: Performance comparison of cache management policies for UC3 during steady-state.....	137
4.16: Comparison of file-based replacement versus Web object-based replacement for UC3 trace.....	148
4.17: Miss analysis of 256MB cache size for UC3.....	151
5.1: Typical scenario on a real system employing Web object-based storage management policy.....	157

5.2: Memory hit ratios of the WOLRU and LRU policies.....	163
5.3: Disk traffic for the WOLRU and LRU policies.....	166
5.4: Disk traffic increase due to WOLRU policy (in comparison to LRU).....	167
5.5: Impact of consecutive and random file assignment on disk response time of the WOLRU policy and the UC3 trace.....	179
5.6: Impact of consecutive and random file assignment on disk response time of the WOLRU policy and the PB trace.....	179
5.7: Impact of consecutive and random file assignment on disk response times of the WOLRU policy for HP2249 disk type.....	180
5.8: Impact of consecutive and random file assignment on the disk seek time of the WOLRU policy for the PB trace.....	182
5.9: Number of disk accesses for the PB trace.....	183
5.10: Disk hit ratios of the NP-WOLRU and LRU policies.....	187
5.11: Impact of consecutive and random file assignment on disk response times of the NP-WOLRU policy for HP 2490 disk and PB trace.....	188

Chapter 1

Introduction

Performance problems in the Web are mostly related to a heavy load on Web servers, high network traffic, inefficiency of the HTTP protocol and slow equipment used by clients for network access. If the protocol performance and speed of client access to the Web improve (for example, by using the new version of HTTP and cable modems), the performance bottlenecks shift towards the network and Web servers. One common way to alleviate these bottlenecks is to use caching at proxies or servers. Caching at proxy servers can be used to reduce network traffic and load on servers. Therefore, improving cache performance leads to improved Web performance. This thesis presents a new cache management strategy for improving the performance of proxy servers.

The main question that this work tries to answer is: can we exploit a Web document structure to improve Web performance? This chapter is organized to briefly explain how this thesis answers this question. Section 1.1 presents an overview of the general methods for improving Web performance. Section 1.2 describes the motivation for studying Web object characteristics. In particular, Section 1.2 explains what Web document characteristics are, and why exploiting Web document structure can lead to cache performance improvement. A brief introduction to our proposed strategy is given

in Section 1.3. Section 1.3 also explains how our strategy improves proxy cache performance. Section 1.4 discusses the outline of the thesis.

1.1 Web Performance Improvement

Some of the commonly used techniques for improving Web performance are summarized here.

- *Web Server Improvement:* Performance of CPU, disk systems and TCP connection handling in the Web servers were analyzed by researchers, and solutions such as using multiprocessor systems, employing caching, reducing disk I/O latency and using connection scheduling on the server were suggested (see [16] and [50]). Performance improvement of Web servers (reducing their response times for example) is an active area of research (see [24] and [47]). The use of parallel Web server hardware is one solution for reducing server response times. For example, IBM has used the SP2 parallel system in the design of Web Servers for the 1998 Olympic Winter games [24]. A cluster of workstations that can share cache information and cache data between the nodes was used by the Alexandria Digital Library (ADL) for implementing a Web server [47]. Other researchers have tried to reduce network bottlenecks through load balancing on multi-node Web servers [47]. Processing capacity rather than network bandwidth is the performance bottleneck for sites that use dynamic pages [24].

- *Proper Design of Web Pages:* Another way of improving Web performance is through the establishment of guidelines for the proper design of Web pages that lead to a more efficient system. Section 3.4.4 presents a number of suggestions, based on our observations of popular Web pages that can be used by content providers for efficient design of Web pages.
- *Employing Caching:* Caching-oriented approaches have often adapted traditional file caching strategies to improve Web server performance. The location of the cache depends on the Web application. The cache can be at the Web server, at the client, or within the network at other locations (e.g., proxy servers). A proxy cache can use the disk or main memory. Thus, improving the cache replacement policy and reducing disk latency in proxy servers are effective ways to improve Web performance. In Chapters 4 and 5 we explain how our results can be used to devise a better strategy for proxy server cache management.

1.2 Web Object Characterization

Fundamental to the goal of improving Web performance is an understanding of the World Wide Web workloads and the characteristics of Web-based applications. There has been a significant research effort on the characterization of Web workloads (see [11] and [15]). For example, characterization of the Web traffic is important in order to devise synthetic workload generators for use in the investigation of effective resource management algorithms. Most characterization of Web documents are based on Web files without considering their inherent structure. In addition, most of the existing research is

based on logs captured on scientific Web sites. Therefore, this characterization is limited by the files that happened to be accessed during the log capture time at the specific sites. Characteristics of Web documents in terms of their components and size have received little attention.

In response to a Web page request, a set of files that includes the Web page source file and its embedded files (i.e., embedded objects) must be transferred. A *Web object* is defined as a Web page source file and its related embedded files. Our goal in conducting this study is to collect data on the structure and size of Web objects. Characterization of Web objects is particularly useful in improving proxy server performance through techniques such as clustering of files that can be used to improve cache performance and to reduce disk latency. We report the results of an empirical study conducted on several popular Web pages. We have chosen the popular Web pages for this investigation because they are more likely to be efficiently designed. In addition, popular Web documents also account for a significant portion of network traffic. We also study a trace of a busy proxy access log in order to characterize Web objects for regular Web environments.

1.3 Using Web Objects to Improve the Performance of Proxy Caching

In Web proxy caching, one or more computers act as a cache of documents for a set of WWW clients. These clients are configured to send Hyper Text Transfer Protocol (HTTP) requests to the proxy. If a requested document is present in the proxy, it is considered as a hit, and the proxy serves that document from its cache. Otherwise, it is

considered as a miss, and the proxy forwards the request to the server containing the source copy of the requested document. Proxy caches reduce the load on servers and reduce the traffic in the network.

Although caching is a popular performance improvement technique, the cache hit ratio that can be achieved by most caching algorithms on Web workloads is in the range of 47% to 52% [62]. This means that for every two requests only one can be found, on average, in the Web cache. One way to improve the hit ratio is by prefetching the files into the cache. To implement prefetching, we need to anticipate future requests for files. There is a key difference between caching and prefetching: the purpose of caching is to provide fast access for the second and subsequent accesses by exploiting temporal locality, whereas the goal of prefetching is to provide fast access even for a first time access to the files. Prefetching files that are accessed in the future, increases the hit ratio at the risk of using a higher bandwidth. Increasing the cache hit ratio reduces the latency that is perceived by the user because of the reduction in the number of retrieved Web files from the original server (this is discussed in more details in Section 2.1.7 of Chapter 2 and in [60]).

There are many factors that affect caching performance. In this research we examine the impact of cache size and cache management policy on performance. This research proposes Web object-based strategies for cache management. The Web object-based cache management strategies proposed in this work have two components: a replacement policy and a prefetching policy. The cache replacement policy determines

which file(s) should be replaced when a cache miss occurs and there is insufficient free space in the cache. We introduce two new Web object-based replacement policies, which are extensions of the Least Recently Used (LRU) and the Least Frequently Used (LFU) replacement policies. The policies proposed in this thesis are described in Chapter 4. These Web object-based policies improve the proxy cache hit ratio by prefetching Web objects from server to proxy.

In this thesis we refer to the delay incurred in serving a hit document (i.e., a document present in the proxy) as cache hit latency. Most proxy caches use main memory together with the disk as the cache. The cache hit latency can be reduced by Web object-based replacement and prefetching between the disk and main memory. A Web object-based policy can also be used to improve storage management of the proxy by prefetching Web objects from disk to the main memory of the proxy. Thus, the proxy provides most of the documents from its main memory instead of its disk when they exist in the proxy. In our prefetching policy, we prefetch all the files associated with the entire Web object in addition to the file that has been requested. A Web object-based file placement policy that stores the members of a Web object in consecutive blocks of the proxy disk is also presented in this thesis. Chapter 5 shows how cache hit latency can be computed and also explains how reducing disk latency reduces proxy cache hit latency.

1.4 Outline of the Thesis

Web servers provide both static and dynamic pages. Our study on structural characteristics of the Web documents focuses on both static and dynamic pages. Caching

is currently used mostly for static Web pages. Our caching policy concentrates on static files. Since each dynamic page also has static components, caching these static parts is another method for gaining performance improvement (see [36] for example). Therefore, our discussion on efficient cache and disk management strategies is applicable in the context of dynamic pages as well.

The rest of the thesis is organized as follows. Chapter 2 presents background information and also discusses previous work in this area. Chapter 3 describes our study on Web object characteristics. The Web object-based cache management strategies are presented in Chapter 4. Chapter 5 proposes expanding the Web object-based strategy to manage the disk and the main memory of proxy caches. Chapter 6 concludes the thesis with a summary of important contributions made and pointers to future research.

Chapter 2 Background Information and Previous Work

2.1 Background Information

This section presents background information related to this thesis. An overview of the issues such as Web document structure, Web workload characteristics, proxy caching, user latency and cache replacement policies is presented in this section.

2.1.1 Web Document Structure

Web servers typically provide both static and dynamic pages. Static documents are stored and retrieved from the file system. Dynamic documents are created on the fly by server programs that are invoked when requests are made. Dynamic pages are essential for situations in which contents of Web pages are constantly changing. For example, IBM developed dynamic pages for the 1998 Olympic Winter games to show the results of the current events on demand [24]. The CPU can be a bottleneck in servers that provide dynamic pages because each page is generated as a result of running a query on that server. Chapter 3 presents our study on Web document structures. Here we provide a short overview on dynamic and static Web pages.

Web Document Structure: Barford and Crovella [15] defined a *Web object* as a collection of all the files that must be transferred to display the complete page. A *Web page source file* is an HTML file containing pointers to zero or more *embedded objects*.

Embedded objects are typically files containing images that users receive as part of the Web page (see Figure 2.1). When a user clicks on a Web object, the corresponding Web page source file and its related embedded objects are sent to the client browser.

Figure 2.1: A typical Web page

Figure 2.1 shows an example of a static Web page with its embedded files. If any of these files are generated dynamically, Figure 2.1 can be considered a dynamic Web page as well. An example of a dynamic Web page is a page that shows the results of a query to a search engine like Yahoo. If the pages that are searched by Yahoo have been changed since the last access to them, the resultant page shows different contents compared to the previous time that Yahoo searched for the same query. Another example of a dynamic Web page is a page that changes very fast. For example, a page showing the stock prices may change each time someone accesses the Web page. On each access to

that page, the server should access a stock price database in order to find the current stock price. By inserting this value in the page and adding other static parts of the page such as images, the server composes the page and sends it to the client.

In the workload study of the 1998 World Cup Web site [10], Arlitt and Jin report that the volume of requests to dynamic files is only 2% of the 1.3 billion total requests for that site. According to another study on the same workload [52], out of 56.8 million requests generated on the busiest day, 12% were to dynamically generated files.

In [16], Barford and Crovella report that apart from the base HTML files (i.e., Web page source files) that reference some embedded objects, there are also some single files in the Web that are neither base HTML files nor embedded objects of any base HTML files. These files, which are large, are compressed files or large Audio and Video files. Small Audio and Video files such as MIDI files can be embedded in a Web page.

In characterizing the Web objects, we studied both dynamic and static embedded objects that belong to a Web object. We also characterized the single Audio and Video files that belong to popular Web sites. The details of our study are presented in Chapter 3.

2.1.2 Characteristics of Web Workload

Some researchers have identified Web workload characteristics by analyzing the Web server access logs. For example, Arlitt and Williamson [11] identified ten workload characteristics by analyzing six server logs. Also, based on characteristics and distribution of the Web requests, researchers have developed tools for simulating Web requests. For example, Barford and Crovella developed a tool for generating a Web workload [15]. The Web workload characteristics and distribution models that are presented by these works are summarized below:

User (Client): Barford and Crovella [15] proposed an ON/OFF model for simulating user behaviour in the Web (see Figure 2.2). There are two kinds of OFF times. Inactive OFF time corresponds to the time between transfers of Web documents. This is also called user "think time". Active OFF time is the time between two successive transfers of components of a single Web object. It is the processing time spent by the browser in parsing a Web file and preparing a connection [15].

File size: Distribution of file sizes in the Web is identified by a distribution that has a lognormal distribution on the body, with a heavy-tailed distribution on the tails [11], [12]. Since Web file size distribution is heavy-tailed, the server's file system must deal with highly variable file sizes. A detailed explanation of heavy-tailed distribution is provided in Section 2.2.1.

Figure 2.2: ON/OFF model (from [15])

Request sizes: The number of bytes transferred during the service of a request is called the request size. Request size is also referred to as transfer size. The distribution of request sizes is observed to be heavy-tailed [31].

Popularity: Arlitt and Williamson [11] showed that 10% of the distinct documents were responsible for 80-95% of all requests received by the server in the access logs analyzed by them. That is why caching is such a powerful tool for improving Web performance. Popular files will typically tend to remain in caches. Popularity distribution for files on a Web server has been shown to follow Zipf's law, which states that if files are ordered from most popular (i.e., rank 1) to least popular, then the number of references to a file is inversely proportional to its rank (r). If we use p to show the number of references to a file, the popularity of a file can be computed by $p = k \times r^{-\alpha}$ for some positive constant k

and exponent α close to 1 [91]. For proxy workloads Arlitt *et al.* [10] report the estimates for α that typically range from 0.5 to 1.

Embedded reference: Active OFF times are typically short, while inactive OFF times are longer. It is therefore important to consider the number of embedded references contained in a Web object in order to model the OFF and ON times in Web traffic. For active OFF times Weibull, and for the number of embedded references in a Web object, Pareto distribution (i.e., not heavy-tailed Pareto, see Section 3.3.2 for details), were suggested as the fitted models [15].

Temporal locality: The principle of temporal locality implies that files referenced recently have a high probability of being referenced in the near future. This is an important characteristic of the system because cache effectiveness can be significantly increased when temporal locality is present in the file references. One way to measure temporal locality is to use the stack distance. This stack is a pushdown stack and is also referred to as a LRU stack. If we assume that the files are stored in a stack, and each request moves the requested file to the top of stack, the depth (position) in the stack for the requested file is the request's stack distance. For example, a file that is placed at the top of the stack has a stack distance of 1. For investigating temporal locality in a sequence of file references, we record the stack distance for a file when it is referenced again. In other words, on each reference to a file that has been referenced at least once, its current stack distance is recorded. Since the stack is a push down stack, bringing a referenced file to the top of the stack pushes down the other files in the stack. This

method is referred to as standard LRU stack-depth analysis [10]. If most of the stack distances for a sequence of the file references are small (i.e., small average or median of stack distances), then the references are mostly to the locations near the top of the stack. Such references demonstrate temporal locality in the file references. Typical distribution of stack distances for a sequence of requests to the Web servers is observed to be lognormal [15]. It has been shown that significant temporal locality is often present in Web request sequences on Web servers.

Geographic location: Another characteristic identified by Arlitt and Williamson [11] is the geographic location of clients that requested documents. By studying Web server logs and identifying the IP addresses of the sites that requested documents, they divided these sites into remote and local sites (relative to the requested Web servers). They showed that the remote sites account for more than 70% of accesses to the server. Also, they reported that Web servers are accessed by thousands of domains, with 10% of domains accounting for more than 75% of accesses.

2.1.3 Impact of Embedded Objects on Web Performance

Effect on the Web traffic: The number and size of embedded objects that belong to a Web object influence the traffic pattern - the more embedded objects in a Web object the more files are transferred. This leads to more active off times in the traffic pattern. The time interval between consecutive transfers of embedded objects consists of processing time spent by the browser in parsing of Web files and time spent for preparing

to start a new TCP connection if needed. Note that different features employed in HTTP protocol can change the traffic pattern when retrieving a Web page. For example, using HTTP/1.1 that provides a single TCP connection, or using HTTP/1.0 when employing multiple TCP connections in parallel, can affect the duration of active off times.

Nielsen *et al.* [85] investigated the impact of embedded objects on performance of the HTTP/1.1 protocol by changing the size of the objects. They showed that changing images from GIF or JPEG format to Portable Network Graphics (PNG) improves network performance when HTTP/1.1 is deployed. PNG has several advantages over GIF/JPEG. It produces smaller files in comparison to GIF/JPEG and renders as quickly on the screen as GIF while producing higher quality pictures.

Effect on download time of the Web page: Size and number of embedded objects in a page also affect the page download time. Microsoft redesigned its home page in 1999 to improve the download time. The size and the number of embedded objects in the new version of their home page are decreased [75].

In addition to the impact of size and type of the embedded objects on the download time of a Web page, the method of serving these embedded objects by the server also affects the total response time. Nadimpalli and Majumdar [79] investigated the use of parallel access to the embedded objects of a Web page when they are distributed on the backend servers. They reported that, by distributing the embedded objects belonging to a Web page on four servers instead of one, and accessing them in

parallel, the time to receive that page by a client can be reduced significantly. They also investigated different allocation methods for the embedded objects on the backend servers. They concluded that a balanced allocation or a replication of the embedded objects on backend servers can also improve system performance.

Effect on Caching: Web caching is based on caching single files. As already noted, the files that are used in the Web include files other than the embedded objects belonging to Web objects. An example of these single files is an Audio or Video file. Caching of Audio and Video files are not as straightforward because of their large size. For example, caching a two hour Video file would exhaust the capacity of most caches. But since Audio and Video files have become more popular, there are recent research directions that suggest methods for caching them. Wu *et al.* [111] suggest a method based on grouping the blocks of a media stream that are received by the proxy into variable size segments. The proxy gives preferences to caching the segments by assigning different values to each segment. This value is based on the reference frequency of each segment and its distance from the beginning of the media. For more details see [111].

Although the number of suggested methods for caching Audio and Video files is increasing, existing Web caching techniques are still based on caching text and image files. Static text and image files are the best candidates for caching as they are small in size and are more popular compared to other types of files available in the Web. Arlitt and Jin reported that out of a total of 1.5 billion client requests to the 1998 World Cup Web site, 98.01% were for either HTML (9.85%) or image (88.16%) files. The remaining

requests (1.99%) were for Audio/Video, compressed, applets, dynamic and other file types [10]. Wills and Mikhailov [104] have measured the rate of change for embedded objects belonging to popular Web pages. They concluded that embedded images do not change as fast as HTML files in a Web page. Therefore, embedded images are good candidates for caching. They also mention the need to associate images with their container (i.e., Web page source file) when they are in the cache. They explain that since the containers change faster than images, their embedded images may not belong to them after they have been changed. They conclude that if there were a relation between containers and embedded images, we would be able to remove the images that do not belong to any page from the cache [104].

Our proposed Web object-based caching policy has this potential by managing Web objects instead of the single files in the cache (see Section 4.3.3 for more details). Since the focus of this thesis is on caching static embedded objects, a detailed explanation of their effect on caching is presented in Section 3.4.1. Besides static embedded objects, some of the embedded objects are dynamically generated files. We explain the effect of dynamic files on caching in more details in the following section.

2.1.4 Caching Dynamic Files

Most of the current Web caching methods prevent caching of dynamic files. Cache policies determine the type of the file before storing them in the cache. Parsing an URL's name, reading file extensions and reading file headers are mechanisms for

recognizing dynamic files. For example, URLs ending with ‘cgi-bin’ or ‘.cgi’ or ‘.pl’ are dynamic files. Also, URLs that contain ‘?’ in their names are query results. The content-type of the files that can be read from the file header also provides information about non-cacheable files such as cookies, application files, etc. Cookies are access authorizations that contain a signed statement from the server. By reading the header information, the files that are defined as non-cacheable by servers can also be identified.

Apart from dynamic files and non-cacheable files, files that change frequently are not good candidates for caching. Current Web cache policies employ a consistency mechanism to avoid serving stale files to the client. One of the common methods for providing cache consistency in the Web is using a time-to-live (TTL) or an expiration time attached to each cached file [76]. When a client wants to read a file from the cache, the cache reads the TTL and ‘last modification time’ from the header of that file. If the expiration time of the file has passed, the cache sends a ‘Get-if-modified-since’ request to the server. The server either replies with a “304” HTTP code that means “not modified”, or a “200” HTTP code that means, “the file is not valid”. In the case of a “200” HTTP response, the server sends the new file to the cache. The problem with this method is that if a group of cached files expires at the same time, each file is revalidated and that increases the load on the server [112]. Arlitt and Jin reported that for the workload they studied, 80.52% of the server responses were successful transmission with 200 HTTP server response, and 19% were 304 HTTP server response [10]. The increasing existence of frequently updated files on the Web makes it necessary to consider better cache coherency policies. In the following paragraphs, we survey a variety of techniques that

provide better cache coherency, and then discuss research related to caching dynamic files.

Some researchers (see [14], [48], [78], [105] for example) suggest using delta-encoding for providing cache consistency. Delta-encoding is a mechanism for transmitting an encoded/compressed form of the changes of documents. Using delta-encoding reduces bandwidth requirements and improves end-to-end performance.

Several researchers (see [65], [66], [112], [113], [114] and [116]) have suggested server-driven consistency. Server-driven consistency is particularly attractive for dynamic Web workloads. In this method, the server informs the clients of updates to the documents. Therefore, the cache can provide the cached files without needing to contact the server. Such cache consistency protocols have been evaluated for both unicast and multicast environments (see [65], [113], [114] and [116] for more details).

Yin *et al.* [112] have implemented a server-driven consistency mechanism for a sporting and event Web site workload that was hosted by IBM during the 1998 Olympic Games. They cached dynamically generated files that accounted for 12% of the 56.8 million total requests on the busiest day [52]. They reported that by using a server-driven consistency protocol, dynamic files were cached as efficiently as static files. They examined issues such as scalability, performance and consistency requirements when using a server-driven consistency approach for caching of both dynamic and static files.

Netscape [81] has added a capability to its browser (Netscape Navigator 1.1) for dealing with dynamic files. This capability, which is referred to as “server push” and “client pull”, enables the browser to display dynamic Web pages whenever they are changing. There is no report concerning the implication of this approach on caching dynamic files.

As well as research that explores methods that deal with cache coherency and updating dynamic contents, there is research that suggests caching only the static parts of a dynamic Web page. In this case, the cache has to be able to separate dynamic and static parts of a Web page when the cache receives that page. Douglass *et al.* [36] extend the HTML language to support the caching of dynamic resources that consist of both static parts and dynamic parts. They refer to the extension as “HTML pre-processing (HPP)” language. They examined HPP for caching the Web pages that are the results of queries made by the client. They report that by using HPP, static parts of pages can be cached, resulting in a reduction of server load. Their proposed method requires minimal changes to Web servers and HTTP protocols.

Cao *et al.* [23] proposed another approach for caching dynamic documents. In this approach, the server supplies a cache applet to be attached to the document. Whenever the proxy receives that document, it invokes its cache applet. Upon receiving a document as a result of a client request, the proxy runs the applet attached to the requested document. Thus, the proxy knows which parts of the document are static and which are dynamic. If the static parts are available in the cache, the proxy provides them from its

cache; otherwise it fetches them from the server and then caches them. For dynamic parts, the proxy provides the requested items from the server. Since the proxy runs the applet, it is able to compose the page and send the results to the client. Running a cache applet in the proxy reduces the processing load on the server site that is required for composing dynamic documents. Cao *et al.* mention that since this approach increases CPU overhead at the proxy site, client latency is increased by a factor of 1.5 to 4.

In the simulation of our proposed Web object-based policy, we cache files with a variable rate of changes. But we consider the effect of the changes of the files on cache performance by checking the changes in the size of cached files during the simulation. We prevent caching of non-cacheable embedded objects belonging to the Web objects by parsing the URL name of the files. Details on this Web object-based policy and cache simulation are presented in Chapter 4.

2.1.5 Proxy Caching

The growth of the Word Wide Web motivated techniques to improve its performance. For Web document caching, several approaches have been proposed. The most popular ones are proxy caching [6], push caching [45] and Harvest caches [18]. In this work we focus on proxy caching, in which one or more computers act as a cache of documents for a set of WWW clients. Proxy caching improves Web performance by trying to reduce user latency while obtaining Web documents. This is because the proxy cache is typically closer to the client than the content provider. Providing documents for

the client by proxy instead of server, lowers network traffic as well. Also, most of the cache hits are directed to the proxy, which lowers service demand on the Web server.

Proxy caching can be implemented as single-level or multi-level proxy caching. In multi-level proxy caching, a cluster of proxies are used in a hierarchy of caches. The reason for using multi-level proxy caching is to improve the performance of caching. In the hierarchical approach, the proxies which are located close to the clients are referred to as lower-level proxy caches, and the proxies that are close to the server are known as higher-level proxy caches. Lower-level proxy caches receive the requests from the clients. If they have the requested document in their cache, they send it to the clients; otherwise they forward the request to the higher-level proxies in the hierarchy.

Roadknight *et al.* [91] report that the effect of lower-level proxy caches is to reduce the temporal locality of requests for higher-level caches. This means that the hit ratio for the proxies that are located at higher levels of the cache hierarchy is less than that for the proxies at the lower levels of the hierarchy. The National Laboratory for Applied Network Research (NLANR) manages IRCACHE that operates in the U.S. IRCACHE is a global cache that consists of parent and sibling caches. Connecting to IRCACHE can improve the performance of caching [53]. For communication and cooperation of proxies in a hierarchy of caches, several protocols have been proposed. We explain some of these approaches here.

The common schema for organizing a hierarchy of proxies is referred to as the static proxy caching hierarchy. This schema consists of parent and child proxies. The goal of a hierarchical cache is to reduce the load on the server or hot spots, besides reducing the access latency [90]. For communication among the proxies in a hierarchy, some protocols have been proposed. Internet Cache Protocol (ICP) [51] is a common protocol that provides communication among the proxies in a hierarchy of caches. ICP is a lightweight message format implemented on top of UDP to provide quick query/reply message exchange. It is used to provide hints about the existence of URLs in the related caches in the hierarchy. Proxies have parent, child and sibling relationships among themselves. Harvest caches and Squid [98] use ICP messages to contact other proxy caches in a hierarchy. For example, if the Squid cache cannot find a requested document, it sends an ICP query to its siblings, and the siblings respond with ICP replies that indicate a HIT or MISS. Squid can use this reply information to choose a cache that it can use to resolve the MISS. Based on this result, Squid retrieves the document from the siblings, parent or original site [97]. More details about ICP are presented in [25], [101], [99] and [102].

Besides the static hierarchy of proxy caching, another schema for using proxy caches in a hierarchy is the dynamic approach. Chiang *et al.* [26] proposed a Caching Neighborhood Protocol (CNP) that describes an infrastructure by which proxy servers are able to build dynamic Web caching hierarchies. In CNP, if the requested file is not available in a proxy cache close to the client, this proxy uses its request forwarding table to select a server to forward this request. The request forwarding table contains

information about all the proxies that are representative of the original server for that requested document. The request forwarding table also contains an estimation of the request latency response for each of the proxies. In other words, a caching hierarchy of proxies is created based on receiving a request from the client. For this reason this approach is considered to be a dynamic schema. However, in the static schema, a proxy can send the requests only to its pre-configured siblings, or its parent, or the original server of the requested document. The question of storing particular servers and representatives in the forwarding table of each proxy, and also the size of the tables, are issues that should be considered in the CNP approach [27]. Chiang *et al.* compare the latency of different Web caching schemes in [28]. They suggest that static schemes can have shorter request latencies if the total number of levels in their hierarchy is one or two. They conclude that, with regard to request latency, CNP outperforms static and other schemes based on the hierarchy of proxy caches.

Another approach for using proxy caches is that of using a group of proxy caches together with some network devices. These devices, also referred to as switches, are responsible for directing the requests to each of the proxy caches. An example of such switches is the ArrowPoint content smart Web switches [13]. These switches are usually located between clients and a group of proxies. When the ArrowPoint switches receive HTTP requests for a document, these requests are processed by them. ArrowPoint switches read the information such as the URL and file extension that is included in the header of the request, and also read the IP address and TCP port of that request. By analyzing these requests, the switches learn about the content of the document that is

requested, and direct the request to the appropriate cache. For example, the switches are able to direct the requests to appropriate caches based on the domain name or URL name of the document. As a result, each cache in a group contains a non-overlapping set of documents. The switches also bypass the caches for requests to the non-cacheable objects and send these requests to the original server. Since the requests for non-cacheable objects are sent to the original server together with the IP address of the requested client, the server is able to send the result directly to the clients. Also, since caches only receive requests for cacheable objects, they do not have to decide whether or not an object is cacheable.

