A Performance Study of Internet Web Servers

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in the

Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Martin F. Arlitt

Fall 1996

Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 0W0

A Performance Study of Internet Web Servers

Candidate: Martin F. Arlitt

Supervisor: Carey L. Williamson

Master of Science Thesis

Presented to the College of Graduate Studies and Research

University of Saskatchewan

Fall 1996

Abstract

The phenomenal growth in popularity of the World-Wide Web (WWW or "the Web") has made WWW traffic the fastest growing component of network packet and byte traffic on the NSFNET backbone. This growth has triggered much recent research activity aimed at improving Web performance and scalability: reducing the volume of network traffic produced by Web clients and servers by using file/document caching, and reducing the latency for WWW users by using improved protocols for Web interaction.

Fundamental to the goal of improving WWW performance is an understanding of WWW workloads. This thesis presents a workload characterization study for World-Wide Web servers. Six different data sets are used in this study: three from academic (i.e., university) environments, two from scientific research organizations, and one from a commercial Internet provider. These data sets represent three different orders of magnitude in server activity, and two different orders of magnitude in time duration, ranging from one week of activity to one year of activity.

The research in this thesis is divided into two parts. In the first part, emphasis is placed on finding workload invariants*: observations that apply across all data sets. Ten invariants are identified. These invariants are deemed important since they (potentially) represent universal truths for all World-Wide Web servers. The second part of the thesis uses the invariants identified in the workload characterization to suggest performance enhancements that seem promising for World-Wide Web servers. In particular, a performance study of file caching is conducted, using trace-driven simulation, to determine the effects of different replacement policies, threshold policies, and partitioning policies on file cache performance. The results show that simple caching strategies using a variation of the LFU (Least Frequently Used) replacement policy perform the best for the Web server workload traces considered.*

Acknowledgements

I am indebted to a large group of people, all of whom have helped me in some way during my Master's program. I would specifically like to thank:

Carey Williamson, my supervisor, for his guidance, wisdom, encouragement and promptness in reviewing my work. I am also grateful to the members of my committee: Professors Richard Bunt, Anthony Kusalik, and Don Drinkwater (External). Their comments and suggestions have dramatically improved the quality of my thesis.

John Dilley (HP Labs), for his constructive comments. Stephen Balbach (ClarkNet), Jim Dumoulin (NASA), Earl Fogel (U of S), Robert Fridman (U of C), Jamie Hodge (U of W), and Robert McGrath (NCSA), for making available the Web server logs that were so crucial to my research.

Professor Carl McCrosky, for helping me to see the light by convincing me to give Computer Science a shot. Leanne Breker, for her support and encouragement, as well as for making everyday life just a little more enjoyable. John Punshon, for teaching me much of what I know about computers. Russ Muzzolini, the LaTeX guru (and suicidal snowboarder), for his tremendous help with the formatting of my thesis. Kevin Froese, Remi Gurski, Tim Harrison, Brad Mazurek, and Greg Oster, for helping with the analyses and simulations. Professor W. Laverty, for refreshing my memory on statistics.

The Natural Sciences and Engineering Research Council of Canada (NSERC), for funding my post-graduate education.

The St. Walburg Eagles, Central Butte Flyers, CJWW Radio Waves, and CS/EE Flying Circuits hockey teams, for allowing me to release the stress that built up after a long day at school.

Finally, my parents, for teaching me the values of hard work and determination.

Table Of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The World-Wide Web (also called WWW, W3, or "the Web") is a distributed hypertext-based information system developed at CERN [2, 7, 72]. Originally developed to allow researchers to share their ideas on high-energy physics, the Web is now responsible for the fastest growing component of packet and byte traffic on the NSFNET backbone [51]. WWW traffic has increased from 74 Megabytes per month in December 1992, which ranked it 186th (in packets) on the Internet activity list, to 3.2 Terabytes per month in December 1994, ranking it second behind only the file transfer protocol (FTP).[1]

There are many reasons behind the explosive growth in Web traffic. These reasons include: the ease of use of the Web; the availability of graphical user interfaces (called Web browsers) for navigating the Web; the functionality provided by hypertext; the availability of editors for creating and "publishing" Web documents; the machine independent nature of the languages and protocols used for constructing and exchanging Web documents; an emerging trend among researchers, educational institutions and commercial organizations to make the Web the standard mechanism for disseminating information in a timely fashion; and a continuing (exponential) increase in the number of Internet hosts and users [50, 56].

The phenomenal growth in Web traffic has sparked much research activity on

[1] Unfortunately, more recent statistics are not available [13].

"improving" the World-Wide Web. The key motivations behind these improvements are the following two performance objectives: a desire to reduce the volume of network traffic produced by Web clients and servers, and a desire to improve the response time (i.e., reduce latency) for Web users.

Fundamental to the goal of improving Web performance is a solid understanding of WWW workloads. While there are several studies reported in the literature, including a synthetic workload model for Web client activity [3], no paper provides a definitive description of Web-related workloads. Most studies present data from only one measurement site, making it difficult to generalize results to other sites. Furthermore, most studies focus on characterizing Web clients, rather than Web servers. It is precisely this void that this work is intended to fill.

The first part of this thesis presents a detailed workload characterization study of World-Wide Web servers, similar to earlier studies of wide-area TCP/IP traffic [14, 15, 17, 74]. Access logs from six different Web servers are used: three from academic (i.e., university) environments, two from scientific research organizations, and one from a commercial Internet provider. The data sets represent three different orders of magnitude in server activity (ranging from 776 requests per day to 355,787 requests per day), and time durations ranging from one week of activity to one year of activity.

Throughout the workload characterization, emphasis is placed on finding workload *invariants*: observations that apply (or seem to apply) across all six data sets studied. These invariants are deemed important since they (potentially) represent universal truths for World-Wide Web servers. Ten candidate invariants are identified [4]. These invariants are summarized in Table 1.1 for ease of reference, and are discussed in more detail within the thesis itself.

The second part of this thesis uses the candidate invariants to suggest and evaluate performance enhancements that seem promising for Web servers. Trace-driven simulations are used to evaluate the performance of a file cache, and to determine the effects of replacement policies, threshold policies, and partitioning policies on the performance of the file cache.

The remainder of this thesis is organized as follows. Chapter 2 provides back-

Table 1.1: Summary of Invariants Found in Web Server Workloads

Invariant	Name	Description
1	Success Rate	Approximately 88% of requests to the server result in the successful return of the document
2	File Types	HTML and image files account for 90-100% of requests
3	Mean Transfer Size	Mean transfer size $\leq$ 21 kilobytes
4	Distinct Requests	Among all server requests, less than 3% of the requests are for distinct files
5	One Time Referencing	Approximately one-third of the files and bytes referenced in the log are referenced only once in the log
6	Size Distribution	The file size distribution is heavy-tailed; the tail of the distribution is Pareto with $0.40 < \alpha < 0.63$
7	Concentration of References	10% of the files accessed account for 90% of server requests and 90% of the bytes transferred
8	Inter-Reference Times	File inter-reference times are exponentially distributed and independent
9	Remote Requests	Remote sites account for $\geq$ 70% of the accesses to the server, and $\geq$ 60% of the bytes transferred
10	Wide Area Usage	Web servers are accessed by 1000's of domains, with 10% of the domains accounting for $\geq$ 75% of usage

ground information on the World-Wide Web, Web clients, Web servers, and Web proxies. Chapter 3 discusses Web performance, the goals of this research, and other research efforts on "improving" the World-Wide Web. Chapter 4 focuses on the workload characterization study, and identifying invariants in Web server workloads. This section also suggests some performance enhancements that seem promising for Web servers. Chapter 5 outlines the experimental design for a performance evaluation of file caching on a Web server. Chapter 6 presents the results from this performance study. Finally, Chapter 7 concludes this thesis with a summary of the work done, the contributions of this thesis, and some suggested future research directions.

Chapter 2

Background

This chapter presents a brief overview of the World-Wide Web, and describes how Web clients, Web servers, and Web proxies operate.

2.1 The World-Wide Web

The World-Wide Web is a distributed hypertext-based information system developed at CERN [2, 7, 11, 52, 72], and is based on the client-server model [52, 67]. The client-server model, as it applies to the Web environment, is shown in Figure 2.1. A user accesses documents on the Web via a *Web browser* (the client). Two popular Web browsers that use a graphical user interface are `Netscape Navigator` [49] and `Mosaic` [47]. An example of a Web browser that uses a text interface is `lynx` [41]. The Web browser sends the user request to a *Web server*, which responds with the requested document. Although information may be stored in almost any location throughout the world, the Web provides the illusion of a single repository of information, where the terms "local" and "remote" have (almost) lost their meaning.

A Web server can respond to requests from multiple clients. Communication is always in the form of request-response pairs, and is always initiated by the client. Web clients and servers communicate using the HyperText Transfer Protocol (HTTP) [8]. HTTP runs on top of TCP [58], a reliable bidirectional byte stream protocol at the

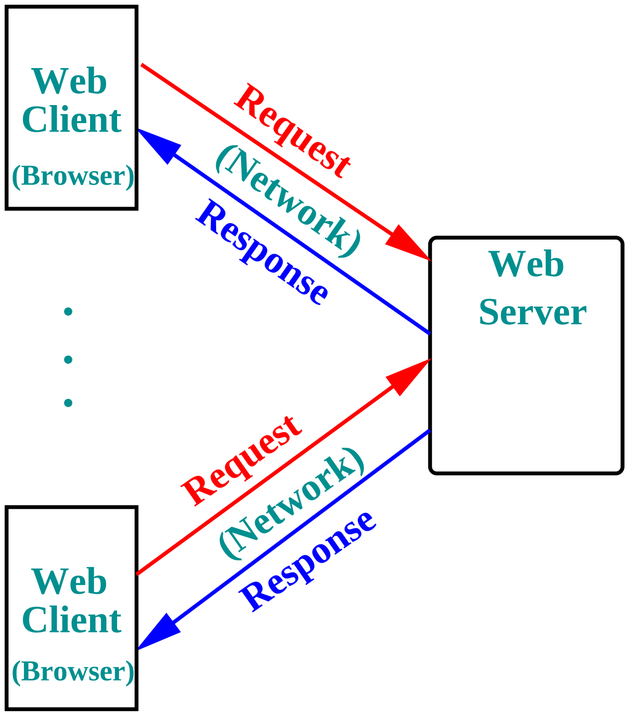


Figure 2.1: Client-Server Model in a Web Environment

transport layer of the OSI Reference Model [67].[1]

Communication between a Web client and a Web server is carried out in the following manner (see Figure 2.1). When a client has a request to make of a particular Web server, the client must contact that server. A TCP connection must be established between the client and server, over which the request and response can be exchanged [8, 52]. Once the connection has been established, the client sends its request to the server. The server parses the request and issues a response. Once the response is complete, the TCP connection between the Web client and Web server is closed. This process is repeated for each document that a client wishes to retrieve from a Web server.

[1] Although Web clients can communicate with different types of servers (e.g., HTTP, FTP, gopher, news), this thesis is concerned primarily with HTTP servers. For the remainder of this thesis, the term "Web server" refers to an HTTP server.

2.2 World-Wide Web Clients

A user can gain access to the information on the World-Wide Web by using a Web browser, such as `netscape` [49], `mosaic` [47], or `lynx` [41]. When the user selects a document (such as an image, sound, video, or postscript file) or a *Web page* (a collection of documents, often consisting of an HTML file and a number of image files) to retrieve (usually by clicking a mouse on a hyperlink), the browser creates a request to be sent to the corresponding server. The request [8] includes: the name of the requested document, expressed as a Uniform Resource Locator (URL) [6]; a set of Hypertext request headers, indicating which data formats the client will accept; and user authentication information, which tells the server which documents the client has permission to retrieve.

An example request is:

```
GET /grads.html HTTP/1.0
Referer: http://www.cs.usask.ca/
User-Agent: Mozilla/1.1N (X11; OSF1 V3.0 alpha)
Accept: */*
Accept: image/gif
Accept: image/x-xbitmap
Accept: image/jpeg
```

In this example, the client has asked the Web server `www.cs.usask.ca` for the document `grads.html`. The client is willing to accept all types of documents (`Accept: */*`). Documents that are of type `gif`, `xbitmap`, or `jpeg` are recognized as images by the client. This request includes several pieces of additional information. First, the `Referer` field shows that this request was selected from the home page of the server. Second, the `User-Agent` field shows that the client is the Netscape Navigator 1.1 (according to Netscape Communications, their company name may be spelled N-E-T-S-C-A-P-E, but it's pronounced Mozilla [49]) for the X11 Windows environment, running on a DEC alpha workstation running the OSF1 operating system.

Once the client has sent the request to the Web server, the client waits for a response. When the response arrives, the browser parses the reply. Depending on the response, the client may make another request to the server, or display the document for the user to view.

2.3 World-Wide Web Servers

The purpose of a Web server is to provide documents to Web clients that request them. A Web server operates in the following way. The server listens on a designated port (usually port 80) for a request from a Web client to establish a TCP connection. Once a TCP connection has been opened and the client has made its request, the server must respond to that request. The response [8] includes a status code to inform the client if the request succeeded. If the request is successful, a document is usually returned with the response. If the request is not successful, a reason for the failure is returned to the client.

A possible response to the example request in Section 2.2 is:

```
HTTP/1.0 200 Document follows
Date: Wed, 07 June 1995 17:05:56 GMT
Server: NCSA/1.4
Content-type: text/html
Last-modified: Tue, 28 Mar 1995 14:50:31 GMT
Content-length: 1215

[document code appears here.]
```

The status code (200) in the first line indicates that the request was successful. The date and time of the response is given, as is the type of server (in this case, it is NCSA `httpd` version 1.4). The requested document is a text file, formatted as html (HyperText Markup Language). This file was last modified at 2:50:31 pm GMT on Tuesday March 28, 1995. The response includes 1215 bytes of document code.

Once the Web server has sent its response and terminated the TCP connection with the client, the server repeats the cycle and begins listening for its next request.[2]

[2]This is a very general overview of how a Web server operates. For more detailed information see `http://hoohoo.ncsa.uiuc.edu/docs/`.

2.4 World-Wide Web Proxies

A World-Wide Web proxy is a special type of Web server. Web proxies were originally designed to provide access to the Web for people on closed subnets who could only access the Internet through a firewall machine [40]. Since newer proxies can also perform caching, they are also useful to users who have full Internet access. Most popular Web client programs have proxy support built in, making it easy to configure an entire group of clients to use the caching proxy. Using the caching proxy can result in a reduction of the external network traffic (i.e., between the proxy and the remote Web server) and in a reduction in response time for user requests [27].

Figure 2.2 shows how a proxy fits into the Web environment. Instead of communicating directly with the remote Web server, the clients communicate with only the Web proxy. The proxy behaves much like a Web server, in that it listens for and responds to client requests. How the proxy fulfills the client requests depends on the type of proxy in use. If the proxy does not support caching, the user request is simply forwarded to the remote server that is specified by the URL in the client's request. Once the proxy receives a response from the remote server, the proxy passes the request to the client. If the proxy uses caching, the cache is searched for the requested document. If the document is found in the cache (a cache hit), a copy of the document is returned immediately to the client. When a cache hit occurs, the remote server does not need to be contacted (although the proxy may contact the remote server to ensure that the cached document has not been modified). If the requested document is not in the cache (a cache miss), the proxy forwards the request to the remote server. When the response from the remote server is received, the proxy adds a copy to its cache before delivering the document to the client [40]. In all communication with remote Web servers, the proxy appears to be just another Web client.

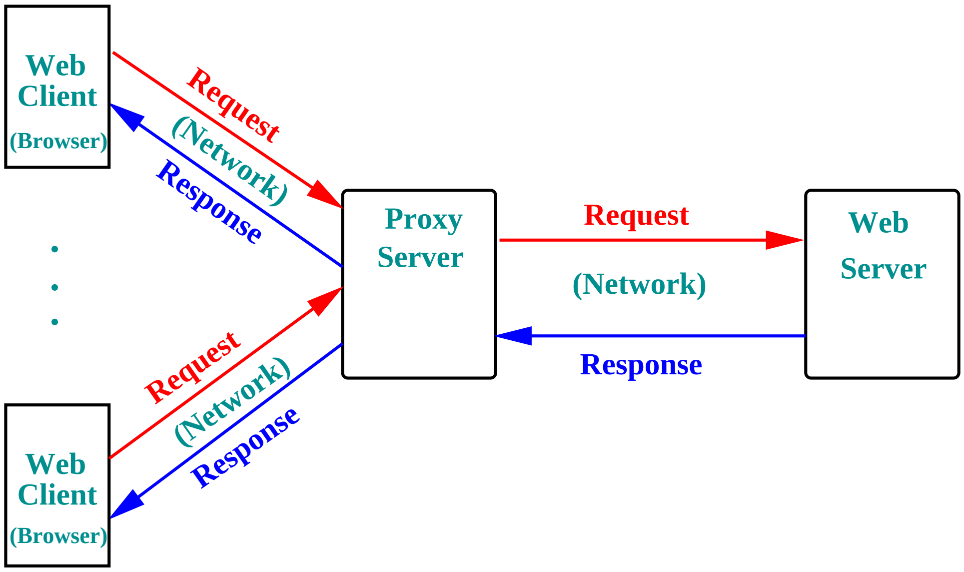


Figure 2.2: A WWW Proxy in a Web Environment

2.5 Summary

The World-Wide Web is a distributed hypertext-based information system based on the client-server model. Users can access documents on the Web using Web browsers (clients). The Web browser sends user requests to a Web server, which responds with the requested documents. Web proxies may be used to allow users from closed subnets to access the Web or to enhance performance for Web users. Web clients, servers, and proxies communicate using the HyperText Transport Protocol, which runs on top of TCP.

Chapter 3

World-Wide Web Performance

The World-Wide Web is a globally distributed application, which uses a large computer network (the Internet). The performance of the World-Wide Web is determined by the extent to which the various components of the Web (clients, servers, proxies, protocols, network links) operate efficiently. This thesis is concerned primarily with issues affecting the performance of Web servers. Section 3.1 discusses issues that affect the overall performance of the Web. Section 3.2 describes the strategies used to investigate Web server performance. Section 3.3 explains the makeup of a Web server log file, and the use of log files in this research. Section 3.4 reviews other research efforts on improving Web performance. Section 3.5 summarizes the chapter.

3.1 Performance Issues

The overall performance of the World-Wide Web is determined by the performance of the individual components that make up the Web: the clients, the servers, the proxies, the network links, and the protocols used for communication between the clients, servers, and proxies. The phenomenal (and alarming) growth in Web traffic has led to performance problems and has sparked much research activity on ways to improve performance. Of most interest are ways to reduce the volume of network traffic produced by Web clients, servers, and proxies, and ways to reduce the latency (i.e., improve response time) for Web users.

3.2 Investigating Web Server Performance

In order to improve the performance of Web servers, knowledge of the workloads that these servers are required to handle is needed. Knowledge of these workloads can be acquired by analyzing logs of Web server activity. These logs are extremely valuable, because they provide a snapshot of actual requests to Web servers. It is these logs that are examined to locate possible workload invariants. These invariants can be used to create a realistic workload model of a Web server, or to identify possible performance enhancements for Web servers.

3.3 Web Server Logs

Web servers can be configured to record information about all client requests. Four log files are common to NCSA httpd version 1.4: an *access* log, an *agent* log, an *error* log, and a *referer* log. The access log records information about all the requests and responses processed by the server. The agent log records the type of browser that was used by the client to issue the request. The error log records unusual Web server events that might require the attention of a Web master or system administrator. The referer log contains information on which Web pages (local or remote) are linked to documents on the (local) Web server. Only the access logs are used in the workload characterization study reported in this thesis.[1]

Each line from the access log contains information on a single request for a document. The log entry for a normal request is of the form:

```
hostname - - [dd/mmm/yyyy:hh:mm:ss tz] request status bytes
```

[1] The error log from the University of Saskatchewan's Web server was used to study the aborted connections in the access log. Aborted connections are discussed in more detail in Section 4.2.3.

From each log entry, it is possible to determine the name of the host machine making the request, the time that the request was made, and the name of the requested document. The entry also provides some information about the server's response to this request, such as if the server was able to satisfy the request (if not, a reason why the response was unsuccessful is given) and the number of bytes transmitted by the server, if any.

An example of a line from an access log is:

```
alfonso.usask.ca - - [15/Aug/1995:13:50:05 -0600] "GET / HTTP/1.0" 200 1265
```

This request came from the host `alfonso.usask.ca` at 1:50:05 pm CST on August 15, 1995. The requested document was the home page ("/") of the Web server. The status code of 200 means that the request was successfully completed by the server, and 1,265 bytes were transferred from the server to `alfonso.usask.ca`.

The access logs provide most of the data needed for workload characterization studies of Web servers, but they do not provide *all* the information that is of interest. For example, the log entries tell only the number of bytes transferred for a document, not its *actual size*; there is no record of the *elapsed time* required for a document transfer; and there is no information on the *complete set of files* available on the server, other than those documents that are mentioned in the logs. Furthermore, there is no record of whether a file access was human-initiated or software-initiated (e.g., by a *Web crawler*), or what caching mechanisms (if any) are in place at the client and/or the server. These issues are outside the scope of this study: the focus of this study is solely on characterizing the workload seen by a typical Internet Web server.

3.4 Related Work

Due to the phenomenal growth in World-Wide Web traffic, there is currently a lot of ongoing research looking for methods to "improve" the Web. Section 3.4.1 discusses some of the current research on improving Web client performance. Section 3.4.2 presents some recent work on Web server performance. Section 3.4.3 describes work that is focused on reducing network traffic. Section 3.4.4 outlines recent work on improving the efficiency of the protocols used in Web client-server communication.

3.4.1 Research on Web Client Performance

The main performance objective for Web browser (client) developers is to minimize latency, without restricting the user in any way. One common approach to the problem is to use file caching. Web browsers such as `Netscape Navigator` [49] use both memory and disk caches in an attempt to reduce the response time for users. Using memory and disk caches also reduces the volume of network traffic between the Web client and server (or proxy). These caches exist for each user.

File caching on the client works in the following way. When a user requests a document, the browser first checks the memory cache for a copy of the document. If the document is not found in memory, the user's disk cache is checked. If the document is not found in the disk cache (a cache miss), the request is sent over the network to the server (or proxy). When the response is received, a copy of the document is added to both of the client's caches. However, even if the document is found in the cache (a cache hit), a request may still be sent to the Web server, to verify that the client's copy of the document is still valid (i.e., the version of the document that the client has is the same version that the server has)[2]. Since the server does not maintain any information about which of its documents are cached or where they are cached, the server cannot notify a file cache when a document is modified on the server (i.e., Web servers do not invalidate cached documents). Therefore, it is the client's responsibility to ensure that it is using the proper version of the document. The client can do this by contacting the proper server and asking the server to verify the document in question. If the client's document is up-to-date, then the server informs the client to use the cached document, otherwise the new version is sent from the server to the client. Since most documents do not change frequently (see Section 4.2.3), and because sending the extra requests to validate

[2]Whether or not this is done is determined by user settings on the Web browser.

documents is inefficient, the client may choose not to validate cached documents every time that the document is requested. The frequency of the validation may be based on a time-to-live (TTL) value (i.e., only validate the document if its TTL has expired), or on a user setting (e.g., check documents only once per session).

Recent work at Boston University [10] analyzed the potential for document caching at a location near the client. More specifically, the study compared the performance of a file cache (using frequency-based replacement) at the session level (a single user), at the host level (several users on a single workstation), and at the network level (multiple users on a single LAN). This study found that while a high hit rate is achievable with these types of file caches, the volume of (external) network traffic is only slightly reduced, because most of the cached documents are quite small. Bestavros *et al.* [10] noted that caching at the session level (which is easy to implement) achieves almost the same level of performance as a LAN level cache. Using trace-driven simulations, they found that session level caching resulted in a 64% reduction in document latency, compared to a 77% reduction in document latency for a LAN level cache. However, the authors point out that the LAN level cache is much more efficient in the resources required (i.e., disk space) to achieve this performance. For example, reducing the volume of network traffic by 37% required a 9 MB cache at the LAN level, 90 MB at the host level, and was not possible at the session level.

3.4.2 Research on Web Server Performance

Efficient Web servers can reduce the latency for user requests by responding to the request as soon as possible. This is particularly important for popular Web servers, which can quickly build up a backlog of requests, as more and more users attempt to access information from the server. Research at the National Center for Supercomputing Applications (NCSA) includes the development of efficient, scalable Web server applications [36, 42]. Kwan *et al.* [36] and McGrath [42] focus on three issues that are needed to provide an efficient, scalable Web server. First, requests sent to the Web server (i.e., `www.ncsa.uiuc.edu`) need to be passed to one of the available servers (i.e., `zeppo.ncsa.uiuc.edu`, `costello.ncsa.uiuc.edu`, etc). Second, all the servers should have access to all documents in the Web database. Third, the requests should be distributed equally among all available servers. NCSA currently implements this using a round-robin domain name system [36].

In order for the Web server to perform well, each component of the server (i.e., hardware components like the CPU, the memory, disk I/O and network I/O, and software elements like the HTTP daemon, the TCP/IP implementation, and the file system) must also perform well [66].

McGrath [46] has evaluated the performance of several Web server platforms. McGrath identified several bottlenecks in the HTTP daemon. First, the HTTP daemon should be run in standalone mode rather than invoked by `inetd`. If the HTTP daemon is in standalone mode, the HTTP daemon listens to the designated port (usually port 80) for incoming requests. All requests are immediately served by the HTTP daemon. If the `inetd` approach is used instead, then `inetd` listens to the port. For each HTTP request it receives, `inetd` invokes a new copy of the HTTP daemon [46]. McGrath also shows that the HTTP daemon should use a helper process to handle multiple requests [46]. One example of a helper process is a multi-threaded server. A multi-threaded server handles each request by a thread within a single global process. While this is a good design that can take advantage of a multi-processor architecture, it is difficult to create a portable implementation. An alternative approach is to preallocate resources. A dispatcher then distributes the arriving requests to these helper processes. The helper processes are created only once, when the HTTP daemon is started. Either of these approaches (multi-threading or preallocation of resources) is preferred to simply creating a new process for each request (by invoking `fork`) [46]. McGrath found that the time to execute the `fork` consumes a significant fraction of the server's processor, and that having one process per request uses up larger amounts of memory [46].