ArrowPoint switches can also work with Reverse Proxy Caching (RPC). RPC acts more as an extension to Web servers rather than being a regular cache. When a client sends a request for a document, the IP address of RPC is returned instead of the IP address of original server. Therefore, RPC serves the client's request transparently and acts as a proxy for the server. RPC also can be used to store specific contents. For example, all requests for a given name can be directed to the RPC. The ability to recognize different requests makes these switches work well in a distributed configuration of proxies. The distributed configuration of proxies together with switches can improve client response time.

In this thesis, we concentrate on single proxy caching and do not deal with issues such as directing requests on a multi-level proxy cache. Also, we only concentrate on the

caching role of the proxies. Proxies are also used for other reasons in the network, such as filtering or blocking client access to specific Web sites, or recording client accesses.

2.1.6 Cache Replacement Policies

Traditional cache replacement policies are generally classified into the following categories: frequency-based, recency-based, size-based or a combination of these three policies. Here, we summarize the representative policies in each group to provide an overview of the replacement policies that have been used in the Web servers and proxy caches. We then explain more recent cache replacement policies that focus on proxy caching.

LRU (Least Recently Used): This policy removes a file from the cache that has not been used for the longest period of time. LRU is commonly used for Web environment and proxy caching. The reason for the popularity of LRU in the Web is because of the good performance of LRU when requests exhibit temporal locality. Temporal locality suggests that files that have been referenced in the recent past are likely to be referenced again in the near future. The advantage of LRU is that it uses temporal locality by simply using a list that is ordered on recency of references.

Some of the Web caching approaches such as Harvest [18] do not implement a pure LRU. Harvest cache initiates the removal algorithm when the total size of cache documents exceeds a threshold value of the cache. To reduce the time that is required for

executing the removal algorithm, Harvest cache sorts only a group of N cached documents instead of all of them. Then, from this sorted document, the least recently used M files are evicted from the cache [108]. Caching applications may also use extra data structures for implementing the LRU list. For example, Squid uses a doubly linked list and a hash table index to implement the LRU list [97].

LFU (Least Frequently Used): This policy maintains a reference count for each file in the cache. On each cache hit, the reference count of the requested file is incremented. On each miss (if there is insufficient space in the cache), the file with the lowest reference count is replaced. In case of a tie, the least recently used file with the lowest reference count is replaced. The advantage of LFU is its simple implementation; the problem with LFU is that some documents can build up extremely high reference counts so that they are rarely replaced, even if these documents are never requested again. This situation is referred to as cache pollution.

LFU-Aging: This policy is similar to the LFU policy, but solves the cache pollution problem by using aging. LFU-Aging reduces the reference counts of the documents occasionally. For example, when the average reference count of the cached files exceeds a limit, all of the reference counts are divided by two. Also, LFU-Aging limits the maximum value of the document reference counts to avoid cache pollution (see [92] and [107] for more details).

LFU*: One of the problems of Web caching is that there are lots of files in the Web that are referred to as one-timers [11]. One-timers are the files that are accessed only once, and thus are poor candidates for caching. The number of references to one-timers can be as high as one third of all of the references to the Web files [11]. Arlitt *et al.* [9] report that the number of one-timers is more than 60% of all of the distinct files in the workload that was analyzed by them. LFU* [12] is a modified version of the LFU, which solves the problem of caching the one-timers. A reference count is maintained for each file in cache, but in the case of a cache miss, the newly requested document is not always added to the cache. Only files with a current reference count of one are removed from the cache, and a new document can arrive in the cache if there is space; otherwise, no replacement is undertaken and the new file does not come into the cache. The problem with this method is that if all file reference counts are greater than one, no replacement occurs. The solution is to use aging (i.e., the reference counts over time can be reduced back to one to have more candidates for removal) [12].

FBR (Frequency Based Replacement): This policy uses a combination of reference frequency and age for replacement. FBR divides the cache into three sections (new, middle and old section). When a file is added to the cache, it is placed in the new section. Since the most recently used file is added in the cache, the Most Recently Used policy is employed at that stage. As this file ages (on the LRU basis) it goes to the middle section. If it continues to age (remains in the cache and is not re-referenced) it goes to the old section. Only files in the old section are candidates for removing from the cache by the replacement strategy. On a cache hit, the referenced file is always moved to the start of

the new section (Most Recently Used). If the file is found in the middle or old section, its reference count is incremented; if the file is already in the new section, its count is not incremented. In the case of a cache miss, which requires a cache replacement decision, the least frequently used file in the old section is replaced. In case of a tie, the least recently used file with the lowest reference count is chosen for replacement ([92], [107]).

The problem with LRU and LFU based algorithms is that they do not consider file size and the download latency. One way to improve the cache hit ratio is to cache many small documents (i.e., put an upper limit on the size of documents that may be added to the cache). To address this size issue, policies that consider size and bandwidth were introduced.

SIZE: This is one of the size-based cache replacement policies. This policy removes large files from the cache in order to make room for small files. If there is a tie with regard to size, size-based policies can use other policies such as LRU or LFU when deciding on the eviction of the files. Size-based policies are used when the goal of a cache replacement policy is to increase the hit ratio. The problem in using these policies is that they may keep small documents for a long time in the cache even if they are never accessed again [84].

Arlitt and Williamson [12] examined different cache replacement policies for Web server caching. LFU*-Aging is observed to be more efficient than the other policies

for improving cache hit ratio and for reducing the volume of data transferred from the hard disk of the server. A detailed description is provided in [12].

GDS (Greedy Dual-Size): Cao and Irani [22] studied several proxy cache replacement policies. They introduced the Greedy Dual-Size replacement algorithm. GDS is a generalization of the Greedy Dual algorithm that was introduced by Young [115] to deal with caching of variable cost pages that have the same sizes. Since the files cached in the Web have variable sizes, Cao and Irani modified the Greedy Dual algorithm to consider variable sizes as well. In GDS, a value, H , is associated with each document in the cache. When a document is brought into the cache the function, $H = \text{cost}/\text{size}$, is calculated. When a replacement has to be undertaken, the document with the smallest H (min- H) is removed from the cache and every document has its H value reduced by min- H . The *cost* is the cost of bringing a file into the cache; therefore, H is the cost of bringing a file into the cache while taking into account the size of that file. The definition of *cost* depends on the goal of the replacement algorithm. For example, if we want to increase the hit ratio, then *cost* is set to 1. In this case, GDS is referred to as GD1. By considering the algorithm of GDS presented in [22], it is clear that if we use the same size files (i.e., by setting the size to 1), GD1 is identical to LRU. If the goal of the replacement policy is to minimize the average latency, then the *cost* is set to the downloading latency. For calculating the average latency, Cao and Irani used the method introduced in [108]. This method is the same as that used by the Hybrid policy, presented in this section. If the goal is to reduce network cost, the *cost* is set to the network cost. The network cost can be considered as the number of packets that are transferred when retrieving the file from the server into the

cache. Cao and Irani examined all of these three extensions to GDS algorithm. They showed that GDS outperforms existing algorithms in many aspects, including hit ratio improvement, latency reduction and network cost reduction. They have also suggested integration of hint-based prefetching with cache replacement policies.

GDSP (GDS-Popularity): Jin and Bestavros generalized the GDS algorithm in [56]. They modified GDS by incorporating popularity information of cached files in GDS. To identify the popularity of the files they define “long term access”, for each file. For each file p that is in the cache, they calculate the value of H as follows: $H = f(p) \times cost(p)/size(p)$, where $f(p)$ represents the long term access frequency for the file p . The resulting algorithm is called GDS-Popularity or GDSP.

They used the formula $f_{i+1}(p) = f_i(p) \times 2^{-t/T} + 1$ to estimate the long term access frequency. In this formula, t is the elapsed time since the last reference to p , and T is a constant that controls the rate of decay. For example, T was set to 2 days in the simulation that was performed by them. To be able to use this formula, a popularity profile (i.e., access frequency and last access time) should be maintained for each file in the cache. HTTP provides the elapsed time (i.e., $cost$), size and the time of last access to a file. Thus, by reading this information, GDSP can compute the value of H , for each cached file in an online manner. Based on the value of H , a decision is made about the file that should be evicted. One of the problems with this method is keeping the popularity profiles in the memory for all of the cached files. To solve the problem of memory space, they keep the popularity information (i.e., access frequency and last

access time) of only a small fraction of cached files. They have evaluated GDSP with $cost(p)$ set to the cost of retrieval of file p for all of the situations in which the goal of the cache is to improve the hit ratio, reduce latency and network traffic. They report the superiority of GDSP for all of these situations compared to LRU, LFU and GDS policies.

GD* (GDS-Popularity): In [57], Jin and Bestavros show that to capture temporal locality, we need to find both popularity and short term correlation of references. For example, two sequences of references, “XAXBXCXDXEXF...” and “GGHHIIJJKKLL...”, both show a temporal locality property. In the first stream, locality results from the popularity of X, whereas in the second stream it is due to correlation of the first and second references to G, H, I, J, K, and L. In more recent work [55], they incorporate long term popularity and short term correlation of references into the GDS algorithm. They referred to the new algorithm as GreedyDual* (GDS-Popularity). GD* uses the following formula to calculate the H value: $H = (f(p) \times c(p)/s(p))^{1/B}$. In this formula $f(p)$ is the approximation for reference counts (i.e., popularity) and constant B is used to estimate the short term reference correlation. To be able to compute B in the implementation of GD* they keep the number of references at different intervals for equally popular objects. They note that since B is stable over time, GD* can also estimate constant B in an online manner as well. They report the superiority of GD* when compared to other Web cache replacement policies that have been proposed in the literature [55].

Hybrid: The Hybrid policy was developed by Wooster and Abrams in [108]. The goal of their proposed algorithm is to reduce the delay that is perceived by a client when

retrieving a file. In this algorithm, the proxy estimates the delay to retrieve a Web file from the server. Based on this estimation, the proxy assigns the cost value to each cached file. The Hybrid policy evicts the files with higher retrieval cost when a cache replacement is necessary. To estimate the page retrieving latency, the latency for stabilizing the connection to the server_{*i*} (*clat_i*) and the connection bandwidth between proxy and server_{*i*} (*cbwserver_i*) are measured. The proxy calculates *cbw* and *clat* for each server when it receives a document from that server. Their computation is briefly explained next.

Although the download time of the document is known to the proxy (through the elapsed time reported by the HTTP request), there is no information about connection latency. The problem is that the cache is running at the application level, and does not have access to the connection latency (*clat*) that is accessible in the lower layers that deal with TCP and packets. Thus, an assumption is made in their algorithm to estimate the connection latency. According to this assumption, if the size of the retrieving document is less than 2 KB, the download time for that document is used for estimating *clat*, and if the size is more than 2 KB, the time is used for calculating *cbw*. More details about this calculation and the related algorithm are presented in [109]. Finally, the Hybrid policy uses the following formula to compute the latency cost of each document (*j*) when the proxy receives it from the server (*i*): $nref_j^{C_2} \times (clat_i + C_1 / cbw_i) / size_j$, in which *C₁* and *C₂* are constants of 8,196 and 0.9, respectively [84]. *nref_j* is the number of references made to the document *j* since it has been brought into the cache and *size_j* is the size of the

document j . They demonstrated that Hybrid works better in reducing the latency compared to LRU and LFU.

Jin and Bestavros [56] report that one of the problems in using the Hybrid formula is that it requires extensive tuning and parameter estimation. The other problem is that Hybrid needs to keep the information of connection time and bandwidth for each server, and this requires too much space in memory. They also note that in calculating file latency, they found inaccuracies when using the Hybrid formula (presented in [108]). Instead of the Hybrid formula, they suggest a simple formula for computing the latency of file (p): $c(p) = t_{conn} + t_{byte} \times s(p)$, in which t_{conn} is the time to establish the connection, t_{byte} is the average time to transfer each byte, and $s(p)$ is the size of file p . To calculate these two parameter values, they measured the average delay for objects of different sizes and then estimated these two parameters by using *least-square fit* for different logs that were used in their simulation.

Arlitt and Cherkasova [9] studied workload characteristics of Web proxy caches. They introduced two new replacement policies that were built upon the strengths of the existing policies. They observed that file size-based policies achieve a higher hit ratio than other policies. Frequency of file access-based policies, however, was discovered to be more effective in improving the byte hit ratio of a proxy cache. Byte hit ratio is the number of bytes served by the proxy cache during cache hits expressed as a percent of the total number of bytes for all requests. Based on their observations, they proposed their new policies to achieve high cache performance.

2.1.7 Latency in the Web

In the research that was undertaken by Krishnamurthy and Wills [60], some of the factors that influence the latency perceived by the users (referred to as *user latency*) were studied. They identified these factors as: effect of using different versions of HTTP protocol (i.e., 1.0 and 1.1), effect of caching and cache validation, network delay, server load and variable sizes of retrieved Web pages. They examined these factors in order to reduce user latency. They concluded that by using HTTP/1.1 instead of HTTP/1.0, using more deterministic management for TCP connections, and employing methods for validating the cached objects, user latency can be reduced. They noted that caching reduces the user latency because it reduces the number of retrieved Web files from the original server for HTTP version 1.0 and 1.1. They also reported other factors that they did not study, but are important for user latency as follows:

- *DNS lookup time*: The time for mapping a server to an IP address.
- *Redirection*: Some sites use the redirection that is provided by HTTP to direct the user request to another URL.
- *Dynamic contents*: The time for generating the dynamic contents.
- *Packet level performance*: The time spent for fragmentation and reassembling of data into the packets and packet level performance issues.

Niclausse *et al.* [84] used *downloading latency* to estimate network latency that is a part of the user latency. They proposed a replacement policy for the proxy caches to reduce downloading latency that leads to the reduction in user latency. Downloading latency consists of two major latencies: The first is the latency of downloading a

document from a server to the proxy, and the second is the latency of sending that document to the user by the same proxy. The minor part of the downloading latency is the communication time spent by user, proxy and server for retrieving that document. They defined the *latency ratio* as the latency of downloading a document from a server to the proxy, divided by the total downloading latency. They suggested the following formula for computing the latency ratio:

$$W^t = \sum_{i=1}^m \text{lati} (1 - \delta_i) / \sum_{i=1}^m \text{lati}$$

where

$\delta_i = 1$ if the requested document is available in the proxy cache and $\delta_i = 0$ if it is not available.

$\sum_{i=1}^m \text{lati} (1 - \delta_i)$ is the sum of downloading time for missing documents from server to proxy.

$\sum_{i=1}^m \text{lati}$ is the sum of downloading time for all documents from server to proxy.

They suggested that latency ratio can be used as a performance metric for comparing proxy cache replacement policies. They proposed a new proxy cache replacement policy that can produce a lower latency ratio than common replacement policies such as LRU, LFU and SIZE [84]. From this formula we can see that if the number of hit documents in a proxy increases, the number of downloaded documents from a server to the proxy reduces. Therefore, the latency of downloading documents from a server to the proxy (i.e., $\sum_{i=1}^m \text{lati} (1 - \delta_i)$) reduces, which causes a reduction in W^t

as well. In Chapter 4, we show that by using our Web object-based caching policy, we increase the hit ratio of the proxy.

Rousskov and Soloviev [93] studied the performance of the Squid caching proxy with HTTP/1.0. They defined the *proxy response time* as the total time it takes to serve a client request by a proxy. They identified two types of the proxy response time: *hit response time* and *miss response time*. According to their study, hit response time is the sum of the following:

- *Client connect time*: The time to receive a client request after the proxy accepts the connection to that client (accept() system call).
- *Swap in time*: The time to load a file from the disk into memory.
- *Proxy reply time*: The time to send the requested file to a client after the proxy receives it from its cache.

They also defined miss response time as the sum of the client connect time and the following:

- *Proxy connect time*: The time to send a HTTP request to a server by the proxy.
- *Server reply time*: The time that it takes to receive a reply from the original server or another proxy after the proxy has sent a request.
- *Proxy reply time*: The time to send a requested file to a client after the proxy receives it from the server or another proxy.

Finally they concluded that disk latency is 30% of the total hit response time. Thus, reducing the disk latency reduces hit response time and, as a result, reduces the user latency. In Chapter 5, we explain how our Web object-based caching policy is useful in reducing the disk latency.

2.2 Previous Work

Chapters 3, 4 and 5 of this thesis discuss the use of Web document structure to improve cache performance. In particular, Chapter 3 discusses Web document characterization and Chapters 4 and 5 present our cache management policy. We classify previous work into three sections related to the issues that are presented in Chapters 3, 4 and 5: Section 2.2.1 describes the previous work on Web document characterization. Section 2.2.2 focuses on the previous work on prefetching. This section also presents a comparison of these works with our prefetching method. Section 2.2.3 describes the previous work that has been undertaken for improving the performance of the Web/proxy server storage systems. This section also explains how our work is different from these works reported in the literature.

2.2.1 Measurements of Web Documents

We classify measurement techniques into two types: source-based and trace-based. In the source-based method, each Web page source file is examined to obtain the data. This method provides data that are more accurate for our purposes. The other

method involves looking at the access logs and deriving the data based on the observed Web traffic. Most of the existing research has adopted the trace-based measurement approach. A representative set of works is discussed next.

Trace-based Measurements: Most of the work has been based on server access logs. Server access logs capture the results of requests made to the server. Arlitt and Williamson [11] analyzed the server access logs of six sites: three from universities, two from scientific research organizations, and one commercial site. They studied the files that were successfully transferred and found that the distribution of file sizes is heavy-tailed. This implies that the server's file system must deal with highly variable file sizes. A heavy-tailed distribution is one whose upper tail declines following a power law. A distribution is heavy-tailed if $P[X > x] \sim x^{-\alpha}$, as $x \rightarrow \infty$, $0 < \alpha < 2$. If $\alpha < 2$ then the distribution has infinite variance, whereas $\alpha < 1$ indicates that the distribution has an infinite mean [31]. Simulation with a heavy-tailed distribution has slow convergence to steady state, and high variability at steady state. For example, the mean of random variables that follow a heavy-tailed distribution can be computed from a large number of samples that include many small observations mixed with a few large observations [32].

Arlitt and Williamson also show that the tail of the Web file size distribution can be modeled with the Pareto distribution [11]. The Pareto distribution, which belongs to the class of heavy-tailed distributions, is characterized by the probability mass function:

$$p(x) = \alpha k^\alpha x^{-\alpha-1}, \alpha, k > 0, x \geq k.$$

Crovella and Bestavros [31] analyzed client-based trace logs obtained by modifying the Mosaic Web browser source code. They studied ON and OFF times (see Section 2.1.2) present in the traffic pattern produced from the log. They found that both ON times and inactive OFF times have heavy-tailed distributions. However, the distribution of ON times is more heavy-tailed compared to the distribution of inactive OFF times. They noted that document transfers correspond to the ON times in the traffic patterns. Since the distribution of file sizes is heavy-tailed, ON times distribution is heavy-tailed as well. They also observed that, despite the presence of some large files such as multimedia files, Web files are mostly small. This leads to a heavy-tailed distribution for the Web file sizes.

They noted that one of the reasons for seeing self-similarity in Web traffic is the heavy-tailed distribution of document sizes. In self-similar traffic, the observable burstiness in the Web traffic is similar across many time scales, such as milliseconds to seconds to minutes or more. This contradicts with the behaviour of Poisson-like traffic, in which the aggregation of the traffic is smoother. The autocorrelation function for a self-similar process decays hyperbolically, which is slower than the traditional traffic models, in which autocorrelation function decays exponentially. The self-similar process shows long range dependence (LRD) in traffic. This means that positive correlation in the burst behaviour exists over many time scales. Long range dependent traffic affects network issues such as packet loss and delay behaviour [31].

Barford and Crovella [15] used a threshold of one second for identifying the embedded objects that belong to a given page. With this assumption, for the same user, two file transfers that are separated by one second or less are considered to correspond to two different embedded objects of a Web object. However, users may interrupt Web page transfers, resulting in a fewer embedded objects in the trace. Furthermore, clients may have already cached some of the embedded objects of a Web page. This again causes transfer of fewer embedded objects. Thus, this method may not provide accurate information on the number of objects in a Web page. Source-based measurements avoid such problems.

Mah [67] analyzed the packet traces taken from a subnet carrying HTTP traffic instead of analyzing server or client access logs. By using this method information of TCP and IP headers of each packet was also available from the HTTP traffic traces. He used a threshold of one second for distinguishing among the TCP connections corresponding to the embedded objects in a Web object. He observed that 80% of the Web pages in their traces required less than four file transfers. He reported a mean of approximately 3 embedded objects per Web object for the traces studied by him.

Source-based Measurement: Wills and Mikhailov [103] studied popular sites to gather statistics on the rate and nature of changes for all the embedded objects and their implications on caching. They found that despite the fact that HTML resources change frequently there is a significant amount of reuse of images. They suggest that cache replacement policies need to associate an image with its container resource. If an image is

no longer used by any container resource then it should be removed from the cache. They suggest that policies for caching resources from the same site should take into account the inherent structure of the page and to discard unneeded image files when the page structure changes. They characterize the number of embedded images per page but do not report their sizes or data on other types of embedded objects. The authors have also suggested the use of a Web object structure in caching [103].

Web page sources were used in the results reported by the Open text index group [19]. The goal of their research was to design an efficient search engine for finding specific keywords in Web pages. In their study, they examined 20,000 HTML Web pages and reported on the number of embedded objects. This is discussed in Section 3.2 of Chapter 3.

2.2.2 Performance Improvement through Prefetching

Since the Web object-based cache management strategies proposed in this thesis use a prefetching policy together with a replacement policy, this section describes the previous work that used prefetching methods in the Web. A representative set of prefetching-based policies is briefly discussed in Section 2.2.2.1. Section 2.2.2.2 explains how our prefetching technique is different from the techniques that were used by these researchers.

2.2.2.1 Prefetching-Based Policies

Caching is an important performance improvement technique used in the Web, and the cache hit ratio that can be achieved by most of caching algorithms is in the range of 30% to 50% [43], [89] and [105]. Kroeger *et al.* [62] report that for the proxy cache, the maximum achievable hit ratio is in the range of 47% to 52%, which is in line with the value reported by Gribble *et al.* in [44]. Caceres *et al.* [21] report that for the proxy cache trace examined by them, the hit ratio can be up to 54.5% if the proxy ignores the cookies.

The average hit ratio of 50%, means that for every two documents accessed only one can be found, on average, in the Web cache. One way to improve the hit ratio is by preloading or prefetching the documents in the local or global caches. Prefetching documents can be based on anticipating the future requests for documents. One difference between caching and prefetching is that the purpose of caching is to provide fast access for the second time the document is accessed, whereas the purpose of prefetching is to provide fast access for the first time that document is accessed.

There are different approaches to prefetching. Some of the important methods found in the literature are described here.

One simple approach to prefetching is geographical push-caching approach [45]. In this method, when the load of a Web server exceeds a limit, the server replicates (pushes) its most popular documents to the other cooperating servers. Therefore, clients

will be able to make future requests from these servers. The problem with this method is that clients need to know from which server to request the documents. This may involve a request to the original server, which adds to the client's latency.

The Top-10 prefetching [72] is based on cooperation between the client and server. The server is responsible for preparing the Top-10 list (i.e., list of its more popular documents) by analyzing the server access log. The frequency of developing this Top-10 list depends on how frequently the content of the server changes. At the client, a prefetching agent is responsible for analyzing the client access log and then creating the prefetching profile for that client. A client profile is a list of servers from which prefetching should be activated. The prefetching starts from client sites that may connect to the different levels of proxies. The title Top-10 does not mean that only 10 documents will be prefetched, but that these documents are the most popular ones.

Bestavros presents a model for speculative service in [17]. In this model, a Web server provides a number of documents or pointers to documents, in addition to the document that was requested by the client. The notion of speculative service is analogous to prefetching in that it is used to improve cache performance in a distributed/parallel shared memory systems, with the exception that the server (not clients) controls when and what to prefetch. In this approach the server maintains the statistical information about each document that it serves. The statistical information denotes the probability of requesting a document (say D_j) after a document (say D_i) has been requested within a limited window of time (say T_w). Two matrices are generated to capture this probability:

The first matrix is $p[i,j]$, which shows the probability of requesting D_j after D_i has been requested within the window time of T_w . The second matrix is $p^*[i,j]$ (the closure of p) that shows the probability that there will be a sequence of dependent requests starting with document D_i and ending with document D_j . Two requests are assumed to be dependent if the same client has generated them within the T_w .

Based on the probabilities of requesting the files in p and p^* matrices, Bestavros defined two types of dependencies for Web resources [17]: embedded dependency and traversal dependency. Two documents D_i and D_j are said to have an embedded dependency with one another if they are always requested together (i.e., a probability of 1 in p^* matrix). If D_j and D_i are sometimes requested together (i.e., a probability of less than 1 in p^* matrix), these documents are said to exhibit a traversal dependency with each other. For example, Bestavros used the value of 5 seconds for T_w and the documents D_j s whose values in the $p^*[i,j]$ was equal to 1, to generate the groups of documents that have embedded dependency with D_i . These dependencies can be used to predict the probability that the two documents are accessed together.

The author reported that an aggressive speculation schema that results in a 50% increase in bandwidth use, yields a 35% reduction in client miss ratios, a 45% reduction in server load and a 40% reduction in service time. He also noted the possibility of using server-initiated prefetching of the files that have embedded dependency with each other. He observed that such a prefetching does not result in any increase in traffic because embedded documents are certainly needed by the client. But he also noted that sending

documents with embedded dependency together with a requested file leads to a small (less than 5%) increase in hit ratio. Note that the definition of embedded dependency here is different from ours. According to our definition the components of a Web object are said to have an embedded dependency with one another. Another difference is that Bestavros reported the client cache hit ratio improvement as a result of prefetching. Our Web object-based policies use Web objects in both prefetching and replacement to improve the cache performance at a proxy server.

Mogul [77] proposed a prefetching model, in which the server makes predictions and transmits these predictions as a hint to the client. The predictions include the expected probability of reference, size and other useful information related to the files that are suggested to be prefetched. The hint is a part of the response to a client's request. The client then uses this hint to decide what to prefetch. Padmanabhan and Mogul described their algorithm in [87] for generating the server hints. In this approach, an interdependency graph that is based on the relation of documents with each other is generated in a server. If a document D_i is requested from the Web server by a client, then probably document D_j will also be requested within a small time interval T_w by the same client. Thus, we have a graph in which the nodes are documents and the edge between document D_i and D_j represents the probability of accessing document D_j after document D_i has been accessed. The researchers reported a 40% reduction in mean latency accompanied by a 40% increase in the total number of bytes transferred when this prefetching technique is used.

Cohen *et al.* [30] suggested techniques for grouping resources that are likely to be accessed together into volumes. A volume is included in the server response to the requesting proxy. Volumes are used to generate hints tailored to individual applications, such as prefetching, cache replacement and cache validation. They introduced two algorithms for volume construction. These algorithms predict the maximum number of resources that can be prefetched as well as the requested resource while reducing false prediction and limiting the size of hints. They named these algorithms Pairwise and Greedy heuristics algorithms [30].

Hine and Wills [46] investigated the use of knowledge of resource dependencies in improving cache replacement policies. They analyzed the access log for identifying the group of related resources. They used both embedded dependency and traversal dependency for generation of the group of related resources. They used these groups to modify the LRU replacement algorithm used at a proxy cache. They reported a 15%-20% improvement in hit ratio compared to LRU. Since we have implemented their algorithm to compare with our work, more details of their algorithm are presented in Section 4.4.

In the research of Duchamp [39], the hyper-link URLs (i.e., HREFs) that exist in a requested page are prefetched together, with that page, into the client cache. The prefetching starts after negotiation between client and server. In this approach, the client sends a usage report together with a request for a page to the server. The usage report consists of history information that shows recent client accesses to the existing hyper-link URLs of the page if it has been accessed by that client before. The usage report also

includes useful information for prefetching, such as the size of each of the hyper-link URLs and all of their embedded images, plus elapsed time between references to the page, and each hyper-link URL on the last accesses to the page by that client. The server collects usage reports from all of the clients who access a page that is provided by that server. Thus, the server has the collective information referred to as the usage profile for each page. A usage profile shows how often hyper-link URLs have been referenced for all of the clients who accessed that page as well as the elapsed times between reference to the page and hyper-link URLs. The server sends the summary of the page's user profile, together with the page, to the requested client who may want to do prefetching.

Based on the information in the usage profile, a client can decide whether or not to prefetch. By using the information in the usage profile, a client can also decide about the method for prefetching. One of the methods prefetches all or some of the hyper-link URLs, without considering their likelihood of being accessed (see [29], [80] and [88] for example). In another prefetching method, the client uses the information in the usage profile to make a graph (similar to [17] and [87]) that shows the access probability for each hyper-link URL of the page. Duchamp notes that by employing the usage profile, a client can learn about popular hyperlinks in a page. He also reports that if a client prefetches the popular hyper-link URLs, the latency perceived by the client can be reduced by 52.3% while the network traffic increases by 24%.

Duchamp explains that one of the drawbacks of his method is that preparing client and server profiles and exchanging them can increase overhead on the network. Also, he

notes that a page can have a large number of hyper-link URLs and if the client wants to prefetch all of them, the number of prefetched files that are useless may increase. The increase in the prefetching can also lead to an increase in network traffic. Another concern with his approach is that his method requires adding new directives to HTTP to be able to do negotiation and information exchange between client and server.

2.2.2.2 Comparison with our Work

The main differences between our work and the work reported in the literature are as follows:

- First, the researchers in [17], [30], [39] and [87] concentrate on prefetching only. They focus on client cache hit ratio and do not report the result of their prefetching approach for the proxy caches. We focus on prefetching the documents at the proxy, because the documents that are prefetched on behalf of one client by the proxy may also be used by another client.
- Second, the difference between our prefetching method and the prefetching methods in [17], [46], [77] and [87] is that we do not use a probabilistic model to find resources to be prefetched. We prefetch embedded objects belonging to a Web object that are more likely to be required by a client after accessing the Web page source file. Our clustering method is based on a Web object that consists of a Web page source file and the embedded objects that belong to that page. The proxy maintains

the information on each Web object and prefetches its related members from the server, in addition to the requested member of that Web object. Moreover, the Web object-based strategies can be implemented online and do not need *a priori* analysis of proxy logs as required in the window-based strategy reported in [46].

- Third, the difference between our work and [39] is that we do not prefetch hyper-link URLs. We only prefetch the embedded objects that belong to a Web page. The advantage is that embedded objects of a page are often stored on the same server with the Web page source file whereas hyper-link URLs of a page can be stored on different servers. Duchamp reports that 28.9% of the hyper-link URLs of the pages that were studied by him were stored on different servers. Therefore, prefetching of embedded objects is faster than prefetching of hyper-link URLs. Also, the prefetched embedded objects are required by a client to be able to see the Web page, which is not true for the hyper-link URLs of that page. For example, a user may read only one page of different Web sites each time without selecting any hyper-link URLs of that page. An example is a user who has the habit of reading only the first page of different Web sites such as CNN and BBC.

Since Hine and Wills' policy [46] uses prefetching and groups of related resources to improve the performance of proxy caches, we have implemented their algorithm and compared its performance with our Web object-based policies. Their policy needs *a priori* analysis of proxy logs and is hard to implement on a real system. A simpler method for identifying the groups to be fetched/replaced is used in our policy (see

Section 3.2 of Chapter 3). Moreover, our Web object-based policies are observed to result in higher hit ratio compared to the policy proposed by Hine and Wills (see Chapter 4).

2.2.3 Proxy Cache Storage Management

To improve the performance of storage systems, several methods have been proposed. Developing a file system that is specifically designed for the workload of a Web/Proxy server is one of these methods. For example, Hummingbird [41], developed at Bell Labs, is a lightweight file system for proxy caches. General distributed file systems such as AFS [49] and DFS [58] are also used to improve storage performance of the Web/proxy servers. Some commercial vendors proposed operating systems with special file systems running on dedicated hardware for proxy caches. For example, CacheFlow [83] provides proxy caches that use an operating system dedicated for caching, called CacheOS, running on specific hardware.

Markatos *et al.* [73] suggest using a set of techniques for proxy storage management. They noted that writing the URLs in a single file, instead of writing them in separate files on the disk, improves the write operation performance because it reduces the head movement necessary for writing the URLs on the free space of the disk. By this method the overhead for keeping the meta-data information of a single URL will be reduced as well. For read operation they suggest that by performing LAZY-READS (i.e., batch read operations) the disk head movements can also be reduced. In this method when a read operation is issued it is stored in an intermediate buffer and after a certain

duration of time all the pending read operations in that buffer (sorted by the location of the data in a file that are to be read) are forwarded to the file system. They suggest that storing the related URLs contiguously on the disk can preserve the spatial locality of the URLs. For doing this, the proxy should maintain the related URLs in some buffers before writing them on the disk. In these buffers the URLs that have the same origin server name are stored and grouped together in the time of writing. By reading the related URLs in a single disk I/O instead of multiple disk I/Os, the performance of read operations can be improved. They report that a proxy application developed by them that incorporated these techniques outperformed Squid by more than a factor of 7 under heavy load.

Wang *et al.* [100] propose a new file system called User-space Custom-made File System (UCFS) to be used for the proxy caches. UCFS eliminates all disk I/Os required for search and update of meta-data by using efficient in-memory meta-data tables. UCFS used a cluster-structured file system to improve read and write operations in proxy caches. In each cluster a group of files that show locality in the reference patterns are stored. They considered the files from the same Web site as the files that are accessed together. These files include the Web page source file, its embedded objects and all the files that can be access by its hyperlinks. In each read or write operation the entire cluster is read or written. The size of each cluster can be from 32 KB to 256 KB. UCFS uses both main memory and disk of proxy as the cache. Main memory contains most recently used files and the data structures required for searching the URLs in cache as well as tables to manage the clusters on disk. They report UCFS achieves 3-19 times better I/O performance compared to different versions of Squid that are proposed for proxy caches.

Some of the differences between these proposals and our work are summarized below:

- Our techniques do not require a dedicated operating system such as CacheFlow.
- Our techniques are general and do not rely on a special file system for proxy caches such as Hummingbird, UCFS and the system proposed in [73].
- We introduce a new storage management policy that can be used in proxy caches such as Squid or Apache. Our proposed caching policy uses the Web object concept to manage the storage that consists of main memory and the disk of a proxy cache. The characteristics of the Web objects are introduced in Chapter 3 and [4].

In Chapter 5 we explain how, by using Web object-based policy, we can improve proxy storage management. These methods have also been presented in [5].

In a recent study, Gabber *et al.* [41] report that Hummingbird uses locality hints generated by a proxy to pack files into large and fixed-sized clusters. A cluster is the unit of disk access in Hummingbird. Files in a cluster are stored on consecutive blocks for reading and writing in one disk access. Although it is mentioned in [41] that an HTML page and its inline images are usually in the same locality set, it seems that Hummingbird analyzes a Least Recently Used (LRU) list to identify the files that belong to each locality set. Hummingbird stores each locality set into a cluster. From [41] it is not clear how the

LRU list is analyzed to form clusters, but it is clear that their grouping method is different from ours.

The differences between our policy and Hummingbird are the prefetching employed in our policy and also the clustering method for finding the group of related files. In our Web object-based policy, the proxy parses each Web page upon receiving it from the server and finds the files that belong to the Web page. According to our policy, each Web object has unique set of embedded objects. The proxy keeps the information of Web object members when they are in the main memory and when it wants to write them on the disk. Therefore, when our Web object-based policy reads the first member of a Web object from the disk it can prefetch the rest of the members of that Web object into the main memory. Thus, the subsequent requests for other members of that Web object are served from the proxy main memory. Using the Web object-based policy can result in substantial reduction in the cache hit latency even when the Web object members are not stored contiguously (see Chapter 5 for more details). In our proposed policy, storing the Web object members in consecutive blocks makes the prefetching even faster. Hummingbird requires a special file system that uses its locality set-based clustering. Our technique produces substantial reduction in hit latency by using conventional file systems.

Chapter 3

Characterization of Web Objects

3.1 Introduction

In this chapter, we present a study that characterizes a set of popular Web documents. This set is based on a weekly announcement of popular Web sites. The files that belong to the front pages of popular Web sites were examined during three different periods of a year. We considered only the front pages of popular Web sites instead of whole sites because this study focuses on popular Web documents and not on the characteristics of entire sites. A front page is usually the most popular page in a Web site. When accessing a site, the front page is downloaded most of the time, but downloading other pages of a site often depends on the choice of the user. To compare our results on popular Web documents with documents in a regular Web environment, a proxy access log belonging to the National Laboratory for Applied Network Research (NLNR) [86] was also analyzed. In this study, the Web documents requested in the access logs are referred to as regular Web documents.

Note that in this chapter popular Web documents refer to the sites that are reported weekly as being the most popular pages among the Web users (more details are in Section 3.2). However, in most published research, popular files are identified by analyzing Web traces. In these traces, a file could be requested by a proxy, the browser or

by a regular user. Proxies request the files that are not available in their caches. Browsers request the files automatically when they parse a Web page. Unfortunately, there is no information in the traces to indicate which file is requested by a user and which one by a browser or proxy. Therefore, the files that are identified as popular files in the traces may not necessarily be the popular files among the users. For that reason, we characterized the weekly popular sites in this chapter. However, we also considered a larger number of popular files by analyzing NLANR traces in Chapter 4. Section 4.2.2 presents a model and detailed discussion for them as well.

In most of the previous work on Web file measurements a trace-based measurement method was used because of a focus on Web traffic characterization. We used a source-based measurement approach to examine the number of embedded objects in a Web object. This approach involves examining each Web page source file, including all of the embedded files in the page irrespective of whether it was present in the access log or not. In analyzing the access log we used a source-based measurement to recognize embedded objects and their corresponding web files. This produces potentially more accurate information about the number of embedded objects in comparison to a trace-based approach.

In trace-based measurements, time thresholds are used to recognize embedded objects. The results therefore, depend upon the selected threshold values. In addition, this method only looks at the number of objects transferred as opposed to the number of objects in the page. A difference between these two numbers can occur because of aborts

by the clients (who may not wait for the complete page to be downloaded and decide to follow a link on the page) for example, or because of the existence of different levels of caching. Information on the association between an HTML main page and the embedded files is important when we want to consider file clustering to improve cache performance. Clustering of related files can potentially reduce disk access latencies as well as communication times.

The remainder of this chapter is organized as follows. Section 3.2 describes the data on popular Web documents collected in our study. An analysis of the results is presented in Section 3.3. Section 3.4 discusses the implications of data characteristics on Web performance improvement. We summarize this chapter in Section 3.5.

3.2 Data Collection and Methodology

Since popular Web documents are expected to have a strong impact on performance, our research is concerned with the characterization of popular Web documents. We studied a number of Web sites. The site <http://www.100hot.com> contains various statistics about the top 200,000 Web sites. This site is responsible for reporting the most popular sites for each week [1]. For this study 3 data sets were used. The first data set was gathered in March 1999 and includes 55 front pages consisting of 1,051 files. These were the front pages of the top ranked (from 1 to 55) sites out of 100 popular sites. The second data set was collected from 100 popular sites in November 1999 and is made up of 86 front pages consisting of 1,490 files. The third data set was gathered in

February 2000 and consists of 92 front pages out of 100 popular sites made up of 1,511 files. Since these data sets were collected from popular Web sites we call them “popular data sets”. We also examined part of an NLANR access log recorded on 25 January 2000 that we call “log data”. It includes 27,546 request entries. After cleaning and reducing the log based on considerations that were suggested by Krishnamurthy and Rexford in [61], the log data ended up with 2,369 unique Web pages. Chapter 4 of the thesis explains more about cleaning and data reduction in working with the access logs. The log data does not correspond to popular Web sites and was studied to compare the characteristics of popular sites with that of regular Web sites.

A program was written to characterize Web objects. This program generates statistics such as the size of a Web object, and also the size and type of all of its embedded objects. The first step in this program is recognition of the embedded objects that belong to the Web page source file. The program parses the Web page source file and finds the appropriate tags that are related to the unique embedded files such as images, frames, applets, embedded Audio/Video files and embedded dynamic files (i.e., the files that contain ‘cgi-bin’ or ‘?’ in their names). Since each frame may have embedded files, this program also reads the text of a frame and finds the appropriate tags to identify the embedded files belonging to that frame. Finally, this program reports size, type and number of embedded objects that belong to a Web object.

In this study, we are interested in identifying the embedded objects that should be transferred to display a complete Web page. To examine the accuracy of our program we

compared the results of the program with manual measurements made on our first data set (i.e., front pages of 55 popular Web sites). We measured the pages belonging to the first data set manually through the Netscape Communicator browser. The Netscape browser can show the information on embedded files that belong to a received Web page. To obtain the data, each page belonging to our first data set was downloaded by the Netscape® Communicator 4.07 browser running in a Windows NT environment. To measure the sizes of the embedded objects in a Web page, the “Page-info” option under the “View” toolbar was used. This option is used to look at the structure and source of the Web pages [82]. The results obtained by our program were the same as the results that were reported by the Netscape browser.

One limitation of our source-based measurement approach is that all the embedded files in a page are considered. Although this method produces an accurate report on the structural properties of the Web page, it is unable to capture the dynamic system behaviour that depends on the input of a particular user. For example, only some of the embedded files are transferred when the user aborts a transfer in the middle. Such dynamic behaviour is aptly captured in the access logs. Also, for dynamic pages that use the conditional inclusion of embedded files in a CGI script, our program may overestimate the total number of embedded objects displayed for a user.

For log data, we ran this program on 2,369 Web pages to find Web objects and their related embedded objects. Based on the embedded objects information, the access log was searched to see how many of these embedded objects were requested. Two series

of statistics were generated: one was based on source-based information on 2369 Web pages and the other was based on requested files that we call request-based information. We ran the program 1 week after the access log date, ignoring the cases where the names of embedded objects were changed.

3.3 Data Analysis

Only data on three data sets (i.e., popular data sets) and the access log (i.e., log data) are presented in this section. The implications of these data are discussed in the following section.

3.3.1 Web Object Sizes

By analyzing the Web pages we observe that some pages have multiple instances of the same embedded objects such as arrows. In this work, a repeated embedded object is considered only once in the measurement of Web object size. This is because a repeated object in a Web page is stored and transferred only once. The actual number of embedded files is computed after considering repeated embedded objects. By adding the sizes of embedded files, the size of a Web object can be obtained.

The cumulative distribution function (CDF) and probability density function (pdf) plots for Web object sizes in popular Web documents are presented in Figures 3.1 and 3.2, respectively.

Figure 3.1: CDF of Web object sizes for popular data sets

Figure 3.1 indicates that more than 90% of Web objects are less than 100 KB. The maximum Web object size is observed to be 285,252 bytes, and the means of Web object sizes are in the range of 50,000 to 60,000 bytes (see Table 3.1).

Table 3.1: Statistics of Web object sizes (in bytes) for popular data sets

Characteristic	Mar 1999	Nov 1999	Feb 2000
Mean	60,061	54,686	50,818
Median	47,454	46,413	41,712
Variance	1.7E+9	1.9E+9	2.1E+9
Maximum	217,839	285,252	283,303
Sample size	55	86	92
Coefficient of variation	0.86	0.80	0.91
Model	Weibull $a=1.64$ $b=67,718$	Weibull $a=1.32$ $b=59,238$	Weibull $a=1.06$ $b=52,004$

Figure 3.2 shows the pdf plot for the popular Web object sizes. The pdf of all three popular data sets show that their underlying distributions are skewed to the right. This observation is supported by the fact that the median is less than the mean for all three data sets (see Table 3.1). The skewed right distributions that can be considered as the candidates for modeling Web object sizes are Weibull and lognormal distributions. However, we cannot assess how well they fit the distribution just by observing the pdf shape of the Web object sizes. Note that based on the knowledge of the previous work on characterization of the Web files (presented in [15] for example), Weibull and lognormal distributions are used for modeling the Web object characteristics in this chapter.

Figure 3.2: pdf of Web object sizes for popular data sets

To be able to assess how well the lognormal and Weibull models fit the distributions of the popular data sets, the parameters for Weibull and lognormal distribution should first be measured. The Weibull density function is:

$$f(x) = ab^{-a} x^{a-1} e^{-(x/b)^a}$$

where, a is a shape parameter and b is a scale parameter [64]. The lognormal density function is:

$$f(x) = \frac{1}{\sigma x \sqrt{2\pi}} e^{-(\ln x - \mu)^2 / 2\sigma^2}$$

where the parameters μ and σ are the mean and standard deviation of $\ln(x)$, respectively.

Maximum-likelihood Estimators (MLE) [64] can be used to estimate the parameters of the candidate models for each data set. For example, for the February 2000 data set when using lognormal model, MLE calculates the parameters of μ and σ equal to 10.28 and 1.49, respectively. For the November 1999 and the March 1999 data sets the lognormal parameters are ($\mu=10.55$, $\sigma=1.17$) and ($\mu=10.82$, $\sigma=0.59$), respectively. The parameters of Weibull distribution estimated by MLE, are listed in Table 3.1.

Visual comparison of the CDFs of these models and CDFs of data sets are presented in Figures 3.3, 3.4 and 3.5. Figure 3.3 shows that for March 1999 data set both Weibull and lognormal fit well. Although it is difficult to distinguish the different lines in Figure 3.3, visual observation of Figures 3.4 and 3.5 shows that the Weibull distribution provides a better representation for popular Web object sizes.

Figure 3.3: CDF comparisons for March 1999 popular data set

Figure 3.4: CDF comparisons for November 1999 popular data set

Figure 3.5: CDF comparisons for February 2000 popular data set

Because it is sometimes difficult to compare the models by visual observation (for example, see Figure 3.3), and also to assess the fitness with a more accurate method, the Kolmogorov-Smirnov (K-S) goodness-of-fit method was used in this chapter instead of visually comparing CDF plots. The K-S method measures the largest vertical distance between the CDF of the fitted model $F^{\wedge}(x)$ and the empirical distribution $F_n(x)$ of the data samples. The value of this distance is defined as:

$$D_n = \max \{D_n^+, D_n^-\},$$

where n is the number of samples and

$$D_n^+ = \max \{i/n - F^{\wedge}(x_i)\}$$

and

$$D_n^- = \max \{F^{\wedge}(x_i) - (i-1)/n\} \text{ for } 1 \leq i \leq n.$$

The value of D_n shows the deviation between the model and data. Therefore, a smaller value of D_n indicates a better fit. In this work we used the K-S method to find the D_n values for different models to be able to suggest a good candidate based on visual fit. Comparing the D_n values produced by the K-S method to find a better fitted model was also used in [37].

D_n values (i.e., K-S statistics) for Weibull and lognormal models are presented in Table 3.2. A comparison of these values indicates that although for the March 1999 data set lognormal is a better model, Weibull is a better candidate than lognormal for the November 1999 and the February 2000 data sets. Thus, based on the graphical plots and the K-S statistics, we conclude that the Weibull distribution provides a good visual fit for popular Web object sizes. As shown in Table 3.1, the shape parameter of the Weibull distribution decreases for more recent data sets. Decreasing the shape parameter in the Weibull distribution reflects a leftward movement of the body of the distribution. This is supported by the plot of the pdfs of popular Web object sizes in Figure 3.2. For example, the body of the pdf of the February 2000 Web object sizes is more leftward compared to the pdf for the March 1999 data sets.

The fact that a Weibull distribution is not considered a heavy-tailed distribution suggests that the distribution of Web object sizes for popular data sets analyzed in this section is not heavy-tailed. A detailed discussion of this issue follows.

Table 3.2: K-S statistics for popular Web object sizes

Model	March 1999	November 1999	February 2000
Lognormal	0.067	0.167	0.186
Weibull	0.12	0.095	0.088

In order to visually inspect whether or not the data set has a heavy-tailed distribution, LLCD plot was used. LLCD plot is a log-log complementary distribution that illustrates the tail behaviour. The complementary distribution (CD) is shown by $P[X > x]$ and obtained by computing $1 - F(x)$, where $F(x)$ is a CDF of the data set. LLCD plot shows $\log(P[X > x])$ versus $\log(x)$. As noted in Section 2.2.1, heavy-tailed distributions are characterized by:

$$P[X > x] \sim x^{-\alpha}, \text{ as } x \rightarrow \infty, 0 < \alpha < 2$$

Using derivation on log-transform of $P[X > x]$ results:

$$d\log(P[X > x]) / d\log(x) \sim -\alpha \text{ for } x > k$$

This means that on the LLCD plot for the large values of x (i.e., $x > k$), the upper tail of heavy-tailed distributions is linear with the constant slope of $-\alpha$. Thus, by measuring the slope of LLCD plots for the large values of x , the shape parameter α for the true distribution can be estimated. Since selecting k (i.e., the beginning of the tail) is done visually, estimating α is not very accurate by this method [33].

Figure 3.6 shows the LLCD plot for popular data sets. The value of k should be selected for estimating the slope. Then for the values of $x > k$ the linear behaviour of tail should be tested for the examined data sets. From Figure 3.1 the value of 100,000 can be chosen for the k that corresponds to the start of the tail decay for the three data sets. In the CDF plot the start of the tail is the place where the graph flattens out before reaching 1.

This area shows the large values of x , which have the small probabilities. The tail area is amplified in Figure 3.6, for which the slope should be measured. To examine linear behaviour as well as to estimate the slope in the LLCD plot, the least-square regression fit [54] was used for the Web object sizes greater than 100 KB.

Figure 3.6: LLCD of Web object sizes for popular data sets

The estimated α values (i.e., slopes) and R^2 values (i.e., coefficient of determination) that assess the goodness of fit for linear regressions are presented in Table 3.3. A R^2 value that is closer to 1.0 indicates a stronger fit, which means more linear behaviour. For example, the February 2000 data set indicates more linear behaviour compared to the other data sets. Since slopes are greater than 2, there is not enough evidence to claim that popular Web object sizes have heavy-tailed distribution.

Table 3.3: Estimates of α for popular Web object sizes

Item	March 1999	November 1999	February 2000
α	2.88	2.98	2.11
R^2	0.88	0.86	0.92

Figures 3.7 and 3.8 show the CDF and pdf of Web object sizes for the log data. For the log data, our source-based measurement shows that the maximum Web object size is 5,993,115 bytes and the mean and median of the Web object size are 77,319 and 20,628 bytes, respectively. In the request-based measurement the maximum Web object size is 1,329,907 bytes and the mean and median of the Web object size are 24,046 and 9,865 bytes, respectively. It is important to note that in both of these measurements approximately 90% of Web objects are less than 100 KB in size. This agrees with our observation on popular Web objects (see Figure 3.7). The reason for the difference between the source-based and request-based measurements for the Web object size is explained in Section 3.3.2.

Figure 3.8 shows the pdf of the Web object sizes for the log data. For both sourced-based and request-based Web object sizes, the values of the means are greater than the values of the medians, which suggests that their probability distributions are skewed to the right. The pdf shapes for the source-based and request-based Web objects also confirm that their distributions are skewed to the right. The pdf shapes of Figure 3.8 suggest that one candidate for modeling can be a lognormal distribution whose density function can take similar shapes to that of Figure 3.8. For request-based Web object sizes when using a lognormal distribution, maximum likelihood estimators

Figure 3.7: CDF of Web object sizes for log data

Figure 3.8: pdf of Web object sizes for log data

calculate the parameters of $\mu=9.03$ and $\sigma=1.50$. These parameters are $\mu=9.31$ and $\sigma=2.72$ for source-based Web object sizes

Using the K-S method for the lognormal model generates the D_n values of 0.05 and 0.18 for request-based and source-based Web objects, respectively. The smaller D_n values show the better fit for request-based Web object sizes compared to source-based Web object sizes. This can be explained by comparing the tails of the pdfs in Figure 3.8. As indicated in this figure, the pdf of source-based Web object sizes has a longer tail compared to the pdf of request-based Web object sizes. Such a long tail is observed for the Web file sizes in [15], and has been modeled as a mix of two distributions: lognormal for the body and Pareto for the tail. More analysis of the tail characteristics of the Web object sizes for the log data is provided next.

Figure 3.9 shows the LLCD plot of Web object sizes for the log data. This figure also shows that the distributions of the source-based Web object sizes have a heavier tail compared to the distribution of the request-based Web object sizes. By using a least-squares regression fit, the α values are estimated as 1.02 and 1.48 for source-based and request-based Web objects sizes respectively (see Table 3.4). Having lower values of α confirms that the distribution of source-based Web objects sizes has a heavier tail compared to the distribution of the request-based Web object sizes. By considering the start of the tails of distributions from Figure 3.7 the values of k were selected equal to 140KB and 50KB for source-based and request-based Web object sizes, respectively. The fitted straight lines generated by regression are shown in Figure 3.9.

Figure 3.9: LLCD of Web object sizes for log data

The estimated R^2 values are listed in Table 3.4. These values show a very strong fit (R^2 s are close to 1) that supports the linear behaviour of the tails. Thus, the results of the least-square regression fit indicate that both request-based and source-based Web object sizes in the log data have heavy-tailed distributions.

Table 3.4: Estimates of α of Web object sizes for log data

Item	Request-based	Source-based
α	1.48	1.02
R^2	0.98	0.96

In this study we have also used Aest, the software that was developed by Crovella and Taqqu [33] to measure the tail weight (α). Aest uses the scaling method to compute

α and gives a result that can be observed with Gnuplot. The result of running Aest on log data gives an α of 1.233 for the source-based measurement of Web object sizes and an α of 1.227 for the request-based measurement of Web object sizes. Since $\alpha < 2$, the Web object size distribution in the log data is also heavy-tailed, which agrees with our observation from using LLCD to measure the tail weight. Being heavy-tailed suggests that, although most Web objects are small, there can be some extremely large Web objects.

3.3.2 Number of Embedded Objects

The number of embedded objects in a Web page is the number of unique files that must be transferred to display the page at the client site. The statistics on the number of embedded objects for popular data sets are presented in Table 3.5. Figure 3.10 presents the CDF plot for the data that corresponds to the 100 popular sites. From this data, we observe that 95% of Web pages have fewer than 40 embedded objects.

Figure 3.11 presents the pdf plot for the number of embedded objects in the popular data sets. This pdf plot shows that the underlying distributions are skewed to the right. This observation is supported by the fact that the values of their median are less than the values of the mean for all three data sets (see Table 3.5).

Table 3.5: Statistics of number of embedded objects for popular data sets

Characteristic	Mar 1999	Nov 1999	Feb 2000
Mean	17.8	17.5	16.1
Median	16	15	14
Variance	126.3	152.2	144.7
Maximum	45	59	57
Sample size	55	86	92
Coefficient of variation	0.63	0.71	0.75
Model	Weibull $a=1.61$ $b=19.8$	Weibull $a=1.42$ $b=19.2$	Weibull $a=1.32$ $b=18.1$

Figure 3.10: CDF of the number of embedded objects for popular data sets

Weibull and lognormal distributions can have shapes similar to Figure 3.11. To find the best candidate for modeling, the K-S method was used to generate the largest

Figure 3.11: pdf of the number of embedded objects for popular data sets

vertical distances (i.e., D_n) between these models and distributions of the number of embedded objects for popular data sets. Based on the K-S statistics shown by Table 3.6, Weibull distribution is a better fit for the number of embedded objects of popular data sets. The estimated parameters of the Weibull model generated by MLE are listed in Table 3.5.

Table 3.6: K-S statistics for number of embedded objects in popular data sets

Model	March 1999	November 1999	February 2000
Lognormal	0.12	0.11	0.10
Weibull	0.06	0.04	0.08

Figure 3.12 shows the LLCD of the number of embedded objects for popular data sets. Figure 3.10 and 3.11 show that tails begin after the value of 30. If a distribution has

a heavy tail, the tail in the LLCD plot should be linear. To examine the linearity and also to estimate the slopes, least-square regression fit was used for the number of embedded objects greater than 30 (i.e., $k=30$). The estimated α and R^2 values are listed in Table 3.7. The estimated values for the slopes are greater than 2, which suggests that the number of embedded objects in the popular data sets do not have a heavy-tailed distribution. Table 3.5 shows that the variance of the number of embedded objects in a Web object is small, which supports this conclusion.

Table 3.7: Estimates of α for the number of embedded objects in the popular data sets

Item	March 1999	November 1999	February 2000
α	4.79	4.43	3.92
R^2	0.97	0.98	0.91

Figure 3.12: LLCD of the number of embedded objects for popular data sets

Figure 3.13, which shows the log data, presents the results for both request and source-based measurements. As shown in Figure 3.13, for the request-based measurement, 95% of requested Web pages contain 10 embedded objects or fewer. For source-based measurement, 70% of Web pages contain 10 embedded objects or fewer. We found that, on average, 25% of the embedded objects belonging to a Web object were requested in the log data. As a result, we have two different lines for source-based and request-based measurements on the Web object size and the number of embedded objects

Figure 3.13: CDF for the number of embedded objects for log data

in all the graphs related to the log data. In their study, Crovella and Barford [15] used a threshold of 1 second to recognize the embedded objects. Based on this threshold they estimated the number of embedded objects and suggested a Pareto distribution with parameters $k=1$ and $\alpha=2.43$. Note that since $\alpha > 2$, the distribution is not heavy-tailed. In

a similar work by Deng [34], a threshold of 60 seconds was used for the recognition of embedded objects but no distribution was suggested.