In a second paper [44], McGrath investigates the performance problems of running the NCSA `httpd` software in the Solaris operating system. McGrath found that the problems are caused by the TCP implementation in Solaris. In order to fix these problems, the source code of the `httpd` software must be modified [44].

Another active area of research related to Web server performance is Web server benchmarking [45, 66, 68]. Benchmarking is used to evaluate the performance of a Web server under certain conditions. The difficulty with benchmarking is determining what conditions to test. The benchmarking results can then be used to compare the Web servers of different vendors. Developers can use benchmarks to test the performance of their Web servers, as well as to market their products. Customers can use the benchmark results to help them make a decision about which product to purchase. System administrators can use benchmarks in capacity planning situations. In 1995, Silicon Graphics released a Web server benchmark called WebSTONE [68]. Sun Microsystems is also working on a Web server benchmark [66], as is the Standard Performance Evaluation Committee (SPEC), a leading standards organization [45].

A researcher at StarNine Technologies has developed a simple, high-level open queueing network model of a Web server [61]. Using this model, a number of performance bottlenecks were identified. Slothouber [61] found that there is a clear upper bound on the serving capacity of Web servers, and that this upper limit is particularly sensitive to the average size of the files being served. As the Web server workload approaches this limit, the server enters a state resembling deadlock, where it attempts to serve more and more files, but at a slower and slower rate [61]. Slothouber states that limiting the number of simultaneous connections to the Web server can help prevent the occurrence of this condition [61].

3.4.3 Research on Reducing Web Traffic

As the number of hosts (and users) on the World-Wide Web increases [50], so does the amount of traffic passing over the Web. The amount of traffic will likely increase even further, as multimedia becomes more common on the Web, and new languages like Java[3] or VRML[4] become more popular. In order that a reasonable level of performance be maintained, more bandwidth will be required. One approach to obtaining more bandwidth is to upgrade all network links. However, this approach is both time-consuming and expensive. A second approach is to make better use of the available bandwidth. This can be achieved through the use of (network) file caches.

A number of recent works [12, 22, 27, 28] have looked at the use of network file caching. File caching can reduce the volume of network traffic, reduce the load on Web servers, and reduce the latency for Web users. Braun and Claffy [12] analyzed requests to NCSA's Web server. They found that through the use of geographically distributed file caches, the volume of traffic transferred by this server could be reduced by up to 70%. Danzig *et al.* [22] show that the volume of traffic on the NSFNET backbone could be reduced by 21%, if all FTP files are cached. In addition, if all FTP files were automatically compressed by the FTP client and server software, these savings could increase to 27% [22]. A caching relay at Digital Equipment Corporation in Palo Alto, California is found to reduce user response times by up to 83%, while reducing external network traffic by up to 50% [27]. Abrams *et al.* [1] studied the performance of caching proxies. The authors report that a proxy has a maximum hit rate of 30-50% no matter how it is designed. They also report that the Least Recently Used (LRU) replacement policy does not perform well, but simple variations to this policy can dramatically increase the hit rate while reducing the cache size [1, 69]. Gwertzman [28] proposes *geographical push-caching* as a method for reducing network traffic. This method involves replicating files. The degree to which a file is replicated is proportional to the file's global popularity, which can be determined from the global knowledge of the server. A similar approach is suggested by Bestavros [9].

One problem that arises with the use of network file caching and replication of files is the locating of the cached copy by the client, without communicating with

[3] See `http://java.sun.com/` for more information.
[4] See `http://vrml.wired.com/` for more information.

the server. Current research into this problem includes the development of Uniform Resource Names (URN), which provide location independent naming, unlike URLs. URNs define a uniformly hierarchical namespace, which supports dynamic relocation and replication of resources [38, 62].

A second problem with network file caching is maintaining *cache consistency* [12, 22]. This means that a document stored in a cache should always be the newest version of the document. If a document is revised, all cached copies need to be updated, in order to prevent a user from receiving a *stale* (old) copy of a file. However, the situation may not be as bad as it seems. First, many Web pages are not frequently modified, which reduces the chance of a cache containing stale files. Second, a user may not care whether or not the document is current, as long as a version of the document is available (i.e., if the remote Web server that has the new document is down, the old document can still be retrieved from the file cache [27]).

Braun, Claffy, Danzig and Schwartz are currently working on a project entitled "Distributed Testbed for National Information Provisioning"[5], a collaborative effort with DEC to develop and deploy a scalable caching and replication system across strategic network locations within the United States.

3.4.4 Research on Web Client-Server Communication

Currently, World-Wide Web clients and servers communicate using the HyperText Transfer Protocol [7]. A significant performance problem with HTTP is that it uses a separate TCP connection for each user request (the request-response pairs use the same connection). Using a separate connection for each request results in an increase in response time [53, 64]. For example, if a user requests a Web page that consists of one HTML file and four image files, five separate TCP connections are required. Significant increases in latency can be attributed to two sources of delay. First, each separate TCP connection requires time to be opened and time to be closed. Latency due to connection setup could be significantly reduced if

[5]See `http://www.nlanr.net/Cache/index.html` for more information.

multiple requests were pipelined into a single connection [53, 64]. In the example above, only one TCP connection would be needed, saving the costs of opening and closing the other four connections. The second source of latency can be attributed to the congestion control algorithm *Slow-Start* that is used by TCP. This algorithm prevents TCP from fully utilizing the available bandwidth for the first few round trips of a connection. This approach increases the latency for short connections [53]. If multiple requests went over a single connection, latency would again be reduced as the single larger transfer would receive more bandwidth than the separate smaller connections [53].

Some Web clients and servers now support *KeepAlive* connections [48]. If both the client and the server have KeepAlive support, the client can pipeline all of its requests across a single TCP connection. The server responds to each of these requests across this same connection. A similar approach to KeepAlive is *HTTP Session Extension* [30]. Not only does HTTP Session Extension allow for multiple requests to be sent over a single TCP connection, but it also allows the client to specify what file types it accepts only once, at the start of the first request. The server then saves this information, and uses it for all requests received on this connection. This approach reduces the number of bytes that are sent across the connection [30].

Work on a new release of HTTP (Version 1.1) is in progress [8]. This version will allow for the pipelining of multiple requests over a single TCP connection. HTTP 1.1 will also have caching directives, which will specify rules to be obeyed by all caching mechanisms along the transfer path. These directives will identify cacheable files, non-cacheable files, private files, and TTL values [8].

Many problems similar to those facing World-Wide Web researchers have been tackled by researchers in wide-area file systems. Spasojevic *et al.* [63] suggest using a wide-area file system, such as the Andrew File System (AFS), within the World-Wide Web. AFS has built in mechanisms for caching, replication, location transparency (i.e., a global name space) and authentication, which facilitate improved performance, reliability and security. Spasojevic *et al.* also found that the AFS transfer protocol Rx performs better than HTTP, as Rx requires less state to establish a connection, and uses bandwidth more aggressively [63].

3.5 Summary

The World-Wide Web, a globally distributed application, is growing rapidly. As this growth continues, efficient Web clients, servers, proxies, network links, and protocols become increasingly important. Other research efforts are currently looking to improve the efficiency of Web clients, proxies, network links, and protocols, as well as Web servers. The research in this thesis is concerned primarily with locating invariants in Web server workloads, which can provide insight into improving the performance of Web servers.

Chapter 4

Workload Characterization

This chapter includes two parts: first, the identification of a number of (possible) workload invariants, and second, a discussion of caching and performance issues for World-Wide Web servers. In total, ten candidate invariants for Web server workloads are identified. These are summarized in Table 1.1 in Chapter 1. Section 4.1 describes the process of data collection, reduction, and analysis. It also includes a summary of the statistical characteristics of the data sets. Section 4.2 presents a detailed analysis of file referencing behaviours on World-Wide Web servers, as well as a detailed look at file size distributions, file modification times, the effects of user aborts, and the presence of self-similarity in Web server workloads. Section 4.3 discusses the second part of this chapter, namely the performance implications for Web servers. Section 4.4 concludes this chapter with a summary of the results.

4.1 Data Collection, Reduction, and Analysis

This section presents an overview of the six separate data sets used in this workload characterization study. Section 4.1.1 describes the data collection sites. Sections 4.1.2 and 4.1.3 present the "raw" log contents, while Section 4.1.4 discusses the reduction of the raw data from the access logs into more manageable form. An analysis of the documents by type and by size is given in Section 4.1.5. Section 4.1.6

summarizes the statistical characteristics of the six data sets.

4.1.1 Data Collection Sites

The access logs used in this research were obtained from six World-Wide Web servers: a lab-level Web server at the University of Waterloo (Shoshin Research Lab in the Department of Computer Science); a department-level Web server at the University of Calgary (Department of Computer Science); a campus-wide server at the University of Saskatchewan; the Web server at NASA's Kennedy Space Center; the Web server from ClarkNet, a commercial Internet provider in the Baltimore - Washington D.C. region; and the Web server at the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, Illinois.

During the data collection, the Web server for the Shoshin Research Lab in the University of Waterloo's Department of Computer Science was a SUNsparc2, running NCSA httpd version 1.3. The Shoshin lab consists of 31 faculty, staff and students [29]. The Calgary Web server was a SUN 4/490, serving about 1300 faculty, staff and students [25]. The Web server at the University of Saskatchewan was a DECstation 5000/133, running NCSA httpd version 1.4 [24]. The university has a population of approximately 21,000 students, faculty and staff. The NASA server consisted of 4 DEC Alpha 2100 servers on an FDDI ring, each running NCSA httpd version 1.4 [23]. The ClarkNet Web server was a SUNsparc10 with two 60 MHz processors, running Netscape's Commerce server version 1.1. At the time of the data collection, ClarkNet was providing Internet access for some 5,000 people [5]. The NCSA server consists of 8 HP 735 workstations, used in a round-robin fashion to provide Web service [43, 36]. The workstations all run NCSA httpd, and use AFS for file access over an FDDI ring.

The access logs were obtained in a number of ways. The Calgary, NCSA, Saskatchewan and Waterloo data sets were obtained after contacting people at each institution directly; the NASA data set was acquired after contacting the Web master for the NASA site; and the ClarkNet data set was obtained after the Vice President of ClarkNet responded to an article posted to a newsgroup. Although six data sets were made available for this study, most of the sites that were contacted refused to provide access to their Web server log files, due to privacy and security issues. ClarkNet was the only site that responded to the news article. Privacy was also a concern for a number of sites that did provide access logs, particularly Calgary. Therefore, the Calgary data set was "sanitized" to remove information about the names of files (each file was given a unique integer identifier, and maintained the same file extension, so that no file type information was lost) and to remove the host names of computers accessing the Web server (each host name was identified as being either a local host or a remote host; this did result in the loss of some information).

4.1.2 Raw Data

The access log for the Shoshin Lab Web server at the University of Waterloo's Department of Computer Science contained 188,636 requests, for the eight-month period from January 1st to August 31st 1995. The access log for the Web server at the University of Calgary's Department of Computer Science held 726,739 requests, for the one-year period from October 24, 1994 to October 11, 1995. The access log from the University of Saskatchewan Web server contained 2,408,625 requests, collected over a seven-month period from June 1st to December 31st 1995. The NASA access log contained 3,461,612 requests, collected between July 1st and August 31st 1995.[1] The access log for the ClarkNet Web server held 3,328,587 requests, for the two-week period of August 28th to September 10th 1995. Finally, the NCSA data set held 2,490,512 requests, for the week-long interval from August 28th to September 3rd 1995. Table 4.1 summarizes the overall information for the six data sets, including the average number of requests per day that each server received, and the average number of bytes per day that each server transferred.

The six access logs provide information on Web servers with very differing work-

[1] The NASA Web server was shut down for a period of 38 hours between August 1st and August 3rd, due to Hurricane Erin.

Table 4.1: Summary of Access Log Characteristics (Raw Data)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Access Log Duration	8 months	1 year	7 months	2 months	2 weeks	1 week
Access Log Start Date	Jan 1/95	Oct 24/94	Jun 1/95	Jul 1/95	Aug 28/95	Aug 28/95
Access Log Size (MB)	17.9	49.9	222.6	355.8	327.5	267.7
Total Requests	188,636	726,739	2,408,625	3,461,612	3,328,587	2,490,512
Avg Requests/Day	776	2,059	11,255	56,748	237,756	355,787
Bytes Transferred (MB)	2,071	7,577	12,343	62,489	27,647	28,268
Avg Bytes/Day (MB)	8.5	21.5	57.7	1,024.4	1,974.8	4,038.3

loads. Table 4.1 shows that the Waterloo server had a very light workload, while the Saskatchewan server had an order of magnitude more requests to handle. The ClarkNet and NCSA Web servers had very heavy workloads, more than an order of magnitude greater than the Saskatchewan server. The level of server activity represented in the six logs varies by almost three orders of magnitude, so that the search for invariants covers light (e.g., thousands of requests per day), medium and heavy (hundreds of thousands of requests per day) workloads. The logs also span different time durations, so that short term (e.g., daily), medium term, and long term (e.g., monthly) aspects of Web server file referencing activity can be studied.

Although the six Web servers under study have very different workloads, they have at least one thing in common. All six of these servers provide access to a relatively static set of documents; that is, there are not many documents that change frequently (e.g., requests to a search engine), or documents that are of interest for only a short period of time (e.g., the scores from last night's hockey games). Access logs from these types of servers could not be obtained, although several sites of these types were contacted.

4.1.3 Access Log Analysis

The first step in the data analysis process was to study the response codes in the Web server access logs. There are many possible responses to client requests. These include:

1. *Successful*: a valid document, which the client has permission to access, was found on the server and returned to the client

Table 4.2: Breakdown of Server Responses (All Data Sets)

Response Code	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Successful	87.8%	78.4%	91.1%	89.6%	88.8%	93.1%
Not Modified	8.2%	13.5%	6.3%	7.7%	8.1%	4.1%
Found	1.6%	4.2%	1.7%	2.1%	0.9%	0.3%
Unsuccessful	2.4%	3.9%	0.9%	0.6%	2.2%	2.5%
Total	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

2. *Not Modified*: the client, which already has a copy of the document in its cache but wishes to verify that the document is up-to-date, is told that the document has not been modified at the server (thus no data bytes need to be transferred)

3. *Found*: the requested document is known to reside in a different location than was specified by the URL provided by the client, so the server responds with the new URL (but not the document)

4. *Unsuccessful*: either no such document exists, the client did not have permission to access this document, or an error occurred (at the server or during network communication)

Table 4.2 provides an overview of the response code frequencies observed in the access logs. From Table 4.2, the first candidate invariant in Web server traffic can be identified. *Successful* responses made up approximately 88% of all responses in each of the six logs. Cache related queries that result in *Not Modified* account for about 8% of the responses.

4.1.4 Data Reduction

Since the *Successful* responses are responsible for all of the documents transferred by the server, only these responses are used in the remaining analyses. That is, all information on the *Successful* responses is retained. This simplification provides a slight reduction in the size of the data sets to be processed, and focuses the workload characterization on the most common events. Further, more significant reductions

Table 4.3: Summary of Access Log Characteristics (Reduced Data)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Access Log Duration	8 months	1 year	7 months	2 months	2 weeks	1 week
Access Log Size (MB)	10.4	20.9	143.9	221.2	195.5	172.6
Total Requests	163,112	567,795	2,165,415	3,087,043	2,940,712	2,289,510
Avg Requests/Day	671	1,608	10,119	50,607	210,051	327,073
Distinct Requests	3,406	8,370	18,849	9,355	32,294	23,855
Total Bytes Transferred (MB)	2,071	7,577	12,330	62,483	27,591	28,268
Avg Bytes/Day (MB)	8.5	21.5	57.6	1,024.3	1,970.8	4,038.3
Mean Transfer Size (bytes)	13,313	13,997	5,970	21,224	9,838	12,947
CoV of Transfer Size	3.45	8.01	11.19	3.62	3.84	6.92

are obtained by removing all extraneous information from the original logs, and representing the necessary information in a more efficient manner. For example, information such as "- -" was not needed in the study, so could be removed without reducing the information available in the reduced logs. Converting the date and time information to an integer value (representing the number of seconds since midnight of January 1st, 1970) resulted in further reductions. This conversion also simplified all analyses involving the use of the timestamps.

Table 4.3 provides a statistical summary of the reduced data sets. This table shows that the number of unique documents requested from the server is significantly lower than the total number of documents requested, implying that some (or all) documents are requested many, many times. The mean size of the documents transferred is quite small (5 - 21 Kbytes), as might be expected. The table also shows that there is a high degree of variability (measured by the coefficient of variation, CoV) in the transfer size, particularly for the Saskatchewan data set.

4.1.5 Analysis of Document Types and Sizes

The high degree of variation in document size is due in part to the wide variety of document types accessed on the server (e.g., HTML, gif, postscript, audio, MPEG). The next step in the analysis was to classify documents by type. Six generic categories were used:

1. HTML (`.html` file types)

2. Images (e.g., `.gif`, `.jpeg`, or `.xbm` file types)

Table 4.4: Breakdown of Document Types and Sizes (All Data Sets)

Waterloo Data							
Item	HTML	Images	Sound	Video	Dynamic	Formatted	Other
% of Requests	38.7	50.1	0.01	0.0006	0.3	3.7	7.184
% of Bytes Transferred	35.0	18.9	0.10	0.10	0.2	25.2	20.5
Mean Transfer Size	12,036	4,961	120,973	2,232,051	6,465	90,444	42,130
CoV of Transfer Size	1.82	3.45	1.10	0.00	1.46	1.83	1.89
Calgary Data							
Item	HTML	Images	Sound	Video	Dynamic	Formatted	Other
% of Requests	47.1	50.3	0.1	0.3	0.04	1.0	1.16
% of Bytes Transferred	13.2	50.2	1.3	11.4	0.01	21.7	2.19
Mean Transfer Size	3,929	13,971	258,196	496,992	4,702	305,444	27,112
CoV of Transfer Size	1.86	3.95	1.49	1.60	1.26	2.77	4.09
Saskatchewan Data							
Item	HTML	Images	Sound	Video	Dynamic	Formatted	Other
% of Requests	55.6	36.5	0.1	0.004	6.7	0.02	1.076
% of Bytes Transferred	50.7	36.6	1.5	2.6	4.4	0.1	4.1
Mean Transfer Size	5,447	5,980	84,154	3,602,176	3,969	36,055	22,441
CoV of Transfer Size	2.19	2.77	2.62	2.29	2.91	0.08	11.30
NASA Data							
Item	HTML	Images	Sound	Video	Dynamic	Formatted	Other
% of Requests	30.7	63.5	0.2	1.0	2.6	0.01	1.99
% of Bytes Transferred	18.8	48.1	1.1	29.7	0.3	0.07	1.93
Mean Transfer Size	12,981	16,059	110,311	439,151	2,817	136,436	26,349
CoV of Transfer Size	2.71	2.37	0.80	0.84	0.68	1.85	2.55
ClarkNet Data							
Item	HTML	Images	Sound	Video	Dynamic	Formatted	Other
% of Requests	19.9	78.0	0.2	0.007	1.2	0.01	0.683
% of Bytes Transferred	15.0	76.6	2.4	2.4	0.8	0.04	2.76
Mean Transfer Size	7,433	9,669	135,082	3,514,759	6,630	36,199	37,138
CoV of Transfer Size	2.14	1.66	1.24	0.35	3.31	1.03	4.25
NCSA Data							
Item	HTML	Images	Sound	Video	Dynamic	Formatted	Other
% of Requests	51.1	48.1	0.2	0.1	0.01	0.006	0.484
% of Bytes Transferred	51.1	36.0	3.5	6.2	0.06	0.2	2.94
Mean Transfer Size	12,950	9,679	197,605	594,796	6,535	369,590	103,783
CoV of Transfer Size	3.56	2.46	5.79	2.18	6.69	2.60	4.38

3. Sound (e.g., `.au` or `.wav` file types)

4. Video (e.g., `.mpeg`, `.avi` or `.mov` file types)

5. Dynamic (e.g., `.cgi` or `.perl` file types)

6. Formatted (e.g., `.ps`, `.dvi` or `.doc` file types)

Any type that did not fit into one of these six categories was placed in the Other category.[2] For each of the data sets in Table 4.3, statistics on the type of document requested were calculated. The results for each are given in Table 4.4.

[2]Not all of the file types for each category are listed, as in some cases there are many. For example, statistics for all tiff files would be included in the Images category and not the Other category.

Using Table 4.4, a second candidate invariant can be identified in Web server workloads. Across the six data sets, HTML and Image documents accounted for 90 - 100% of the total requests to the server.[3] This observation is consistent with results reported by Sedayao [60], by Cunha *et al.* [20], and by Kwan *et al.* [37]. All of these papers reported that over 90% of client requests were for either HTML or Image documents.

Table 4.4 also indicates that most transferred documents are quite small, which is a third invariant. This phenomenon was also observed by Braun and Claffy [12] and by Kwan *et al.* [37] for requests to NCSA's Web server. Despite the fact that Web browsers provide support for the use of multimedia objects like sound and video, documents of these types accounted for only 0.01 - 1.2% of the requests in the six data sets. However, these types of files account for 0.2 - 30.8% of the bytes transferred, since these files tend to be much larger than other file types. In the NASA data set, one video file was responsible for 14.3% of the bytes transferred by the NASA server, although only 0.133% of the requests were for this file. It is obvious that future growth in the supply of (or the demand for) multimedia documents could have a significant impact on the workload of a server (as well as an impact on Invariants 2 and 3).

The large variation in the mean file sizes for the different document types helps to explain the large coefficient of variation (CoV) values reported in Table 4.3. The CoV values per document type are much lower in Table 4.4.

Table 4.5 illustrates two additional workload invariants. First, only 0.3 - 2.1% of the requests and 0.4 - 5.1% of the bytes transferred are for distinct documents. This observation implies that caching documents (at the server, at the client, or within the network) could greatly improve the performance of the server, as has been pointed out by Braun and Claffy [12]. Second, in all six data sets, approximately one-third (e.g., 22.6 - 42.1%) of all the distinct documents are requested only once,

[3] In the available data sets, there is no invariant for HTML documents alone, or for Image documents alone. In fact, the usage of HTML and Image document types differs dramatically for the Saskatchewan and ClarkNet data sets.

Table 4.5: Statistics on Distinct Documents for All Data Sets

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Distinct Requests/Total Requests	2.1%	1.5%	0.8%	0.3%	1.1%	1.0%
Distinct Bytes/Total Bytes	5.1%	3.8%	2.1%	0.4%	1.5%	2.7%
Distinct Files Accessed Only Once	29.1%	22.6%	42.0%	42.1%	31.9%	35.0%
Distinct Bytes Accessed Only Once	22.8%	19.8%	42.5%	14.3%	24.7%	39.1%

and one-third (e.g., 14.3 - 42.5%) of the distinct bytes are transferred only once. This observation is somewhat surprising given that the six data sets represent time durations ranging from one week to one year. This "one-time" referencing behaviour has obvious implications on the maximum possible effectiveness of document caching policies. Further discussion of these implications is deferred until Section 4.3.

4.1.6 Summary

This section has summarized the statistical characteristics of the six data sets used for this workload characterization study. While the six access logs differ greatly in duration and server activity, five workload invariants have been identified. These invariants are summarized in the first five rows of Table 1.1. The next section presents an in-depth study of file referencing patterns and file size distributions for World-Wide Web servers, looking for further invariants.

4.2 Web Server Workload Characterization

This section presents a detailed analysis of file referencing behaviours on World-Wide Web servers, as well as a detailed look at file size distributions, file modification times, the effect of user aborts, and the presence of self-similarity in Web server workloads. An analysis of file size distribution starts this section.

4.2.1 File Size Distribution

Figure 4.1 shows the (cumulative) file size distribution for each of the six sites. While there are a few very small files (< 100 bytes) at each of the sites, most files appear to be in the range of 100 - 100,000 bytes, while a few files (< 10%) are larger than

100,000 bytes. This distribution is quite consistent with the file size distribution on NCSA's Web server [12].

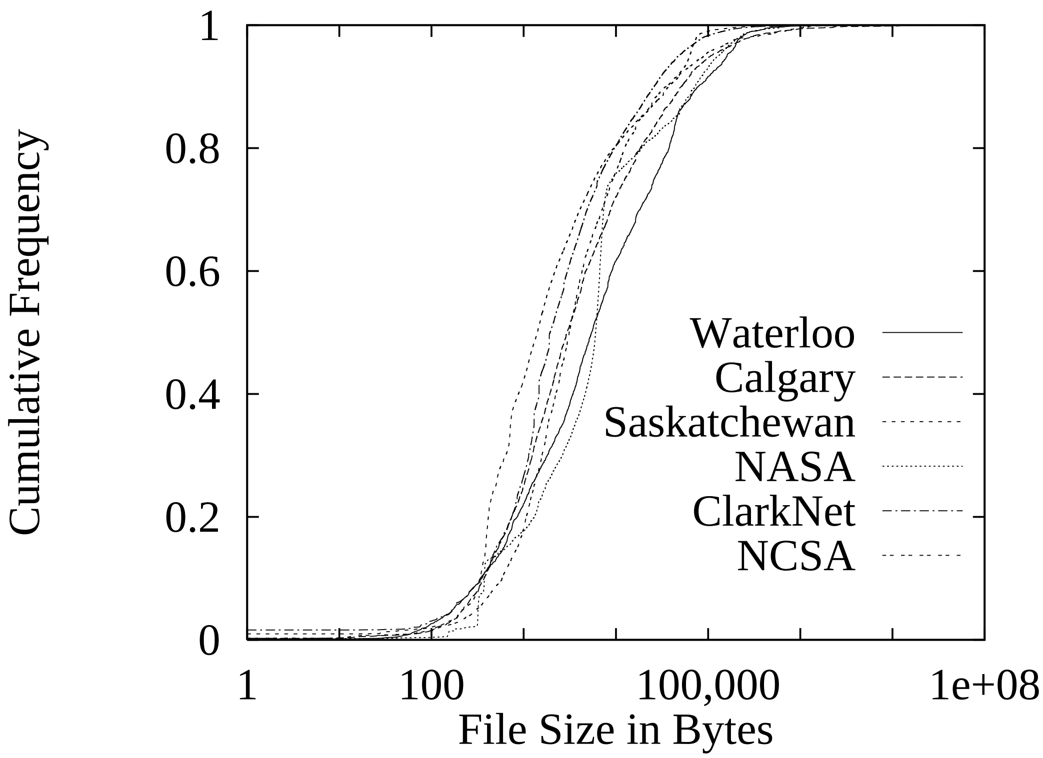


Figure 4.1: Distribution of File Sizes, by Server

Figure 4.1 shows that the distribution of filesizes is *heavy-tailed*. A distribution is deemed to be heavy-tailed if (regardless of the behaviour for small values) the asymptotic shape of the distribution is hyperbolic [18, 33, 55]. The simplest heavy-tailed distribution is the *Pareto* distribution. This distribution was originally applied to socioeconomic applications, such as the distribution of income [33]. More recently, this distribution has been used to model the distribution of file sizes [18] and FTPDATA bursts [54].