In another study by the Open text index group [19], the number of image references in the Web page source files was counted to determine the number of embedded objects. For our purposes, the information obtained by counting the number of image files is inadequate. First, an image file may be repeated in a Web page several times. Thus, counting the number of image files is not an accurate method for determining the number of embedded files. Note that only one copy of an embedded object is stored independent of the number of times it is used in a Web page. Secondly, there are other types of embedded objects, which are not included in their study. Finally, they do not present data on the mean number of embedded images in a Web page.

Figure 3.14 presents pdf plots for the number of embedded objects in the log data. The pdf of the number of embedded objects in the request-based log data is similar to the shifted exponential distribution. The pdf of the number of embedded objects in the source-based log data indicates that besides exponential, Weibull and lognormal distributions can be used as candidates for modeling. Note that an exponential distribution $\text{expo}(b)$ is the same as Weibull distribution $\text{Weibull}(1,b)$, when shape parameter a is equal to 1 [64]. The pdfs of underlying distributions are skewed to the right. By comparing the D_n values generated by the K-S method for these models, Weibull distribution appears to be a better fit for both request-based and source-based number of embedded objects for the log data. The D_n values are 0.67 and 0.14 when

using the Weibull model for request-based and source-based data respectively. The estimation of the parameters of the models by MLE are $(a = 1, b = 1.87)$ and $(a = 0.88, b = 7.85)$ for request-based and source-based data respectively.

Figure 3.14: pdf for number of embedded objects for log data

Figure 3.15 shows the LLCD plot of the number of embedded objects in the log data. Figure 3.13 shows that the tail begins after the values of 10 and 24 for request-based and source-based log data in CDF plot, respectively. Therefore, the least-square regression fits were used for the number of embedded objects greater than 10 and 24, for request-based and source-based data in the LLCD plots of Figure 3.15. The estimated α and R^2 values are shown in Table 3.8. Since the estimated values for the slopes are greater than 2, the distribution of the number of embedded objects in the log data is not a heavy-tailed distribution.

Figure 3.15: LLCD for number of embedded objects for log data

Table 3.8: Estimates of α for number of embedded objects for log data

Item	Request-based	Source-based
α	2.86	4.26
R^2	0.97	0.98

The result of running Aest on our log data gives an α of 2.1 for the source-based measurement, and an α of 2.22 for the request-based measurement, for the number of embedded objects in a Web object. Since α is higher than 2, the distribution is not heavy-tailed. Also, using Hill's estimator [40] confirms that the distribution is not heavy-tailed.

3.3.3 Size of Embedded Objects

In the study of the first data set of popular sites, March 1999 (1,051 image and HTML files), the mean file size is observed to be 3,256 bytes. It is interesting to note that the coefficient of variation of Web file size is large (see Table 3.9). The larger coefficient of variation for the embedded object size may be due to a mix of text and image files used in constructing a popular Web page.

Table 3.9: Statistics of Web file sizes (in bytes) for popular data sets

Characteristic	Mar 1999	Nov 1999	Feb 2000
Mean	3,256	2,563	2,772
Median	877	1013	947
Variance	4.9E+7	2E+7	2.6E+7
Maximum	101,333	52,193	49,679
Sample size	1,051	1,490	1,511
Coefficient of variation	2.15	1.74	1.9
Model	Lognormal $\mu = 6.93, \sigma = 1.59$	Lognormal $\mu = 6.78, \sigma = 1.62$	Lognormal $\mu = 6.76, \sigma = 1.66$

Figure 3.16 presents the CDF for embedded object (i.e., file) sizes of popular sites. From this data, we observe that 90% of files are fewer than 10,000 bytes. Figure 3.17 presents pdfs for the embedded object sizes in the popular data sets.

Figure 3.16: CDF of file sizes for popular data sets

Figure 3.17: pdf of file sizes for popular data sets

The pdfs of the embedded object sizes for November 1999 and February 2000 data sets are very close to each other. Observing the shape of the pdf of these two data sets suggests that Weibull and lognormal distributions can be used for modeling them. Comparing the Dn values generated by the K-S method for Weibull and lognormal distributions, indicates that the lognormal model is the better fit for the file sizes of the popular data sets (see Table 3.10).

Table 3.10: K-S statistics for popular embedded object sizes

Model	March 1999	November 1999	February 2000
Lognormal	0.04	0.05	0.05
Weibull	0.09	0.06	0.07

Table 3.10 shows that lognormal distribution is a better model compared to Weibull for file sizes of popular data sets. The estimation of the parameters of the Weibull model generated by MLE are $(a = 0.66, b = 2234.2)$, $(a = 0.68, b = 1935.2)$ and $(a = 0.65, b = 1955.8)$ for the March 1999, November 1999 and February 2000 data sets respectively. The parameters of the lognormal distribution generated by MLE are listed in Table 3.9. All the pdfs are skewed to the right with the long tails that suggest heavy-tailed distributions. However, computing α is important to recognize a heavy-tailed distribution [63]. A detailed discussion follows.

Figure 3.18 shows the LLCD plot of the embedded object sizes for the popular data sets. Figure 3.16 shows that tails begin for file sizes greater than 10,000 bytes, which corresponds to about 90% of the data points in the CDF plot. Therefore, the estimation of

the slopes by using least-square regression fit was done for the embedded objects having sizes greater than 10,000 bytes (i.e., $k=10,000$).

Figure 3.18: LLCD of file sizes for popular data sets

The estimated α and R^2 values are shown in Table 3.11. The lower R^2 value of February 2000 suggests that in the LLCD plot it shows less linear behaviour compared to the other two data sets. This conclusion is supported by visual observation of the tail behaviour of the February 2000 data set in Figure 3.18. The estimated values for all the slopes are greater than 2, which suggests that the size of embedded objects in the popular data sets does not have a heavy-tailed distribution.

Table 3.11: Estimates of α for size of embedded objects in popular data sets

Item	March 1999	November 1999	February 2000
α	2.09	2.31	2.01
R^2	0.99	0.97	0.89

Running Aest on the popular data sets of March 1999 (1051 Web files), November 1999 (1490 Web files) and February 2000 (1514 Web files) produces an α of 1.596, 1.52 and 1.45, respectively. Based on these Aest results, since $\alpha < 2$, the popular Web file size distribution is heavy-tailed that contradicts our results when using the LLCD. However, the developers of Aest suggest that it should be used together with the graphical plots of the tail. They also mentioned Aest is suitable for large data sets [33]. In a recent paper Downey [38] reports that Aest cannot distinguish between long tailed and lognormal distribution. This means Aest may wrongly report a heavy-tailed distribution for the data that come from a lognormal distribution. Therefore, we do not consider the Aest results for the file sizes in popular data set. Thus, based on the LLCD plot, the CDF, the pdf and estimated values of the slopes we conclude that the distribution of the file sizes in the popular data sets considered here is not a heavy-tailed distribution.

Many researchers have characterized the file sizes for regular Web documents. For example, for files that are larger than 10,000 bytes, Crovella and Bestavros [31] reported an α of 1.27 for all files and an α of 1.59 for text files. Also Arlitt and Williamson [11] observed that the size of files larger than 1024 bytes are Pareto

distributed with $0.40 < \alpha < 0.63$. These results suggest that file size distribution is a heavy-tailed distribution for regular Web documents.

3.3.4 Types of Embedded Objects

Embedded objects are mostly image files with a GIF or JPEG format. In the popular data set recorded on March 1999, from 1051 observed embedded objects, 71 HTML text files account for 33% of the total size of Web objects and 980 picture files account for 67% of the total size of Web objects. The HTML file size mean is 16,054 bytes whereas the picture file size mean is 2,329 bytes. Arlitt and Williamson [11] report a mean of 9,123 bytes and 10,053 bytes for text and picture files, respectively. If we compare the sizes of popular Web documents with the sizes of the regular Web documents reported in [11], we observe that on average:

- Image files of popular Web files are smaller than the image files of regular Web files
- Text files of popular Web files are larger than the text files of regular Web files

Analyzing the popular data set of March 1999 shows that about 93% of the files are picture files and the remaining 7% are text files. A key difference between the characteristics of popular Web files and regular Web files is that the picture files in the popular Web documents tend to be much smaller in size (2300 bytes versus 10,000 bytes).

3.3.5 Other Web File Types

As noted in Section 3.3.4, the Web objects in the popular Web documents that were investigated in this work consist of only picture and text files and none of them contained applet or other types of files. In the Web there are also very big files (such as Audio/Video file). Since most of the Audio and Video files are large in size, instead of being embedded objects of the pages, they are usually retrieved through the hyper links from the Web pages. Some of the Audio and Video files are real-time streams and their links are redirected to other servers that do not provide information on size and type. However, we studied popular Web pages containing links to Audio and Video files with MP2, MP3, MIDI and AGIF formats. The links in these Web pages that were selected from 100hot.com sites can be used to download the Audio and Video files. After downloading these files, they were stored and the sizes of stored files were measured. We examined 92 stored Audio and Video files from popular Web files. Statistics on their sizes are presented in Table 3.12.

Table 3.12: Statistics on popular Audio/Video files

Mean (bytes)	2,976,110
Median (bytes)	2,883,584
Variance	4.96E+12
Minimum (bytes)	26,624
Maximum (bytes)	10,171,187
Coefficient of Variation	0.75
Sample size	92

3.4 Implications of Characteristics on Web Performance

The data presented in this chapter is based on 233 popular Web sites that were observed on three different dates that span 11 months. We expect that these data capture the general characteristics of popular Web files.

Our data on the number of embedded objects and Web object sizes show the utility of devising Web object-based caching and prefetching techniques to improve cache management policy. Our data on the number of embedded objects and Web file sizes (see Tables 3.5 and 3.9 and Figure 3.16) show the possibility of reducing disk latency on Web or proxy servers. Sections 3.4.1 and 3.4.2 explain both of these issues, and Chapters 4 and 5 describe an implementation of these results for proxy caching. Section 3.4.3 and Section 3.4.4 describe other implications that are important for Web performance improvement.

3.4.1 Improving Web/Proxy Server Cache Management

A number of observations on the structure and sizes of Web objects and files are presented in Section 3.3 of this chapter. In this section, we summarize the implications of our observations to devise the better policies for cache management in Web/proxy servers. Most of our discussion here focuses on text and picture files.

Web object size: Measurements made on popular Web objects and regular Web objects show that 90% of Web objects are less than 100,000 bytes in size (see Figure 3.1 and Figure 3.7). This indicates that, in most cases, entire Web objects can be effectively cached in the main memory at client workstations and thus can be re-accessed with low latency. The reasonable size of popular Web objects also implies that multiple popular Web objects can be cached for different clients on a global cache (proxy) serving multiple clients. Note that most of the embedded objects belonging to the same page are accessed together when that page is requested. This means that the cache replacement policy at the server/proxy may be able to use the cache space more efficiently by considering a Web object as a unit of cache replacement instead of using a Web file as a unit of cache replacement. We have devised Web object-based policies for cache management that prefetch the entire Web object when any of its component files is accessed. When replacement is necessary, instead of replacing individual files our cache management strategy replaces an entire Web object. A simulation-based study indicates that such Web object-based strategies can outperform traditional cache replacement strategies such as LRU and LFU (see Chapter 4 for more details).

Number of embedded objects: Popular Web pages were observed to contain 17 embedded Web files on average (see Table 3.5). Each of the embedded files needs to be transferred to the client for displaying the contents of the entire Web page. Thus, if the constituent Web files of a single Web object can be prefetched in proxy/client cache upon receiving a request for a Web page source file, the cache hit ratio can be increased. The advantage of prefetching embedded files belonging to a Web page is that the prefetched

files are more likely to be referenced. A mean of 17 embedded objects per page indicates a good potential for increasing the cache hit ratio. The improvement in cache hit ratio by using such a Web object-based prefetching is described in Chapter 4.

Web file size: A heavy-tailed distribution of file sizes for regular Web documents means that we have mostly small files but should expect a few large files as well [32]. Thus, if we consider file size as one of the factors in a cache replacement decision, it can result in a better cache replacement policy. For example, evicting a larger file may be preferable to evicting multiple small files because, by evicting a large Web file, space for multiple smaller Web files can be created in the cache. Similarly, not placing a file with a size larger than a threshold in a cache, may improve cache performance. Since Web object sizes also have a heavy-tailed distribution for regular Web documents, the same consideration of size in file-based cache management is also applicable for Web object-based cache management.

Although the sizes of Audio/Video files exhibit less variability in comparison to those of the picture and text files (see Table 3.9 and Table 3.12), their mean size is orders of magnitude higher than that of the text and picture files. This implies that caching Audio/Video files can consume most of the cache capacity.

3.4.2 Reducing Disk Latency in Web/Proxy Servers

Since proxies mostly use disk-based caching, and some Web-based applications perform significant disk I/O, reducing disk latency can improve the proxy/Web server performance. In this section, we discuss the implication of our results on reducing disk I/O latency in Web/proxy servers.

Number of embedded objects: Proxy caches often employ both disk and memory. Servers can also use main memory as caches. Section 3.3.2 shows that the average number of embedded files in a Web object is 17. This means that for retrieving a Web object stored on the disk of a proxy or a server, a conventional file based strategy can give rise to multiple file transfers. However, when one of the members of the Web object is accessed it is possible to prefetch all the other members from disk to main memory. Since most of the Web object members are accessed together the follow-up requests for the other members of the same Web object are satisfied from the main memory.

Web file size: The distribution of Web file sizes is observed to be characterized by a high coefficient of variation (see Table 3.9). A small mean and large variability in the Web file size implies that a small proportion of the available files is large, whereas most of the files are small in size (see Figure 3.16). Note that 90% of Web files are less than or equal to 10,000 bytes (see Table 3.9 and Figure 3.16). Since most files that belong to the Web objects are small in size the files can be packed into consecutive blocks on a disk.

Clustering of the Web files that belong to the same Web object can reduce disk seek time, which in turn reduces disk latency.

3.4.3 Implication of Results for Dynamic Web Pages

The Web documents that were investigated in this work mostly consist of static files. Sections 3.4.1 and 3.4.2 present a discussion on how the knowledge we have gained about their structural characteristics can be used to improve the performance of caching and disk I/O on proxy and server systems that store static Web pages. Generating pages dynamically often requires incorporating databases into Web servers. These databases can be accessed by Common Gateway Interface (CGI) scripts or Java servlets. Since the execution of these programs require additional CPU cycles, using faster processors is important for attaining high system performance (see [24] and [47]). Since each dynamic page also has static components, caching these static parts is another way to obtain performance improvement (see [36] and [23] for example). The discussion in the previous section on structural characteristics of static Web pages and on efficient cache and disk management strategies are thus applicable in the context of dynamic pages as well.

3.4.4 Efficient Web Page Design

Characterization of popular Web documents can lead to a better understanding of the methods underlying the efficient design of a Web page. Keeping the page download

time reasonable is important for Web page designers. The median value of Web object sizes observed in our study for popular documents varies from 40 to 46 KB (see Table 3.1). Ignoring network congestion, this corresponds to a page download time of under 13 seconds for a user with a 28.8 Kbits/sec modem and to a page download time of under 7 seconds for a user with 56.6 Kbits/sec modem. This is in line with the popular belief that most Web page designers try to keep the page download time below 10 seconds for a user with an inexpensive modem.

Instead of a monolithic construction, a Web page is observed to consist of multiple embedded files. The mean number of embedded objects per page varies between 16.1 to 17.8 for popular data sets considered in this study (see Table 3.5). One of the reasons for using multiple embedded objects is the existence of different types of information such as text and images in the same page. In our study we have also observed multiple occurrences of the same embedded object within the same page. Embedded files mostly have sizes less than 10 KB (see Table 3.9). Using small components in multiple locations of a page (i.e., front page as well as other pages) leads to lower network traffic, because fewer unique embedded files need to be transferred to download that page. Also, caching of small files that are used repeatedly in a Web page is more efficient because less space is needed to place that Web page in the cache. Thus, using small embedded objects that can be repeated in different locations of a Web page is recommended for efficient Web page design.

Another observation for popular Web sites is that the designers of these sites used only images and text embedded files in the front pages (see Section 3.4). Other types of embedded Web files such as Audio/Video files and applets were not present in the front page of popular Web sites. This is because of the larger size of Audio/Video files compared to text and image files (see Table 3.12) can lead to larger download times.

3.5 Summary

The difference between this study and previous work is that in this study Web objects are characterized. In previous work, Web files were studied without considering the inherent structure of the Web page that contains multiple Web files. We studied the Web objects in popular Web documents and compared our results with the characterization of Web objects in regular Web documents. Since popular Web documents are accessed more often than others, their characteristics are expected to have a strong impact on performance. A summary of the important characteristics of Web objects for popular and regular Web documents is presented in Table 3.13. This section highlights the important observations presented in the chapter. The conclusion are based on the data analyzed in this chapter.

We found that the mean number of embedded objects for popular Web documents studied in this chapter is approximately 17. Also, we observed that the distribution of the number of embedded objects is not heavy-tailed, which agrees with the results reported by other researchers on regular Web sites. Little data exist on the measurement of Web

object size and the characteristics of its distribution. In this study, the distribution of Web object sizes for regular documents is observed to be heavy-tailed, and 90% of them have a size that is less than 100,000 bytes. The distribution of popular Web file sizes is not heavy-tailed. This can be explained by the fact that the designers of the popular pages want to keep page download time reasonable. Therefore, they do not use large embedded objects that lead to longer page download times. The Weibull distribution is suitable for modeling the distributions of Web object sizes and number of embedded objects in popular documents.

Lognormal appeared to be suitable for modeling the distribution of file sizes in popular Web documents.

Table 3.13: Summary of Web object characteristics

Characteristic of Web object	Popular Web documents	Regular Web documents
Distribution of Web object sizes	Weibull distribution	No data in previous work heavy-tailed in log data
Web object size statistics	90% less than 100 KB	No data in previous work 90% less than 100 KB in log data
Distribution of number of embedded objects	Weibull distribution	Not heavy-tailed ([15]) not heavy-tailed in log data
Mean number of embedded objects	16.14 to 17.78 (see Table 3.5)	90% included 17 or less with median of 11 [10] mean of 8 in log data
File size distribution	Lognormal distribution	Heavy-tailed ([11], [31] and [15])
File size statistics	90% of the files are less than 10,000 bytes	Most of the files are in the range of 100 – 100,000 bytes [11]

We observed that text files in popular Web documents were larger in size than text files of regular Web documents reported in [11]. On the other hand, image files of popular Web documents were smaller in size than image files of regular Web documents described in [11].

We have also discussed the utility of the results reported in this chapter. Our data on the number of embedded objects and Web objects sizes can be useful in the investigation of Web object caching and prefetching techniques. We have implemented a Web object-based proxy caching strategy. This is discussed in Chapter 4. We have also examined how the characteristics reported could be exploited in reducing disk latency on proxy servers. An investigation of this issue is presented in Chapter 5.

Chapter 4

Using Web Objects to Improve the Performance of Proxy Caching

4.1 Introduction

This chapter demonstrates that by using Web document structure, a significant performance improvement for proxy caching can be achieved. We introduce two cache management strategies: Web object-based Least Recently Used (WOLRU), and Web object-based Least Frequently Used (WOLFU), which are simple and implementable on proxy servers.

This chapter is organized as follows. Section 4.2 describes the NLANR Web access logs used in this research. In addition to the NLANR trace that was used in Chapter 3, two other NLANR traces are used in this chapter to study our proposed policies. We used one more NLANR trace in Chapter 5. Since we used the same process for cleaning these logs, Section 4.2 describes all the NLANR traces and the cleaning process. Section 4.2 also presents workload characteristics such as popularity patterns and one-timer statistics for the NLANR traces used in the simulation of the cache management strategies. The proposed cache management strategies are described in Section 4.3. This section also explains the clustering method used in our cache management strategies. Section 4.4 discusses the simulation details, performance metrics

and caching policies that have been simulated. Analysis of the results is presented in Section 4.5. A discussion of the prefetching component of the policy is presented in Section 4.6. This section also includes a more detailed analysis of the cache misses. Section 4.7 gives an overall summary.

4.2 Proxy Log Characteristics

We have used four access logs collected by NLANR for this research. Three traces are from a cache at the University of Illinois at Urbana-Champaign (UC), and one trace is from the University of Pittsburgh (PB) [86]. NLANR manages IRCACHE, a high-level proxy cache serving other caches [53]. In this chapter the three traces belonging to the University of Illinois at Urbana-Champaign (referred to as UC traces) are used for simulations. Each of these three access logs captures the cache accesses made on a specific day. The first trace is the first 12 minutes of an access log that was recorded on November 23, 1999 and consists of 8,410 request entries. The second trace is the first 81 minutes of an access log that was recorded on January 25, 2000 and is made up of 27,546 request entries. The third trace is an entire access log recorded on March 12, 2000 and includes 453,152 request entries. We call these traces UC1, UC2 and UC3 respectively (see Table 4.1). The three ranges of these trace lengths enable us to investigate the performance of our policies over three different time durations (minutes, an hour and a day). In Chapter 5 the University of Pittsburgh (PB) trace is used. The PB trace is the first half (the first 12 hours) of the one-day access log that was recorded on December 25, 2001 and is made up of 566,539 request entries. We used the NLANR

access logs because they are busy proxy access logs that provide the URL names, which we use in our method for finding Web objects. Other available proxy access logs provide a number instead of a requested URL because of confidentiality concerns, and are not useful in the simulation of our proposed policies. In Section 4.5.1, we have used the Boeing proxy access log that includes numbers instead of URL names. This trace is used for simulation of another prefetching-based policy from the literature that does not need URL names. More details on the Boeing access log and this policy are presented in Section 4.5.1.

Table 4.1: Statistics on the NLANR traces

Trace name	Recording date of the trace	Duration of requests	Number of requests	Number of requests after data reduction	No. of Web objects after data reduction
UC1	Nov 23, 1999	12 minutes	8,410	2,888	411
UC2	Jan 25, 2000	81 minutes	27,546	12,752	1,947
UC3	Mar 12, 2000	24 hours	453,152	191,470	25,780
PB	Dec 25, 2001	12 hours	566,539	220,124	25,450

Data Reduction: We considered a URL to be cacheable if it did not contain substrings such as ‘cgi-bin’ or ‘?’ and if it did not have a file extension such as ‘.cgi’. We reduced each trace by removing non-cacheable URL request entries. We simulated only cacheable URLs whose origin server response contained HTTP GET request with a status code of 200 that indicates a successful transfer. Since each URL may have a large string format, we converted an URL into a unique number. For identifying the same URL that occurred in different formats within the trace, we used the suggestions made by Krishnamurthy

and Rexford [61]. Their suggestions help to resolve difficulties in working with URLs in the access logs. These difficulties are due to the existence of a range of acceptable formats for a potentially long URL string. The same page may appear in the log with slightly different URLs. For example, both `http://www.scs.carleton.ca` and `www.scs.carleton.ca` can appear in the log. Both URLs refer to the same page and the first is replaced by a string (`www.scs.carleton.ca`) during the cleaning process before conversion to a unique number.

The summary of our data sets is presented in Table 4.1. In this table, the figures presented in the column titled “number of Web objects” is the number of Web objects in each trace identified by our Web object identification program (explained in Section 4.3). Note that each Web object is a group consisting of a Web page source file and its related embedded objects. We also simulated a policy that is proposed by Hine and Wills [46] for the traces listed in Table 4.1. Since in this policy we need to know the name of a server in each request entry, we removed the request entries that did not include the name of destination servers. Table 4.1 (5th column) shows the number of requests in each trace after data reduction.

4.2.1 One-time Referencing

The statistics related to one-timers are presented in Table 4.2. One-timers are the files in a log that are accessed only once. As shown in this table, the number of requests generated for one-timers is in the range of 70% to 80% of the total number of requests for

the traces used in this research. Except for UC1, which is the shortest trace, the number of bytes transferred by one-timers is in the range of 72% to 86% of the total number of bytes transferred. The number of one-timer files is in the range of 93% to 95% of the total number of distinct files. The proportion of one-timer files in NLANR traces is higher than that observed with other proxy servers. The reason for this can be explained by the filtering effects that exist in the Web caching hierarchies [106].

Williamson [106] analyzed the effects of having multiple levels of caches on file referencing characteristics. In multiple levels of caches, the proxies in the lower levels send requests to the higher level proxies only for the files that are not present in their cache. He reported that if lower level proxies use larger cache sizes, their requests to the higher level proxies are filtered for most of the popular documents. This means most of the requests sent to the higher level caches are for one-timers. This causes higher level proxies to store mostly one-timer files in their caches. For example, by using a trace-driven simulation he measured the number of one-timer files stored in the higher level caches when lower level proxies use the cache sizes in the range of 1 to 4 GB. The number of one-timer files stored in the higher level caches was in the range of 87% to 95% of the total number of distinct files.

Therefore, filtering effects cause a large number of one-timers to be present in the used NLANR traces. As noted before, there is no benefit to cache the one-timers because they are accessed only once. In spite of the large number of one-timers, there are approximately 5% to 7% of the documents responsible for 15% to 20% of the requests in

our traces. These documents are accessed more than once and are referred to as popular files. For NLANR traces the requests for popular files might be due to the overlap in requests from different proxies or clients. The characterization of popular files is presented next.

Table 4.2: Statistics on one-timers for NLANR traces

Item	UC1	UC2	UC3	PB
No. of distinct files	2,574	9,857	142,762	184,298
No. of one-timer files	2,459	9,233	133,822	175,776
One-timer/distinct files (%)	95.53	93.67	93.74	95.36
No. of total requests	2,888	12,752	191,470	220,124
No. of one-timer requests	2,459	9,233	133,822	175,776
One-timer/total requests (%)	85.16	72.40	69.89	79.85
Total bytes (MB)	43.40	300.74	4,199.33	3,722.13
Total one-timer bytes (MB)	41.36	219.19	3,626.05	3,016.82
One-timer/total bytes (%)	95.30	72.88	86.35	81.05

4.2.2 Popularity Characterization

As described in Section 2.1.2, a distribution similar to Zipf (i.e., Zipf-like) can be used to characterize the popularity of files in the Web. A Zipf-like distribution expresses a power-law relationship between popularity P (number of references) of a file, and its rank r :

$$P = c/r^\beta, \text{ where } c \text{ is a constant}$$

If β is exactly equal to 1, it is referred to as Zipf's law and if it is close to 1, it is referred to as a Zipf-like distribution [20] and [69]. For example, Web proxy workloads, in which β is in the range of 0.5 to 1 have been said to exhibit a Zipf-like pattern [10]. The value of β is important because it affects the popularity curve. The popularity curve

shows concentration of references to the popular files that affects the cache performance. The value of β depends on the location of a cache in a hierarchy. It has been observed that the higher level proxies have less β values compared to the lower level proxies in a cache hierarchy [91]. Since the Zipf-like distribution is characterized by a power-law relationship, its shape is linear with the slope equal to $-\beta$ on a log-log plot of popularity versus rank.

4.2.2.1 Popularity Analysis for Files

In order to examine whether the popularity of the files requested in each trace follows a Zipf-like distribution, the number of references to the files versus rank was plotted on the logarithmic (both X and Y axes) scale graphs for each trace. Figures 4.1 (a), (b), (c) and (d) show the popularity patterns (i.e., frequency of references versus ranks) for the UC1, UC2, UC3 and PB traces respectively. In these figures, the plots of the measured data are referred to as empirical. The model in these figures refers to a fitted line that is explained further in the following paragraphs. Except for UC1, the empirical plots for all other traces are approximately linear and similar to the Zipf-like distribution. For UC1, Figure 4.1 (a) indicates that most of the files are referenced fewer than 10 times and some of them have been referenced equally. One possible reason for such a behaviour is the short duration of the trace that captures only 12 minutes of references to the cache. Section 4.4, which focuses on the simulation warm-up period, shows the UC1 captures only the transient part of a cache simulation. As a result, re-references to the files that are typical in proxy cache references, are not captured in this

(a) UC1

(b) UC2

(c) UC3

(d) PB

Figure 4.1: Popularity patterns for (a) UC1, (b) UC2, (c) UC3 and (d) PB traces

trace. Figures 4.1 (c) and 4.1 (d) show a good similarity between the model and the empirical distribution for PB and UC3.

Visual observation indicates that UC2, UC3 and PB demonstrates Zipf-like behaviour. To estimate the exponents of these Zipf-like distributions, a least square fit was used on a modified data set for each trace. The modified data set was formed by a method that was used in [91] and [69]. In this method all files that have the same number of references are grouped together and assigned the same rank. For example, if three files are referenced 100 times and were ranked 2001, 2002 and 2003, then in the modified data set these three files are considered as a single data point with the rank of 2002 (i.e., the average of the three ranks) and number of references equal to 100. Table 4.3 shows the estimated β and the goodness of fit R^2 . According to these values the estimated β is in the range of 0.75 to 0.84 for UC2, UC3 and PB traces. This confirms that popularity patterns for these three data sets are characterized by Zipf-like distributions. The low value of β for UC1 suggests that the popularity pattern for this trace demonstrates less Zipf-like behaviour in comparison to the other three traces. Based on the higher β values and also the slightly lower proportion of the one-timers in UC2 and UC3 (see Tables 4.2 and 4.3) compared to the other traces used, we expect to have a better cache performance for UC2 and UC3 compared to UC1 and PB traces.

Table 4.3: Estimated slopes for popularity characteristics

Item	UC1	UC2	UC3	PB
β	0.51	0.76	0.84	0.75
R^2	0.93	0.93	0.98	0.98

4.2.2.2 Popularity Analysis for Web Objects

In characterizing the popularity of Web objects, we consider the source-based Web objects identified in Table 4.1. In addition, each of the remaining files in the trace is treated as a single file Web object. Based on this characterization 86.91% of distinct Web objects are accessed only once. The number of requests to one-timer Web objects is 47.8% of the total number of the requests in this trace. In order to examine whether the popularity of Web objects follows Zipf-like distribution, the number of references to the Web objects versus rank is shown in Figure 4.2. Using the method described in Section 4.2.2, the β value was estimated as 0.71 and the goodness of fit R^2 was equal to 0.98. This confirms that the popularity pattern for the Web objects is characterized by Zipf-like distribution.

Figure 4.2: Web object popularity pattern for UC3

4.2.3 Analysis of Request Arrivals

An analysis of the arrivals of requests recorded in a trace as a function of time is presented in this section. The first column of each trace is a time stamp showing the time at which a Web file is requested. For NLANR traces, it shows the time at which the client socket is closed [53]. Time series analysis of the traces is useful for finding the busy and quiet periods of access to the caches. Observing the cache behaviour during the simulation for bursty and quiet periods provides a good insight into system performance. Further analysis, such as finding long-range dependency in the request arrival patterns, is also possible with time series analysis. In this work only a graphical presentation of bursty and quiet periods of the traces is presented. Observing cache performance (by considering the cache hit ratio) during these periods is also discussed. Since UC3 captures the longest duration of the client accesses in the traces used in this research, time series analysis is presented only for this trace.

The time stamps of the traces show the cache reply times to the clients. To be able to analyze arrival times of client requests, another time series that captures the time of arrival of requests was generated. The time series for the arrival of requests were generated by subtracting the elapsed time (i.e., the time between opening and closing of a client socket [53]) from the time stamp for each entry of the UC3 trace. After that the entries in the UC3 trace were sorted based on the request arrival times. Figures 4.3, 4.4 and 4.5 show the results of request arrivals analysis. We found that these graphs and the graphs that correspond to request replies (generated by considering the time stamps in the

trace) were similar to each other. Therefore, we only present here the graphs that correspond to request arrivals.

Figure 4.3 presents the arrival counts for one minute intervals for UC3. This arrival count is the total number of request arrivals in one minute. This graph shows the arrival counts per minute from midnight of the day of which UC3 was recorded (March 12, 2000) to midnight of the next day. The average of the arrival counts is 133 per minute and the peak of arrival counts is 491 per minute (which happened around 6:00 a.m.) The visual observation of this graph indicates that from midnight until 6:00 a.m. the workload is relatively stable in comparison to the activity observed from 6.00 a.m. to 10.00 a.m. during which there are some bursty periods of request arrivals. From 10.00 a.m. to 4.00 p.m. request arrivals are less bursty and between 12:00 a.m. to 1:00 p.m. there is less activity. Around 6:00 p.m. and 8:00 p.m. there are also some peaks in the arrivals.

The arrival behaviour that is captured in Figure 4.3 shows that there are no significant idle periods within 24 hours. To see arrival counts for a longer time granularity, the request arrival counts for 30 minute intervals are also generated. The resulting graph is presented in Figure 4.4. The data in Figure 4.4 show that the average arrival counts tend to increase slowly throughout the day. The reason for not having any idle period in the examined NLANR trace compared to the traces from other proxies (described in [93] for example), throughout 24 hours, can be explained by the fact that

Figure 4.3: Request per interval time series for UC3 (interval: 1 minute)

other proxies plus regular clients connect to the NLANR caches. These proxies may be situated in different parts of the world with different time zones and can give rise to request arrivals all of the time within the 24 hour period. This observation agrees with the results of the performance study of Squid for root proxies such as NLANR in [93]. Rousskov *et al.* [93] report that since a root proxy serves international traffic for all time zones its activity is not correlated to the time of the day as observed in the other proxies.

Figure 4.5 shows the average hit ratios for 30 minute intervals when using a 1 GB cache and a LRU replacement policy. The hit ratio for each interval was calculated by dividing the number of hits by the number of arrivals for that interval. The interval hit ratio can be used to show how a cache behaves with different arrival rates. About 120

Figure 4.4: Request per interval time series for UC3 (interval: 30 minutes)

Figure 4.5: Periodic hit ratio of 1 GB cache size for UC3 (interval: 30 minutes)

minutes after midnight, there is a peak in arrival counts but a decrease in the hit ratio. Around 270 minutes after midnight, there is a peak in the hit ratio and a dip in arrival counts. At 6:00 a.m. there are peaks for both the hit ratios and request arrivals. After 5:30 p.m., although there are some fluctuations in arrival counts, the variation in hit ratio appears to be smaller. One of the possible reasons for the low hit ratio observed in the trace is that most of the users of the cache are other proxies rather than human users. These proxies may not need to re-reference the files because they cache the popular files locally.

Based on these observations it seems that there is no strict correlation between the hit ratio and request arrivals. For the trace used here when the number of client requests increases it does not always increase the hit ratio.

4.3 The Proposed Cache Management Strategy

In this section, we present the Web object-based strategies for cache management.

4.3.1 Web Object-Based Grouping

We used the Web object identification program discussed in Section 3.2, to identify the Web objects in our traces. As discussed earlier, a Web object, which consists of the Web page source file and all its related embedded objects, is considered as a group.

We processed the reduced traces to identify unique HTML Web files. The Web object identification program uses these unique HTML files to find all the Web objects for each trace. Table 4.1 (last column) shows the number of Web objects. Section 4.3.3 explains how the embedded objects can be determined on-line in a real system without *a priori* processing of Web objects. In our simulation, however, we had to run a Web object identification program in advance, because the tags were not available in the traces.

Our approach involves examining each Web page source file and finding its cacheable embedded objects. To find cacheable embedded objects, we used the same program that we used for Web object identification with a modification. We modified the program to identify only cacheable embedded objects belonging to a Web object. This means that when the program parses a Web page source file, it only considers the embedded objects whose URL names do not contain ‘cgi-bin’ or ‘?’ and do not have ‘.cgi’ file extension. Our method to identify a Web object is the source-based approach because it accesses the Web sites hosting a Web page and parses the source HTML Web file to identify its constituent embedded objects.

Thus, before performing a cache simulation with each reduced trace, we ran the Web object identification program that accessed the host sites to identify the Web objects. Since NLANR traces were recorded before the time of running the Web object identification program, the name of some embedded objects could have changed during this time interval. Also, as noted in Chapter 3, some of the embedded objects are not requested within the logs because of caching at a client site or an other level of proxy

caching. For example, the Web object identification program found, on average, 7 embedded objects for each Web object requested in UC3. By analyzing UC3, we found that, on average, 2.5 out of 7 embedded objects belonging to a Web object are requested in that trace. We excluded the embedded objects that were identified by the Web object identification program but were not present in the traces.

Note that a client requires all of the embedded objects during the initial access to a Web object. However, in a subsequent access to the same Web object by that client, it is possible that some of the embedded objects have already been cached by that client or other connected proxies. In that case, the implementation of our policy prefetches more embedded objects than are needed. We ignored this (i.e., prefetching of not requested embedded objects) in the simulation because we prefetch the embedded objects that were present in the used traces.

Similar trace-driven simulation studies such as [30] that suggest a prefetching method also made the same assumption. They only considered the files that were present in the trace for simulation purposes and ignored the effects of caching by the client or other proxies. The research presented in [39], that proposes an online prefetching, has suggested that by contacting the client, the online prefetching algorithm can identify the files that have already been cached and can therefore avoid prefetching them.

For all of the NLANR traces analyzed in this chapter, Web objects ended up, on average, with 2.5 embedded objects. Our method for finding embedded dependency is

different from the work described in [17] and [46]. In [17] and [46] a threshold of time was used to identify embedded dependencies. In this approach, if requests to two files generated by the same client in the log are consecutive and occur within a window of time less than the threshold, and are always requested together in the trace, then the files are said to exhibit an embedded dependency. Since the policy proposed by Hine and Wills [46] is implemented for our traces, we used their method for finding embedded dependent groups to compare with our results. This policy generated 619 embedded dependent groups with an average of approximately 3 members in each group for UC3. This policy generated embedded dependent groups with an average of approximately 3 and 4 members in each group for UC1 and UC2, respectively. The number of members in each group generated by this approach is comparable with our result, but the number of generated groups in this approach is much lower than the number of Web objects obtained by our program. We discuss this issue in more detail in Section 4.5.1, in which another trace was used for the investigation of Hine’s algorithm. Further discussion on trace-driven simulation is provided in Section 4.4.

4.3.2 Web Object-Based Cache Management Policy

In this section we introduce an approach for improving the performance of proxy caches. Our approach uses Web objects as units for fetching and replacement. In our proposed policy, we consider a reference to a member of a Web object as a reference to all the members of the Web object. If the currently referenced file is not in cache, a Web object-based policy fetches the referenced file. In addition, all the other members of the

Web object to which the requested file belongs are prefetched into the proxy cache. When file replacement is necessary to accommodate an incoming Web object, all the member files of one or more Web objects are removed. The number of Web objects to be removed depends on the size of the incoming Web object. Selection of a Web object for replacement depends on the cache replacement policy. In the WOLRU policy, the least recently used Web object is removed, whereas in the WOLFU policy the least frequently used Web object is replaced.

The WOLRU algorithm that we implemented in our simulation model is given in Figure 4.6. As in LRU, the algorithm uses a stack for managing the cache. The stack records the order in which Web objects are accessed. The most recently used Web object is at the stack top, whereas the least recently used Web object is at the bottom of the stack. In Figure 4.6, we use p for referring to a requested file. *Web object(p)* is the Web object to which p belongs. Each entry of the access log is a reference to a specific p . We used two data structures: *group* for keeping the information on Web objects (i.e., name and size of the related embedded objects) and *cache* for simulating proxy cache behaviour. As noted in the previous section, we ran the Web object identification program to generate information of the Web objects. This information is stored in the *group* before performing the simulation. During the simulation, for each request to p if it exists in the *cache*, *Web object(p)* is placed at the top of the stack. Otherwise *Web object(p)* is fetched from the *group* and placed in the cache. For each request to p , its size is read from the log and if p exists in the *cache*, this size is compared with the current size of p . If two sizes are different it means p has been changed since the last request to it. In

that case, the new size of p is written in the *group* and p is fetched from *group* and overwritten in the *cache*. This means *Web object*(p) is updated both in the *group* and in the *cache*. Since the current request is for p , *Web object*(p) is placed at the top of the stack.

For the WOLFU policy, we use a similar algorithm and data structures, except for the fact that Web objects are sorted based on their frequency of reference. The Web object with the highest frequency of reference is at the top and the one with the lowest frequency of reference is at the bottom of the stack. If two Web objects have the same frequency of references the most recently used Web object is placed in the higher position of stack. In the implementation of Web object- based policies, the proxy cache can hold the information related to Web objects. How and when the Web object information is integrated into proxy caches is discussed next.

WOLRU Algorithm:

for each request **do**

begin

p = document accessed in current request .

if p is already in *cache* ,

if size of p is changed

Update p and *Web object*(p) in *group*.

Evict the Web objects from the bottom of the stack if needed.

Bring p into *cache* and Update *Web object*(p).

Move *Web object*(p) to top of stack

else

while there is not enough room in *cache* for *Web object*(p),

Evict the Web object from the bottom of the stack.

Bring *Web object*(p) into *cache* and place at the top of stack.

end

end.

Figure 4.6: Web object-based LRU algorithm used by the simulation model

4.3.3 Implementation Implications

Two different possible implementation alternatives are discussed. The first does not need the server to recognize Web objects. The server in the second uses entire Web objects as units of transfer (discussed in the last paragraph of this section).

In a real system when using a Web object-based policy, for each request for file p , the proxy sends a request for p to the related server if p is not in the proxy cache. If p is the Web page source file, the Web object policy scans the Web page source file once it arrives from the server and uses the appropriate tags to recognize its embedded objects (for example, inline images). While parsing the page for each embedded object in the Web page, the proxy sends a request to the server for that embedded object. Thus, when the proxy sends the page to the client, all of the requests for its embedded objects have been sent to the server. A typical scenario is shown in Figure 4.7. If p is an embedded object, and it is not available in the cache, the proxy fetches it and considers it as a Web object with only one member. The proxy can maintain pointers for all of the members of a Web object. Note that according to our policy a file in a cache can only belong to one Web object, which means that each Web object has a unique set of embedded objects.

We assume that the proxy cache uses a faster machine than the client's machine. Thus, it can parse the page in a short time. However, it does not guarantee that when the client's requests for embedded objects of p arrive at the proxy, these are available in the proxy cache. There are many parameters that affect the availability of the embedded

1. Client sends request for p that corresponds to the main HTML Web page
2. Proxy receives the request for p and since it is not in cache, it sends a request to server
3. Server sends p to proxy
4. Proxy reads p and by identifying the tags, it sends requests for all the embedded objects of p
5. Proxy sends p to client
6. Embedded objects of p are sent to proxy and proxy places Web object(p) in its cache
7. Client requests embedded objects of p
8. Proxy sends embedded objects of p

Figure 4.7: Typical scenario for a Web object-based cache management strategy

objects in the proxy cache before the arrival of client requests. The parameters such as location of proxy, the network latency, the size of the Web object and bandwidths of the network between server and proxy and between proxy and the client can affect this availability.

When the location of the proxy is closer to the server than the client, it is more probable that the prefetched embedded objects are present in the proxy cache before the arrival of client requests for them. If the proxy location is closer to the client and client request for a particular embedded object reaches the proxy before the embedded object, the Web object-based policy still reduces the client latency because requests for

embedded objects were generated by the proxy earlier than the client. Note that the Web object-based policy prefetches the embedded objects that are not available in the proxy cache. Also some embedded objects belong to more than one page (this issue explained in more details in the next paragraph). Therefore, it is possible that some embedded objects requested by a client are present in the proxy cache and need not to be prefetched from the server. These embedded objects may belong to the other Web pages that are already requested by the other clients. In that situation even if the proxy location is close to the client a Web object based policy can reduce client latency.

As already noted, a Web object-based policy prefetches the embedded objects that may not be requested by the client. The number of such “not-requested” embedded objects depends on how many of them are already cached in the proxies closer to the client in the cache hierarchy and on the client’s machine. Some embedded objects appear on more than one page. For example, Arlitt *et al.* (page 44 of [10]) report that out of the Web pages that were studied by them, 90% of the embedded objects are used in 15 or fewer Web pages. Since an embedded object can be shared by more than one Web object, it may already be present in the proxy cache as a result of an earlier request to another Web object. Therefore, the total number of embedded objects that will be prefetched by a Web object-based policy will not necessarily be equal to the number of embedded objects in the Web object. However, some of the prefetched embedded objects may not be requested. To avoid prefetching such not-requested embedded objects, the proxy can communicate with its client (after step 4 in Figure 4.7) and avoid requesting the embedded objects that have already been cached by that client. In the hierarchy of proxy

caches this communication can be in the form of ICP messages that are used by commonly used proxies such as Squid (see Section 2.1.5 for the details).

Note that the number and size of the embedded objects affect the time required to retrieve all the embedded objects of a Web object. The fact that the Web page source file and its embedded objects are usually kept in the same server causes less communication delay to retrieve Web object members when the proxy prefetches them from the server.

Another approach to implementation of the Web object-based strategies is to send the Web page source file as well as the embedded objects as a single cluster from the server. This approach needs cooperation from the server. The proxy sends only one request corresponding to the source file of a Web object to the server. After receiving this request the server sends all the files for the Web object to the proxy. This implementation will drastically reduce latency when the proxy and server are far apart. This is because a number of separate sets of message exchanges, one for each member of a Web object is reduced to only one that corresponds to the entire Web object. Moreover, with this approach, the embedded objects will be available on the proxy when the client request for these arrive. This approach however, will require additional support from the server software.

Two approaches for implementation of the Web object-based strategies were discussed. The overhead associated with the first approach includes the time required for parsing each Web page source file by the proxy. The overhead of the second approach includes the time needed for identifying the Web objects by the server. Moreover, both

approaches give rise to an increase in network bandwidth that is discussed in Section 4.5.2.

4.4 Simulation

To evaluate the WOLRU and WOLFU policies, we used a trace-driven simulation using the NLANR UC1, UC2 and UC3 traces shown in the Table 4.1. For each trace, we first ran the Web object identification program discussed in Section 4.3.1 and stored the information related to the Web objects. The Web objects and the groups considered in the policy proposed by Hine and Wills [46] are considered to have unique members. After generating the groups, the trace-driven simulations were run.

During simulation, if any requested resource was unknown to our Web object-based policies or the policy proposed by Hine and Wills [46], it was considered as a group with only one member. An unknown resource is a file that is requested in a trace and cannot be associated with any Web object. For example, Audio/Video or compressed files that can be retrieved through the hyperlinks of the Web pages are considered as unknown resources. The same approach is used in [46].

For verification of the simulator, we used it with a small number of requests as input. We compared the hand-calculated result for that small number of requests and the simulation result and found that they agreed with each other. The performance metrics

captured by the simulation program and the parameters for the simulation model are presented next.

4.4.1 Simulation Parameters

Cache size: We experimented with a range of sizes: 16 MB, 64 MB, 250 MB, 1000 MB, 4000 MB and 64000 MB. The performance achieved with 64000 MB cache size is considered to be the performance expected from an infinite-sized cache because no file replacement occurred with the 64000 MB cache size.

Cache management policies: Both the WOLRU and the WOLFU strategies were simulated. To compare the Web object-based policies with file-based policies, a pure LRU policy and a pure LFU policy (explained in Section 2.1.6 of Chapter 2) were also simulated. Note that in case of LRU and LFU, only a single file is fetched and placed in the cache at a time. Since our proposed Web object-based policies use prefetching, a prefetching-based cache replacement policy that is proposed by Hine and Wills [46] was also simulated with our traces. We refer to this policy as Hine's algorithm. A brief explanation of Hine's algorithm is given here.

In this policy, a trace is analyzed *a priori* to identify groups of related resources. In this algorithm, two parameters are important for generating the groups. The first is a window of time parameter. All the consecutive requests from the same client to the same server generated within a time duration less than or equal to this window are said to

constitute a session. For generating the sessions, the requests in the trace were first sorted by destination and then by client ids. According to this policy a session is defined as a sequence of requests that are made by a single source to a specific destination. Since the trace was already sorted by time, by comparing the time of two consecutive requests in the sorted trace with the window time parameter, each session was identified. The end of each session was determined whenever the client name in each log request entry was not same as the previous one, or the time between two consecutive requests was more than the window time. The second parameter, P_t , is a threshold that shows the probability of requesting the files together in the sessions. For example, by setting P_t to 1 and with an appropriate window time one can generate the embedded dependent groups explained in Section 4.3.1. By using lower values for P_t , more files can be grouped together.

Hine's approach for identifying the groups is briefly presented (a more complete description is available in [46] pp. 17-19). Let a and b be two files in the trace that appeared together in N_{ab} sessions. Let N_a and N_b be the number of sessions that contain a and b respectively. a and b are allocated to the same group if:

$$N_{ab} / N_a \geq P_t \text{ and } N_{ab} / N_b \geq P_t$$

Different values of the time and probability threshold P_t were used in [46]. We have chosen 120 seconds for the time threshold and 0.1 for P_t in simulation. Hine and Wills [46] suggested that these values are likely to produce the best performance for their algorithm.

4.4.2 Performance Metrics

We considered two performance metrics: hit ratio and byte hit ratio. *Hit ratio* is the ratio of the number of cache hits and the total number of cache requests. A hit is said to have occurred when the referenced file is in the cache and its size has not changed since the last access to it. *Byte hit ratio* is the number of bytes that the proxy cache served directly by providing from its cache as a percent of the total number of bytes for all requests. It corresponds to the ratio of the sum of the sizes of the requested files found in the cache, and the sum of the sizes of all the requested files.

A hit is said to have occurred when the referenced file is in the cache. A cached file may have changed in content or size since the last access to it. Because in common-format proxy logs there is no data on file modification times, we ignore the fact that contents of cached files may have changed. But we can recognize if the size of a cached file has changed since the last access to it. Each time a requested file is read from the trace, the retrieved size can be compared with the current size of the file in the cache. If the two sizes are different it means that the cached file is not valid. Such a situation is considered to be a cache miss and this file is fetched into the cache.

For NLANR traces the size that is reported in the trace is the total number of bytes transferred from the proxy to the clients, which includes HTTP headers. Thus, for NLANR traces, the size can change when header size changes occur, or when clients abort the requests [53]. Therefore, a size change does not necessarily imply a change in

the file. Thus, as in [59] we ignored size changes and disabled the file size checking in our simulation program. The second “if” in the nested “if” statement in the algorithm of Figure 4.6 is not executed for NLANR traces. As in [59] we have set the size of a file equal to the maximum number of bytes transferred for that file recorded in the used traces.

We have measured the number of occurrences of files with the same name and different sizes in the used traces. It was less than 8% in all the NLANR traces used in this research. We also repeated the simulations for policies after enabling size checking in our simulator for the used NLANR traces. The performance improvements produced by the Web object based-policy were observed to be comparable to those obtained when size checking was disabled. In this thesis we report the results of our simulations, in which size checking was disabled for all of the policies and traces.

4.4.3 Simulation Warm-up Period

One important issue is the length of the simulation warm-up period. The length of the warm-up period for cache simulation is important because it should be selected properly in order to remove the transient state of the cache behaviour in the used traces. This method of simulation is known as the proper initialization, which requires the start of the simulation to be close to the expected steady state [54]. Note that during the initial state of simulation many cache misses occur because the cache is empty. In most simulations the statistics in the steady-state are of interest. However, identifying the end

of the transient state and start of steady-state of cache behaviour is not always easy. This difficulty makes all the transient removal methods to be heuristic methods [54].

Cache behaviour is expected to be stable (i.e., steady-state period) after the initial periods. In the steady-state period, the statistics of the initial conditions do not change the cache performance. In the simulation, for example, we do not want to add the misses that occurred during the warm-up period, to the total miss ratios.

Figures 4.8, 4.9 and 4.10 are used to identify the steady-state and transient state, based on the hit ratio for our traces. For each trace a cache simulation using LRU policy was conducted for different cache sizes. Note that 64 MB, 1 GB and 4 GB cache sizes correspond to an infinite cache for UC1, UC2 and UC3 respectively. To see the performance of the hit ratio throughout the traces, each trace was divided into several intervals. Dividing a trace into equal time intervals for finding the length of the warm-up period has already been used in [8]. Since the three traces have a different number of requests, different interval durations were used in order to visually observe the hit ratios for the traces. The selected intervals are 50, 100 and 1000 requests for UC1, UC2 and UC3 respectively. The hit ratio is referred to as a cumulative hit ratio because it is computed after adding the number of hits and the number of requests for an interval to the number of hits and number of requests of all the previous intervals. The cumulative hit ratio is plotted as a function of the interval for each trace. Finally, visual inspection was undertaken to determine a point where cumulative hit ratio shows a stable behaviour. Note that in all the graphs the lines are in the same order as the legends.

UC1: Figure 4.8 shows the result for UC1, which is the smallest trace. A visual inspection indicates that the cumulative hit ratio increases with the number of references, and starts to flatten off at the end of the trace. This behaviour indicates that this trace captures only the transient part of the cache behaviour. The small duration of the trace (12 minutes) is shorter than the time required to capture the re-references to the requested files presented in the cache. This is consistent with the large number of one-timers (see Table 4.2) and a lesser Zipf-like behaviour (see Table 4.3) observed for UC1.

UC2: Figure 4.9 shows a plot of cumulative hit ratios for UC2 as a function of the number of requests. UC2 is the second largest trace (81 minutes). By a visual inspection of the graphs for all cache sizes, it can be seen that the cumulative hit ratios start settling down after 3900 references. Note that although this trace is short, it has captured re-references to the requested files present in the cache.

UC3: This is the largest trace with a duration of 24 hours. Traces of comparable duration have been used in other studies of cache simulation (see [84] for example). Figure 4.10 shows the cumulative hit ratio for UC3. Initially the hit ratios can be seen to vary with the number of requests, but they tend to settle down as more requests are considered. From a visual inspection of the graphs for all the cache sizes, the hit ratios can be seen to start settling down after 49,000 requests. It means that the transient period ends after 49,000 requests.

Figure 4.8: Transient identification for UC1

Figure 4.9: Transient identification for UC2

Since UC3 is the longest trace, the cumulative hit ratio is also presented as the function of time in Figure 4.11. A similar plot is used in [71] for identifying the warm-up of cache simulation for the traces. Figure 4.11 presents a plot of the cumulative hit ratio for the 1 GB cache size. The time interval for computing the cumulative hit ratio is one minute. By comparing this graph with the graph presented in Figure 4.10 for 1 GB cache size, it is apparent that the steady state starts 490 minutes after the start of the simulation (i.e., 8:10 a.m.) that corresponds to 49,000 requests in UC3 trace.

Since UC1 and UC2 are small in comparison to UC3 with a substantial warm-up period, only simulation results for the entire traces are presented. The result for these two traces show the performance of Web object-based policies during short levels of time granularities (12 and 81 minutes) of cache behaviour.

For UC3, the largest trace, both the results of the simulation for the entire trace as well as the results for only the steady-state period are presented. Simulation with the whole trace is performed because the result includes the effects of the transient behaviour of the cache, which may be of interest in certain situations.

Figure 4.10: Transient identification for UC3

Figure 4.11: The variation of cumulative hit ratio with time for UC3

4.5 Results

4.5.1 Performance of the Web Object-Based Policies

The performance of Web object-based policies, file-based policies and Hine's algorithm are presented in Figures 4.12, 4.13, 4.14 and 4.15. Each figure consists of two graphs: one gives the hit ratio and the other the byte hit ratio for different cache sizes. Note that the graphs are in the same order as the legends in all the figures.

A comparison of hit ratios produced by WOLRU and LRU indicates that the hit ratio for WOLRU is approximately two times that for the LRU policy. To show the improvement more precisely, the increases in hit ratios are computed by subtracting the hit ratios of LRU policy from the hit ratios of WOLRU policy. Figures 4.12(a), 4.13(a), 4.14(a) and 4.15(a) indicate that WOLRU produced an increase in hit ratios in the range of 11% to 21% over LRU for all the three traces for various cache sizes. For example, the performance improvements of 11.95%, 21.8%, 19.73% and 19.46% in hit ratios are achieved by WOLRU over LRU for a 256 MB cache size, with UC1, UC2, UC3 (whole trace) and UC3 (steady-state) respectively.

The results show that the increases in byte hit ratio achieved by WOLRU are in the range of 6% to 8% for different cache sizes (see Figures 4.12(b), 4.13(b), 4.14(b) and 4.15(b)). The increases in byte hit ratios are computed by subtracting the byte hit ratios of

(a)

(b)

Figure 4.12: Performance comparison of cache management policies for UC1

(a) Hit ratio (b) Byte hit ratio

(a)

(b)

Figure 4.13: Performance comparison of cache management policies for UC2

(a) Hit ratio (b) Byte hit ratio

the LRU policy from the byte hit ratios of WOLRU policy. A comparison of hit ratios for WOLFU and LFU shows that an increase in hit ratios in the range of 11% to 20% for WOLFU. The corresponding increases in byte hit ratios range from 2% to 8% for all the three traces.

UC1 is the smallest trace for which all of the requested files can fit into a cache of size 64 MB. Thus, there is no cache replacement for the cache sizes equal to or greater than 64 MB. This causes the graph of each policy to flatten out when the cache sizes reach 64 MB.

Figure 4.13 shows that the graphs of each policy start to flatten off from cache sizes greater than 256 MB for the UC2 trace. This can be explained by the fact that the total size of requested files in the UC2 trace is less than 256 MB. Note that when there is no replacement, only the prefetching part of the Web object-based policies is responsible for the performance improvement.