The Pareto distribution has the probability density function:

$$p(x) = \alpha k^{\alpha} x^{-\alpha-1}, \qquad \alpha, k > 0, x \geq k \tag{4.1}$$

Table 4.6: Maximum Likelihood Estimates of α (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
$\hat{\alpha}$	0.40	0.49	0.54	0.45	0.63	0.55

The cumulative distribution function for the Pareto distribution is:

$$F(x) = P[X \leq x] = 1 - (k/x)^{\alpha} \tag{4.2}$$

In both equations, α is the shape parameter and k is the smallest possible value of the random variable. As α decreases, an arbitrarily large portion of the probability mass may be present in the tail of the distribution; this means that a random variable that follows a heavy-tailed distribution can give rise to extremely large values with non-negligible probability [18]. As k increases (i.e., as the smaller values in the distribution are ignored), only the tail of the distribution is modeled.

There are numerous methods for estimating the value of α for an empirical distribution. For this study, estimates of α (denoted as $\hat{\alpha}$) were determined using *Maximum Likelihood Estimators* [31, 32]. Using this approach, $\hat{\alpha}$ is estimated by calculating:

$$\hat{\alpha} = n[\sum_{j=1}^{n} log(X_j/k)]^{-1} \tag{4.3}$$

Table 4.6 shows the estimates of α that were calculated using Maximum Likelihood Estimators (with $k = 1000$ bytes). Other methods for estimating α include *Least Squares Estimators* [32], *Estimators from Moments* [32], *log-log complimentary distribution plots* [18], and the *Hill estimator* [71]. Although there are many methods for estimating α, there does not appear to be one "good" method [19]. Researchers at Boston University are currently developing a tool to provide more accurate estimates for the shape parameter α [19].

Figure 4.2 shows the cumulative distribution of file sizes on the NCSA Web server, along with the curves of two different Pareto distributions. The first Pareto distribution, labelled "alpha = 0.55", has shape parameter $\alpha = 0.55$ and location parameter $k = 1000$. This curve uses the shape parameter that was determined by the Maximum Likelihood Estimator. The "alpha = 0.55" curve has too much weight in the tail, which suggests that a larger value is needed for α. The second Pareto distribution, denoted "alpha = 0.63", has a much better fit to the tail of the empirical distribution. Across the six data sets, the tails of the file size distributions (for files larger than 1000 bytes) appear to be Pareto with $0.40 \leq \alpha \leq 0.63$. This characteristic is present in all six data sets, and is thus added to Table 1.1 as the sixth candidate invariant.

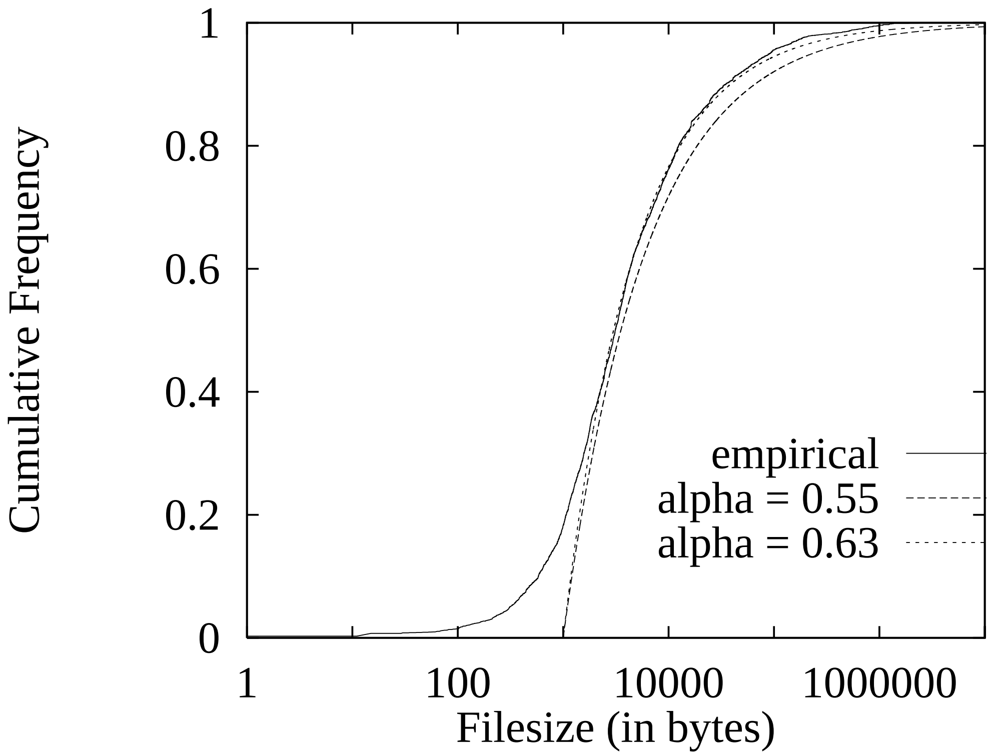


Figure 4.2: Comparison of Empirical and Pareto Distribution of File Sizes (NCSA Data)

It is important to be able to match the upper tail of the file size distribution, as these large files are responsible for a significant fraction of the bytes transferred by the server. However, the Pareto distribution does not match well with the entire distribution of file sizes. A lognormal distribution may characterize the lower and middle ranges of the file size distributions more closely [18]. Since most of the requests are for files in the 100 - 100,000 byte range, it will be important to model this range accurately as well. However, since a lognormal distribution is not heavy-tailed [18], an accurate model of the entire range of file sizes may require the use of several combinations (e.g., a lognormal distribution for the lower and middle ranges and a Pareto distribution for the upper range of the file size distribution), similar to that used by Paxson for modeling FTPDATA bursts [54]. A more sophisticated heavy-tailed distribution might also work well for modeling the entire file size distribution. The creation of a model (either scaled to the particular distribution under study or unscaled, to approximate Web server file size distributions in general) that represents the entire range of file sizes is left for future work.

4.2.2 File Referencing Behaviour

This subsection looks at a number of different characteristics in the file referencing patterns at World-Wide Web servers. The analysis focuses on five general characteristics: frequency of reference, concentration of references, temporal locality, inter-reference times, and geographic distribution of references.

4.2.2.1 Frequency of Reference

The first analysis focuses on the frequency of reference for different Web documents. Clearly, not all Web documents are created equal. Some are extremely "hot" and popular documents, accessed frequently and at short intervals by many clients at many sites. Other documents are accessed rarely, if at all.

To illustrate this non-uniform referencing behaviour (or *concentration*), the list of unique files is sorted into decreasing order based on how many times the unique files were accessed. A graph is then created by plotting the cumulative frequency of requests (starting with the most popular documents) versus the fraction of the total files referenced. The resulting plot for all six data sets is shown in Figure 4.3(a).

Figure 4.3(a) illustrates the non-uniform pattern of file referencing behaviour: 10% of the unique documents were responsible for 80-95% of all requests received by the server, at each of the six sites. The NCSA data set shows the most concentration, while the Calgary data set shows the least. This referencing pattern is similar to the Bradford-Zipf distribution identified in library science applications, as well as the referencing behaviour identified in the context of program behaviour, file systems, and virtual memory systems [27, 33, 34, 35, 57, 73]. Figure 4.4 shows the frequency distribution of requests to the unique files at the NCSA Web server, compared to the Zipf distribution.

This concentration phenomenon is another invariant in the Web server logs, and is thus added to Table 1.1. Braun and Claffy [12] and Kwan *et al.* [37] reported similar results for NCSA's Web server.

The same concentration pattern applies to bytes transferred as well. When the documents are sorted by the number of bytes that they caused the server to transfer (Figure 4.3(b)), 10% of the unique documents were responsible for approximately 90% of the bytes transferred by the server at each of the six sites.

A simple analysis was performed to determine if there is any relationship between the size of a file and the number of times that a file is referenced. A histogram of the total number of references versus file size is shown in Figure 4.5(a) for each of the data sets. This figure shows that most references are for small files (i.e., files in the range 100 to 10,000 bytes). However, when the data is normalized (i.e., when the average number of references per file of a given size is calculated, shown in Figure 4.5(b)), it can be seen that a small file (files in the range 100 to 10,000 bytes) is only slightly more likely to be requested than a large file is. To verify this, a correlation coefficient is calculated for each data set. By definition, the value of the correlation coefficient always lies between -1 and $+1$ [31]. A correlation coefficient close to $+1$ would indicate that large files are more likely to be requested than small files, while a correlation coefficient close to -1 would indicate that small files are more likely to be requested. If there is no relationship between file size and frequency of reference, a correlation coefficient of 0 will occur.

The correlation coefficients for each of the six data sets are shown in Table 4.7. In all cases, the correlation coefficients are negative, but are extremely close to 0. These correlation coefficients are consistent with the earlier observation that small

Table 4.7: File Size - Frequency of Reference Correlation Coefficients (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Correlation Coefficient	-0.003	-0.005	-0.003	-0.008	-0.005	-0.003

files are only slightly more likely to be requested than are larger files.

4.2.2.2 Inter-Reference Times

The next analysis focuses on the inter-reference time for documents that are accessed more than once. The analysis is performed for two different definitions of "time". One definition is "wall-clock time", which uses the timestamps in the access log to determine the number of seconds between repeated accesses to a given document. The second definition is "gap time", the number of other (intervening) document requests that occur between repeated references to a specific document. The advantage of using these two different definitions of time is the ability to distinguish between (wall clock) time-related effects (e.g., user think time), which should be common for all logs, and multiplexing effects (e.g., server load, server response time) that may not be common across all data sets.

The inter-reference times are computed for each unique document that is referenced more than once, and then combined together to form the cumulative distribution of inter-reference times for all documents that are accessed more than once. The cumulative frequency distributions are shown in Figure 4.6.

Figure 4.6(a) clearly illustrates the different workload levels for the six servers. On the lightly-loaded Waterloo server, documents tend to be accessed at long intervals (hours to days). Documents on the busy NCSA server are accessed on a seconds or minutes basis. For inter-reference gaps (Figure 4.6(b)), the relative order (and shape) of the lines changes; the Waterloo site has the smallest set of unique documents and a low level of activity; the ClarkNet site has a much larger set of unique documents, and a higher level of activity. The remaining sites fall somewhere in between.

A separate statistical analysis (shown in Figure 4.7 for the NCSA data) suggests that file inter-reference times (i.e., the time between requests for the same file) are exponentially distributed and independent. However, the distribution of file inter-reference times does not pass the Anderson-Darling goodness-of-fit test [21] for the exponential distribution. The reason for this failure can be attributed to fluctuations in the number of requests by time-of-day, or by day-of-week. In other words, there is too much variation to model the file inter-reference times with a simple, fixed rate Poisson process (this is similar to the problem faced by Paxson for Telnet and FTP session arrivals [55]). The next simplest model, as suggested by Paxson [55], is to postulate that the file inter-reference times during fixed interval lengths can be well modeled by homogeneous Poisson processes. Using this approach, the file inter-reference times were again studied, this time over one hour intervals. With this approach over 95% of the intervals passed the Anderson-Darling Goodness-of-Fit test for exponential arrivals. This observation applies for all six data sets, and is added to Table 1.1 as an invariant.

It is understandable that the inter-reference times for individual files would be exponentially distributed and independent, as each request for a (particular) file is generally from a new user. This behaviour is much like uncorrelated, memoryless behaviours (i.e., a Poisson process) [55].

It is probable that Web session arrivals are also Poisson processes, as each session arrival would indicate a new user. This is similar to FTP session arrivals, which are Poisson [55]. Request arrivals within a Web session would not be Poisson, as they are the actions of one individual. Similarly, Paxson found that FTPDATA connections are not Poisson [55]. However, because there is no method to distinguish between Web session arrivals and request arrivals from the data in the access logs, these statements cannot be confirmed.[4]

[4] Paxson reports a similar problem exists for X11 traffic. [55].

4.2.2.3 Temporal Locality

Access logs were analyzed to look for temporal locality in the file referencing behaviour. Temporal locality refers to the tendency for the same document to be re-referenced frequently within short intervals.

Temporal locality can be measured using the standard LRU (Least Recently Used) stack-depth analysis. When a document is initially referenced, it is placed on top of the LRU stack (i.e., position 1), pushing other documents down in the stack by one location. When the document is subsequently referenced, its current location in the LRU stack is recorded, and then the document is moved back to the top of the stack (pushing other documents down, as necessary). When the entire log has been processed in this fashion, temporal locality in referencing behaviour is reflected in a high probability of references to locations at or near the top of the LRU stack.

Figure 4.8 shows the results of our LRU stack-depth analysis for all six data sets. The Calgary data set shows the highest temporal locality, while the ClarkNet data set shows the least. There is thus no invariant evident in these data sets for temporal locality.

It is possible that the level of multiplexing in a busy Web server is large enough to mask any evidence of temporal locality in the access logs. Client-side caching mechanisms may also serve to remove temporal locality from the reference stream seen at the server, as was demonstrated in [26].

4.2.2.4 Geographic Distribution

The final analysis of file referencing behaviour examines the geographic distribution of document requests. This analysis makes use of the IP addresses of the requesting hosts in the access log. In particular, the network component of the IP address (based on a Class A, Class B, or Class C address) is used to determine if a requesting host is *local* or *remote* (relative to the Web server). The network identifier in each IP address is further used to classify requesting hosts into *domains* (not to be confused

Table 4.8: Geographic Distribution of Requests (All Data Sets)

Local Hosts						
Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
% All Requests	22.3	46.4	24.9	6.3	1.9	1.2
% All Bytes	18.3	36.6	24.8	2.7	1.6	0.5
Remote Hosts						
Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
% All Requests	77.7	53.6	75.1	93.7	98.1	98.8
% All Bytes	81.7	63.4	75.2	97.3	98.4	99.5

with *domain names*) that have the same network address.

Table 4.8 shows the geographic distribution of requests and bytes transferred at the six sites. For example, 77.7% of all the requests to the Waterloo server came from remote hosts, while local hosts generated the remaining 22.3% of the requests. In terms of bytes transferred, 81.7% of the requested bytes were transferred to remote hosts, with 18.3% to local hosts. The rest of the table is organized in a similar manner.

On all six Web servers, remote hosts send the most requests and receive the most data. Remote hosts account for over 75% of requests on all but one server (Calgary), and well over half of the total bytes transferred on all servers. This observation is reported in Table 1.1 as another invariant.

The local access patterns at the Saskatchewan and Waterloo servers are quite similar. This is likely caused by the use of the Web in teaching and research activities. The access patterns at NCSA, NASA, and ClarkNet are similar, with remote accesses accounting for almost all of the requests and transferred data. This is substantially different from the access patterns at Saskatchewan and Waterloo. The likely explanation for this behaviour is that there are very few "local" hosts for these organizations.

Figure 4.9(a) shows the distribution of references by the number of domains accessing the Web server.[5] A small number of domains account for a significant portion of the requests, while the remaining requests are received from several thou-

[5]The Calgary data set is not shown since the "sanitized" log did not show host names or IP addresses.

sand domains. In all six data sets, 10% of the domains accounted for at least 75% of the requests (Invariant 10 in Table 1.1). Similarly, Figure 4.9(b) shows that 10% of the domains are responsible for at least 75% of the bytes transferred by the Web servers.

4.2.3 File Modification Times

Several Web documents appeared in the access logs multiple times, with the same URL each time, but with different file sizes at different points in the logs. These "file modification" events are analyzed separately in this section.

There are two possible causes for these "anomalies" in the Web server access logs. First, a user may edit and physically change a Web document at any time. Second, a Web client may *abort* a connection in the midst of a file transfer (i.e., the user clicks on the "Stop" button on the Web browser).

From the standpoint of access log analysis, aborts appear the same as a file modification because the access logs record the number of bytes transferred by the server, not the actual size of the file that was requested. However, aborted connections *are* reported in the Web server error log. The error log for the Saskatchewan data set was analyzed to determine the effects of user aborts on the file modification data reported in this work.[6]

An analysis was thus performed on file modification times. Figure 4.10 shows the cumulative distribution of file modification times. Most files tend to go unmodified for extended periods of time. The small number of files that are modified only a few seconds after the previous modification are likely the result of a user creating or modifying a Web page, viewing the page, then editing and re-viewing the page until the desired modifications are complete.

Figure 4.11 shows the distribution of file modifications times for the Saskatchewan data set. The "aborts included" curve shows the distribution when aborts

[6]The error logs for the Calgary, ClarkNet, NASA and Waterloo Web servers were not available. Since the error logs were not available for these four data sets, the error log for the NCSA server was not analyzed (although it was available).

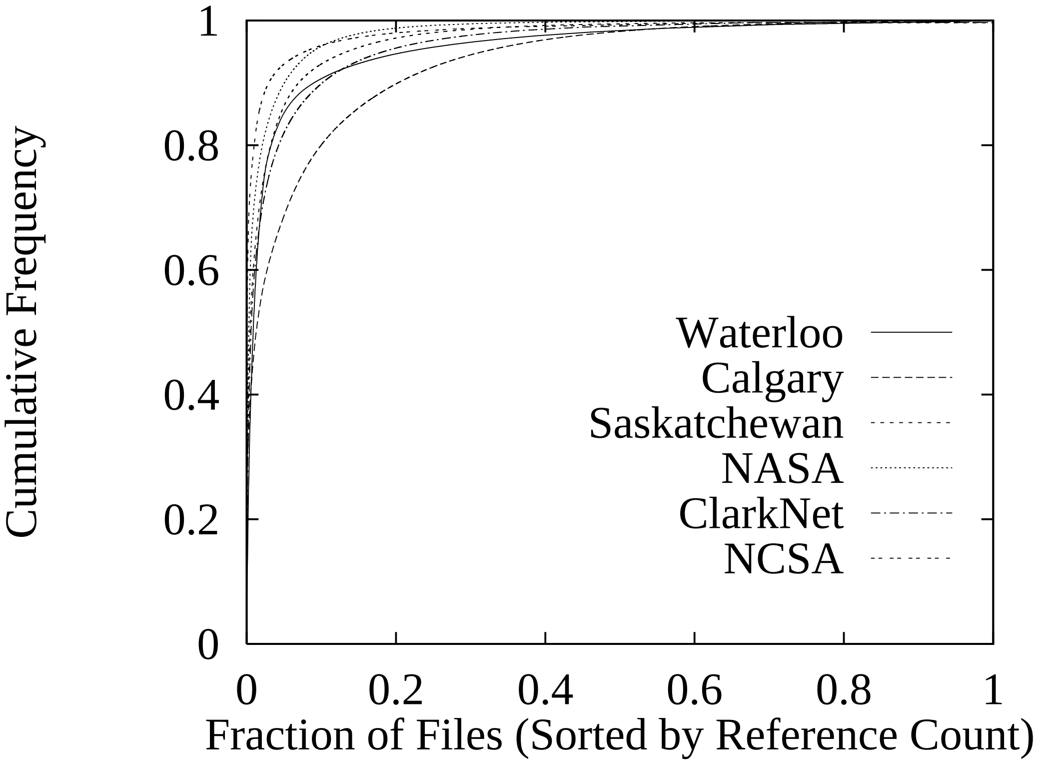
are considered to be file modifications, while the "aborts removed" curve gives the distribution of file modification times when all aborted connections (1.1% of all requests) have been removed. The "aborts removed" curve is significantly flatter, showing that files are modified less often than the raw data in the logs suggests. Removing the aborted connections from the other five logs would likely show similar results.

Table 4.9 summarizes information about the aborted connections in the Saskatchewan data set. While the number of aborted connections is quite low, the number of bytes transferred by aborted connections is relatively large. Furthermore, remote users are more likely to abort a connection than are local users, as expected.

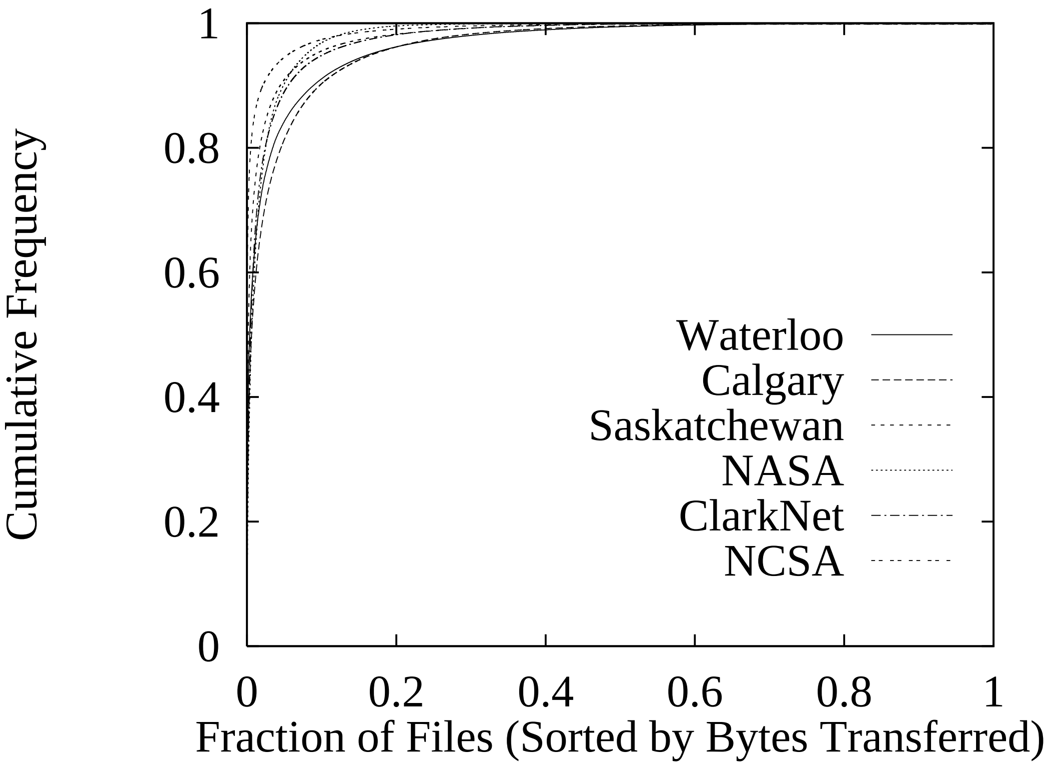
Table 4.9: Aborted Connections (Saskatchewan Data)

Item	Local	Remote	All
% of Connections Aborted	0.9	1.2	1.1
% of Bytes Transferred	4.0	5.7	5.1

Figure 4.12(a) shows the number of aborted connections versus the size of the file being transferred. This figure shows that most of the aborted connections were for small files. However, when the probability of a connection being aborted (based on the size of the file being transferred) is calculated, it becomes clear that larger files (i.e., files $>$ 100,000 bytes) are more likely to be aborted than are smaller files. This relationship is shown in Figure 4.12(b). The correlation coefficient for this data set is 0.13, which confirms the statement that connections transferring larger files have a slightly higher probability of being aborted than do those connections transferring smaller files. Thus, there is statistical evidence of a "user impatience" factor.



(a)



(b)

Figure 4.3: Concentration of References: (a) Sorted by Number of Requests, and (b) Sorted by Bytes Transferred

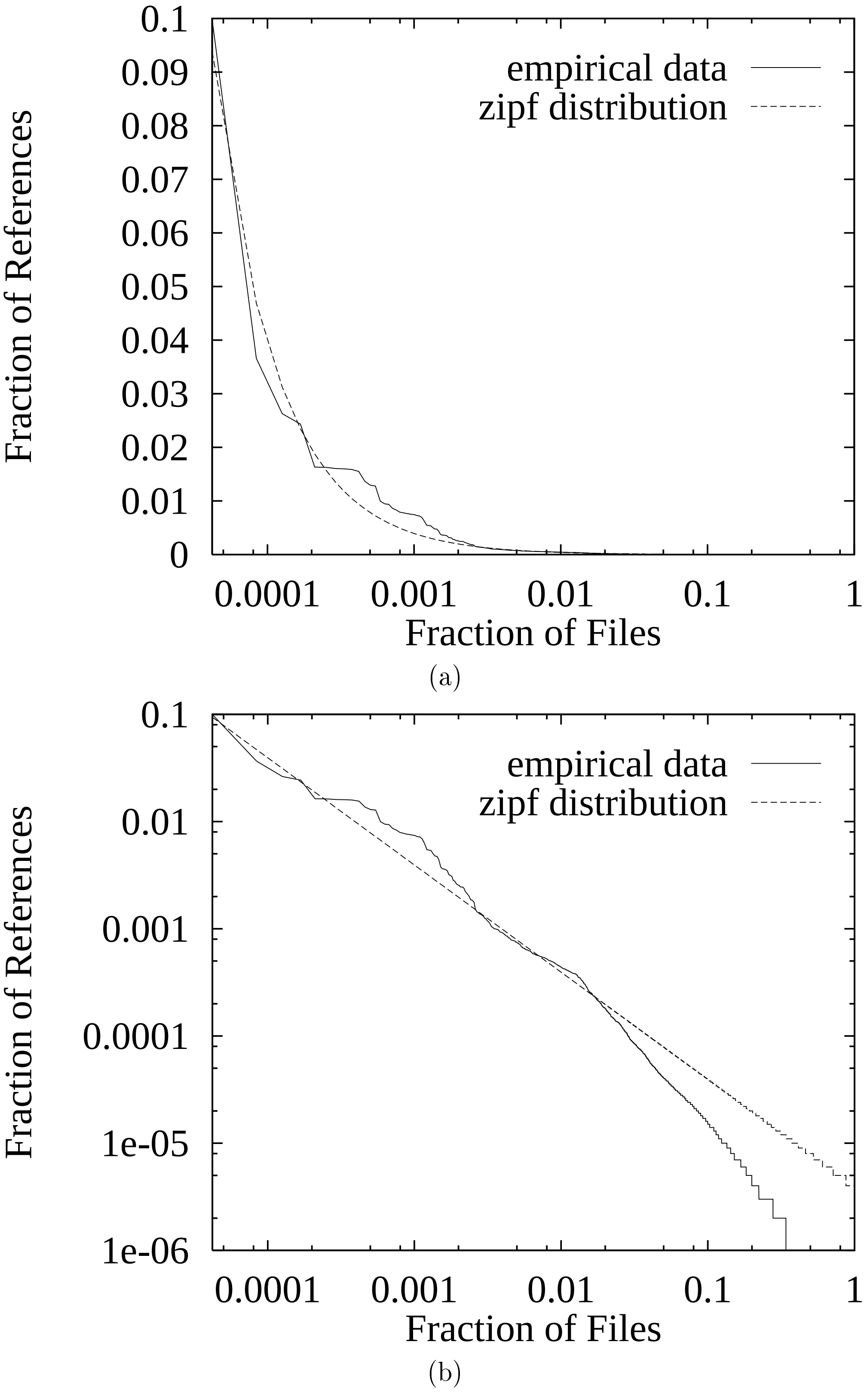


Figure 4.4: Comparison of Empirical and Zipf Distribution of Frequency of Reference (NCSA Data)

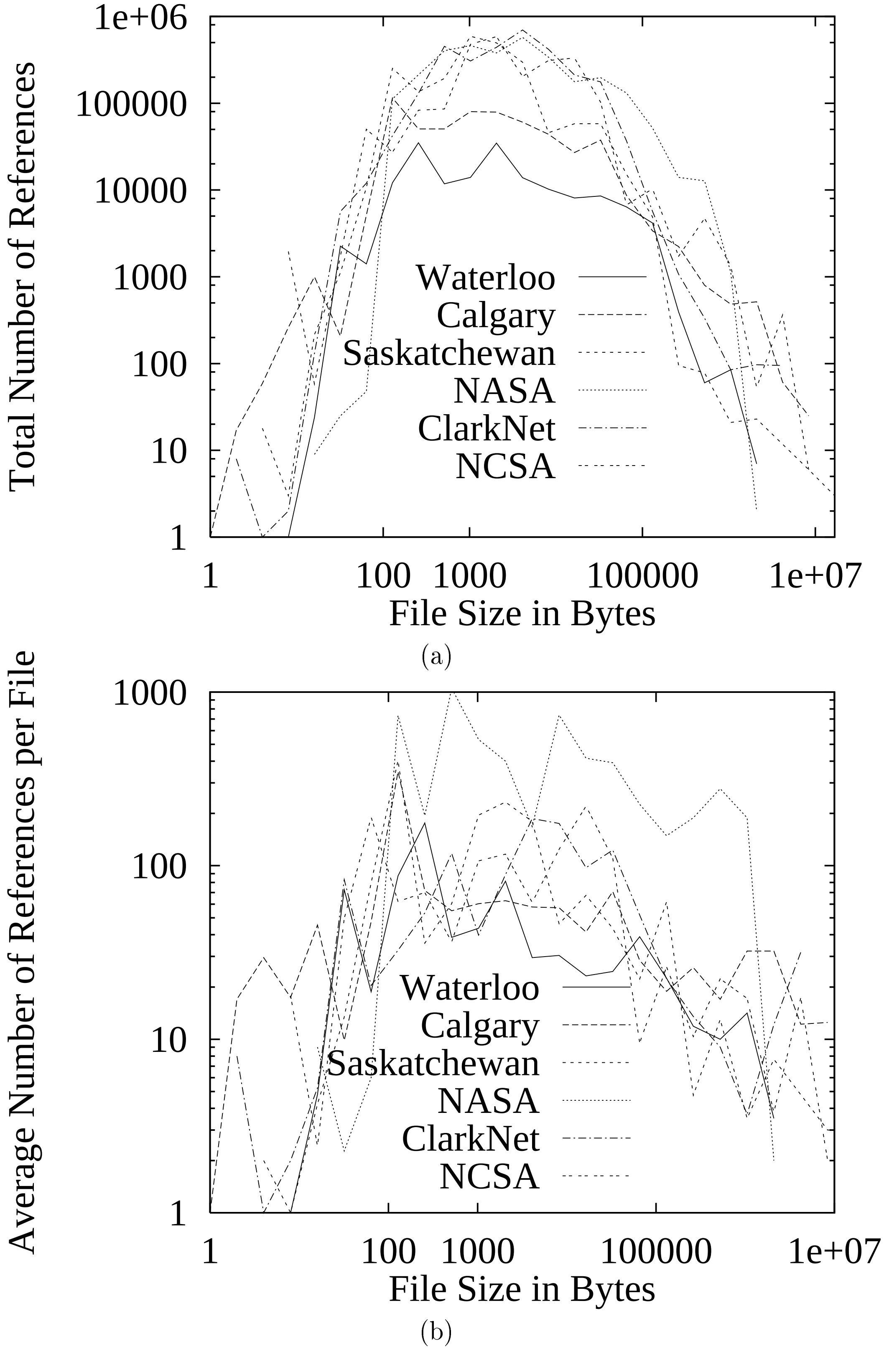
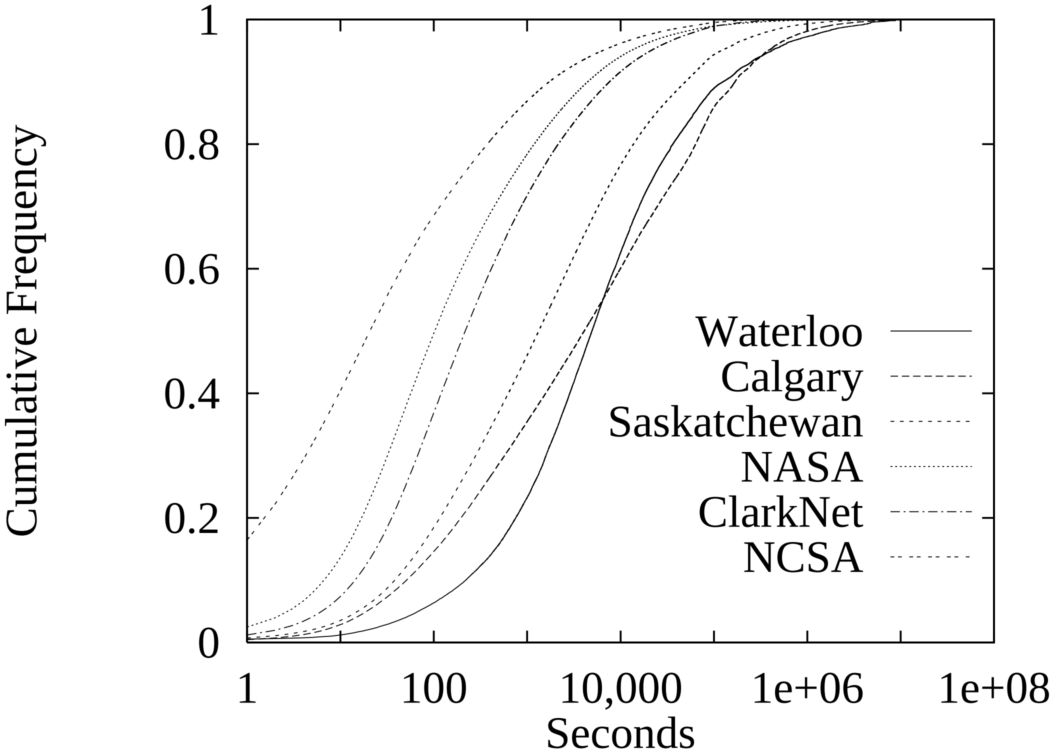
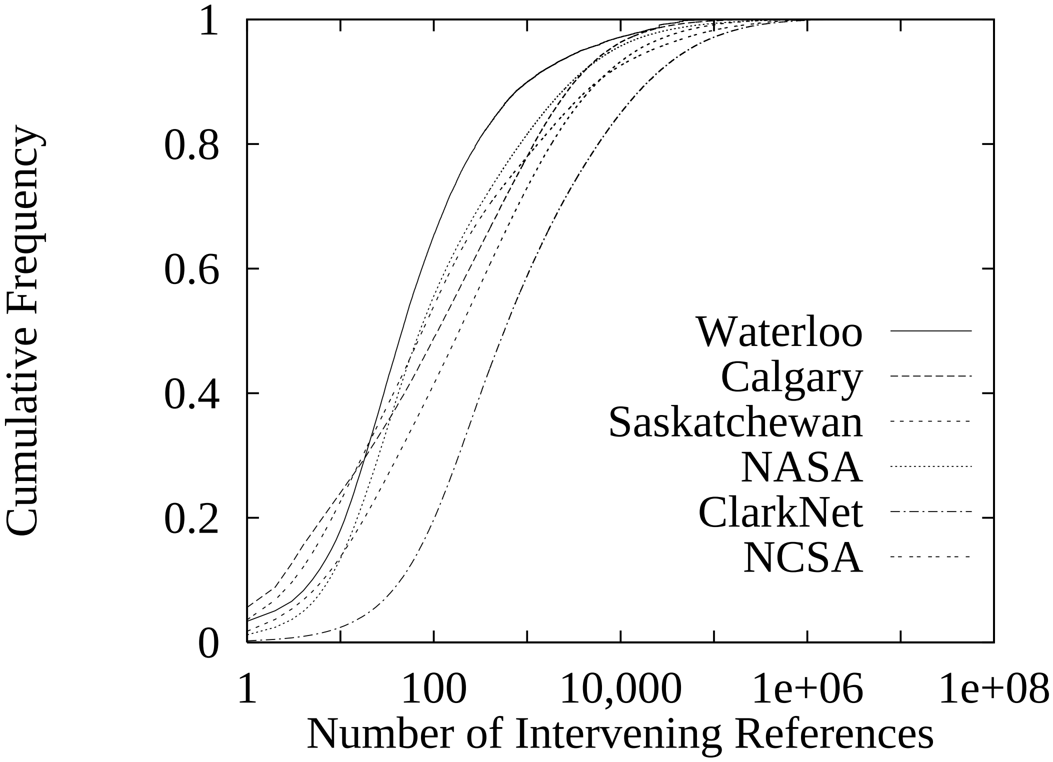


Figure 4.5: References to a File as a Function of File Size: (a) Aggregate Data (b) Normalized Data



(a)



(b)

Figure 4.6: Distribution of: (a) Inter-reference Times, and (b) Inter-reference Gaps

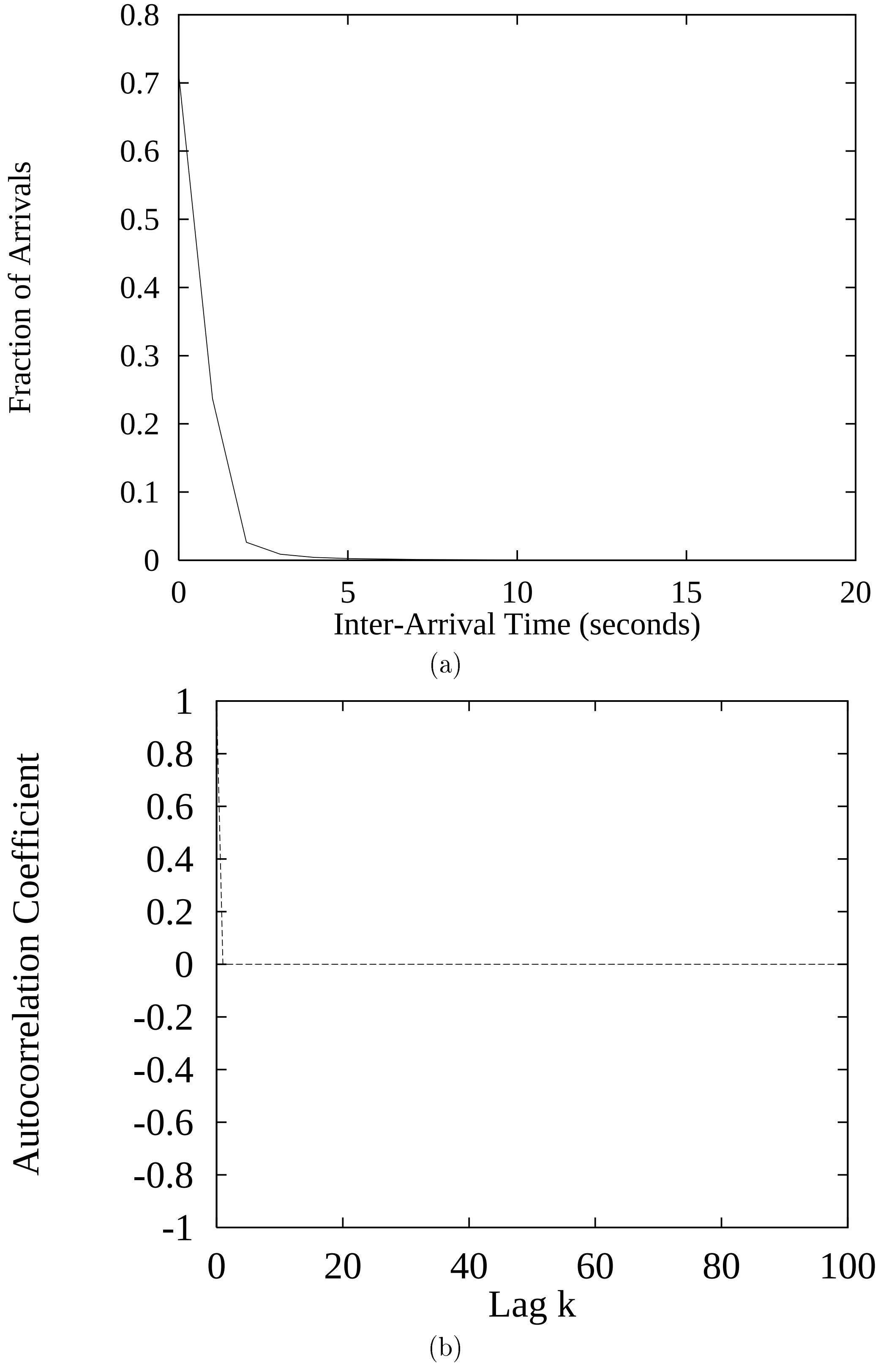


Figure 4.7: NCSA Inter-Reference Times: (a) Frequency Distribution (b) Autocorrelation

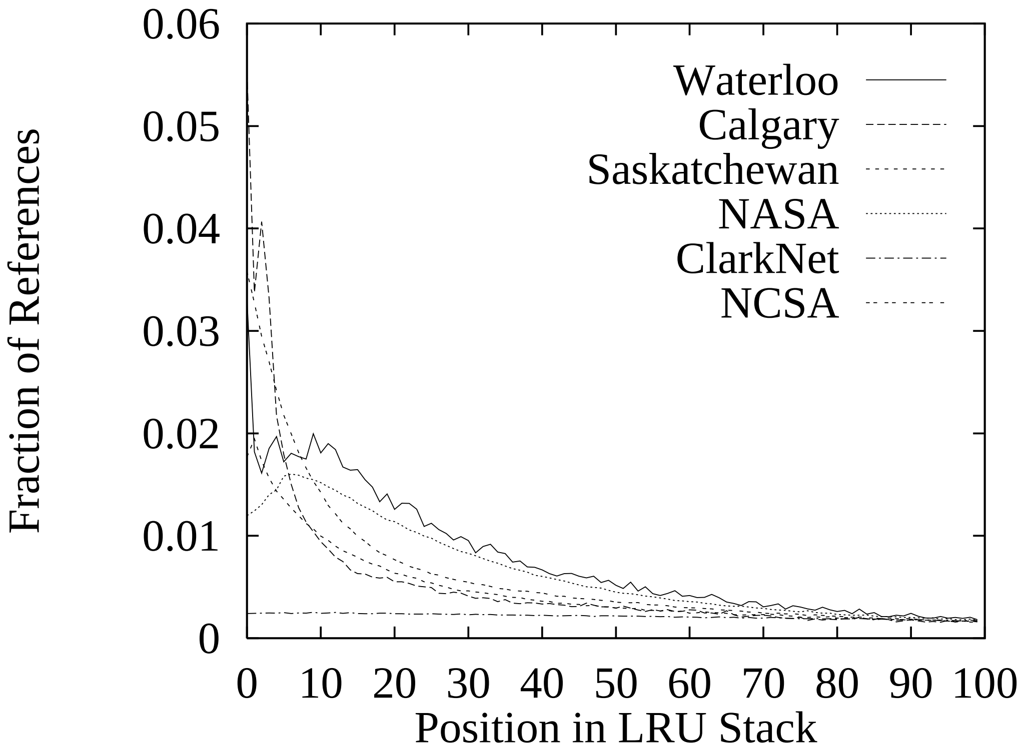


Figure 4.8: Temporal Locality Characteristics

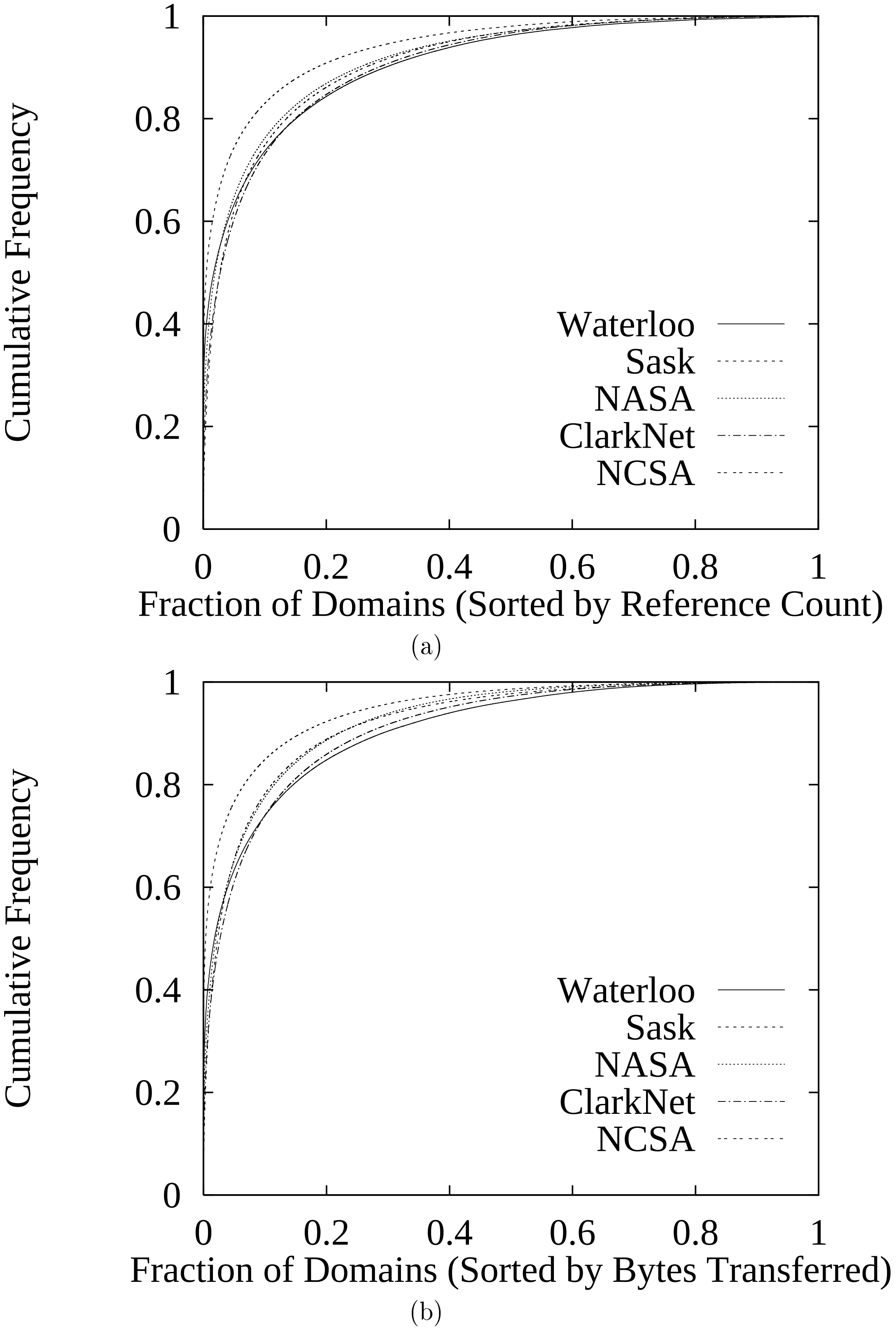


Figure 4.9: (a) Distribution of References by Domain, and (b) Distribution of Bytes Transferred by Domain