Figure 4.14 presents the performance achieved for the whole UC3 trace whereas Figure 4.15 shows the steady-state performance for UC3. The steady-state results obtained by removing the transient component of UC3 are comparable to those obtained with the whole trace. The differences between the hit ratios and byte hit ratios for the steady-state and the whole trace simulation were less than 2% for all policies and all cache sizes. The effect of the transient part on system performance does not seem to be strong.

(a)

(b)

Figure 4.14: Performance comparison of cache management policies for UC3

(a) Hit ratio (b) Byte hit ratio

(a)

(b)

Figure 4.15: Performance comparison of cache management policies
for UC3 during steady-state (a) Hit ratio (b) Byte hit ratio

For the UC3 trace, the maximum achievable hit ratio (achieved with an infinite cache) when using LRU is 25.43%. By enabling the size checking of our simulation, this value is close to 18%. The hit ratio observed with LRU for the one day NLANR trace UC3 is comparable to the hit ratio obtained with the LRU policy from a larger NLANR trace (i.e., 21 days) that was used in [68].

The maximum observed hit ratio for LRU in [68] by the same definition of miss (i.e., the files with the same names and different sizes during simulation considered as cache misses) was close to 19%. Niclausse *et al.* [84] have also reported a maximum achievable hit ratio of 20% for another NLANR trace that was used by them. They have observed that NLANR traces produce a lower hit ratio with LRU compared to other proxy log traces. They noted that this is due to the fact that NLANR is at a higher level in the hierarchy of proxy caches.

Although the performance of LFU and LRU are comparable, LFU demonstrates slightly higher performance. A similar observation is also made in [12]. Since WOLRU demonstrates the best performance for all three different traces, we analyzed the cost associated with this policy, which is presented in Section 4.5.2.

A comparison of the graphs for all the traces indicates that using the Web object-based policies double the hit ratios of the file based policies for both small traces and a large trace (i.e., UC3). They also demonstrate a substantial performance improvement over Hine's algorithm. Note that the Web object-based strategies and Hine's algorithm

have two components: one for prefetching and the other for replacement. The file-based policies such as LRU and LFU do not have the prefetching component.

As indicated in Section 4.2.2 the smaller proportion of one-timers and more Zipf-like behaviour of UC2 and UC3 resulted in a better cache performance for these traces compared to UC1 (see the hit ratios of the traces with LRU policy in Figures 4.12, 4.13 and 4.14).

The increases in hit ratios achieved by Hine's algorithm over LRU are in the range of 1.48% to 4.50%. The corresponding increases in byte hit ratios range from 0.43% to 1.10%. The increases in the hit ratios observed with Hine's algorithm for our traces are lower than the 15%-20% increases in hit ratio reported by Hine and Wills in [46]. One of the possible reasons for this difference lies in the types of the traces used in the simulations reported in [46] and this work. NLANR is a busy access log, containing requests for many servers. For example, the third trace (UC3) contains requests sent to 9,583 servers. Hine and Wills [46] reported that in the trace used by them (Victoria University access log), most of the requests were sent to only four busy servers. Thus, their proxy log includes requests to a smaller number of servers in comparison to our proxy logs. By considering their algorithm for generating groups (see [46] pp. 17-19), it is more likely that resources that are requested from the same server have a higher probability of being related and therefore being accessed together. This is a possible reason for Hine's algorithm giving rise to a high performance improvement over LRU for

the trace in [46] compared to that observed with the NLANR traces. The results produced by Hine’s algorithm for UC1 and UC2 agree with the result achieved with the UC3 trace.

We have performed some additional experiments using Hine’s algorithm with different probability thresholds for another trace. We used an access log from the Boeing proxy cache that captures the requests that are generated for this proxy in one day. The trace is recorded on March 3, 1999 and consists of 201,913 requests sent to 5,682 unique servers. More details about this log are provided in Table 4.4.

Table 4.4: Characteristics of Boeing access log

Item	Value
Duration	One day
Date	March 3, 1999
Total no. of requests in reduced trace	131,205
Total no. of transferred bytes	1.1 GB
No. of distinct files	64,128
No. of one-timers files	54,496
No. of one-timers /total requests (%)	41.53
No. of one-timers/distinct files (%)	84.98

Unfortunately, the Boeing trace does not allow us to use our source-based method for finding embedded dependency, because URLs are encrypted for security reasons. Therefore, we only simulated Hine’s algorithm on this trace (see Table 4.5). The simulations were performed for an infinite cache size and for the entire trace. Techniques used for data reduction and all the assumptions, including the ignoring of size changes for

the same file, are the same as in the simulations that were performed for the NALNR traces.

Using different probability thresholds leads to generation of the groups with a different degree of dependency between the members. For example, using the probability threshold of 1 means that the members of the generated groups always have been accessed together in the trace. Using the probability threshold of 0.1 means that the members of the generated groups have a small degree of dependency with each other. Hine and Wills [46] suggested that by using the probability of 0.6 and window time of 120 seconds generates most of the groups that have the dependent members. Using the groups generated with a window time of 120 seconds and a probability threshold of 0.6 leads to marginal improvement in hit ratios. Generating more groups that have more members leads to better hit ratios. For example, the number of the generated groups for the Boeing access log with a probability threshold of 1 and 120 seconds is 1457 groups with an average of 4 members in each group. As noted in Section 4.3.1, using Hine's algorithm with the same parameters for UC3 generates 619 groups with an average of 3 members in each group. Using 120 seconds with a probability threshold of 0.1 generates 923 groups with an average of 4 members for the UC3 trace. The larger number of groups generated for the Boeing trace compared to the UC3 trace, causes more files to be prefetched. Thus, a greater improvement in hit ratios is obtained by using Hine's algorithm for the Boeing trace.

Table 4.5: Hit ratios for the Boeing trace when using Hine's algorithm with a window time of 120 seconds

LRU policy	Probability Threshold = 1	Probability Threshold = 0.6	Probability Threshold = 0.1
50.95%	56.52%	57.46%	58.79%

Table 4.5 indicates that the increases in hit ratios achieved by Hine's algorithm over LRU are in the range of 5.57% to 7.84%. The results show that Hine's algorithm performs better for the proxy traces, in which fewer unique servers are requested by the clients. It also shows that by reducing the probability threshold, the achieved hit ratio increases marginally.

Note that prefetching the embedded dependent groups (second column of Table 4.5) generated by Hine's algorithm provided close to 11% improvement in the hit ratio over LRU for this trace. The maximum achievable improvement in hit ratios achieved by Hine's algorithm when using the probability threshold of 0.1 is close to 16% over LRU for the Boeing trace. This indicates that the contribution of the prefetching of the embedded dependent groups to this maximum improvement that can be achieved by Hine's algorithm is 11% out of a total improvement of 16%. Since we showed that our method for generating embedded dependent groups is more efficient than Hine's algorithm, we expect that the WOLRU policy would generate a higher improvement in hit ratio for the Boeing access log.

4.5.2 Cost of the WOLRU Policy

Since WOLRU prefetches Web objects, it can potentially transfer more bytes from the server to proxy when compared to the LRU strategy. The cost of using WOLRU is captured by two parameters: traffic ratios and bandwidth increase. These values are computed for the 64 MB cache size (see Table 4.6).

For a given trace, the bandwidth used in transferring data between the server and the proxy is computed by dividing the total number of bytes transferred by the duration of the access log. The cost of WOLRU in terms of the bandwidth increase is presented in the last row of Table 4.6. The bandwidth increase is computed by subtracting the bandwidth used by LRU from that used by the WOLRU. The largest increase of 10.89 Kb/s, observed with the UC3 trace, is small in comparison to the capacity of the currently available networks. The computation of bandwidth increase is based on the assumption that data transfer is distributed uniformly during the entire duration of the log. The intensity of Web access in a real system is likely to vary. As a result, a period of high activity may demand a higher data transfer capacity from the network than the average we have computed here.

Traffic ratio reflects the increase in the number of bytes transferred from server to proxy due to the WOLRU policy. It is computed by dividing the number of bytes transferred by WOLRU, by that transferred by LRU. Thus, for the WOLRU policy and a 64 MB cache, the traffic ratios are 1.011, 1.009 and 1.028 for UC1, UC2 and UC3 traces

respectively (see Table 4.6). This indicates only a marginal increase in the network cost associated with the WOLRU policy.

Note that the cost of prefetching reported here may be optimistic for a real system. Our cost calculation is for the observed cost of prefetching in the simulation. As mentioned in Section 4.3.1, the embedded objects which were present in the trace have been used for prefetching. This means that the prefetched embedded objects are accessed at least once. In real systems it is possible that the cost of the embedded objects that are prefetched but not requested (because of caching by client or other proxies, discussed in Section 4.3.3) will be added to this cost. The prefetching cost reported by the trace-driven simulation of WOLRU policy occurs because some of the prefetched embedded objects are not accessed while they were in the cache.

Since the prefetching method used by WOLFU is the same as WOLRU, and the hit ratios produced by them are comparable, we expect that WOLFU demonstrates a cost characteristic that is similar to WOLRU.

Table 4.6: Data transfer characteristics for LRU and WOLRU with a 64 MB cache size

Caching Policy	Total Number of Bytes Transferred between Server and Proxy		
	UC1 trace	UC2 trace	UC3 trace
LRU	43.00 MB	248.50 MB	3.98 GB
WOLRU	43.49 MB	250.75 MB	4.10 GB
Traffic ratio	1.001	1.009	1.028
Bandwidth Increase	5.61 Kbits/sec	3.78 Kbits/sec	10.89 Kbits/sec

4.6 Discussion

We showed that by using a Web object-based policy we can gain better cache performance. The superiority of the Web object-based policy over file-based policies are observed for the traces with different duration (UC1, UC2 and UC3 for example). The smaller traces used in the simulation mostly consist of transient periods. Therefore, a Web object-based policy leads to a higher performance not only during steady state but also during the transient phase that corresponds to a proxy cache warm up. The reason for this improvement is the use of a Web object instead of a single file as a unit of transfer. A Web object-based policy has two components: prefetching of the Web objects and replacement of Web objects when the cache is full. Section 4.6.1 explains the effect of each of these two components on WOLRU performance. Section 4.6.2 provides a detailed analysis of miss hit ratios observed in WOLRU and LRU policies.

4.6.1 Web Object-Based Prefetching

The hit ratio obtained with a 64 GB cache size (infinite cache for our traces) is presented in Table 4.7. Since there is no removal of files from an infinite cache, this table shows the maximum achievable hit ratios for LRU and WOLRU. For the UC3 trace, for example, the hit ratio that can be generated by WOLRU is 19.57% higher than the hit ratio that can be achieved by LRU. This means that if we only prefetch the Web object members without replacing them, we can increase the hit ratio by approximately 20%. Note that the performance of LRU is equal to LFU for the infinite cache. For our three

traces when using LRU, the maximum achievable hit ratio varies from 10.87% to 25.43%. One of the reasons for the low hit ratio for LRU is the large proportion of one-timers in our traces. Reference to a one-timer is never a hit for LRU but sometimes is a hit for WOLRU. Table 4.2 shows that approximately 93% of the distinct files are one-timers in the used traces. Each of these one-timers generates a miss in the file-based policies. However, for the Web object-based policy because of prefetching, not all the one-timers lead to cache misses. Miss analysis for both file-based and Web object-based policies are presented in the next section. The maximum achievable hit ratios observed with the Web object-based strategies are approximately two times that observed with the LRU policy. For the PB trace, the improvement in hit ratios when using the Web object-based policy is also two times that achieved with the LRU policy (see the fourth column of Table 5.5 of Chapter 5).

Table 4.7: Hit ratios for the traces with an infinite cache

Caching policy	UC1	UC2	UC3
LRU	10.87%	22.71%	25.43%
WOLRU	22.30%	42.81%	45.00%

Table 4.7 demonstrates that even when there is no Web object replacement, the prefetching part of WOLRU still significantly improves the hit ratios of the proxy cache. The reason for this is as follows: in order to see a Web page, all of its embedded files should be requested and downloaded to the user's computer. Therefore, in most of the

cases, all or most of the members of a Web object are requested by the user. Thus, prefetching the members of a Web object increases the hit ratios.

The second part of the Web object-based policy is the replacement of the Web objects when the cache is full. To show how much improvement we can gain from the prefetching part, the cache replacement part of the WOLRU policy is modified. In this simulation, we changed the cache replacement part of the WOLRU policy and used a conventional file-based replacement instead of the Web object-based replacement in the cache. The prefetching part of the modified WOLRU policy works in the same way as before. We refer to this new policy as the WOLRU File based Replacement (WOLRU-FR). WOLRU-FR prefetches the related Web objects corresponding to a file from the server when that file is fetched from the server. Also WOLRU-FR uses LRU as the cache replacement policy and evicts files instead of Web objects from the cache when the cache is full.

Figure 4.16 presents the comparison between WOLRU and WOLRU-FR for the UC3 trace simulated for the whole trace. Figure 4.16 indicates that the hit ratios of WOLRU and WOLRU-FR are very close to each other. Thus, by using a file-based replacement instead of the Web object-based replacement we do not improve the hit ratio of WOLRU significantly. Our results (i.e., the hit ratio when using WOLRU instead of WOLRU-FR) show that the increase in hit ratio produced only by the Web object-based replacement component of WOLRU is marginal.

Figure 4.16: Comparison of file-based replacement with Web object-based replacement for UC3

In Chapter 5 we extend the use of the WOLRU for managing both the main memory and the disk of a proxy cache. In our proposed policy, by using Web object-based replacement and prefetching between disk and main memory, the cache hit latency can be decreased. Therefore, we prefer to keep the Web object-based replacement part of WOLRU. Section 5.5 discusses the cache hit latency reduction caused by the Web object-based replacement component of WOLRU policy.

4.6.3 Miss Analysis

Figure 4.16 indicates that prefetching is the main source of improvement in the hit ratios. Prefetching increases the hit ratio by changing some cache misses to cache hits. Cache misses are classified into cold misses and capacity misses. A cold miss occurs

when a requested file has never been seen by the proxy [8]. If a requested file has already been stored but evicted from the cache at a later time, the miss is a capacity miss. Therefore, the number of capacity misses depends on the cache size and the cache replacement policy. When a cache has an infinite size all of the misses are cold misses. Prefetching attempts to reduce the number of cold misses by predicting the files that will be requested in the future.

There is another kind of miss that is referred to as a consistency miss, which occurs when the document in the cache is stale [8]. In the simulation algorithm we identify a stale file by comparing its current size with its size in the previous access. However, because of inaccuracy in the reported sizes for NLANR traces (discussed in Section 4.4.1), we disabled the size checking part of our simulation algorithm for all of the policies and traces. This means that consistency misses in the cache simulation for NLANR traces are not considered.

Figure 4.17 (a) and Figure 4.17 (b) show the performance of LRU and WOLRU with a 256 MB cache simulated for the whole UC3 trace. Note that the graphs are in the same order as the legends in the figures. In these figures the graphs of cumulative miss ratios are shown as a function of the number of requests. The total miss ratio consists of cold miss and capacity miss ratios. For both Figure 4.17 (a) and Figure 4.17 (b) the cold miss ratio is the main contributor to the total miss ratio.

The large number of one-timers observed in NLANR traces is the reason for the large number of cold miss ratios. Figure 4.17 (a) and Figure 4.17 (b) show that cold miss ratios tend to decrease, whereas the capacity miss ratios tend to increase slightly with the number of requests in the traces. The re-referencing behaviour of popular files (discussed in Section 4.4.2) affects the number of capacity misses. For both Figure 4.17 (a) and Figure 4.17 (b), capacity miss ratios achieved with LRU and WOLRU are close to each other. Performing simulation with smaller cache sizes produces higher capacity misses.

Figure 4.17 shows that the main difference between the performance of LRU and WOLRU is the difference in cold miss ratios. The hit ratio shows the performance of two policies. Cold miss ratio is decreased by approximately 20%, and the same amount is added to the hit ratio for WOLRU in comparison to the LRU policy. The prefetching part of WOLRU converts some of the cold misses to hits. This is the main reason why it outperforms the LRU policy. Also, this can explain why Web object-based policies are effective for small traces such as UC1 that captures only the transient period.

In a transient period there are lots of cold misses that reduce the hit ratio of file-based policies, but the prefetching part of Web object-based policies convert some of them to hits. Thus, Web object-based policies can gain performance improvement during both steady-state and transient periods.

(a)

(b)

Figure 4.17: Miss analysis for a 256 MB cache size for UC3

(a) LRU policy (b) WOLRU policy

Table 4.8 shows the total miss ratios and capacity miss ratios for both of the WOLRU and LRU policies with all cache sizes simulated for only the steady-state period of the UC3 trace. The total miss ratio for a 4 GB cache size is equal to the cold miss ratio for each policy because there is no capacity miss with a 4 GB cache size. Note that the cold miss ratio for a given policy is independent of the cache size. The values of capacity miss ratio for both of the policies are close to each other. Thus, prefetching part of WOLRU is the main source of performance improvement for both UC3 (steady-state) and UC3 (whole trace).

Table 4.8: Miss analysis for UC3 (steady-state)

Cache Size	WOLRU		LRU	
	total miss ratio (%)	Capacity miss Ratio (%)	Total miss ratio (%)	capacity miss ratio (%)
16 MB	63.79	8.78	83.32	9.07
64 MB	61.14	6.14	80.65	6.40
256 MB	58.93	3.91	78.39	4.13
1 GB	56.72	1.71	76.05	1.80
4 GB	55.01	0	74.25	0

4.7 Summary

This chapter has proposed a new cache management approach based on Web objects. The strategy has two components: one for prefetching and one for replacement.

Based on a trace-driven simulation, we observe that for a number of different traces used in this chapter and different cache sizes, two Web object-based policies (i.e., WOLRU and WOLFU) outperform the conventional file-based (i.e., LRU and LFU) strategies that do not have a prefetching component, both in terms of hit ratio and byte hit ratio, by a substantial margin.

The prefetching part of the Web object-based policy is the main reason for improving cache performance. By converting some of the cold misses to hits, the prefetching part of Web object-based policies produces a higher hit ratio in comparison to the file based policies. Compared to other existing prefetching techniques, Web object-based prefetching has the advantage of prefetching files that are most likely to be referenced. This is because all the embedded files in a Web page are to be fetched when the Web page source file is referenced. Thus the performance of the Web object-based strategies is expected to be directly proportional to the average number of embedded objects per page. The Web object-based strategies are also observed to outperform another file group based strategy from the literature. One of the reasons for this superiority is that the source-based identification of file groups in our policy is more effective than the window-based approach used in the other strategy.

The Web object-based strategies presented in this chapter can be implemented on a real system. The Web object-based policy does not need *a priori* analysis of proxy logs as required by the window-based strategy reported in [46]. We analyzed the cost of using WOLRU, by measuring the increase in network traffic between the proxy cache and the

server in comparison to its file-based counterpart LRU. This cost calculation is the observed cost of prefetching in the simulation. The observed increase in traffic ratio and bandwidth usage is not a significant problem for current networks.

Chapter 5

Web Object-Based Storage Management for Proxy Caches

5.1 Introduction

Disk delay is a part of the hit response time, which is the delay perceived by a client when a proxy serves hit documents. Rousskov *et al.* [93] added extra code to Squid [98] running at a number of proxies, in order to measure the disk delay contribution to the hit response time. They report that the disk delay contributes about 30% to the total hit response time. They concluded that the disk storage subsystem may become a potential bottleneck in proxies. Note that reducing the hit response time of a proxy can reduce the latency that is perceived by the client. Section 3.4.2 of Chapter 3 describes how the results on Web object characterization are useful to reduce the disk response time in proxy servers.

Section 5.2 presents the proposed storage management policy and Section 5.3 discusses the proposed technique to reduce the disk latency in proxy caching. Section 5.4 presents a discussion on the cache replacement policy component of the Web object policy. Section 5.5 explains the reduction in cache hit latency by using the Web object-based storage management. A summary of this chapter is given in Section 5.6.

5.2 Web Object-Based Storage Management Policy

Most of the common proxies use main memory together with the disk as a cache. In this work storage means both main memory and the disk of a proxy. The storage management policy presented in this chapter is an extension of the Web object-based caching strategy discussed in the last chapter. This storage management has two components. Each of them is explained next.

Web object-based storage management uses the Web object as a unit for replacement and prefetching between the disk and main memory. The purpose of the prefetching part employed by this policy is to provide more documents from the proxy main memory instead of its disk to reduce the hit response time, and is discussed in more details in Section 5.2.1. Web object-based storage management also has a second component in which a Web object-based approach is used for file-level clustering. This component is referred to as a Web object-based file placement policy and is discussed in Section 5.3. Grouping of related files into clusters is a well-known technique. In Section 5.5 we investigate whether or not using Web object-based file placement can improve the performance of the proxy cache. The results of the simulations indicate that most of the performance improvement accrues from the use of Web object-based prefetching and replacement component. For example, approximately 50% reduction in cache hit latency is produced by this component whereas only 5% additional performance improvement results from the Web object-based file placement component.

Section 4.3.3 explains how we can implement a Web object-based policy for proxy caching. Since our Web object-based storage management policy manages the internal storage of a proxy cache, it acts in the similar way as the Web object-based policy discussed in Section 4.3.3. A typical scenario, shown in Figure 4.7, is used to explain the behaviour of Web object-based storage management policy. Figure 5.1 illustrates this scenario indicating file transfers between main memory and the disk of the proxy cache when implementing the Web object-based storage management strategy. The Web object based replacement and prefetching is discussed next.

Figure 5.1: Typical scenario of Web object-based storage management policy

The overhead of this Web object-based storage management policy is similar to the overhead of the Web object-based policy that was discussed in Section 4.3.3. Since Web-object based replacement and prefetching increase the file transfers between disk and main memory, there is an additional overhead that is discussed in Section 5.2.4.

5.2.1 Web Object-Based Replacement and Prefetching

A document that is provided by the commonly used proxy applications such as Squid or Apache [7] may exist either in the main memory or the disk. The first step to reduce the latency of hit documents is to try to provide more documents from the main memory rather than the disk of the proxy cache. For this reason, Web object-based storage management tries to efficiently manage the documents that exist in the proxy disk. In the proposed policy, we use Web objects instead of files as units of fetching and replacement in the proxy disk system.

If a proxy cache serves a hit document from the disk, that document together with its related Web object members are brought from the disk to the main memory. Therefore, the following requests for the Web object members are served from the main memory instead of the disk. If the main memory is full, Web object-based replacement policy creates free space in the main memory. A replacement policy such as LRU is used to determine eviction of Web objects from the main memory. When a Web object is evicted from the main memory it is transferred to the disk if a copy of it does not exist on the disk. In transferring a document from main memory to disk and vice versa Web object-based storage management transfers all the members of the related web object. To study

the performance of this policy, we conducted several trace-driven simulation experiments. More details on the policy and trace-driven simulations are presented next.

5.2.2 Simulation

The simulation programs are modifications of the cache management simulation programs that we used in Chapter 4. A description of how the programs can simulate a disk-based proxy is given next.

When a proxy receives a file or a Web object, it is stored in the main memory of that proxy. We have chosen this scenario for the simulation instead of a scenario in which the proxy writes the received file directly to the disk. The reason for this selection is the behaviour of the common proxy cache applications such as Squid and Apache. Squid keeps a part of the proxy main memory for the hot objects. Squid considers the hot objects as the most recently used files. Therefore, whenever a file is read from the server or a file is hit on the proxy disk, Squid places that file in the main memory [97]. Also Apache can be configured to use both memory and disk as the cache [7]. Therefore, it can be configured to write the receiving file first into the main memory cache. The size of the main memory is adjustable for both Squid and Apache.

If we need to free space in the main memory, files or Web objects are transferred from the main memory to the disk if they are not already stored on the disk. We limited the disk writes to a situation in which memory becomes full to be able to compare the

number of disk writes for the examined caching policies with variable main memory sizes. In a real system, the file transfer to disk can be performed before the main memory becomes full, or the proxy can perform asynchronous write operations while retrieving files from the server to improve system efficiency. Note that the LRU policy uses files, and the WOLRU policy uses Web objects, as units of transfer. If the files or Web objects exist on the disk, their sizes are compared when transferring the files or Web objects from the main memory to disk. The existing files or Web objects are overwritten only if the sizes of the incoming and existing entities are different (which indicates that the files or Web objects were modified when reallocated in the main memory). A modified file is one whose copy on the disk or main memory of the proxy is stale because it has changed in the server. When a file is changed in a server, the proxy fetches it from the server upon the client's request. As mentioned in Section 4.4.1, we did not consider the size change as an indication of file modification for the PB and UC3 traces.

To discover if a requested file is available in the proxy, a search for that file is undertaken in both the main memory and disk. If the requested file exists in the main memory, the number of hits on the main memory is incremented. Otherwise, if the requested file exists on the disk, the number of disk hits is incremented and a Web object containing that file is brought into the main memory. In our simulation model, the most recently used files are always placed in the main memory. Therefore, if main memory becomes full, least recently used files or Web objects are transferred from the main memory to the disk to make room for the incoming file or Web object. In other situations, the program acts according to the Web object-based policy, as explained in Chapter 4.

For example, if a requested file does not exist both on the disk and the main memory of the proxy, the request is considered to have generated a miss and the Web object corresponding to the file is prefetched from the server. When the file or Web object is received from the server it is placed in the proxy main memory.

Cache management policies: Both LRU and WOLRU strategies are modified to simulate disk-based caching (see the previous description of the simulation).

Traces: Trace-driven simulation uses reduced UC3 and PB NLANR traces (see Table 4.1 for details). We used our Web object identification program to identify Web objects in the reduced UC3 and PB traces (see Section 4.3 for details). As noted previously, these Web objects consist of unique and cacheable embedded objects that have appeared in the traces.

Performance metrics: We considered two performance metrics to compare the LRU and WOLRU disk-based caching policies: memory hit ratio and disk hit ratio. Memory hit ratio (MR) for a policy is computed by dividing the number of memory hits (MH) by the total number of hits generated by the policy (TH). Similarly, disk hit ratio (DR) is computed by dividing the number of disk hits (DH) by the total number of hits achieved (TH) by the policy.

$$MR = MH / TH \quad (5.1)$$

$$DR = DH / TH \quad (5.2)$$

It is clear that the sum of memory hit ratio and disk hit ratio is equal to one. The goal of the Web object-based policy is to increase the memory hit ratio. A higher memory hit ratio means a better cache performance.

Cache sizes: The cache consists of both main memory and disk in our simulation. Experiments were performed with a number of different main memory sizes: 16 MB, 64 MB, 256 MB, 1 GB, 4 GB. We assume that the disk size is more than total size of requested files in each trace. Since the total size of requested files in each trace is less than 4 GB this assumption is realistic. Therefore, we do not have to evict any files or Web objects from the disk. Thus, we can focus on the transfers between main memory and disk.

5.2.3 Experimental Results

Figure 5.2 shows the memory hit ratios for WOLRU and LRU for the UC3 and PB traces. It can be seen from this figure that WOLRU offers significant performance gains for the lower memory sizes. As the memory size increases, the difference between the performance of the two strategies decreases because most of the hit documents are provided from the main memory. In particular, for memory sizes greater than or equal to 4 GB, the replacement policy does not make any difference, as all the files are present in main memory. Cache replacement policies make a difference when there is contention for main memory.

(a)

(b)

Figure 5.2: Memory hit ratios of the WOLRU and LRU policies

(a) PB trace (b) UC3 trace

Simulations were performed for UC3 and PB, and for each of them statistics were collected for the entire trace. Note that the observed differences between memory hit ratios for WOLRU and LRU result from two different types of prefetching performed by the WOLRU policy. The first is the prefetching of files when a miss has occurred. In this case, WOLRU prefetches the embedded objects of the requested Web object from the server. Since the prefetched embedded objects are placed in the main memory, the subsequent requests for them are satisfied by the proxy main memory. This increases the number of memory hits.

The second type of prefetching that increases the number of memory hits for WOLRU policy is prefetching of the Web object members from the disk to the main memory when one of its members is hit on disk. The main memory serves the subsequent requests for the members of the prefetched Web object. The improvement in memory hit ratio of WOLRU over that of LRU is the sum of the memory hits achieved by these two types of prefetching. For example, when the memory size is 64 MB, the WOLRU policy provides a memory hit ratio improvement of 12.5% for the UC3 trace and nearly 20% for the PB trace over the LRU policy.

Figure 5.2 indicates that for small memory sizes, the performance differences between LRU and WOLRU are more than that observed for the large memory sizes for both traces. One reason for this difference is the greater number of memory hits that the LRU policy can achieve for the larger memory sizes. When memory size is large more popular files can be kept by LRU in the main memory. In addition, when WOLRU uses

the larger memory sizes it can keep more popular files in the main memory that causes fewer disk hits, and, as a result, fewer files are prefetched from disk to main memory. This decreases the improvement caused by the second type of prefetching (i.e., between disk and main memory) in WOLRU.

A comparison of Figure 5.2 (a) and Figure 5.2 (b) shows that the memory hit ratio of LRU for PB is less than the memory hit ratio achieved for UC3, specially for small memory sizes (i.e., 16 MB and 64 MB). The reason can be explained by considering the workload characteristics of each trace. Table 4.2 of Chapter 4 shows that the number of requests to one-timers of total requests in the PB trace is 10% more than that in the UC3 trace. Table 4.3 also shows that the popularity pattern for UC3 is more Zipf-like compared to PB. Both of these characteristics affect the re-referencing patterns (i.e., temporal locality) of the files in each trace. Therefore, more re-referencing of files stored in memory can be the reason for a higher memory hit ratio observed for UC3 compared to PB, when using LRU with small memory sizes.

5.2.4 Disk Traffic

Since WOLRU prefetches members of Web objects from the disk to main memory, and moves Web objects from the main memory to disk, it may transfer a higher number of files in comparison to LRU. To analyze the traffic between disk and main memory we measured the sum of the number of transferred files between disk to main memory and main memory to disk for both WOLRU and LRU policies. The results are presented in Figure 5.3.

(a)

(b)

Figure 5.3: Disk traffic for the WOLRU and LRU policies (a) PB trace (b) UC3 trace

The traffic between the disk and main memory is referred to as disk traffic. As shown in Figure 5.3, an increase in the memory size results in a decrease in the disk traffic for both policies. When the main memory size is 4 GB, all the requested files in each trace can be placed in the main memory. Thus, there is no need to transfer files or Web objects between the main memory and disk for a 4 GB main memory size. We define traffic ratio as the ratio between the number of files transferred by WOLRU and that transferred by LRU. The traffic ratios are in the range of 1.02 to 1.08 for different memory sizes for both UC3 and PB traces. We can see that WOLRU does not increase the disk traffic substantially. Figure 5.4 shows the percentage increase in disk traffic due to WOLRU (over LRU). It can be seen from this figure that the increases are less than 6% when the main memory size is equal to or greater than 64 MB. Even with a 16 MB memory, the increase in the disk traffic is less than 9% for both UC3 and PB traces.

Figure 5.4: Disk traffic increase due to WOLRU policy (in comparison to LRU)

5.3 Web Object-Based File Placement Policy

Placement of files plays an important role in determining the disk access time. In this section, we investigate the relationship between file placement and the performance of a proxy cache. This section begins with an introduction in order to provide the background information. The details of Web object-based file placement policy and the proposed clustering method are presented in Section 5.3.2. Sections 5.3.3 and 5.3.4 discuss simulations and results, respectively.

5.3.1 Introduction

This section provides an overview of file placement and clustering methods that are used in proxy applications and disk performance evaluation.

5.3.1.1 Overview of the Storage of Files

Most of the proxy caches run on systems with Unix or similar operating systems. The operating system usually provides an interface for writing and reading data on disk. The physical disk consists of cylinders. Each cylinder is divided into tracks. A track is divided further into sectors. One or more sectors form a logical block [94]. Data transfers between the main memory and disk are in the units of logical blocks. The size of a logical block is usually 512 bytes or larger, although the disk can be formatted to have different logical block sizes [94]. The total time for accessing a file consists of the time to move

the head to the right track (seek time), the time to find a correct sector (rotational delay), and the time to transfer data (transfer time). Disk seek time contributes more to the total delay of accessing the files, especially when files are not stored in contiguous blocks. Thus, if we know which files are accessed together and if we store them in contiguous blocks, we can reduce the disk seek times. As a result, we can decrease the average disk response time. The file system is responsible for storing a file on disk. Usually a file system provides two levels of clustering that are presented next.

Block-level clustering: At this level, the question is how a single file should be stored. For example, a single file can be stored in contiguous blocks (i.e., contiguous allocation), or as a linked list of blocks. Other techniques that include index allocation are described in [95].

File-level clustering: This is a higher level of clustering and concerns the clustering of files into groups and placing the group members close to each other in the disk. The group members are usually the files that are accessed together.

For example, the Fast File System (FFS, a Unix file system) uses the cylinder grouping technique to provide both block-level and file-level clustering. In the cylinder grouping technique, users or applications have to place the related files into a directory. The files of the directory are allocated in one or more consecutive cylinders to reduce disk seek time. In the cylinder grouping technique, files belonging to a directory are

stored on consecutive blocks on disk(s). With the same approach, FFS also tries to store a single file in consecutive disk blocks [74] and [95].

5.3.1.2 Using File-level Clustering by Proxy Applications

Some proxy applications have used the consecutive allocation of the file members of a “locality set” that was defined by these applications. For example, in Hummingbird the unit of the disk access is a cluster, in which the members of a locality set are stored in the consecutive disk blocks. As noted in Section 2.2.3, it is not clear how Hummingbird determines each locality set.

Squid and Apache use the Unix standard file system UFS, a descendant of the 4.2 BSD Unix Fast File System (FFS) [74]. FFS and UFS both use the cylinder grouping technique to store the files on disk. In a Web server, a directory is created by a user to store the related files on the disk. In the proxy server, the cache applications (such as Squid or Apache) create the directories when writing the files on disk [96]. Some researchers report that when Squid or Apache sends the locality set information to the FFS to create the directories, these directories are not efficient. For example, Squid tries to balance the directories, but it destroys the locality sets. Apache maps the files with the same origin server to the same directory, which causes some directories to have many entries. The cylinder grouping technique used by FFS does not always guarantee consecutive block allocation of the files belonging to the same directory. Consecutive blocks are used only if a set of consecutive blocks of required size is available. Thus,

even storing a single file on the consecutive disk blocks is not guaranteed in FFS [96] and [70].

5.3.1.3 Overview of Disk Drive Performance Evaluation

Secondary storage performance is usually measured by disk drive behaviour. A disk drive contains a set of rotating platters. The upper and lower surfaces of each platter are covered with magnetic media. Data are written in circular tracks on each surface and are written in units of sectors. The disk arm holds a set of read/write heads that can be located in a requested address that consists of specific cylinder, surface and sector. Current disk drives have other components such as an intermediate controller, buses, device drivers, request schedulers, and disk block caches that can affect the total disk response time.

Researchers mostly use traces that capture the workloads of a real system for evaluating the performance of storage devices. Such a trace can be a log that captures the requests from a storage system activity or a trace produced by a synthetic disk request generator [42]. Another reason for the interest in disk drive behaviour is the examination of system level policies such as disk scheduling algorithms [110], or application level policies that change disk I/O requests. For all of these reasons, a comprehensive disk storage model that captures all of the details of disk drive attributes is needed.

5.3.2 Contiguous Allocation of Web Object Members

The Web object characterization presented in Chapter 3 shows that embedded objects belonging to Web object members are accessed together most of the time. Thus, if we store files of a Web object on contiguous disk blocks, we can reduce the disk response time. As discussed in the previous section clustering related files and placing them near one another on disk has been used earlier to improve the disk performance. Whether or not the members of a Web object can be used as a file cluster for improving performance needs investigation. To the best of our knowledge the answer to this question is not available in the literature. To test this idea, we performed a number of simulation experiments that are described in Section 5.3.3.

In particular, we want to know how much additional improvement can be achieved by incorporating consecutive allocation of the Web object members (i.e., our definition of locality set) in WOLRU. In the simulation, therefore, consecutive allocation of the Web object members is compared with the random allocation of Web object members on disk. The justification for the choice of using random allocation as a basis of comparison is presented next. Note that in both of these types of file allocations, a single file is stored in the contiguous blocks. The benefit of such a simulation for a real system is explained next.

As noted in Section 2.2.3, the Web object-based file placement policy is a part of our proposed caching management strategy that can be used by a proxy. However, if we

want to incorporate Web object-based file placement in the Web object-based storage management policy, we should be able to send and receive the information of the related files (i.e., Web object members) to/from the underlying file system.

When the Web object-based policy uses a file system such as FFS in the best possible scenario, Web object members will end up in consecutive disk blocks. If the operating system used by the Web object-based policy does not use such a file system or uses a file system with a different file-level clustering strategy, in the worst case the files of a given Web object will end up as far as possible from each other. Thus, assuming an average “random” placement of Web object members is a reasonable choice for the simulation experiments used in this investigation.

To summarize, the main benefit of doing this simulation is to determine the effectiveness of Web object-based file placement (i.e., contiguous allocation of Web object members on disk). If it is effective it can be incorporated in a file system used by caching application.

5.3.3 Simulation

We performed a trace-driven simulation, but this time we focused on disk performance. We evaluated disk performance with the help of a disk simulator, called DiskSim [35]. A brief overview of DiskSim used in the simulation is presented next.

Disk Simulator: DiskSim was developed at the University of Michigan to support research work on various aspects of the storage subsystem architecture. It includes modules to simulate disks, intermediate controllers, buses, device drivers, request schedulers, disk block caches and disk array data organizations. DiskSim can be used as a part of a comprehensive system model, such as an operating system simulator, or as a standalone subsystem. It has been validated against 10 different disk models from 5 different manufacturers [35]. It allows simulation with an I/O trace as the input and provides disk performance parameters such as a disk average response time as output.

To run DiskSim, we had to configure it to simulate a specific disk type [35]. We configured DiskSim to simulate three disk types. The simulation of each of these three disk types has been validated against the real disk drives (Table 5.1 shows the basic parameters of these disk types) by the developers of DiskSim. Each disk accepts an ASCII format trace as an input. The default parameter files for simulating each of these disk types are provided by the developers of DiskSim. We used the latest version of DiskSim (DiskSim2.0) for the simulations. By using a parameter file for the simulated disk type and an I/O input trace, DiskSim performs the simulation. At the end of simulation, DiskSim generates an output file that gives the statistics on the performance of the simulated disk.

Table 5.1: Basic parameters for simulated disk drives

Disk	Model Number	Data Surfaces	RPM
HP2249A	C2249A	13	5,400
HP2490A	C2490A	18	6,400
Seagate Elite	ST41601N	17	5,400

Trace: The I/O input trace for DiskSim can have different formats. We used the ASCII input trace format. For an ASCII input trace, the default format of each request entry consists of “request arrival time, device number (i.e., the storage component that is accessed by the request), block number, request size and a request flag that indicates disk read or write. As in the simulations described in Section 5.2.2, we used the UC3 and PB traces. We wrote two programs to generate the I/O input traces in the ASCII format from these access logs. Details about these programs are given next.

Programs: First, we modified the program that was used for the trace-driven simulation in Section 5.2.2. The modified program simulates a Web object-based storage management policy and generates the appropriate disk operations for each caching policy. Thus, the program takes the UC3 or PB NLANR trace as input and generates a high-level trace that captures each disk read and write operation. Each line of the resulting trace contains a disk read or write event, the time of the event that is read from the trace, the size and the name of the file that is read from or written to the disk, and the Web object information for that file.

We wrote two additional file placement programs, both of which use the high-level trace as the input and generate the ASCII format traces that can be accepted as input by DiskSim. The first program generates a trace, in which all the members of a Web object are stored in consecutive blocks. The second program generates a trace, in which the members of a Web object are stored on the same device but in random block positions. Other characteristics of the two programs are similar and are summarized below:

- Device numbers are generated randomly.
- Each file is stored on consecutive blocks.
- The size of a block is assumed to be 512 bytes.

Simulation Parameters: We performed the simulations with the same range of main memory cache sizes that were used in the simulation described in Section 5.2.2. We configured DiskSim to perform the simulation for three disks whose basic parameters are shown in Table 5.1. For each disk, we used the default parameter files that are suggested by DiskSim [35].

Performance metrics: We used average disk response time as the performance metric. It is the mean response time for a disk request [35]. In the other words, for each disk request, such as read or write, disk response time indicates the total time that spent by the components of the simulated disk to serve this request.

5.3.4 Results

The performance obtained with the placement of Web object member files on consecutive blocks is compared with that of a random placement. For each simulated disk, simulations were run to measure the disk response times for all the main memory sizes and traces.

Figure 5.5 shows the average disk response times for the WOLRU policy when the Web objects file members are stored in consecutive and random blocks for the HP 2490 and the Seagate disk systems. The corresponding results for the PB trace are presented in Figure 5.6. Note that the total size of the files that belong to the UC3 or the PB traces is less than 4 GB. This means that no files are read from the disk when main memory size is 4 GB. For this reason the 4 GB memory size is not used in this simulation.

These results show that the average disk response time can be reduced by approximately 1 to 4 msec for the simulated disks. For example, when the memory size is 256 MB, we noticed a reduction of 10.31% in disk response time for the HP 2490 disk when consecutive placement strategy is used. The reduction in disk response time for the Seagate disk is 11%. Note that this is the only reduction in the disk response time that is a part of the hit latency due to consecutive allocation of Web object members on disk blocks. WOLRU has already achieved a major reduction in the hit latency by providing more documents from the main memory instead of the disk.

Note that for reducing clutter in the graphs we have presented the data for only two disk types in Figures 5.5 and 5.6. A similar performance improvement was obtained by consecutive block allocation of the Web object members on HP 2249 disk.

Figure 5.7 shows the average disk response time of HP 2249 disk for the PB and UC3 traces. A comparison of the average response times achieved with the simulated disks indicates that HP 2490 produces a lower response time compared to the other disks. This is possibly because HP 2490 is a newer disk drive [35].

It is interesting to observe that, irrespective of a reduction in the number of disk operations, the mean disk response time increases with an increase in memory size (see Figures 5.5, 5.6 and 5.7). We observed this phenomenon for all our simulated disk types and traces. In WOLRU and LRU policies, queuing delay increases when the memory size increases (see third column of Tables 5.2 and 5.3). One possible reason for this increase in the queuing delay could be because of burstiness in the disk access pattern with increasing memory sizes [42].

Figure 5.5: Impact of consecutive and random file assignment on disk response time for the WOLRU policy and the UC3 trace

Figure 5.6: Impact of consecutive and random file placement on disk response time for the WOLRU policy and the PB trace

(a)

(b)

Figure 5.7: Impact of consecutive and random file placement on disk response times of the WOLRU policy for HP2249 disk type (a) UC3 trace (b) PB trace

Table 5.2: Queuing characteristics observed with LRU policy and PB trace for HP2490 disk

Main memory size	Average response time (ms)	Average queuing delay (ms)	Number of disk idle periods	Total number of disk operations
16 MB	25.82	9.70	165,929	199,674
64 MB	31.36	15.18	155,745	190,048
256 MB	37.59	21.63	142,098	175,890
1 GB	48.90	32.52	96,613	121,606

Table 5.3: Queuing characteristics observed with WOLRU policy (consecutive allocation) and PB trace for HP2490 disk

Main memory size	Average response time (ms)	Average queuing delay (ms)	Number of disk idle periods	Total number of disk operations
16 MB	17.55	3.44	202,308	217,039
64 MB	19.11	4.72	186,720	200,880
256 MB	20.05	5.48	169,255	182,516
1 GB	23.55	8.44	116,293	125,787

To examine the effects of consecutive block allocations of Web object members on disk performance, we performed more experiments for PB trace. Figure 5.8 presents the disk seek time results. Figure 5.8 indicates that the allocation of Web object members on consecutive blocks reduces disk seek times by 1.3 msec for the HP disk and by 2 msec for the Seagate disk. This means that the reduction in the average disk seek time for both the disks is 20% on average.

Figure 5.8: Impact of consecutive and random file assignment on the disk seek time of the WOLRU policy for the PB trace

Figure 5.9 shows the number of disk read and write operations in the WOLRU storage management policy for different main memory sizes (for the PB trace). As expected, the total number of disk reads and writes decreases as memory size increases. Figure 5.9 also indicates that the number of disk write operations is much higher than the number of disk read operations for all of the main memory sizes for the WOLRU disk

based policy. At the beginning of the simulation, when the cache is not full there are no read or write operations on the disk. When the replacement policy starts to make free space in the main memory, it transfers files to the disk. Transferring files from main memory to disk means performing disk write operations. Although a file is written on disk during the simulation if it is not already present there, the large numbers of the disk writes show that in most cases the files are transferred from main memory to the disk by the WOLRU policy. Note that the large numbers of the disk writes are not the result of the modification of the files. As noted in Section 4.4.1, because of the presence of other bytes in the file sizes reported by NLANR traces, we did not consider size change as an indication of the file modification in the simulation for the PB and the UC3 trace. Therefore, a file is written to the disk only if it is not already present on disk.

Figure 5.9: Number of disk accesses for the PB trace

The main reason for the large difference between number of write and read operations is the presence of a large number of one-timers in the PB trace: 80% of the total number of requests (see Table 4.2). One-timers are initially placed in the main memory and later when the memory becomes full, they are written on the disk. However, they are never read again from the disk.

Another reason for the large difference is that write operations are performed more than read operations in the simulation. In the case of a miss, when the main memory is full there are write operations on disk to make free space for the incoming file or Web object from the server. Disk read operations are needed only when a file is hit on the disk. In this case, its related Web object is transferred to the main memory. Note that, if the main memory is full, when a disk hit occurs, there will be disk writes because of file transfers from memory to disk (if a copy of the transferred file is not already present on disk).

To summarize, since the write operations are needed both in case of a miss or a disk hit, and disk reads are undertaken only when there is a disk hit (file is not one-timer), there are more write operations than read operations. Table 5.4 presents more results on the disk performance for the different allocation methods of the Web object file members for PB trace. This table shows that the average time of disk read requests (4th column) and disk write requests (5th column) decrease as a result of using the contiguous file allocation policies.

Table 5.4: Comparison of file placement policies for the PB trace

Policy	Disk Type	Main Memory	Disk Read time Avg. (ms)	Disk Write time Avg. (ms)	Disk Res. Time Avg. (ms)
WOLRU	Seagate	64MB	59.09	28.44	31.58
WOLRU , Consecutive	Seagate	64MB	55.71	25.96	28.99
WOLRU	HP 2490	64MB	39.95	18.68	20.85
WOLRU , Consecutive	HP 2490	64MB	36.00	17.18	19.11
WOLRU	Seagate	256MB	79.07	33.39	36.33
WOLRU , Consecutive	Seagate	256MB	53.66	30.86	32.33
WOLRU	HP 2490	256MB	53.12	20.24	22.36
WOLRU , Consecutive	HP 2490	256MB	36.21	18.94	20.05

5.4 Discussion

Section 4.6 provided a discussion of the prefetching component of WOLRU. It concluded that the prefetching component of WOLRU is the main contributor to the improvement in the hit ratio. This conclusion was the result of analyzing a variation of WOLRU policy (called WOLRU-FR), in which a file-based replacement was used instead of the Web object-based replacement. In this section we discuss a Web object-based replacement component of the WOLRU storage management policy. To focus only on the replacement component of the Web object-based storage management policy prefetching between the server and the proxy is eliminated. In this approach, the proxy only identifies the Web object members but does not prefetch them. We modified the program that was used for the simulation in Section 5.2.2. The modified WOLRU policy is called No-Prefetching-WOLRU (NP-WOLRU). This approach can be used to measure

the improvement that Web object-based policy gains by only prefetching the Web object members from disk to the main memory.

To evaluate the improvement that can be obtained by using this approach, we performed a trace-driven simulation using the UC3 and PB traces as input. We modified the program that was used for the simulation in Section 5.2.2. In the modified program, we excluded the prefetching of Web object members from the server to the proxy.

We used the same parameters and performance metrics used in Section 5.2.2 in this simulation. The performance of the NP-WOLRU policy is presented here. Figure 5.10 shows the increase in the memory hit ratios. When using the NP-WOLRU instead of LRU, prefetching the members of a Web object from the disk to main memory caused an improvement in memory hit ratios. The increase in memory hit ratios led to the reduction in disk hit ratios. For example, Figure 5.10 (a) shows that for UC3 trace when we use NP-WOLRU instead of LRU with 256 MB of main memory, the reduction in disk hit ratio (i.e., the increase in memory hit ratio) is 12.5%. Figure 5.11 presents the disk response time and disk seek time as a result of using consecutive and random block allocation of the Web object members in NP-WOLRU policy.

Figure 5.11 (b) presents the disk seek time results. This figure indicates that for a main memory size of 256 MB, allocation of Web object members on consecutive blocks, reduces disk seek time by 1.35 msec for the HP disk and the PB trace. This means that the reduction in the average disk seek time is 20% when using the NP-WOLRU policy

(a)

(b)

Figure 5.10: Memory hit ratios of the NP-WOLRU and LRU policies
(a) UC3 trace (b) PB trace

Figure 5.11: Impact of consecutive and random file allocation on disk response times of the NP-WOLRU policy for HP 2490 disk and PB trace
 (a) Average disk response time (b) Average disk seek time

with consecutive block allocation for the Web object members. Cache hit latency achieved with this policy is presented in the next section.

5.5 Impact on Proxy Cache Hit Latency

We define cache hit latency as the delay for serving a hit document by a proxy cache. Cache hit latency is just a part of the total latency that is perceived by a client (i.e., hit response time) when the proxy serves the hit document. As noted in Section 2.1.7, the hit response time consists of the time for reading the client request after she/he has connected to the proxy (i.e., the delay from an `accept()` system call on proxy to the receiving of the client HTTP request), the time for sending the document to the client, and proxy cache hit latency [93]. Here we only concentrate on cache hit latency. Since the proxy cache uses both disk and main memory, cache hit latency can be calculated by adding the latency of hit documents that are provided by the main memory and latency of the hit documents that are provided by the proxy disk. We assume that the latency for a document provided from the main memory is negligible compared to that provided from the disk. Thus, to compute the cache hit latency, we need to know the disk response time for the documents which are provided from the disk. Under this assumption, the average cache hit latency (HL) for a proxy can be computed by multiplying the disk hit ratio (DR), which can be computed by using Equation 5.2, by the average disk response time (DT).

$$HL = DR \times DT \quad (5.3)$$

In order to determine the reduction in the cache hit latency, the results for the PB and UC3 traces (when simulating HP 2490 and Seagate disks with a 256 MB main

memory) are presented in Table 5.5. WOLRU with consecutive block allocation of Web object members provides the most reduction in cache hit latency in comparison with the other policies.

For example, for the UC3 trace on the Seagate disk, the average latency of the hit documents, given by Equation 5.3, is 5.39 msec and 2.72 msec for LRU and WOLRU, respectively (see the last column of Table 5.5). Thus, WOLRU reduces the latency for hit documents by 50% for the UC3 NLANR trace. Note that these values are smaller compared to the client expected hit latency on a real system. These values include only memory and disk latencies. These values do not include the network latencies that are incurred in receiving the client request and in transferring the requested documents from the proxy cache to the client.

By using Equation 5.3, the average cache hit latency for WOLRU is computed as 2.45 msec when the Web object members are stored in consecutive blocks for the UC3 trace (see the last column of Table 5.5). Thus, by using WOLRU and consecutive placement, the hit latency for the UC3 trace is reduced by 55% from the hit latency observed with LRU.

The results for the UC3 trace suggest that, of the 55% reduction we obtained for the cache latency for the hit documents, a reduction of 50% is achieved by using WOLRU instead of LRU, and 5% of the reduction was due to the consecutive file placement policy. When using NP-WOLRU (explained in Section 5.4) and contiguous

file allocation, instead of LRU, the reduction in cache hit latency is 22%, out of which 4% results from the allocation of the Web object members in the consecutive disk blocks.

Table 5.5: Performance comparison of policies with 256MB of main memory

Policy	Trace	Disk type	Cache Hit Ratio	Disk Hit Ratio	Avg. Disk Res. Time (ms)	Avg. Cache Hit Latency (ms)
LRU	PB	HP 2490	13.86%	15%	37.59	5.64
NP-WOLRU	PB	HP 2490	13.86%	13.5 %	22.04	2.98
NP-WOLRU, Consecutive	PB	HP 2490	13.86%	13.5%	20.95	2.83
WOLRU	PB	HP 2490	31.92%	7%	22.36	1.6
WOLRU, Consecutive	PB	HP 2490	31.92%	7%	20.05	1.4
LRU	UC3	Seagate	22.10%	13%	41.43	5.39
NP-WOLRU	UC3	Seagate	22.10%	11.5%	38.71	4.45
NP-WOLRU, Consecutive	UC3	Seagate	22.10%	11.5%	36.65	4.21
WOLRU	UC3	Seagate	41.84%	7%	38.9	2.72
WOLRU , Consecutive	UC3	Seagate	41.84%	7%	35.05	2.45

For the PB trace, the reduction in cache hit latency is 75% by using WOLRU (with contiguous allocation of Web object members) compared to LRU. 4% of the reduction is because of the contiguous allocation of Web object members. When using

NP-WOLRU and contiguous file allocation the reduction of cache hit latency is 49% (compared to LRU), out of which approximately 3% is due to the allocation of the Web object members in consecutive disk blocks.

5.6 Summary

In this chapter we investigated the Web object-based policy (WOLRU) for improving the performance of proxy cache storage management. By using the WOLRU storage management policy we can improve proxy cache performance by providing more of the hit documents from the main memory, as opposed to the disk. The reason for this improvement is that most of the members of a Web object are accessed together. Thus, when a member of a Web object that is on the disk is requested, WOLRU prefetches the rest of the members of that Web object from disk to main memory. Therefore, follow-up requests for the other members of the same Web object are satisfied from the main memory, which increases the proxy memory hit ratio.

We investigated a No-prefetching WOLRU (NP-WOLRU) policy, in which there is no prefetching between the server and the proxy. Analyzing NP-WOLRU showed that even if we eliminate prefetching between the server and the proxy, this approach is still able to reduce the cache latency for the hit documents. For example, when the size of proxy main memory is 256MB, NP-WOLRU obtained a reduction of 22% in cache hit latency for the UC3 trace. The results show that 18% of this reduction is the result of prefetching of the Web object members from the disk to main memory when one of its

members is hit on disk. And 4% of the reduction results from the allocation of the Web object members in the consecutive disk blocks.

The results of trace-driven simulations show that WOLRU achieves reduction in cache hit latency in comparison to LRU. For example, when the size of the proxy main memory is 256 MB, a reduction of 50% in cache hit latency is achieved for the UC3 trace. Our results also show that we can achieve more reduction in disk response time by allocating Web object members on consecutive disk blocks. Although such an allocation reduces the average disk seek time by approximately 20% (observed for all the memory sizes and simulated disks for PB trace), but additional reduction in cache hit latency by incorporating this method, is only modest. For example, for the UC3 trace, when using 256 MB of main memory, we obtained an additional 5% reduction in cache hit latency by storing the Web object members on consecutive blocks of the proxy disk.

These results show that the main reduction in the cache hit latency results from the Web object-based replacement and prefetching between the disk and main memory. Therefore, if this policy does not use contiguous allocation of Web object members on disk, it can still reduce the cache hit latency significantly.

Chapter 6

Conclusions

6.1 Web Object-Based Proxy Cache Management

Proxy caches are essential for improving system performance and enhancing user satisfaction in the World Wide Web. Appropriate cache management strategies are crucial for achieving these goals. This research has proposed a new cache management approach based on exploitation of Web document structure. Our policy takes the Web document structure into account by using the Web object as a unit for system management. A Web object, consisting of the Web page source file and all of its constituent embedded objects, is used as a unit for prefetching and replacement in our policy.

Chapter 3 presents a study on Web object characteristics. Web object sizes as well as the number and size of embedded objects in Web objects were measured. The data on Web object sizes presented in the chapter show 90% of the Web objects are less than 100 KB in size. To the best of our knowledge, there is no similar work on measuring the size of Web objects. Most of the embedded objects are observed to be small: 90% of them are less than or equal to 10 KB. This data on the size of Web objects and the number of embedded objects per Web object indicate that Web objects can be used on the proxy cache as a unit of replacement and prefetching to improve performance.

Several trace-driven simulations were performed and are described in Chapter 4. Based on the results, we observe that for a number of different traces and cache sizes, two Web object-based policies, WOLRU and WOLFU, outperform the conventional file-based LRU and LFU strategies respectively, both in terms of hit ratio and byte hit ratio by substantial margins.

The Web object-based strategies have two components: one for Web object prefetching and the other for Web object replacement. The prefetching part of the Web object-based policy is observed to be the main source of performance improvement in the simulation experiment described in Chapter 4. The prefetched embedded objects are required by a user to be able to see the Web page. One of the reasons for this improvement in performance is the presence of a large number of one-timers. By prefetching Web object members, the Web object-based strategies effectively convert some of the cold misses occurring with a file-based strategy such as LRU or LFU into hits.