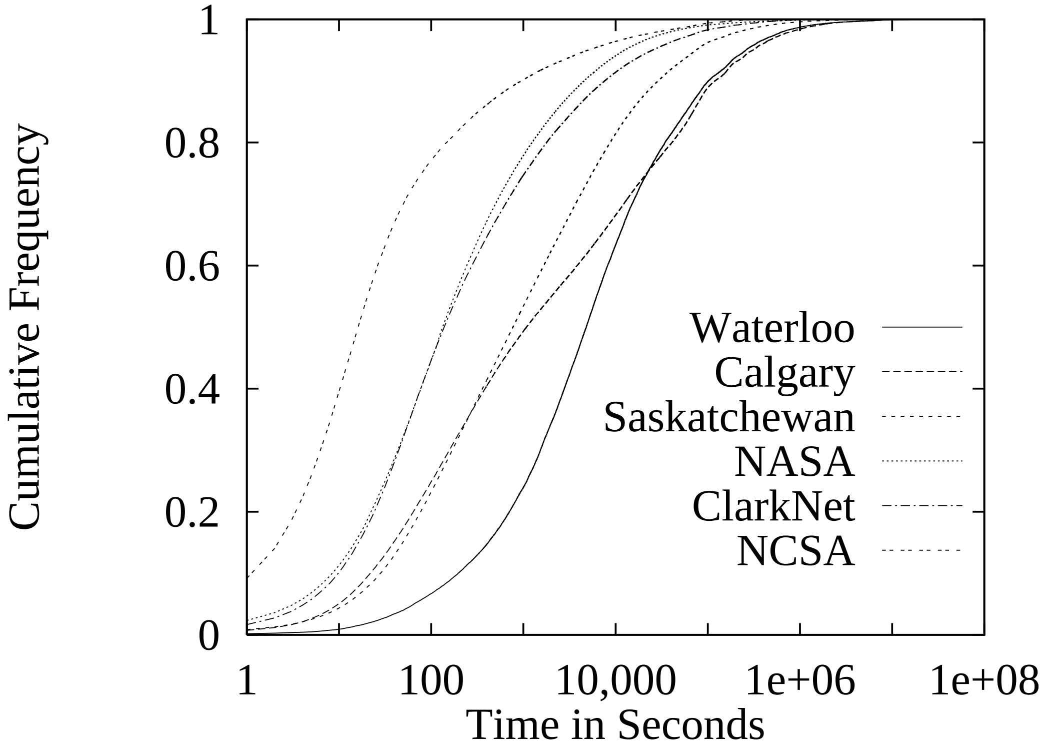


Figure 4.10: Cumulative Distribution of File Modification Times

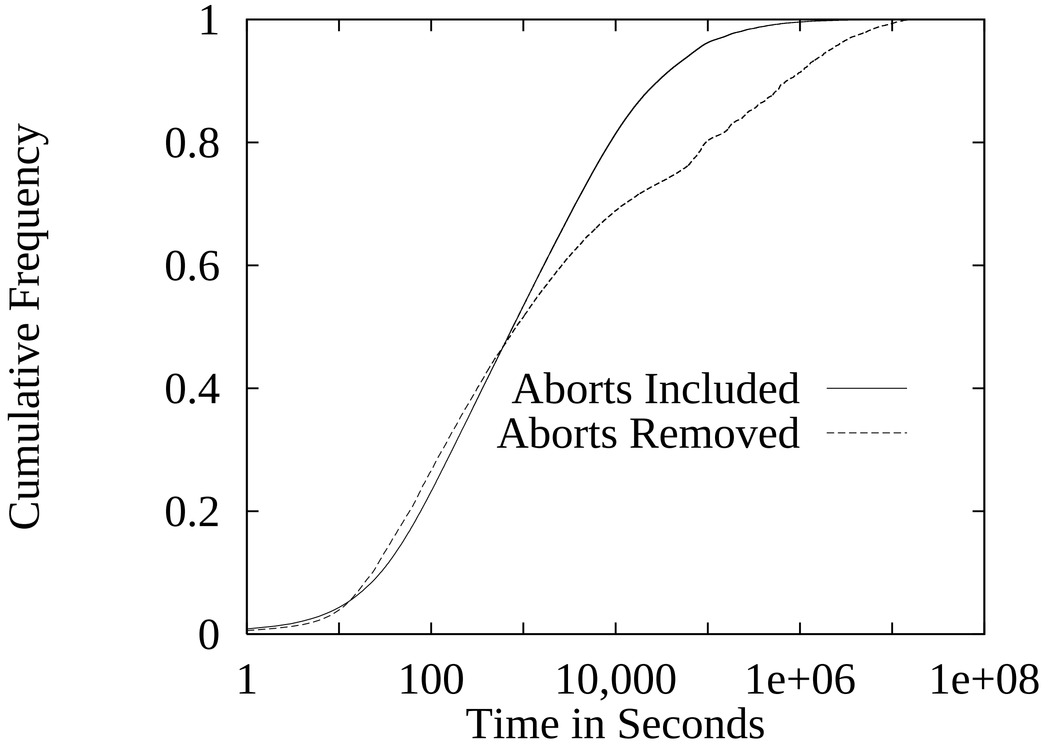
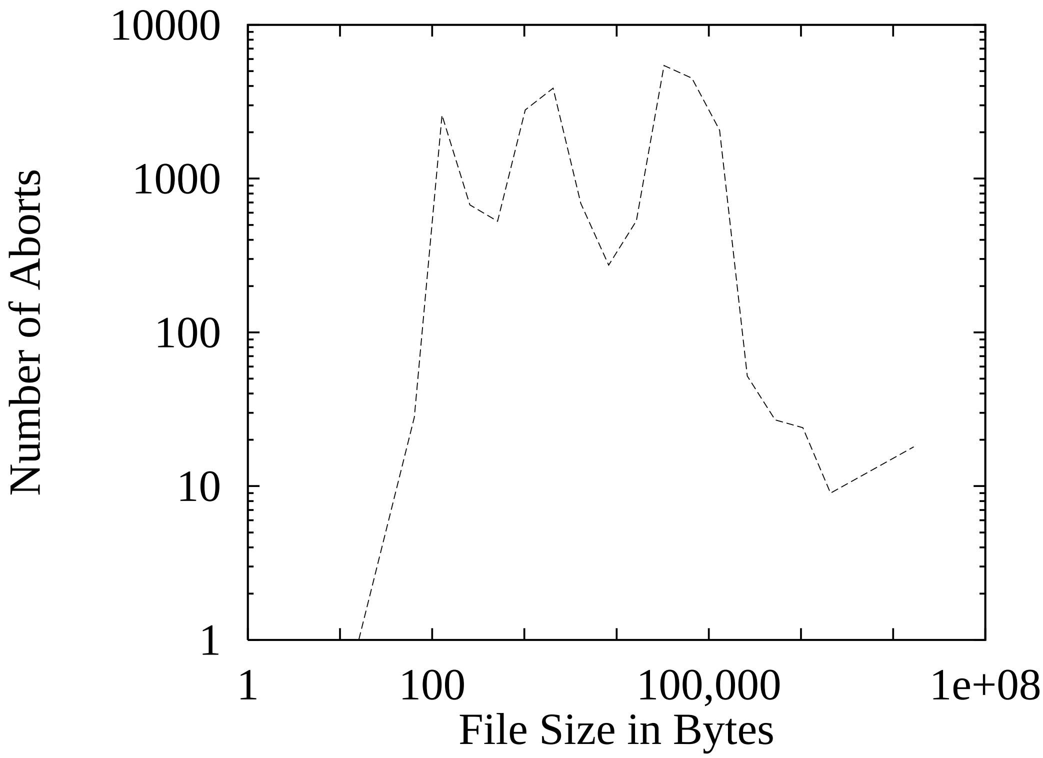
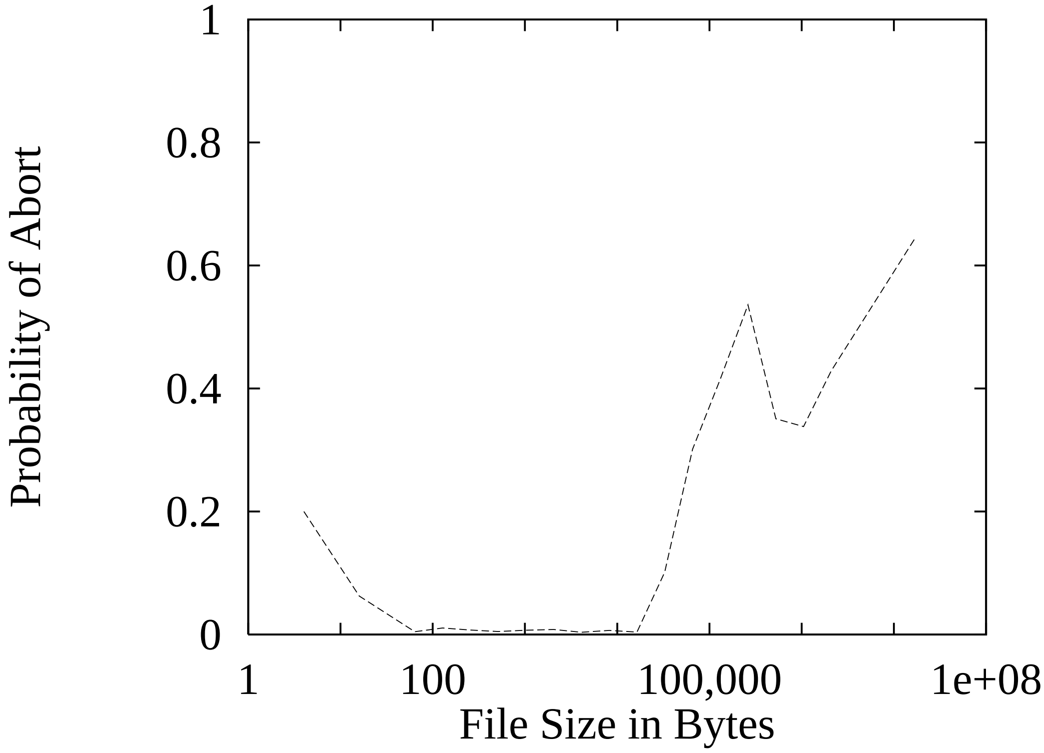


Figure 4.11: Cumulative Distribution of File Modification Times (Saskatchewan Data)



(a)



(b)

Figure 4.12: Probability of Aborted Connections as a Function of File Size: (a) Aggregate (b) Normalized (Saskatchewan Data)

4.2.4 Self-Similarity

Recent work has suggested that World-Wide Web traffic may be self-similar [18]. This section presents an analysis to test for the presence of self-similarity in Web server workloads. Section 4.2.4.1 presents the main characteristics of self-similar processes. Section 4.2.4.2 discusses the impact of self-similar processes on performance. Section 4.2.4.3 lists the procedures used to identify self-similar processes. Section 4.2.4.4 provides the results of the tests that were performed to check for self-similarity in the workloads of the Web servers under study.

4.2.4.1 Characteristics of Self-Similarity

There are two important characteristics of self-similar processes. One characteristic is burstiness across many time scales. For network traffic, this means that the traffic does not get smoother when it is aggregated [18, 39, 55, 71]. A second characteristic of self-similar processes is long-range dependence. This means that the observed values for traffic volume (e.g., bytes per one second interval) at any time instant are correlated with values at all future instances. This long-range dependence is identified by a hyperbolically decaying autocorrelation function [18, 39, 55, 71].

4.2.4.2 Impact of Self-Similar Traffic

Traditional network models have not modeled the burstiness of traffic across many time scales, nor have they modeled the long-range dependence that exists within the traffic. The failure of these models to capture these characteristics have caused performance studies to be overly optimistic, meaning that performance measures like packet delay or maximum queue size are drastically underestimated [39, 55, 71]. Thus, it is important to determine if self-similarity exists in Web server workloads, in order to perform more realistic studies.

An attractive feature of self-similar processes is that only a single parameter, called the Hurst parameter H, is needed to model the degree of self-similarity. Possible values for the Hurst parameter are in the range $0.5 < H < 1.0$ [55]. For processes that have only short range dependence (e.g., Poisson), a Hurst parameter of $H = 0.5$ would be used. As the degree of self-similarity increases, the value of H would increase towards 1.0.

4.2.4.3 Testing for Self-Similarity

A number of tests are used to identify self-similar processes. Since these tests are thoroughly described in the literature [18, 39], only a short overview is given here.

The first test is to determine the autocorrelation function. As mentioned in Section 4.2.4.1, self-similar processes have long-range dependence. This long-range dependence can be identified by a hyperbolically decaying autocorrelation function [18, 39, 55, 71]. If no evidence of long-range dependence is found, then the process is not self-similar.

If evidence of long-range dependence is found, several additional tests are conducted. First is the Variance-time plot, which plots the observed variance in the traffic arrival process versus the time window of observation, on a log-log scale. This graph is used to detect slowly decaying variance. For a self-similar process, the resulting plot should be a straight line with slope greater than -1 [18, 39]. A second test, called the rescaled range statistic or R/S plot (log-log scale), is used to acquire an estimate for the Hurst parameter H [18, 39].

4.2.4.4 Results of Self-Similarity Tests

For this thesis, four tests (a visual test, to look for burstiness over different time scales, and three statistical tests, described in Section 4.2.4.3) were performed to identify if long-range dependence and self-similarity existed in Web server workloads. Only the analysis of the ClarkNet data is described here.

The first test, a visual inspection of the bytes transferred per unit of time by the Web server, was used to identify burstiness across different time scales. The results are shown in Figure 4.13, for four different time scales. The topmost graph (1000 second intervals) shows a full week of ClarkNet data, in which a distinct daily usage pattern is seen. Each subsequent graph "zooms in" on a selected (shaded) subinter-

Table 4.10: Values for the Hurst Parameter H, for All Data Sets

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
H	0.51	0.54	0.55	0.63	0.65	0.60

val of the previous graph with a finer time resolution. The bottom graph shows the workload per 1 second interval (the resolution of the access log timestamps). The results suggest that there is some evidence of self-similarity in this workload.

The three statistical tests described in Section 4.2.4.3 are used to confirm this intuition. Since these tests need to be performed on a stationary process [71], an eight hour period of data was used. Figure 4.14(a) shows the autocorrelation function for the ClarkNet data set. While this graph does not appear to be decaying hyperbolically, Figure 4.14(b) (a more detailed view of the same graph) shows that the autocorrelation function remains non-zero, indicating that there is some long-range dependence. Figure 4.15(a) shows the results of the Variance-time plot. This plot has a slope flatter than -1 (the dotted line in the figure), which is a property of a self-similar process [18]. Figure 4.15(b) shows the results of the final test, the R/S plot. The data points generated from the analysis form a band with slope greater than 0.5 (the lower dotted line in the figure) and less than 1.0 (the upper dotted line in the figure). Linear regression on this plot results in a Hurst parameter estimate of $H = 0.65$, suggesting a small degree of self-similarity in the ClarkNet server workload. Table 4.10 gives the estimates of the Hurst parameter H for each of the Web server workloads under study. From this table it does not appear as though self-similarity is an invariant in all Web server workloads, though it does appear to be a property when Web traffic is heavy, as reported in [18].

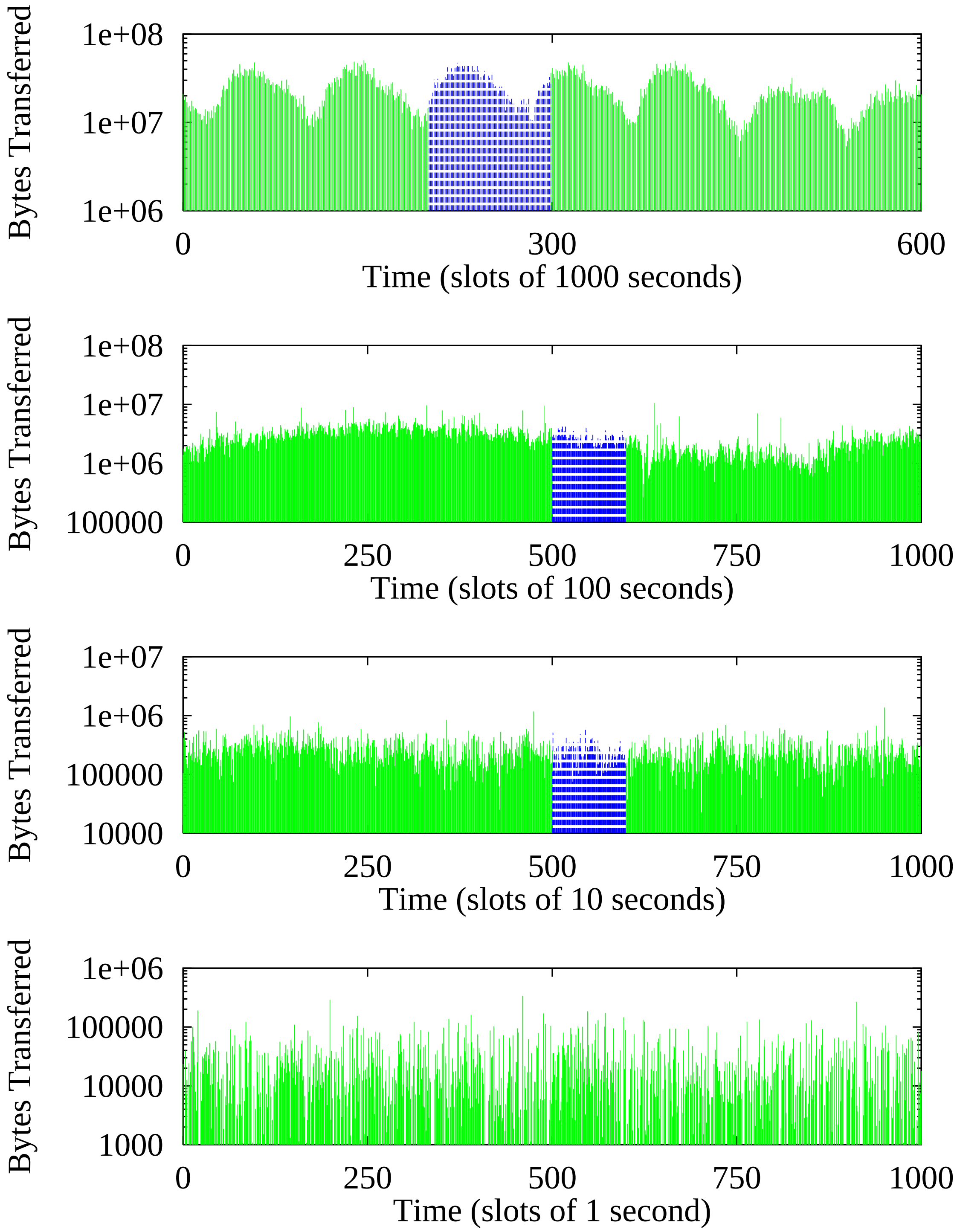
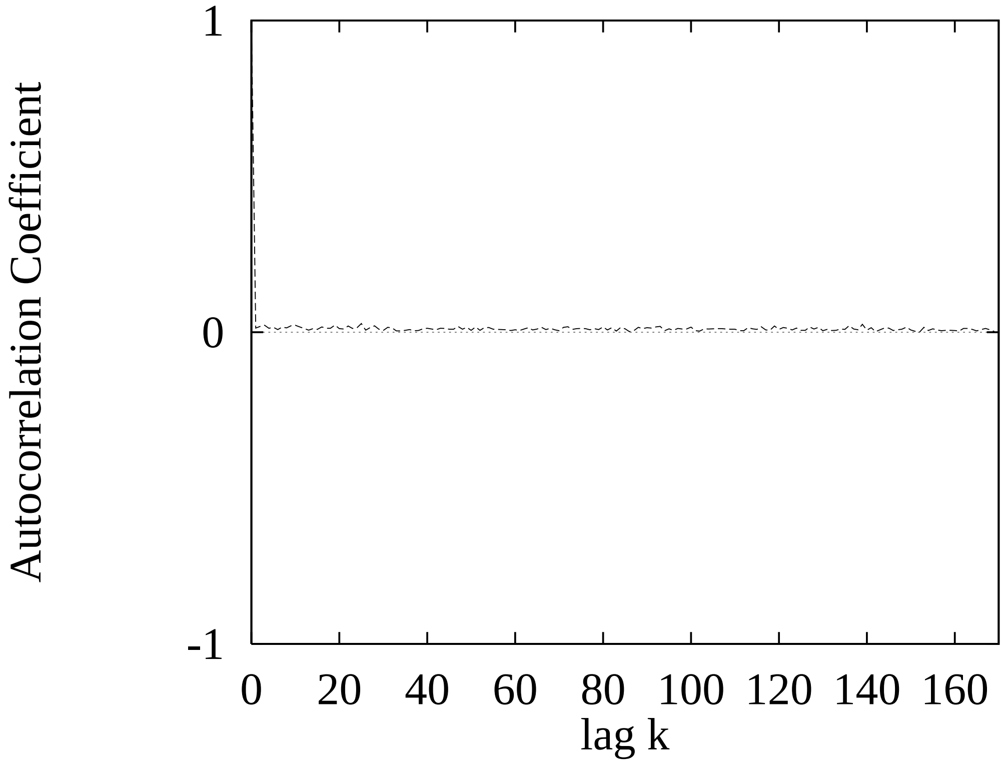
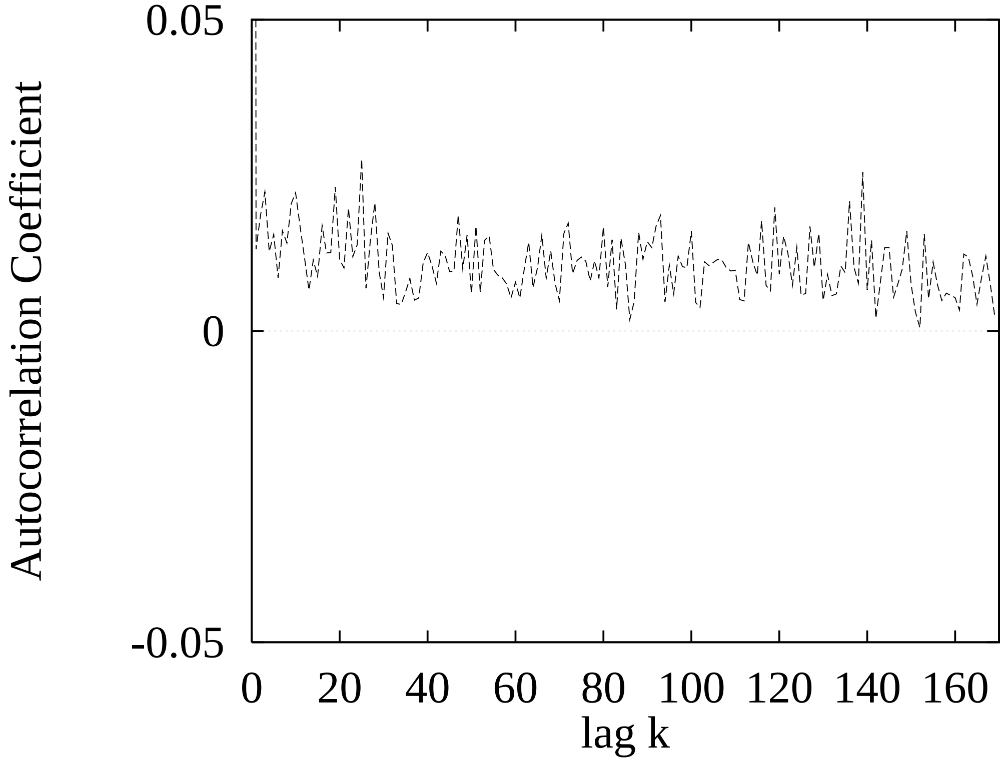


Figure 4.13: Server Workload at Four Different Time Scales (ClarkNet Data)

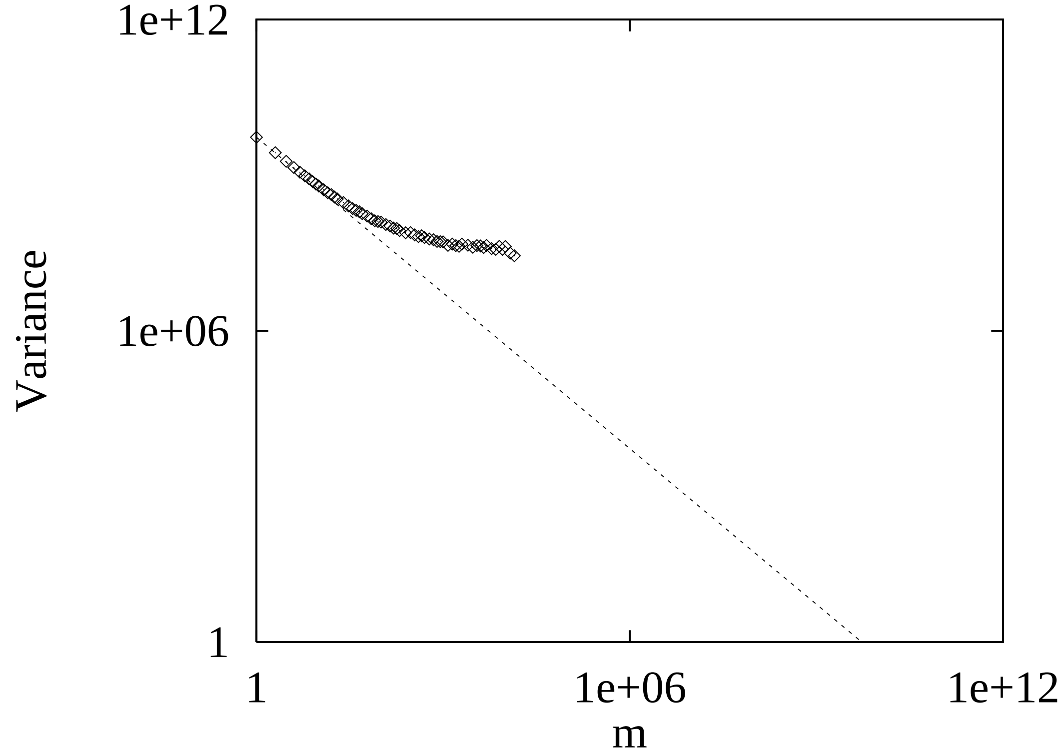


(a)

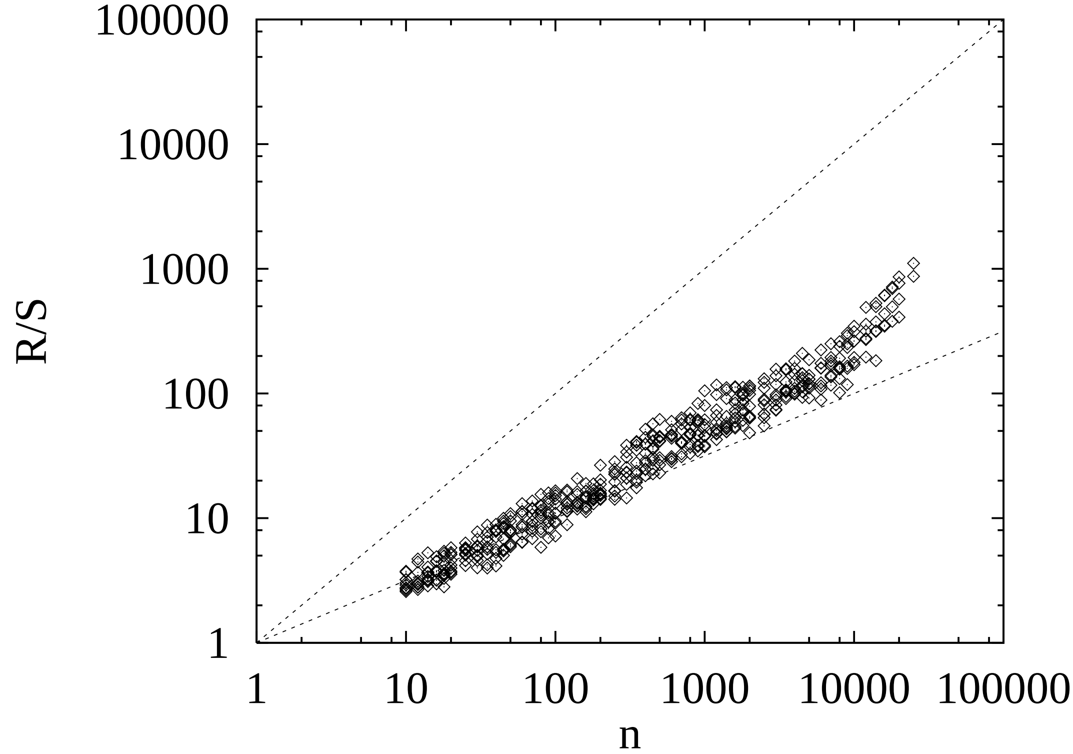


(b)

Figure 4.14: Self-Similarity Tests: (a) Autocorrelation (b) Detailed Autocorrelation (ClarkNet Data)



(a)



(b)

Figure 4.15: Self-Similarity Tests: (a) Variance-Time (b) R/S Plot (ClarkNet Data)

4.2.5 Summary

This section has presented a detailed study of Web server workload characteristics. Results were presented for file size distributions, file referencing patterns, file modification times, and self-similarity in Web server workloads.

From the analyses reported in this section, five additional workload invariants have been identified. These invariants are summarized in the last five rows of Table 1.1.

4.3 Performance Implications

This chapter concludes with a discussion of performance issues (particularly caching issues) for World-Wide Web servers. Caching is clearly a promising approach to improving Web performance because of the large number of references to a small number of documents (Invariant 4 from Table 1.1), the concentration of references within these documents (Invariant 7), and the small average size of these documents (Invariant 3). The location of the cache is intentionally left unspecified (e.g., at the client, at the server, or in the network) as well as its size (e.g., Megabytes or Gigabytes), focusing instead on the use of the workload invariants to estimate the maximum performance improvement possible with Web server caching. For simplicity, the discussion below assumes that all Web documents are read-only (i.e., never modified), and that file-level caching (not block-level caching) is used. Misses due to "cold start" are also ignored.

4.3.1 A Basic Tradeoff: Requests versus Bytes Transferred

There are two main elements that affect the performance of a Web server: the number of requests that a server must process, and the number of data bytes that the server must transfer (i.e., disk I/O's, packets).

There is thus a choice to be made between caching designs that reduce the number of requests presented to Internet Web servers, and caching designs that reduce the volume of network traffic[7]. Both approaches represent possible avenues for improving Web server performance, but optimizing one criterion does not necessarily optimize the other. The choice between the two depends on which resource is the bottleneck: CPU cycles at the server, or network bandwidth.

Figure 4.16 illustrates this tradeoff. While the discussion here focuses only on the ClarkNet data set, similar observations apply for the other data sets (see Figures 4.17 to 4.21).

The topmost graph (Figure 4.16(a)) illustrates the relationship among the size of files on a Web server (from Figure 4.1), the number of references to those files, and the number of data bytes that references to these files generate (i.e., the "weighted value" obtained from the product of file size and number of times that a file is referenced). This graph shows that 80% of all the documents requested from the ClarkNet server were less than 10,000 bytes in size. 76% of all references to the server were for files in this category. Thus, caching a large number of small files would allow the server to handle most of the requests in a very efficient manner. However, Figure 4.16 also points out that the references to files less than 10,000 bytes in size generate only 26% of the data bytes transferred by the server. Furthermore, looking at the tail of the distribution, documents over 100,000 bytes in size are responsible for 11% of the bytes transferred by the server, even though less than 0.5% of the references are to files in this category (Invariant 6). What this means is that in order to reduce the number of bytes transferred by the server as much as possible, a few large(r) files would have to be cached. That is, the server must sacrifice on "cache hits" for many (small) requests in order to save on bytes transferred for large requests.

The remaining two plots in Figure 4.16 illustrate the tradeoff in more detail. The middle plot (Figure 4.16(b)) show the results for a cache designed to maximize cache hits for requests (i.e., to reduce the number of requests to the server). In this graph, the top line represents the cache hit rate for requests, the bottom line represents the

[7]Clearly, reducing the number of requests also reduces the volume of network traffic, but the main focus of the two approaches is different, as will be shown.

cache size, and the middle line represents the potential savings in bytes transferred by the server when the cache is present. In this design, for example, caching 10% of the server's distinct files (namely, the most frequently accessed documents) for the ClarkNet data set results in a cache hit rate of 90% (the top line in the graph). The documents in the cache, which represent the potential savings in bytes transferred, account for 84% (the middle line in the graph) of the bytes transferred by the server, and the cache size would need to hold 8.3% (the bottom line in the graph) of the total distinct bytes referenced in the server access log.

The bottom plot (Figure 4.16(c)) represents the results for a cache designed to reduce bytes transferred. In this graph, the top line represents the savings in bytes transferred, the bottom line represents the cache size, and the middle line represents the cache hit rate. In this design, for example, caching 10% of the server's files (namely, the 10% of the documents that account for the most bytes transferred) results in an 82% cache hit rate (the middle line). The documents in the cache would account for 95% (the top line) of the bytes transferred, but the cache would have to be large enough to contain 52% (the bottom line) of the distinct bytes represented in the server access log. Clearly there is a tradeoff to be made in cache size, cache hit rate, number of server requests, and number of bytes transferred by the server.

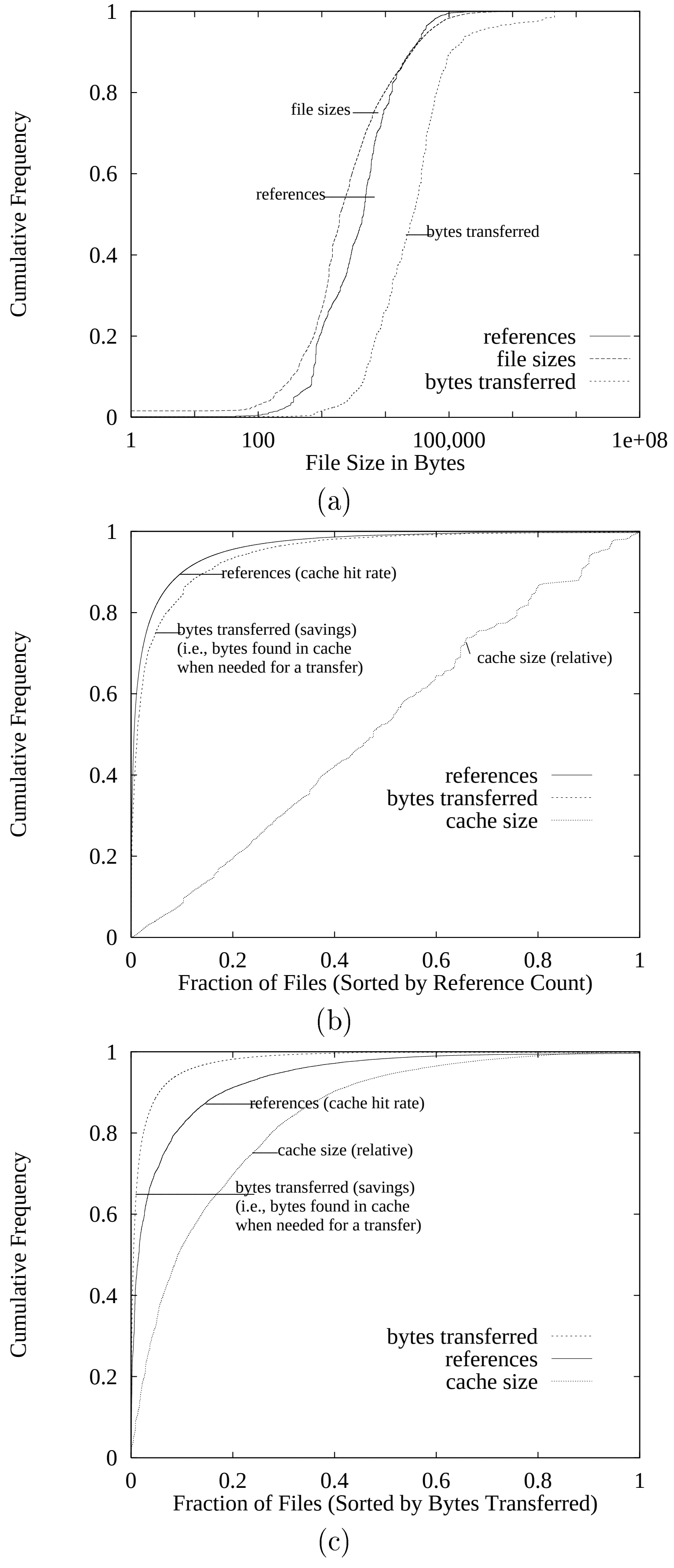
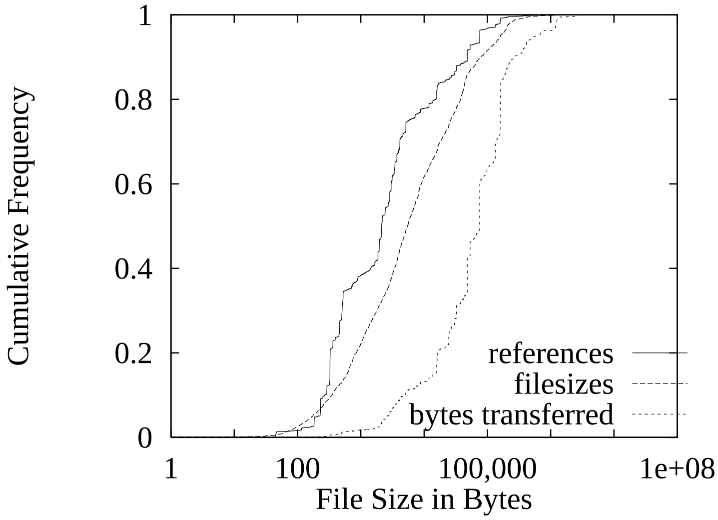
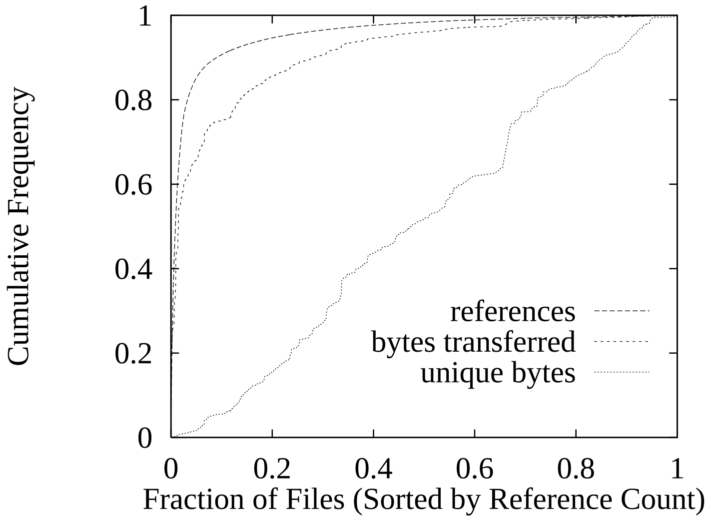


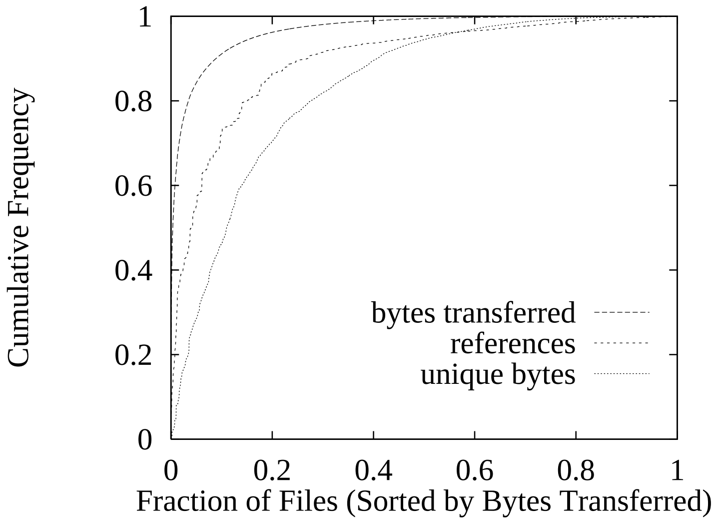
Figure 4.16: Comparison of Caching and Performance Issues (ClarkNet Data)



(a)



(b)



(c)

Figure 4.17: Comparison of Caching and Performance Issues (Waterloo Data)

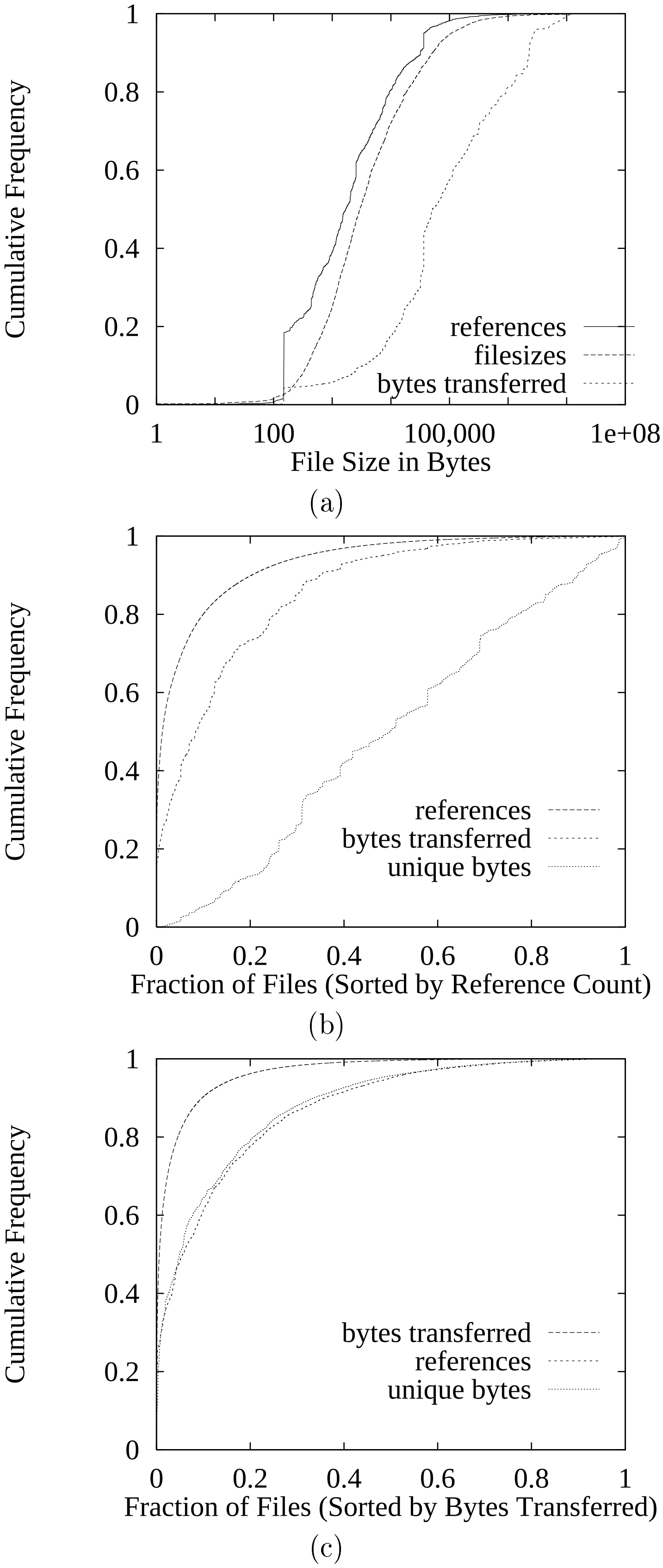
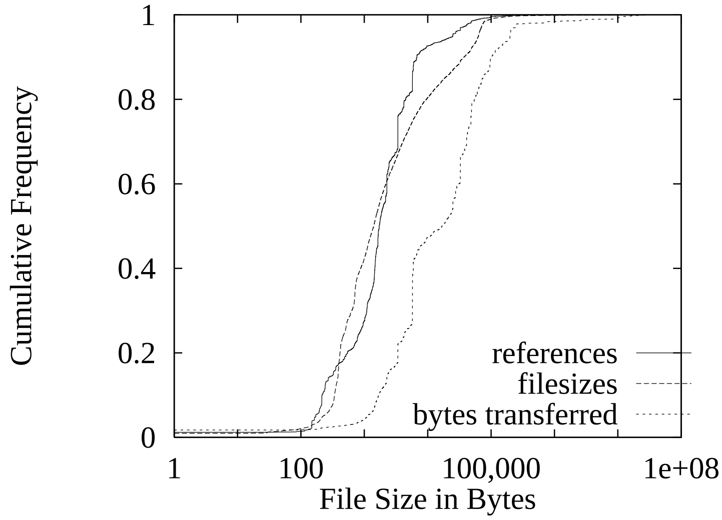
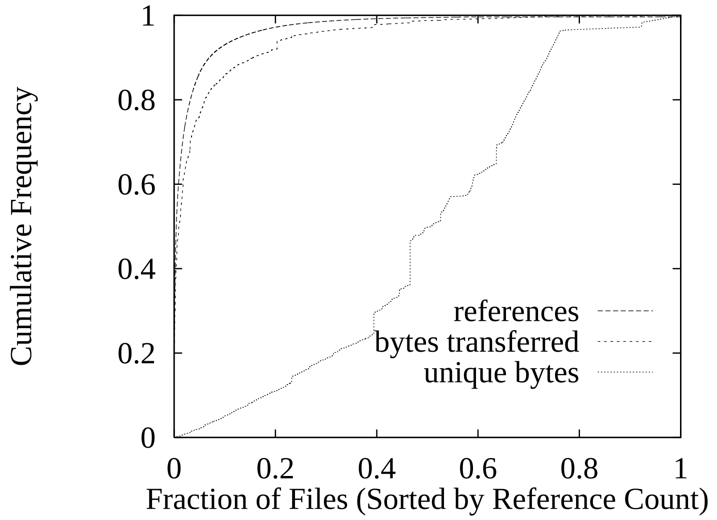


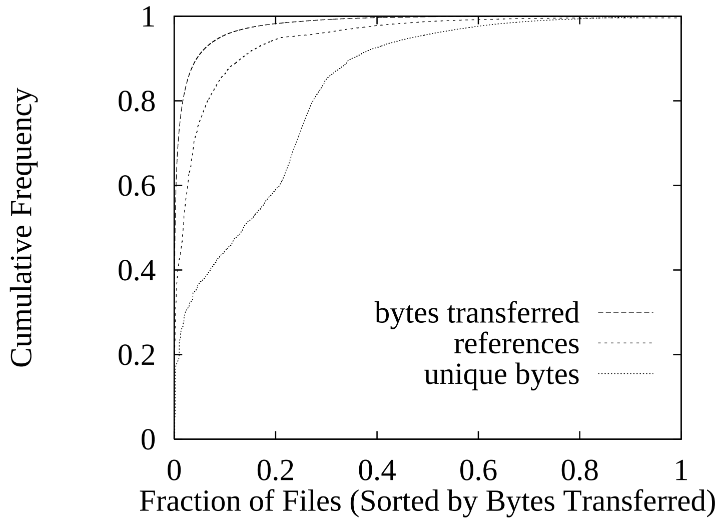
Figure 4.18: Comparison of Caching and Performance Issues (Calgary Data)



(a)



(b)



(c)

Figure 4.19: Comparison of Caching and Performance Issues (Saskatchewan Data)

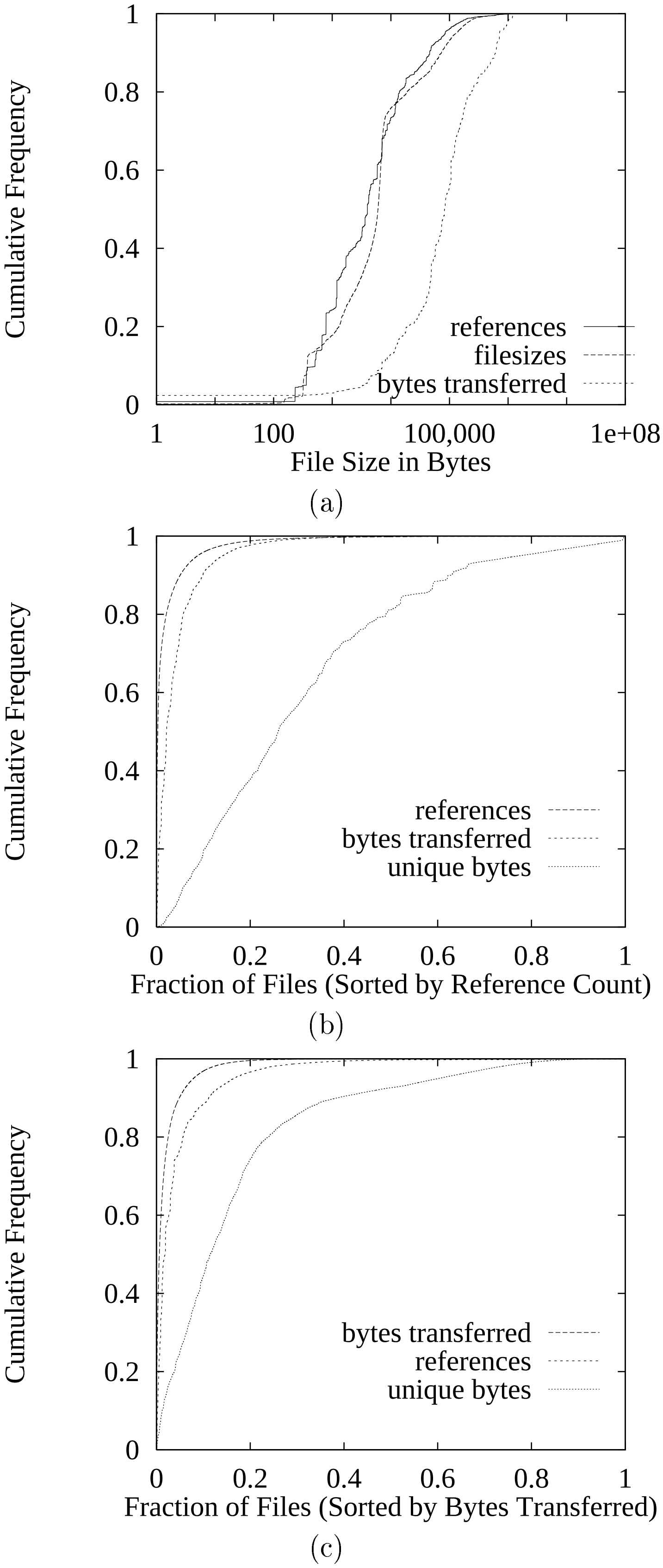


Figure 4.20: Comparison of Caching and Performance Issues (NASA Data)

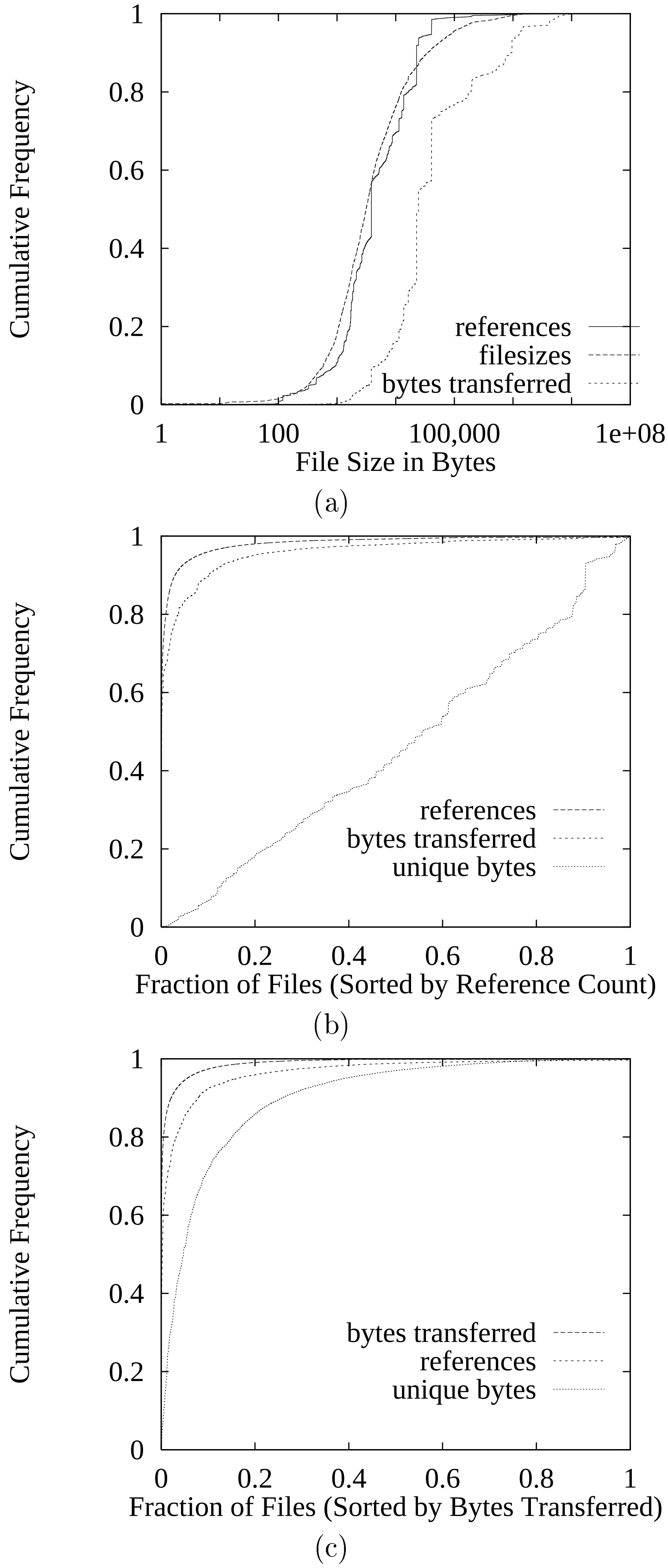


Figure 4.21: Comparison of Caching and Performance Issues (NCSA Data)

4.3.2 Other Issues

The final comments in this section concern "one timers", cache replacement strategies, as well as thresholding and partitioning approaches to cache management. The points raised here are investigated further in Chapters 5 and 6, using the available server workloads.

First, the "one time" referencing nature (Invariant 5 from Table 1.1) of Web documents is a concern.[8] While this effect could be simply an artifact of the finite duration of the access logs studied, or something as innocent as the deletion or renaming of Web documents, the effect is present across *all* access log time durations studied. This one-time referencing behaviour means that if a poor replacement policy is chosen, on average, one-third of a server cache could be cluttered with useless files. Techniques to expunge such files from a cache, such as timeouts on cached files, are highly desirable. Invariant 8 may be useful in setting proper timeout values for documents in the cache. Another approach would be to cache only on the *second* reference to a file within a specified time period, rather than the first.[9]

Second, the fact that temporal locality was *not* present in all data sets suggests that a locality-based cache replacement policy such as LRU may not work well for *all* servers. Frequency-based policies such as Least Frequently Used (LFU) may be more attractive because of the concentration of references (Invariant 7), and also because LFU easily deals with one-timers. Another possible choice is to use a hybrid policy, such as Frequency Based Replacement (FBR) [59].

Third, there may be merit in using "size thresholds" in cache management, to better cope with the "heavy tailed" Pareto distribution of file sizes (Invariant 6), and the issues raised in Section 4.3.1. For example, two such threshold policies might be "never cache a document larger than X bytes" (because it uses up too much cache space, and adversely impacts hit rate), or "never cache a document smaller than Y

[8]The advent of *Web crawlers* may change Invariant 5 to be "N timers", for some small integer N. However, the argument made here still applies.

[9]For N timers, the document would only be cached if it is requested $N+1$ times in the given time period.

bytes" (because it does not save much on bytes transferred by the server).

Fourth, dividing the cache into partitions for different document types may help the performance of the cache. For example, 90 - 100% of the requests to each Web server were for HTML and Image files (Invariant 2). Having a partition in the cache for HTML and Image files would protect these documents from being replaced by less popular files of a different type.

Finally, as a small but practical matter, Web servers should avoid doing name lookups for *each* incoming client request when producing the access log, particularly when successive requests are from the same requesting host. That is, servers should exploit whatever temporal locality exists in the incoming reference stream of requesting hosts to avoid the (slow) name lookup whenever possible. A small cache (e.g., 20 entries) of the results of recent name lookups should suffice (e.g., 75% hit rate). This small refinement alone may help improve response times for heavily loaded Web servers and on associated network services (like name servers). Turning off the name server lookup feature is another option [66].

4.4 Summary

This chapter has presented a detailed workload characterization for World-Wide Web servers. This study used logs of Web server accesses at six different sites: three from university environments, two from scientific research organizations, and one from a commercial Internet provider. The logs represent three different orders of magnitude in server activity, and span two different orders of magnitude in time duration.

From these logs, ten invariants in Web server workloads have been identified. These invariants are summarized in Table 4.11. These invariants are deemed important since they (potentially) represent universal truths for all Internet Web servers.

The invariants were used to identify two possible strategies for the design of a caching system to improve Web server performance, and to determine bounds on the performance improvement possible with each strategy. This study identified

Table 4.11: Summary of Invariants Found in Web Server Workloads

Invariant	Name	Description
1	Success Rate	Approximately 88% of requests to the server result in the successful return of the document
2	File Types	HTML and image files account for 90-100% of requests
3	Mean Transfer Size	Mean transfer size $\leq$ 21 kilobytes
4	Distinct Requests	Among all server requests, less than 3% of the requests are for distinct files
5	One Time Referencing	Approximately one-third of the files and bytes referenced in the log are referenced only once in the log
6	Size Distribution	The file size distribution is heavy-tailed; the tail of the distribution is Pareto with $0.40 < \alpha < 0.63$
7	Concentration of References	10% of the files accessed account for 90% of server requests and 90% of the bytes transferred
8	Inter-Reference Times	File inter-reference times are exponentially distributed and independent
9	Remote Requests	Remote sites account for $\geq$ 70% of the accesses to the server, and $\geq$ 60% of the bytes transferred
10	Wide Area Usage	Web servers are accessed by 1000's of domains, with 10% of the domains accounting for $\geq$ 75% of usage

the distinct tradeoff between caching designs that reduce network traffic and caching designs that reduce the number of requests presented to Internet Web servers. While the two approaches are somewhat at odds with each other, both represent possible avenues for improving Web server performance.