The Web object-based strategies are also observed to outperform another file group-based strategy called Hine's algorithm for a number of traces and cache sizes. One of the reasons for this superiority is the source-based identification of file groups in our policies. The source-based method generates a more accurate group of related files compared to the window-based approach used in the other strategy. Moreover, the Web object-based strategies presented in this research can be implemented on a real system

and do not need an *a priori* analysis of proxy logs as required in most of the window-based prefetching strategies.

In addition to the performance improvements, this research also presents the cost associated with one of the Web object-based policies, WOLRU. Because of prefetching, the Web object-based strategy gives rise to an increase in traffic between the proxy cache and the server in comparison to its file-based counterpart, LRU. For the traces used, the concomitant increase in traffic ratio and bandwidth usage is observed to be an insignificant problem for current networks.

6.2 Web Object-Based Storage Management

Chapter 5 presented an investigation of Web object-based techniques for improving the performance of proxy cache storage systems. Two techniques were proposed for performance improvement of a proxy cache consisting of a two level storage hierarchy: a main memory and a hard disk. In the first technique, a cluster of files that contains the Web page source file and the associated embedded files, called a Web object, is used as a unit for replacement and prefetching between the disk and main memory. Whenever a requested file is not in the main memory, if the request to that file has generated a hit on proxy disk, all the files in the corresponding Web object are brought into the main memory from the disk. This reduces the number of future reads on the disk for the other files that belong to the same Web object residing on the disk. When file replacement is necessary for bringing in a Web object, one or more clusters of files,

each of which corresponds to a Web object, are replaced. The second technique attempts to reduce disk response time by placing the files that belong to the same Web object on consecutive disk blocks.

Using trace-driven simulations, we have analyzed the performance of these storage management techniques for proxy caches. Traces collected from real proxy caches were used in our simulation. Results indicate that for the hit documents that are provided by the proxy cache, a 50% reduction in cache hit latency, can be achieved by using the Web object-based WOLRU in comparison to the conventional LRU replacement policy. By incorporating contiguous allocation of Web object members on the disk an additional 5% reduction in cache latency can be produced when the proxy serves the hit documents.

6.3 Thesis Contributions

The contributions of this thesis are summarized below:

Characterization of Web Objects

- One distinctive feature of this study is our consideration of the *Web object* as a unit for Web document structure characterization and system management. Most existing works do not discuss the structure and size of Web objects. A source-based measurement technique is used in the Web object characterization.

- Chapter 3 identifies a number of Web object characteristics for popular Web documents and suggests distribution models for these characteristics. Weibull and lognormal distributions appeared to be suitable for modeling the distributions of Web object characteristics in popular Web documents analyzed in this thesis.

Proxy Cache Management

- Two new cache management strategies were introduced. The Web object-based policies WOLRU and WOLFU are simple and implementable on real proxy servers.
- Based on trace-driven simulations, Chapter 4 demonstrates that by using these Web object-based strategies a significant performance improvement in hit ratio can be achieved in comparison to conventional cache replacement strategies that are file-based. For example, results of our simulation experiments with NLANR traces show up to 100% improvement in hit ratio in comparison to the conventional file-based cache replacement policies (see Section 4.5.1).
- Chapter 4 also indicates that the main source of achieved improvement in hit ratio is the prefetching part of the Web object-based policy. The prefetching method employed by the Web object-based policy is more efficient than the prefetching method used by a similar prefetching-based policy [46].

Chapter 5 proposed two techniques to improve the storage system performance of proxy caches. Both techniques are based on the concept of Web objects.

- In the first technique, a Web object is used as a unit for replacement and prefetching between the disk and main memory. Our simulation results show that the prefetching part gives rise to a substantial reduction in cache hit latency.
- The second technique uses Web object information to place the files in a Web object contiguously so that the disk access time can be reduced. The simulation results presented in Chapter 5 indicate that only a marginal reduction in cache hit latency is obtained from this technique.

6.4 Future Work

Possible directions for future research are outlined:

- We examined the traces that belong to the NLANR cache, a top level proxy cache in caching hierarchies. As mentioned in Section 2.1.5, the hit ratios for the proxies that are located at a higher level of a cache hierarchy is less than the proxies that are located in the lower level of the hierarchy [91]. This thesis focused on the use of Web object-based policies for proxy caches. It is also possible to use a Web object-based policy for the management of Web server caches as well. Thus, evaluating the performance of a

Web object-based policy for different levels of caching within the network and Web servers is useful.

- Although we identified the Web document characteristics for both static and dynamic pages in this thesis, we examined Web object-based policies for caching static files only. Recent research concerning the improvement of cache coherency suggests that caching of dynamic files can also be useful. Current cache methods prevent caching of the dynamic files because of the frequent changes of these files that require many contacts between caches and servers. The new cache coherency methods, such as server-driven consistency (see Section 2.4.1), enable caches to serve the dynamic cached files without contacting the servers. By using Web object-based policies (instead of the file-based policies such as LRU) in combination with these coherency methods, it may be possible to obtain improvement on cache performance even for dynamic files.
- This thesis investigated the performance improvement of using a Web object-based policy for proxy servers. The overall reduction in latency experienced by a user is not evaluated in this work. As mentioned in Section 2.1.7, in addition to the cache replacement policy there are other factors that impact the overall user latency. A more detailed simulation to evaluate the reduction in this overall user latency produced by the Web object-based policy forms an important direction for future research.
- One of the directions for future research is the use of synthetic Web proxy workload generators to evaluate the sensitivity of the performance of a Web object-based policy

to the changes in characteristics such as temporal locality, popularity and proportion of one-timers.

References

- [1] 100 hot.com, Related address is: <http://www.100hot.com/home.shtml>

- [2] A. Abhari, S. P. Dandamudi and S. Majumdar, "Using Web Objects to Improve the Performance of Proxy Caching," *In Proceedings of the Fourth International Workshop on Web Engineering at the World Wide Web WWW10 Conference*, pp. 82-92, Hong Kong, May 2001. Available at:
<http://acims.uws.edu.au/webe2001/webe-www10-proc.pdf>

- [3] A. Abhari, S. P. Dandamudi and S. Majumdar, "Structural Characterization of Popular Web Documents," *International Journal of Computers and Their Applications*, Volume 9, Number 1, pp. 15-24, March 2002.

- [4] A. Abhari, S. P. Dandamudi and S. Majumdar, "Characterization of Web Objects in Popular Web Documents," *In Proceedings of ISCA 13th International Conference on Parallel and Distributed Computing Systems*, pp. 616-624, Las Vegas, Nevada, USA, August 2000.

- [5] A. Abhari, S. P. Dandamudi and S. Majumdar, "Exploiting Web Document Structure to Improve Storage Management in Proxy Caches," *In Proceedings of 9th International Conference on High Performance Computing*, pp. 89-101, Bangalore, India, December 2002.

- [6] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, "Caching Proxies: Limitations and Potentials," *In Proceedings of the Fourth World Wide Web Conference '95: The Web Revolution*, pp. 119-133, Boston, MA, December 1995.

- [7] Apache Software Foundation. Apache http server project.
<http://www.apache.org/httpd.html>

- [8] M. Arlitt, *A Performance Study of Internet Web Servers*, M.Sc. Thesis, Department of Computer Science, University of Saskatchewan, June 1996.

- [9] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches," *ACM SIGMETRICS Performance Evaluation Review*, Volume 27, Number 4, pp. 3-11, March 2000.

- [10] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," *IEEE Network*, Volume 14, Number 3, pp. 30-37, May/June 2000. Extended version available as Technical Report HPL-1999-35R1 at:
<http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html>

- [11] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants," *In Proceedings of the ACM SIGMETRICS '96 Conference*, pp. 126-137, Philadelphia, PA, April 1996.

- [12] M. Arlitt and C. Williamson, "Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers," *The Society for Computer Simulation Journal*, Volume 68, Number 1, pp. 23-33, January 1997.
- [13] ArrowPoint, "Content Smart Cache Switch," White paper on the ArrowPoint, Web site is: <http://www.arrowpoint.com/solutions>
- [14] G. Banga, F. Douglass and M. Rabinovich, "Optimistic Deltas for WWW Latency Reduction," *In Proceedings of 1997 USENIX Technical Conferences*, pp. 289-303, Anaheim, CA, January 1997. Also Available at:
<http://www.research.att.com/~douglass/papers/optdel.ps.gz>
- [15] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *In Proceedings of ACM SIGMETRICS '98 Conference*, pp.151-160, Madison, Wisconsin, June 1998.
- [16] P. Barford and M. E. Crovella, "A Performance Evaluation of Hyper Text Transfer Protocols," *In Proceedings of ACM SIGMETRICS '99 Conference*, pp. 188-197, Atlanta, Georgia, May 1999.

- [17] A. Bestavros, "Using Speculation to Reduce Server Load and Service Time on the WWW," *In Proceedings of the Fourth ACM Conference on Information and Knowledge Management*, pp. 403-410, Baltimore, Maryland, November 1995.
- [18] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber and M. F. Schwartz, *Harvest: A Scalable, Customizable Discovery and Access System*, Technical Report CU-CS-732-95, University of Colorado, Department of Computer Science, Boulder, Colorado, March 1995.
- [19] T. Bray, "Measuring the Web," *In Proceedings of 5th International World Wide Web Conference*, Paris, France, 1996.
- [20] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", *In Proceedings of IEEE'99 INFOCOM*, pp. 126-134, New York, NY, March, 1999.
- [21] R. Caceres, F. Douglass, A. Feldmann, G. Glass, M. Rabinovich, "Web Proxy Caching: The Devil is in the Details," *1998 Workshop on Internet Server Performance*, pp. 111-118, Madison, WI, June 1998.
- [22] P. Cao and S. Irani, "Cost-aware WWW Proxy Caching Algorithms," *In Proceedings of USENIX Symposium on Internet Technology and Systems (USITS)*, pp. 193-206, Monterey, CA, December 1997.

- [23] P. Cao, J. Zhang and K. Beach, "Active Cache: Caching Dynamic Contents on the Web," *In Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware98)*, pp. 373-388, The Lake District, England, September 1998.
- [24] J. Challenger, P. B. Danzig and A. Iyengar, *A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites*, Technical Report RC21337, IBM Research, T. J. Watson Research Center, 1998.
- [25] A. Chankhunthod, P. B. Danzig, C. Neerdals, M. F. Schwartz and K. J. Worrell, "A Hierarchical Internet Object Cache," *In Proceedings of the USENIX Technical Conference*, pp. 153-163, San Diego, CA, January 1996.
- [26] C. Chiang, M. T. Liu and M. E. Muller, "Caching Neighborhood Protocol: A Foundation for Building Dynamic Web Caching Hierarchies with Proxy Servers," *In Proceedings of International Conference on Parallel Processing (ICPP'99)*, pp. 516-524, Aizu, Japan, September 1999.
- [27] C. Chiang, M. T. Liu and M. E. Muller, "On Request Forwarding for the Dynamic Web Caching Hierarchies," *In Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, pp. 262-269, Taipei, Taiwan, July 2000.

- [28] C. Chiang, M. Ueno, M. T. Liu and M. E. Muller, *Modeling Web Caching Hierarchy Schemes for Performance Studies*, Technical Report OSU-CISRC-6/99-TR17, Department of Computer and Information Science, The Ohio State University, July 1999.
- [29] K. Chinen and S. Yamaguchi, "An Interactive Prefetching Proxy Server for Improvement of WWW Latency," *In Proceedings of the Seventh Annual Conference of the Internet Society (INET 97)*, Kuala Lumpur, Malaysia, June 1997. Available at: http://www.isoc.org/inet97/proceedings/A1/A1_3.HTM
- [30] E. Cohen, B. Krishnamurthy, and J. Rexford, "Efficient Algorithms for Predicting Requests to Web Servers," *In Proceedings of the IEEE INFOCOM*, pp. 284-293, New York, NY, March 1999.
- [31] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transaction on Networking*, Volume 5, Number 6, pp. 835 – 846, December 1997.
- [32] M. E. Crovella and L. Lipsky, "Long-lasting Transient Condition in Simulations with Heavy Tailed Workloads," *In Proceedings of the 1997 Winter Simulation Conference (WSC'97)*, pp. 1005-1012, Atlanta, GA, December 1997.

- [33] M. E. Crovella and M. S. Taqqu, "Estimating the Heavy Tailed Index From Scaling Properties," *Methodology and Computing in Applied Probability*, Volume 1, Number 1, 1999.

- [34] S. Deng, "Empirical Model of WWW Document Arrivals at Access Link," *In Proceedings of the 1996 IEEE International Conference on Communication (ICC'96)*, pp. 1797-1802, Dallas, TX, June 1996.

- [35] DiskSim, Available at: <http://www.ece.cmu.edu/~ganger/disksim>.

- [36] F. Douglass, A. Haro, M. Rabinovich, "HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching," *In Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, pp. 83-94, Monterey, California, December 1997.

- [37] A. B. Downey, "The Structural Cause of File Size Distributions," *In Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 328-339, Cincinnati, Ohio, August 2001.

- [38] A. B. Downey, "Evidence for Long-Tailed Distributions in the Internet," *Internet Measurement Workshop (ACM SIGCOMM)*, pp. 229-241, San Francisco, CA, November 2001.

- [39] D. Duchamp, "Prefetching Hyperlinks," *In Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, pp. 127-138, Boulder, CO, October 1999.

- [40] P. Embrechts, C. Klueppelberg and T. Mikosch, *Modelling Extremal Events for Insurance and Finance*, pp. 330-34, Springer, New York, 1997.

- [41] E. Gabber, L. Huang, E. Shriver and C. Stein, "Storage Management for Web Proxies," *In Proceedings of 2001 USENIX Annual Technical Conference*, pp. 203-216, Boston, MA, June 2001.

- [42] G. R. Ganger, "Generating Representative Synthetic Workloads: An Unsolved Problem," *In Proceedings of the Computer Measurement Group (CMG) Conference*, pp. 1263-1269, December 1995.

- [43] S. Glassman, "A Caching Relay for the World Wide Web," *In Proceedings of the First International World Wide Web Conference*, pp. 69-76, Geneva, May 1994.

- [44] S. D. Gribble and Eric A. Brewer. "System Design Issues for Internet Middleware Services: Deductions From a Large Client Trace," *In Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, pp. 207-218, Boulder, CO, December 1997.

- [45] J. Gwertzman, "Autonomous Replication in Wide-Area Internetworks," B.A. Thesis, Center for Research in Computing Technology, Harvard University, Cambridge MA, April 1995.
- [46] J. H. Hine, C. E. Wills, A. Martel and J. Sommers, "Combining Client Knowledge and Resource Dependencies for Improving World Wide Web Performance," *In Proceedings of INET'98 Conference*, Geneva, Switzerland, July 1998. The details and the proposed policy are available in Martel's MSc thesis at:
<http://www.mcs.vuw.ac.nz/~hine/Papers/Martel/index.html>
- [47] V. Holmedahl, B. Smith and T. Yang, "Cooperative Caching of Dynamic Contents on a Distributed Web Server," *In Proceedings of 7th IEEE International Symposium on High Performance Distributed Computing (HPDC-7)*, pp. 243-250, Chicago, IL, USA, July 1998.
- [48] B. C. Housel and D. B. Lindquist, "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment," *In Proceedings of Second Annual International Conferences on Mobile Computing and Networking*, pp. 108-116, Rye, New York, November 1996.
- [49] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. "Scale and Performance in a Distributed File

System,” *ACM Transactions on Computer Systems*, pp. 51-81, Volume 6, Number 1, February 1988.

- [50] Y. Hu, A. Nanda and Q. Yang, “Measurement, Analysis and Performance Improvement of Apache Web Server,” *In Proceedings of the 18th IEEE International Performance, Computing and Communications Conference (IPCCC'99)*, Phoenix/Scottsdale, Arizona, February 1999.
- [51] ICP, The related address is: <http://icp.ircache.net/rfc2186.txt>
- [52] A. Iyengar, M. Squillante and L. Zhang, “Analysis and Characterization of Large-scale Web Server Access Patterns and Performance,” *In Proceedings of World Wide Web Conference*,” pp. 85-100, Volume 2, Number 1, June 1999.
- [53] IRCACHE, For details see <http://www.ircache.net>.
- [54] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley & Sons, Inc., New York, NY, April 1991.
- [55] S. Jin and A. Bestavros, “GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams,” *In Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon,

Portugal, May 2000. *Also appeared in International Journal on Computer Communications*, pp. 174-183, Volume 24, Number 2, February 2001.

- [56] S. Jin and A. Bestavros, "Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms," *In Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, pp. 254-261, Taipei, Taiwan, April 2000.
- [57] S. Jin and A. Bestavros, "Temporal Locality in Web Reference Streams: Sources, Characteristics, and Caching Implications (Extended abstract)," *In Proceedings of ACM Sigmetrics 2000*, Santa Clara, CA, June 2000. The complete version is available as Computer Science Technical Report BUCS1999-009, Boston University.
- [58] M. Kazar, B. Leverrett, O. Anderson, V. Apostolides, B. Bottos, S. Chutani, C. Everhart, W. Mason, S. Tu and E. Zayas, "Decorum File System Architecture Overview," *In Proceedings of the summer 1990 Usenix Conference*, pp. 151-164, Anaheim, California, June 1990.
- [59] T. P. Kelly, Y. M. Chan, S. Jamin, J. K. MacKie-Mason, "Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value," *In Proceedings of Forth International Web Caching Workshop*, San Diego, California, March 1999.

- [60] B. Krishnamurthy, C. E. Wills, "Analyzing Factors that Influence End-to-End Web Performance," *Ninth International World Wide Web Conference (WWW9)/Computer Networks*, Volume 33, Numbers 1-6, pp. 17-32, June 2000.

- [61] B. Krishnamurthy and J. Rexford, "Software Issues in Characterizing Web Server Logs," *W3C Web Characterization Group Workshop*, Boston, MA, November 1998.

- [62] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," *In Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, pp. 13-22, Monterey, California, December 1997.

- [63] S. Kotz, N. Johnson, *Encyclopedia of Statistical Sciences*, Volume 3, pp. 598-599, John Wiley & Sons, New York, 1982-1988.

- [64] A. M. Law, W. D. Kelton, *Simulation Modeling and Analysis*, Third Edition, pp. 292-402, McGraw-Hill, Boston, 2000.

- [65] D. Li and D. R. Cheriton, "Scalable Web Caching of Frequently Updated Objects using Reliable Multicast," *In Proceedings of the 2th USENIX Symposium on Internet Technologies and Systems (USITS'99)*, Boulder, Colorado, October 1999.

- [66] C. Liu and P. Cao, "Maintaining Strong Cache Consistency in the World-Wide Web," *In Proceedings of the Seventeenth International Conference on Distributed Computing Systems (ICDCS'97)*, pp. 12-21, Baltimore, MD, May 1997.
- [67] B. Mah, "An Empirical Model of HTTP Network Traffic," *In Proceedings of the Conference on Computer Communications (INFOCOM '97)*, Kobe, Japan, April 1997.
- [68] A. Mahanti, D. Eager and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance," *Performance Evaluation* (Special Issue on Internet Performance Modeling), Volume 42, Number 2/3, pp. 187-203, September 2000.
- [69] A. Mahanti, C. Williams, D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy," *IEEE Network Magazine* (Special Issue on Web Performance), Volume 14, Number 3, pp. 16-23, May/Jun 2000.
- [70] C. Maltzahn, K. Richardson, D. Grunwald, "Reducing the Disk I/O of Web Proxy Server Caches," *In Proceeding of the 1999 USENIX Annual Technical Conference*, pp. 225-238, Monterey, CA, June 1999.
- [71] N. Markatchev and C. Williamson, "WebTraff: A GUI for Web Proxy Cache Workload Modeling and Analysis," *In Proceedings of IEEE/ACM International*

Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 356-363, Fort Worth, Texas, October 2002.

[72] E. P. Markatos and C. E. Chronaki, "A Top-10 Approach to Prefetching the Web," *In Proceedings of the INET'98 (Internet Society)*, Geneva, Switzerland, July 1998.

[73] E. P. Markatos, D. N. Pnevmatikatos, M. D. Flouris and M. G.H. Katevenis, "Web-Conscious Storage Management for Web Proxies," *IEEE/ACM Transactions on Networking (TON)*, Volume 10, Number 6, pp. 735-748, December 2002.

[74] M. K. McKusick, W. N. Joy, S. J. Leffler and R. S. Fabry, "A Fast File System for UNIX," *ACM Transactions on Computer Systems*, Volume 2, Number 3, pp. 181-197, August 1984.

[75] Microsoft Corporation, Web page at: <http://www.microsoft.com>.

[76] J. Mogul, "A Design for Caching in HTTP/1.1 Preliminary Draft," Technical Report, Internet Engineering Task Force (IETF), January 1996, Work in Progress.

[77] J. C. Mogul, "Hinted Caching in the Web," *In Proceedings of the Seventh ACM SIGOPS European Workshop*, pp. 129-140, Connemara, Ireland, September 1996.

- [78] J. Mogul, F. Douglass, A. Feldmann and B. Krishnamurthy, "Potential Benefits of Delta Encoding and Data Compression for HTTP," *In Proceedings of SIGCOMM'97*, pp. 181-194, Cannes, France, September 1997.
- [79] S. Nadimpalli and S. Majumdar, "Techniques for Achieving High Performance Web Servers", *In Proceedings of International Conference on Parallel Processing (ICPP'2000)*, pp. 233-242, Toronto, Canada, August 2000.
- [80] NetAccelerator 2.0 software available at:
<http://www.imsisoft.com/netaccelerator/netacc2.html>
- [81] Netscape Inc., "An Exploration of Dynamic Documents," available at:
http://home.netscape.com/assist/net_sites/pushpull.html
- [82] Netscape Communicator 4 User Manual, Professional Edition Commerce and Security, pp. 630 – 675.
- [83] Network cache performance measurements. CacheFlow White Papers Version 2.1, CacheFlow, Sept. 1998.

- [84] N. Niclausse, Z. Liu and P. Nain, "A New Efficient Caching Policy for the World Wide Web," *In Proceedings of the Workshop on Internet Server Performance (WISP'98)*, pp. 119-128, Madison, Wisconsin, June 1998.

- [85] H.F. Nielsen, J. Getty, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie and Chris Lilley, "Network Performance Effects of HTTP/1.1, CSS1 and PNG," *In Proceedings of ACM SIGCOMM'97*, pp. 155-166, Cannes, France, September 1997.

- [86] NLANR, traces are available in: <ftp://ircache.nlanr.net/Traces>.

- [87] V. N. Padmanabhan and J. C. Mogul, "Using Predictive Prefetching to Improve World-Wide Web Latency," *ACM SIGCOMM Computer Communication Review*, Volume 26, Number 3, pp. 22-36, July 1996.

- [88] PeakJet2000, Software available at: <http://www.peak.com/peakjet2long.html>.

- [89] J. E. Pitkow and M. M. Recker, "A Simple Yet Robust Caching Algorithm Based Upon Dynamic Access Patterns", *In Proceedings of the Second International World Wide Web Conference*, pp. 1039-1046, Chicago, IL, October 1994.

- [90] D. Povey and J. Harrison, "A Distributed Internet Cache," *In Proceedings of the 20th Australian Computer Science Conference*, Sydney, Australia, February 1997.

- [91] C. Roadknight, I. Marshall and D. Vearer, "File Popularity Characterization," *In Proceedings of the Second Workshop on Internet Server Performance (WISP'99)*, Atlanta, Georgia, May 1999.
- [92] J. Robinson and M. Devarakonda, "Data Cache Management Using Frequency Based Replacement," *In Proceedings of the 1990 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, pp. 134-142, Boulder, CO, May 1990.
- [93] A. Rousskov and V. Soloviev, "A Performance Study of Squid Proxy on HTTP/1.0," *World Wide Web Journal, Special Edition on WWW Characterization and Performance and Evaluation*, pp. 47-67, June 1999.
- [94] A. Silberschatz and P. Galvin, *Operating System Concepts*, Fifth Edition, pp. 431-433, John Wiley & Sons, 1999.
- [95] A. Silberschatz, P. Galvin and G. Gang, *Applied Operating System Concepts*, First Edition, pp. 377-382, John Wiley & Sons, 2000.
- [96] E. Shriver, E. Gabber, L. Huang, and C. A. Stein, "Storage Management for Web Proxies," *In Proceedings of USENIX Annual Technical Conference*, pp. 203-216, Boston, MA, June 2001.

- [97] Squid, Frequently Asked Questions: <http://www.squid-cache.org/Doc/FAQ/FAQ-12.html>
- [98] Squid, Related address is: <http://squid.nlanr.net>
- [99] Squid Internet Object Cache, For details see: <http://squid.nlanr.net/Squid>
- [100] J. Wang, R. Min, Y. Zhu and Y. Hu, "UCFS-A Novel User-Space, High Performance, Customized File System for Web Proxy Servers," *IEEE Transaction on Computers*, Volume 51, Number 9, pp. 1056-1073, September 2002.
- [101] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), Version 2," RFC2186, September 1998.
- [102] D. Wessels and K. Claffy, "ICP and the Squid Web Cache," *IEEE Journal on Selected Area in Communication (JSAC)*, pp. 345 –357, April 1998.
- [103] C. E. Wills and M. Mikhailov, "Toward a Better Understanding of Web Resources and Server Responses for Improved Caching," *In Proceedings of the Eighth International World Wide Web Conference*, pp. 153-167, Toronto, Canada, May 1999.

- [104] C. E. Wills and M. Mikhailov, "Examining the Cacheability of User-Requested Web Resources," *In Proceedings of the 4th International Web Caching Workshop*, pp. 78-87, San Diego, CA, March/April 1999.
- [105] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, "Removal Policies in Network Caches for World-Wide Web Documents," *In Proceedings of ACM SIGCOMM '96 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pp. 293-305, Stanford, CA, August, 1996.
- [106] C. Williamson, "On Filter Effects in Web Caching Hierarchies," *ACM Transactions on Internet Technology*, Vol. 2, No. 1, pp. 47-77, February 2002
- [107] D. Willick, D. Eager and R. Bunt, "Disk Cache Replacement Policies for Network Fileservers," *In Proceedings of the 13th International Conference on Distributed Computing Systems*, pp. 2-11, Pittsburgh, PA, May 1993.
- [108] R. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delay," *In Proceedings of the 6th International WWW Conference*, pp. 325-334, Santa Clara, CA, April 1997, also available at: <http://research.cs.vt.edu/chitra/docs/www6r/>
- [109] R. Wooster, *Optimizing Response Time, Rather than Hit Rates, of WWW Proxy Caches*, M. S. thesis, Computer Science Dept., Virginia Tech, Dec. 1996.

Available at: <http://scholar.lib.vt.edu/theses/materials/public/etd-34131420119653540/etd-title.html>

- [110] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes, "On-Line Extraction of SCSI Disk Drive Parameters," *In Proceedings of the ACM SIGMETRICS '95 Conference on Measurement and Modeling of Computer Systems*, pp. 146-156, Ottawa, Canada, May 1995. The complete version is available as Technical Report CSE-TR-323-96, University of Michigan, Ann Arbor, December 1996.

- [111] K. Wu, P. S. Yu and J. L. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," *In Proceedings of the Tenth International World Wide Web (WWW10) Conference*, pp. 36-44, Hong Kong, May 2001.

- [112] J. Yin, L. Alvisi, M. Dahlin and A. Iyengar, "Engineering Server-driven Consistency for Large Scale Dynamic Web Services," *In Proceedings of the Tenth International World Wide Web (WWW10) Conference*, pp. 45-57, Hong Kong, May 2001.

- [113] J. Yin, L. Alvisi, M. Dahlin and C. Line, "Using Leases to Support Server-Driven Consistency in Large-Scale Systems," *In Proceedings of the Eighteenth International Conference on Distributed Computing Systems*, pp. 285-295, Amsterdam, the Netherlands, May 1998.

- [114] J. Yin, L. Alvisi, M. Dahlin and C. Line, "Hierarchical Cache Consistency in a WAN," *In Proceedings of the 2th USENIX Symposium on Internet Technologies and Systems*, pp. 13-24, Boulder, Colorado, October 1999.
- [115] N. Young, "The k-server dual and loose competitiveness for paging," *Algorithmica*, June 1994, Volume 11, Number 6, pp.525-41. Revised version of "On-line caching as cache size varies," *In Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 241-250, San Francisco, CA, January 1991.
- [116] H. Yu, L. Breslau and S. Schenker, "A Scalable Web Cache Consistency Architecture," *In Proceedings of the ACM SIGCOMM'99*, pp. 163-174, Boston, MA, September 1999.