This chapter finished with a discussion of the performance implications of "one-timers", cache replacement policies, and thresholding and partitioning policies to cache management. These issues are investigated further in Chapters 5 and 6, which present a performance evaluation of a file cache.

Chapter 5

File Cache Performance Study: Experimental Design

During the workload characterization described in Chapter 4, a number of issues were discussed that suggested file caching could improve the performance of a Web server. This chapter presents a performance evaluation of a file cache, focusing on the issues that were raised in Chapter 4. This performance study uses trace-driven simulations to evaluate the effects of cache replacement policies, threshold policies and partitioning policies on the performance of the file cache. Section 5.1 provides background information on the traces used in this study. Section 5.2 presents the practical matters that affect file cache performance, and describes the assumptions that were made for this study to deal with these matters. Section 5.3 discusses the experimental design for this performance evaluation. Section 5.4 concludes this chapter with a summary of the experimental design.

5.1 Traces

The trace data that is used to drive the file cache simulations is derived from the Web server logs that were used in the workload characterization in Chapter 4. A trace file was generated for each of the six Web servers (Waterloo, Calgary, Saskatchewan, NASA, ClarkNet and NCSA). A summary of the original logs is given in Table 5.1.

Table 5.1: Summary of Access Log Characteristics (Raw Data)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Access Log Duration	8 months	1 year	7 months	2 months	2 weeks	1 week
Access Log Start Date	Jan 1/95	Oct 24/94	Jun 1/95	Jul 1/95	Aug 28/95	Aug 28/95
Access Log Size (MB)	17.9	49.9	222.6	355.8	327.5	267.7
Total Requests	188,636	726,739	2,408,625	3,461,612	3,328,587	2,490,512
Avg Requests/Day	776	2,059	11,255	56,748	237,756	355,787
Bytes Transferred (MB)	2,071	7,577	12,343	62,489	27,647	28,268
Avg Bytes/Day (MB)	8.5	21.5	57.7	1,024.4	1,974.8	4,038.3

As was the case for the workload characterization, only a portion of the original data was needed for the file cache simulations. A number of observations allowed for the log files to be dramatically reduced in size. First, since the *Successful* responses are responsible for all of the documents transferred by the server, only these responses are used in the cache simulations. Second, only selected data from each request is required for the simulations. Recall from Section 3.3 that a normal access log entry has the form: `hostname - - [dd/mmm/yyyy:hh:mm:ss tz] request status bytes`. The only information needed in this performance study is the name of the requested document (`request`) and the size of the requested document (`bytes`). The hostname could be used if a study of geographically distributed caching mechanisms, such as was studied by Braun and Claffy [12], was conducted. The timing information would be required if a timeout-based replacement policy were used. The `status` field is used to identify which requests to include in the trace (i.e., the *Successful* responses), but is not added to the trace file. To further reduce the size, each trace entry consists of only three integers: one to uniquely identify the file, the second to indicate the size of the file, and the third to indicate the type of the file. Table 5.2 provides a summary of the trace files used in the performance study.

5.2 Practical Matters and Assumptions

There are a number of issues that must be addressed before designing the experiment. Each of these issues is discussed separately. Section 5.2.1 discusses the

Table 5.2: Summary of Trace Characteristics (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Duration	8 months	1 year	7 months	2 months	2 weeks	1 week
Start Date	Jan 1/95	Oct 24/94	Jun 1/95	Jul 1/95	Aug 28/95	Aug 28/95
Size (MB)	1.7	5.8	22.8	30.9	31.8	22.8
Requests	163,112	567,795	2,165,415	3,087,043	2,940,712	2,289,510

location of the file cache with respect to the Web server. Section 5.2.2 looks at the issue of cache consistency. Section 5.2.3 discusses cache security.

5.2.1 Cache Location

The concept of file caching in a Web environment has been widely discussed [10, 12, 27, 28, 36, 37, 42, 63]. Possible locations for such file caches include at the client (i.e., in the Web browser), in the network, or on the Web server. The two main goals of these caches are to reduce the volume of network traffic, and to reduce the latency (i.e., response time) for Web users.

The purpose of this performance study is to determine how effective a file cache can be in reducing the workload on a Web server. While any cache in the Web environment can help to reduce the workload on a Web server, for the purpose of this study the file cache is assumed to be on the Web server itself (i.e., in the memory of the Web server). A file cache in this location can see all the requests destined for this Web server, while client and network file caches see only a subset of these requests. This assumption was made because the traces contain information on all the requests the Web server receives.

Client and network file caches can reduce the number of requests that reach a Web server, as well as reduce the volume of network traffic between the client and the server (on a client cache hit) or between the network cache and the server (on a network cache hit). A file cache on the Web server does not reduce the number of requests that the server must process, nor does it reduce the volume of traffic on the network links between the client and the server. Instead, a file cache can improve the performance of the server in the following way. On a cache hit (i.e., a copy of the requested document is found in the file cache), the file cache reduces by one the number of requests sent to secondary storage, where the Web documents are permanently stored (this may be on a local disk or on a file server elsewhere on the local network). Each cache hit also results in a reduction of the number of bytes transferred to the server from the location where a permanent copy of the document is stored (i.e., a reduction in the data volume). On each cache hit, the time that the Web server requires to prepare a response to a user's request is reduced, as the Web server can retrieve a document from memory much more quickly than from secondary storage. This reduction in response time assumes that the transfer of documents from secondary storage represents the performance bottleneck. In this scenario, the file cache does not reduce the load on the Web server CPU. If the CPU is believed to be the bottleneck, a different approach to file caching may be required (e.g., a hierarchy of network file caches).

5.2.2 Cache Consistency

One problem that must be addressed with the use of file caching is that of *cache consistency*: making sure that the copies of documents in the file cache are consistent (i.e., up-to-date) with respect to the copies of those documents at the server. The difficulty of this problem lies in knowing when a document has been changed at the server. The difficulty of this problem also depends on cache location: since the Web server has no knowledge of the location or contents of network or client file caches, the Web server cannot inform these caches when a document has been modified. The problem is not as difficult for a file cache on the server, as the server has knowledge of when its files have been modified.

During the workload characterization described in Chapter 4, it was found that few documents are modified. Table 5.3 shows the percentage of (potentially) modi-

Table 5.3: Percentage of Documents Modified (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
% of Documents Modified	2.5	3.6	1.0	0.5	0.8	0.2

fied[1] documents in the access logs of each server.[2] The highest percentage of modified documents observed in the six access logs was 3.6%, for the Calgary server. In four of the traces, the percentage of modified documents was less than or equal to 1.0%.

The results in Table 5.3 suggest that document modifications will have minimal impact on the performance results of the file cache simulations. Thus, to simplify the performance study, all files are assumed to be read-only (i.e., never modified).

5.2.3 Cache Security

Another problem that arises with the use of file caching is *cache security*. While most documents on the Web are intended for viewing by any interested party, there are some documents that are intended only for restricted access. The owners of these "restricted" documents may not want copies of these documents to be stored in a file cache, where users who are not part of the privileged group could obtain access to them. One possible solution would be to allow documents to be marked as "uncacheable" by their owners, so that the file is not added to any file cache between the server and the requesting client. However, this approach assumes that all the file caches (if any) along the path check for such a flag. If the owners of these "restricted" documents are still concerned for the security of their documents, a second option would be to use encryption. With this approach, even if the document were cached and accessed by an unauthorized user, the contents of the document could not be easily compromised. For this performance study, cache security is not considered; all documents are treated as being cacheable.

[1]From the information in the access logs, it is difficult to determine when a file has been modified. One method is to search for a change in file size. However, a change in file size can also be the result of a user aborted connection.

[2]Files of size 0 are not counted, as these are the result of aborted connections.

5.3 Experimental Design

This section describes the design of the file cache performance study. Section 5.3.1 defines the system to be simulated. Section 5.3.2 presents the metrics used to evaluate the performance of the file cache. Section 5.3.3 lists the parameters, factors and levels that are involved in the simulation. Section 5.3.4 describes the evaluation techniques used in this study.

5.3.1 System Definition

For this performance study, a very simple system is assumed, with the file cache located in the memory of a Web server. In this position, the file cache is able to see all requests to the Web server. The decision to locate the file cache in this position is described in Section 5.2.1.

The system under study operates in the following way. When a request from a Web client arrives at the Web server, the server checks in its file cache for a copy of the requested document. If a copy is found (a *cache hit*), the server can return the document to the client immediately. If a copy of the requested document is not found in the file cache (a *cache miss*), the Web server must retrieve a copy from the local disk or file server on which the document is permanently stored. When a copy of the document is returned to the Web server from secondary storage, a copy of it may be added to the file cache (depending on the replacement, threshold and partitioning policies for the file cache). At this point the Web server sends a response to the client that made the original request.

Throughout this study, file-level caching is investigated. File-level caching differs from block-level caching in one small but important way: a file cache holds a variable number of files (which can have different sizes), while a block cache contains a fixed number of equal-sized data blocks. On a cache miss, a file cache removes zero or more files from the cache, until there is sufficient space to add the new file. This differs from block-level caching, which replaces at most one data block on a cache miss. In a file cache, only complete documents are cached; when a file is added to the cache, the whole file is added, and when a file is removed from the cache, the entire file is removed. The file cache never keeps only part of a file, unlike a block-level cache.

5.3.2 Performance Metrics

Two performance metrics are used to evaluate each file cache configuration: cache hit rate, and the data volume. The cache hit rate is the fraction of requests to the server that are satisfied by searching the file cache for a copy of the requested document. The data volume refers to the fraction of the requested bytes that must be transferred between the Web server and secondary storage, relative to the number of bytes transferred when no cache is used. The cache configuration that achieves the highest cache hit rate and the lowest data volume is considered to be the file cache with the best overall performance.

A third possible metric, response time (i.e., latency), is not included in this study, as the data sets do not contain the necessary information. However, the Web server that achieves the highest hit rate and the lowest data volume should also have the shortest response time (assuming that communication with secondary storage, and not the server's CPU, is the performance bottleneck).

5.3.3 Simulation Parameters, Factors and Levels

Four system parameters and two workload parameters are identified in this study. The system parameters are the cache size, the replacement policy, the threshold policy, and the partitioning policy. The workload parameters are the set of client requests and the length of the cache *warm-up* period. Additional parameters, such as disk-access times and network speeds, do not impact the system under study (see Section 5.3.1) and are therefore omitted.

5.3.3.1 Cache Sizes

The first factor in this study is the size of the file cache. Since entire files are being cached, rather than data blocks, the size of the file cache is given as a number of megabytes (MB), rather than as a number of blocks. In total, 12 different levels are used. They are: 1, 2, 4, 8, 16, 32, 64, 96, 128, 256, 512, and 1024 MB. The largest file cache size (1024 MB) is large enough to cache the entire document set on any of the Web servers under study. The results for the 1024 MB file cache can therefore be interpreted as the results for an infinite-sized file cache, or a file cache that never replaces a document once the document has been added to the cache. Using a cache of this size indicates the maximum achievable cache hit rate, and the minimum achievable data volume. The performance of smaller cache sizes can then be compared to the performance of the infinite cache. Such a comparison can be used to justify how large a file cache should be, and to determine at what point increasing the cache size does not result in significant performance improvement.

5.3.3.2 Cache Replacement Policies

The second factor in this study is the cache replacement policy. This policy determines which document (or documents) should be removed from the file cache in order to create sufficient space for the most recently requested document (documents are replaced only when a cache miss occurs, and there is insufficient free space in the file cache to add the new document). Four replacement policies are tested in this study: Least Recently Used (LRU), Least Frequently Used (LFU), Least Frequently Used with Aging (LFU-Aging), and Frequency-Based Replacement (FBR).

The LRU policy removes from the cache the file that has not been used for the longest period of time. LRU works well in reference streams that exhibit the property of *temporal locality* (this property suggests that files that have been referenced in the recent past will likely be referenced again in the near future). The LRU policy is frequently used in computer systems.

The LFU replacement 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 cache miss (if there is insufficient space to add the requested file) the file with the lowest reference count is selected for replacement. In the case of a tie, the least recently used file with the lowest reference count is replaced.

The LFU-Aging policy is similar to the LFU policy in that the least frequently used document is replaced when space is required for a new document. However, LFU-Aging attempts to deal with a problem of LFU. With LFU, some documents can build up extremely high reference counts so that they are never replaced, even if these documents are never requested again. The LFU-Aging policy attempts to ensure that this condition does not occur by limiting and aging (i.e., occasionally reducing) reference counts [59, 70].

To combat reference count buildup, the LFU-Aging policy uses two parameters, M_{refs} and A_{max} [59]. M_{refs} is the maximum reference count that one file can acquire. If a file in the cache is referenced, but its reference count has already reached the maximum allowable value, the reference count is not incremented. This modification helps to prevent the reference counts of files from becoming too large [59]. The second parameter, A_{max}, is used to "age" the reference counts of files in the cache. The aging process works by dynamically maintaining the sum of all reference counts for files in the cache. Whenever the average number of references per file exceeds A_{max}, the reference count C of each file in the cache is reduced to $\lceil C/2 \rceil$ [59, 70]. In this study, a number of different levels for M_{refs} and A_{max} are tested. This process is described in Section 6.2. However, the reference counts are always decreased by a factor of two; other values were not tested.

The FBR policy makes a replacement decision using a combination of reference frequency and block age [59, 70]. FBR divides the cache into three sections: a new section, a middle section, and an old section. The sizes of these sections are set using two parameters, F_{new} and F_{old}. F_{new} specifies the fraction of cache space dedicated to the new section, while F_{old} specifies the fraction of cache space for the old section. All remaining space is allocated for the middle section.

Figure 5.1 shows the three sections of a FBR cache. When a file is initially added

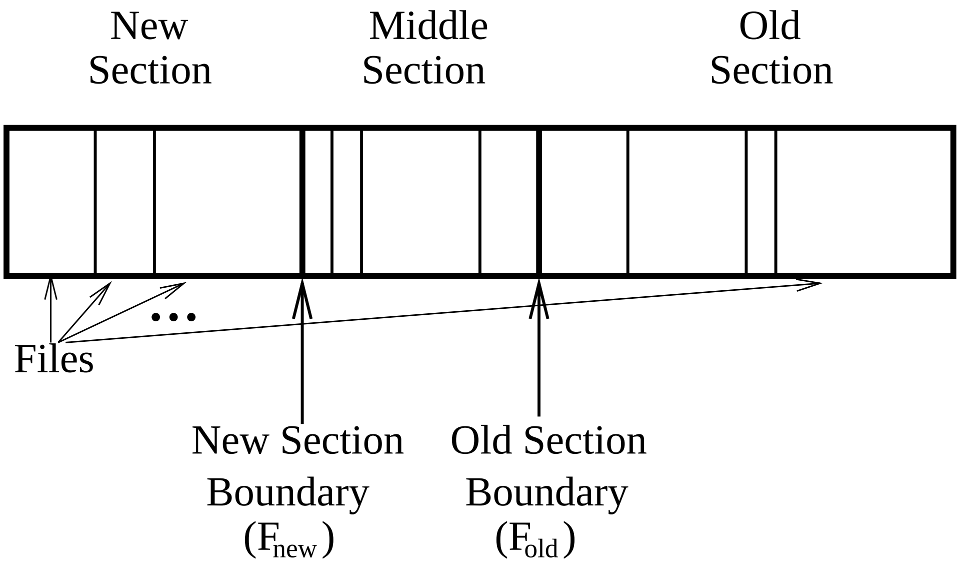


Figure 5.1: Sections of an FBR File Cache

to the cache, it is placed at the start of the New Section (most recently used, or MRU). As the file ages (on an LRU basis), it will be removed from the New Section and added to the Middle Section. If the file continues to age (i.e., it remains in the cache but is not re-referenced) it will eventually be removed from the Middle Section and added to the Old Section. Only files in the Old Section are candidates for removal from the cache by the replacement strategy. On every cache hit, the referenced file is always moved to the start of the New Section (MRU). If the file was found in either the Middle or Old Sections, its reference count is incremented. If the file was already in the New Section, its reference count is not incremented [59, 70]. On a cache miss that requires a cache replacement decision, the least frequently used file in the Old Section is replaced (on a tie the least recently used file with the lowest reference count is chosen). The FBR policy also implements the aging policy on reference counts that was described for LFU [59, 70].

For the entire performance study, the parameters for the FBR policy were fixed. F_{new} was set to 0.25 and F_{old} was set to 0.60 (leaving 0.15 for the Middle Section). These values were used by both Robinson and Devarakonda [59] and Willick *et al.* [70]. The same aging parameters were used for both the LFU and the FBR replacement policies. The values used for these parameters are discussed in Section 6.2.

5.3.3.3 Threshold Policy

The third factor used in this study is a threshold policy that places constraints on the sizes of documents allowed in the cache. A requested document is added to the file cache if and only if the size of the requested document is within the acceptable range. During the workload characterization in Chapter 4, it was determined that in all data sets, most client requests were for small documents, while a significant portion of the data traffic was caused by requests for larger documents. Therefore, it might be possible to improve the cache hit rate by caching a large number of small documents (i.e., put an upper limit on the size of document that may be added to the cache), or to reduce the data volume by caching a few larger documents (i.e., place a lower limit on the size of document that may be added to the cache).

In total, four levels are examined for this factor. The first level caches all files (no threshold policy). The second level places an upper limit on the size of documents that may be added to the cache, while the third level uses a lower limit on document size. The fourth level used for this factor implements both an upper and a lower bound on the size of file that may be added to the cache.

5.3.3.4 Partitioning Policy

A partitioning policy is the fourth factor in this study. This policy divides the file cache into a number of partitions based on document types. This approach is considered to determine if preventing files of one type from replacing another type in the cache has any effect on performance. In particular, the workload characterization in Chapter 4 found that most requests were for either HTML or image documents.

Three levels are tested in this study. The first level uses only one partition (the cache is shared by all document types). The second level uses three partitions: one for HTML documents, one for image documents and one for all other document types. The third level uses seven partitions, one for each document type identified in Chapter 4 (HTML, Images, Sound, Video, Dynamic, Formatted, and Other).

5.3.3.5 Additional Factors

Two workload parameters are also factors in this performance study: the access logs and the warmup period. The access logs from all six data sites are used to create trace files to drive the file cache simulations. These trace files are summarized in Section 5.1.

The second factor is the length of the "warm-up" period, used to reach steady-state. Only the steady-state performance is of interest, since during the initial (or transient) state, many cache misses occur, simply because the cache is empty. However, identifying where the transient state ends and the steady-state begins is a difficult problem [31].

For this study, steady state is determined as follows. A trace-driven simulation is conducted, using an infinite sized file cache. In this scenario, once a file enters the file cache it is never replaced. Thus, all cache misses occur because the file has never been referenced before. The traces are then divided into small intervals. The cache hit rate and the data volume are then calculated separately over each interval. The results from each interval are then plotted and visually inspected to determine a point when the averages do not change much from interval to interval (based primarily on hit rate). Figure 5.2 shows such a plot for the NCSA trace. The vertical line denotes the value that was chosen as the start of the steady-state period. Figure 5.3 presents the transient plots for all six data sets. In most cases, a conservative estimate for the end of the transient period was chosen. Table 5.4 shows the length of the warmup period chosen for each of the six traces. On average, steady state was reached after 33% of the requests in the logs. The relatively long warmup period (compared to trace length) required to reach steady state suggests two observations. First, there is a massive volume of documents stored on (and accessed from) a Web server. Second, the long warmup period suggests that it would be undesirable to completely flush a server cache at any time (e.g., on a daily

Table 5.4: Length of Transient Period Chosen (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Requests	50,000	250,000	500,000	1,000,000	900,000	900,000
% of Trace	30.7	44.0	23.1	32.4	30.6	39.3

or weekly basis).

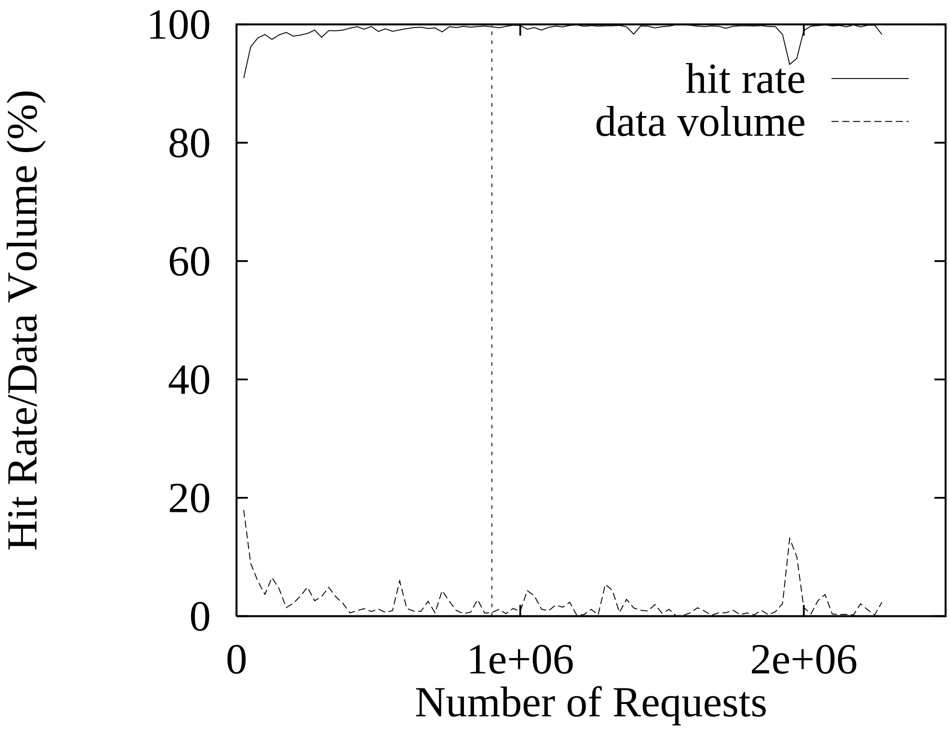


Figure 5.2: Transient Identification (NCSA Data)

All remaining experiments use the warmup periods indicated in Table 5.4. During the warmup period, the file caches behave normally, with files being added and removed as needed, according to the replacement, threshold and partitioning policies that are in place at the cache. However, statistics concerning the performance of the file cache are not collected until the cache has entered into the (predetermined) steady-state.

5.3.4 Evaluation Technique

In order to observe the full range of interactions between the different factors, a full-factorial design is required. However, due to the large number of factors and

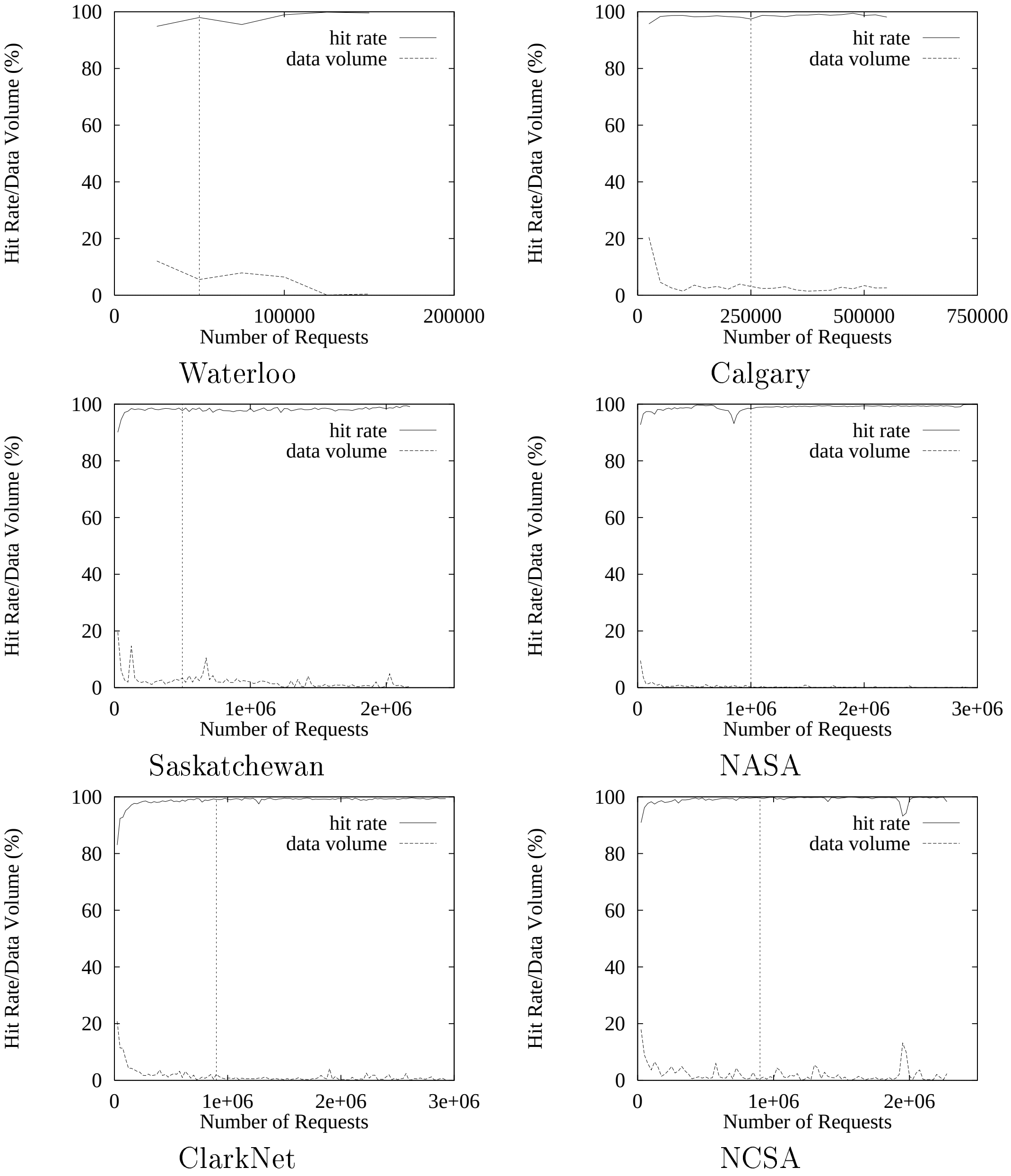


Figure 5.3: Transient Identification (All Data Sets)

levels being studied, this can result in a very time and resource consuming process. Thus, to reduce the overall size of the study, several levels for some of the factors are eliminated at an early point in the study.

The main factors affected by the decisions were aging parameters, cache replacement policies and cache threshold and partitioning parameters. During the initial stages of the study, values for the aging parameters A_{max} and M_{refs} were tested for each trace. The values determined during these tests were then used for the remainder of the experiments. During the comparison of the replacement policies, it was clear that the LFU*-Aging policy always outperformed the LFU-Aging policy, so for all of the remaining experiments, LFU-Aging was excluded, and only three replacement policies (LFU*-Aging, LRU and FBR) were used. During the evaluation of the threshold and partitioning policies, only a small number of values were tested for each level. While these tests may not uncover the full potential of these policies, these few values are sufficient to identify some of the benefits and drawbacks of these policies. Throughout all of the experiments, all 12 different cache sizes were simulated.

5.4 Summary

This section has presented the experimental design for the performance study of a file cache on a Web server. This study focuses on the effects of cache size, replacement policy, threshold policy and partitioning policy on the performance of a file cache. The next chapter provides the results of the study.

Chapter 6

File Cache Performance Study: Simulation Results

This chapter presents the simulation results from the file cache performance study. Section 6.1 explains an additional replacement policy, LFU*, that was examined in the performance study. Section 6.2 describes the process for determining the aging parameters that are used in both the LFU-Aging and FBR cache replacement policies. Section 6.3 provides the results for file cache performance for the different replacement policies that were tested. Section 6.4 presents the performance results of a file cache that uses a threshold policy. Section 6.5 discusses the results obtained when a partitioning policy is used. Section 6.6 summarizes the results of the performance study.

6.1 LFU* and LFU*-Aging Replacement Policies

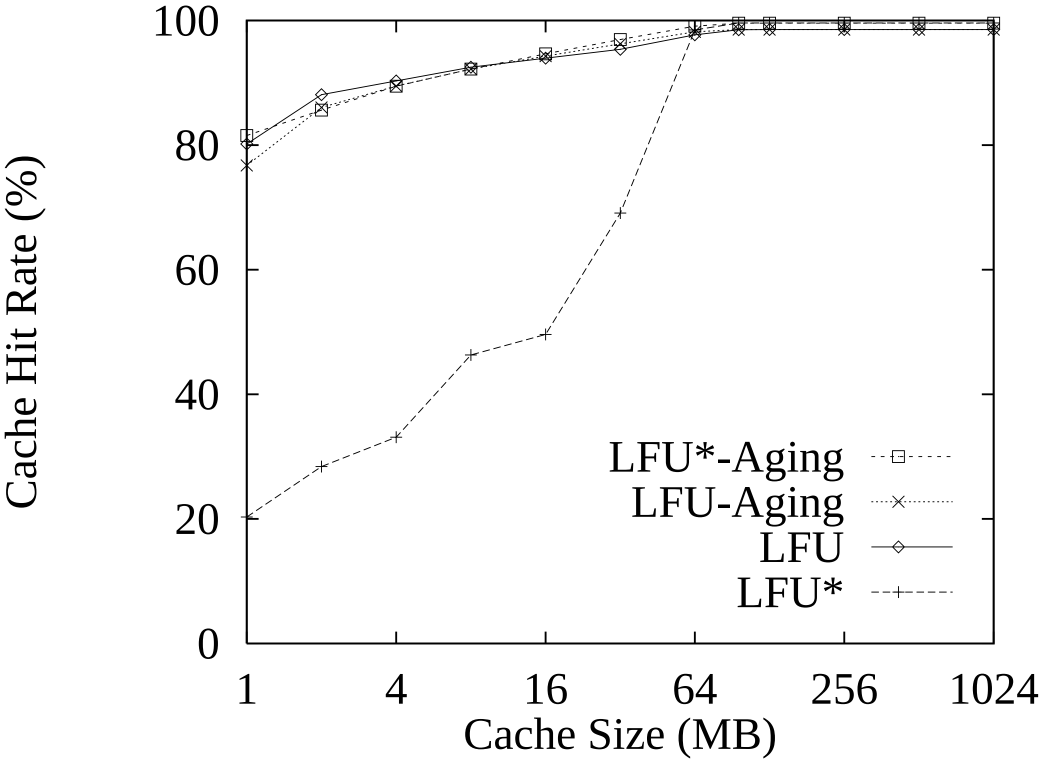
During the workload characterization study in Chapter 4, it was observed that a large portion (about one-third) of the unique documents on the Web server were requested only once during the duration of the access log (these are referred to as "one-timers"). Obviously, caching these documents would only degrade the performance of the file cache.

In an attempt to minimize the performance impact of these “one-timers”, a modification was made to the LFU replacement algorithm. The modified algorithm is denoted LFU*, to distinguish it from the standard LFU algorithm. LFU* is similar to the LFU policy in that a reference count is maintained for each file in the cache. The LFU* policy differs from the LFU policy in that on a cache miss, the newly requested document is not always added to the cache. More specifically, only files with a reference count of one in the cache are candidates for replacement. The amount of space in the cache that is occupied by files with a reference count of one is maintained dynamically. On a cache miss, the file cache compares the size of the requested file to this amount. If the file size is less than this amount, then zero or more old files (with a reference count of one) are removed, and the new file is added. If the file size is larger than this amount, the new file is not added, and the old files are left in the cache.

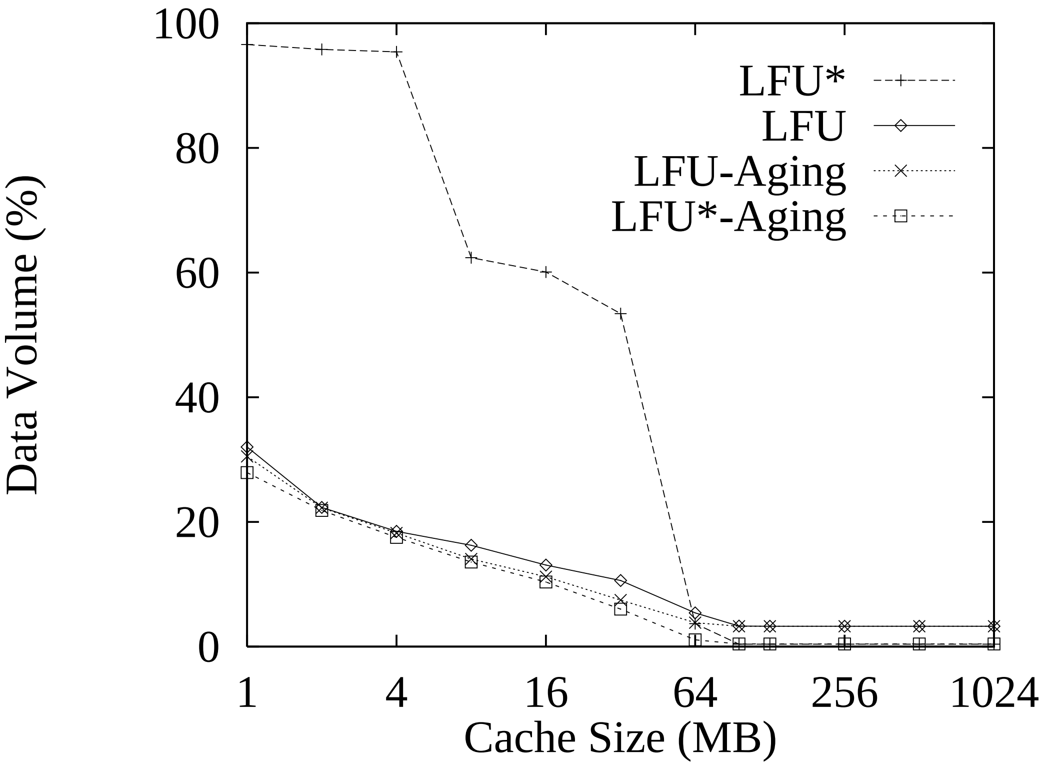
Like LFU, LFU* can also suffer from files building up high reference counts. In fact, once all files in the cache have reference counts of two or more, no new files are ever added. This is not a problem if the cached documents remain popular, but would be disastrous if the popular document changed. To alleviate this problem, the aging policy that is described in Section 5.3.3.2 is used, so that the reference counts, over time, can be reduced back to one, making more documents candidates for replacement. This second new policy is denoted as LFU*-Aging. For the purpose of comparison, both LFU-Aging and LFU*-Aging are tested with the same aging parameters. The A_{max} value used for each data set is shown in Table 6.1, while no limit was used for the M_{refs} parameter.

Figures 6.1 to 6.6 show the performance of the LFU, LFU-Aging, LFU* and LFU*-Aging replacement policies. Across all of the data sets, several general trends can be seen. In terms of cache hit rate, the LFU*-Aging policy performs the best, while in most cases, LFU* performs the worst. The LFU and LFU-Aging policies perform similarly, usually better than LFU* but not as well as LFU*-Aging. The one notable exception to the observation that LFU* is the worst replacement policy is the NCSA data set (Figure 6.6). For this data set, LFU* is the second best policy behind LFU*-Aging. One possible explanation for this discrepancy is that the trace duration is too short (one week) for LFU* to exhibit its true behaviour, namely that the performance of the LFU* cache degrades with time. That is, after a short period the LFU* cache fills up with a popular set of documents. These documents are never removed from the LFU* cache. The cache performs well as long as this set of documents remains popular. As the popularity of the cached documents diminishes, so too does the performance of the LFU* cache. By using the aging policy, LFU*-Aging is able to make some changes to its (cached) document set, to avoid this performance degradation. At the same time, the LFU*-Aging policy appears to avoid some of the performance impact of the "one-timers", which LFU and LFU-Aging suffer from. Since LFU* performs so poorly, it is excluded from all remaining experiments.

The LFU-Aging and LFU*-Aging replacement policies perform about the same in terms of reducing data volume. In several of the data sets (Calgary, shown in Figure 6.2(b), and NASA, shown in Figure 6.4(b)), LFU-Aging has a slightly lower data volume than does LFU*-Aging, while in a number of the other data sets (Waterloo, shown in Figure 6.1(b), ClarkNet, shown in Figure 6.5(b), and NCSA, shown in Figure 6.6(b)), LFU*-Aging is marginally lower than LFU-Aging. In the remaining dataset (Saskatchewan, shown in Figure 6.3(b)), the two policies perform almost identically.

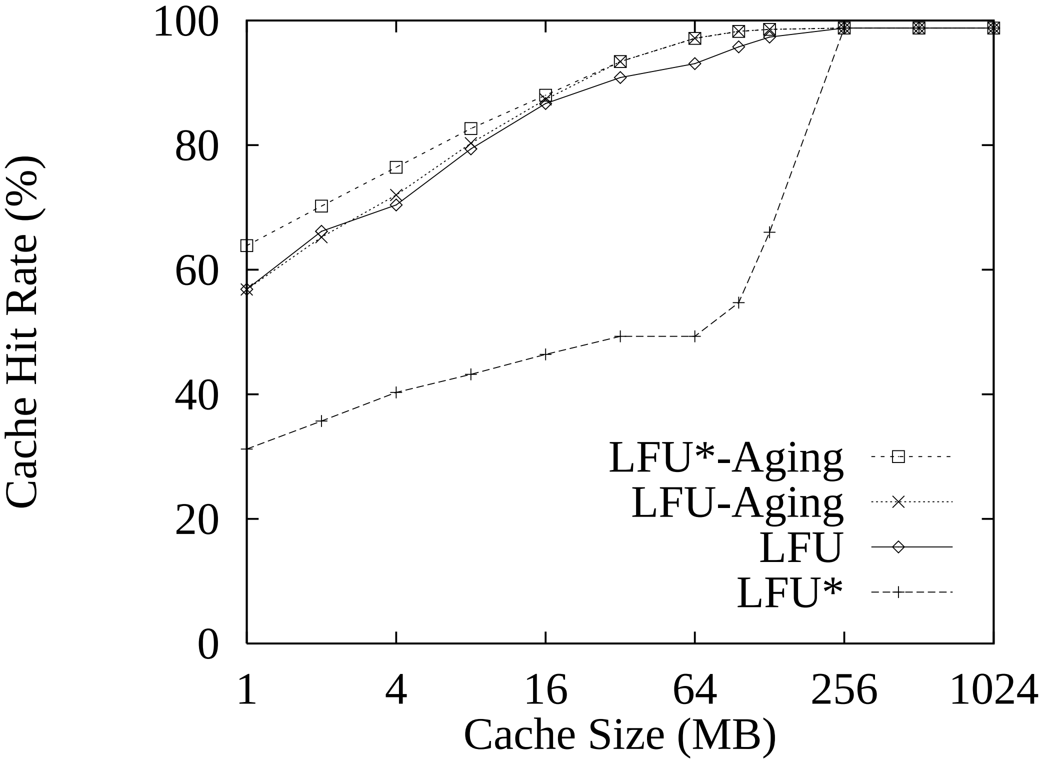


(a)

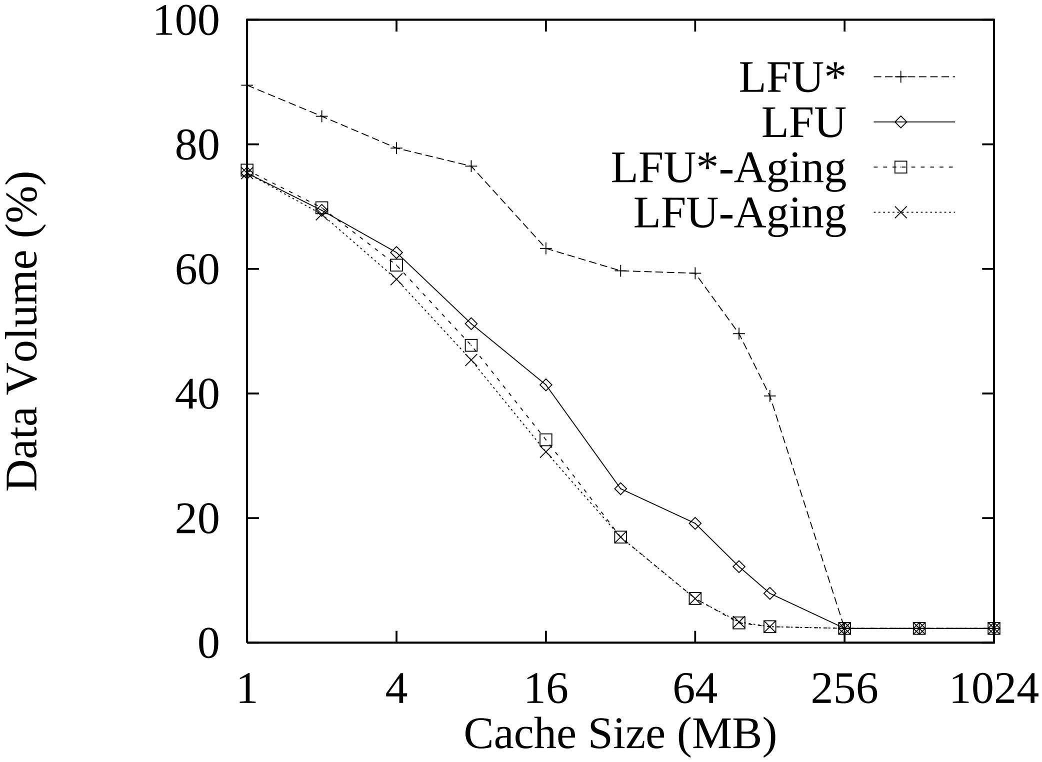


(b)

Figure 6.1: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Waterloo Data)

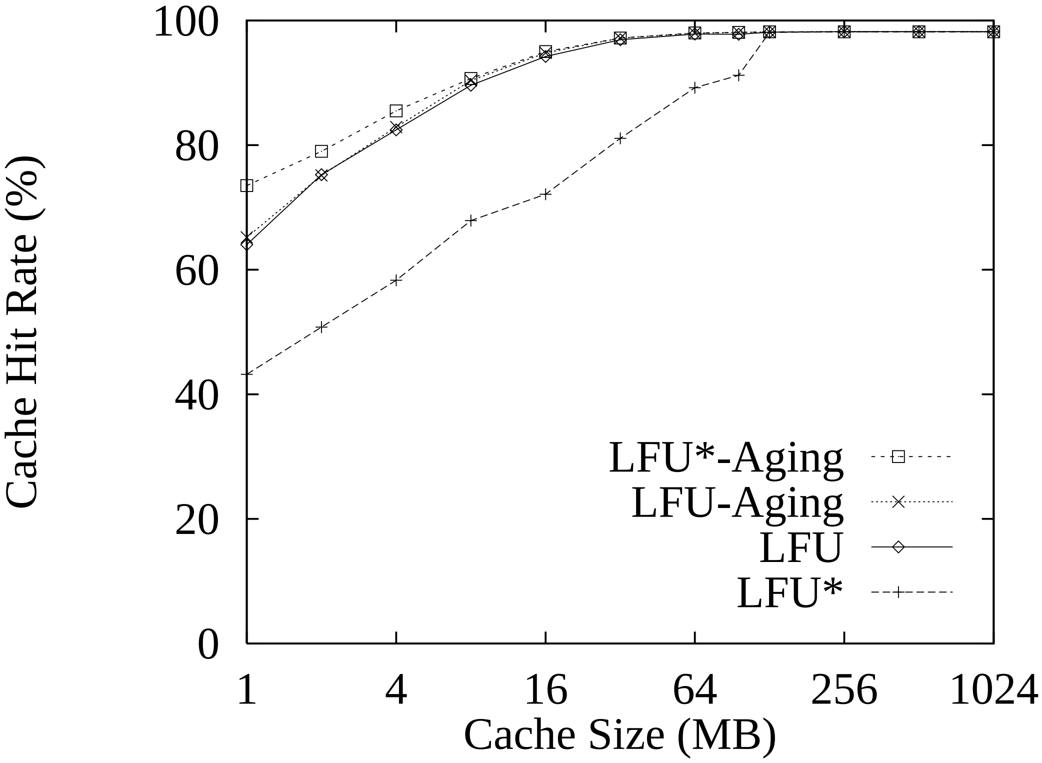


(a)

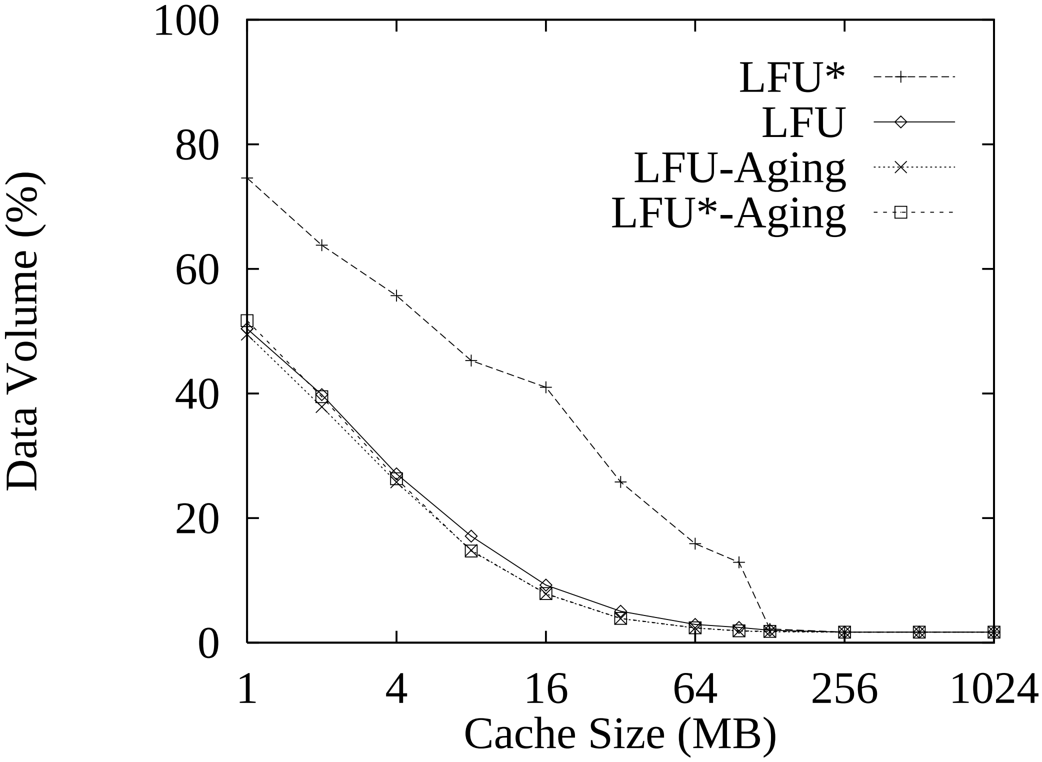


(b)

Figure 6.2: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Calgary Data)

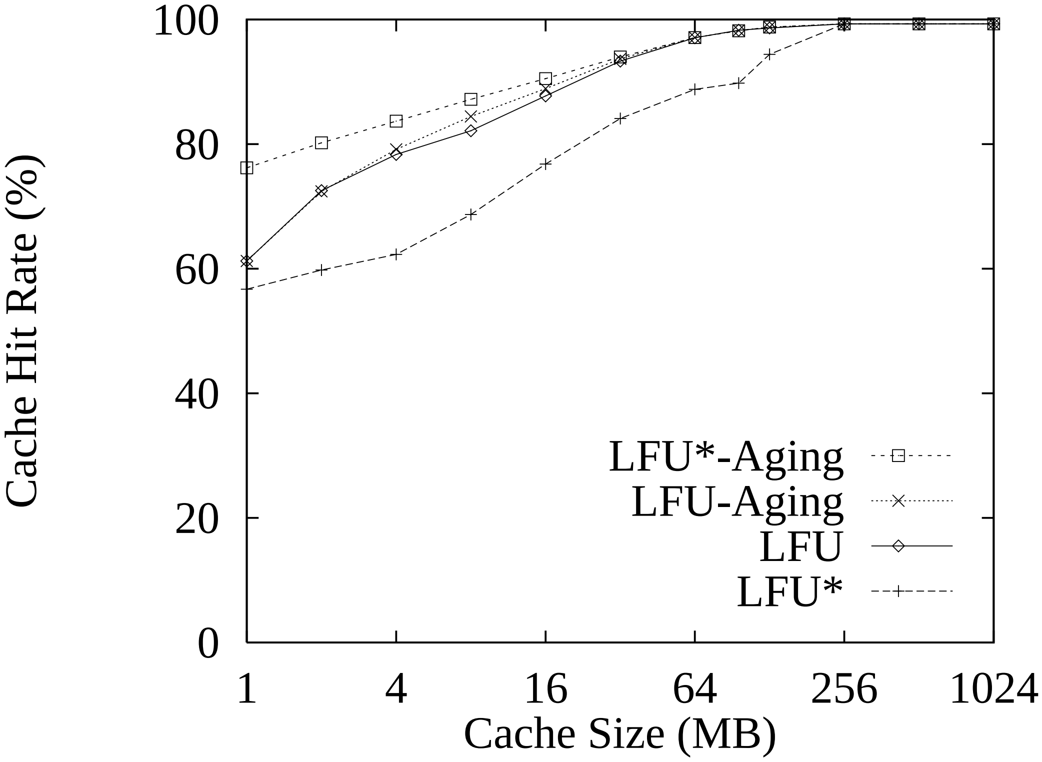


(a)

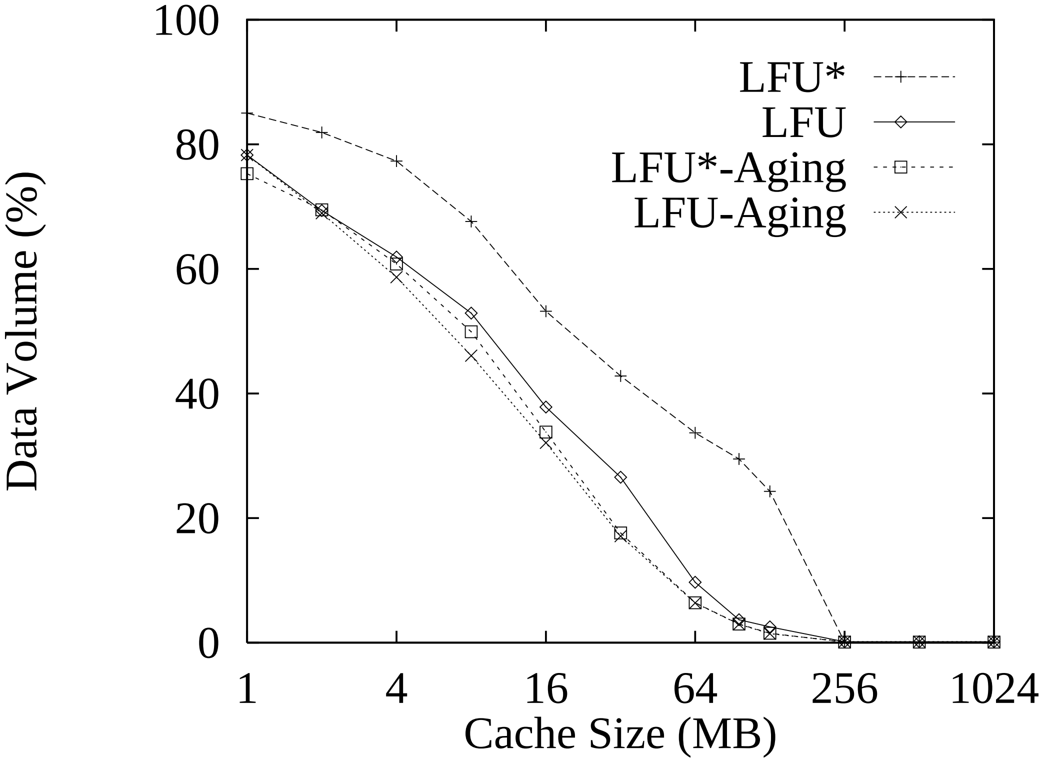


(b)

Figure 6.3: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Saskatchewan Data)

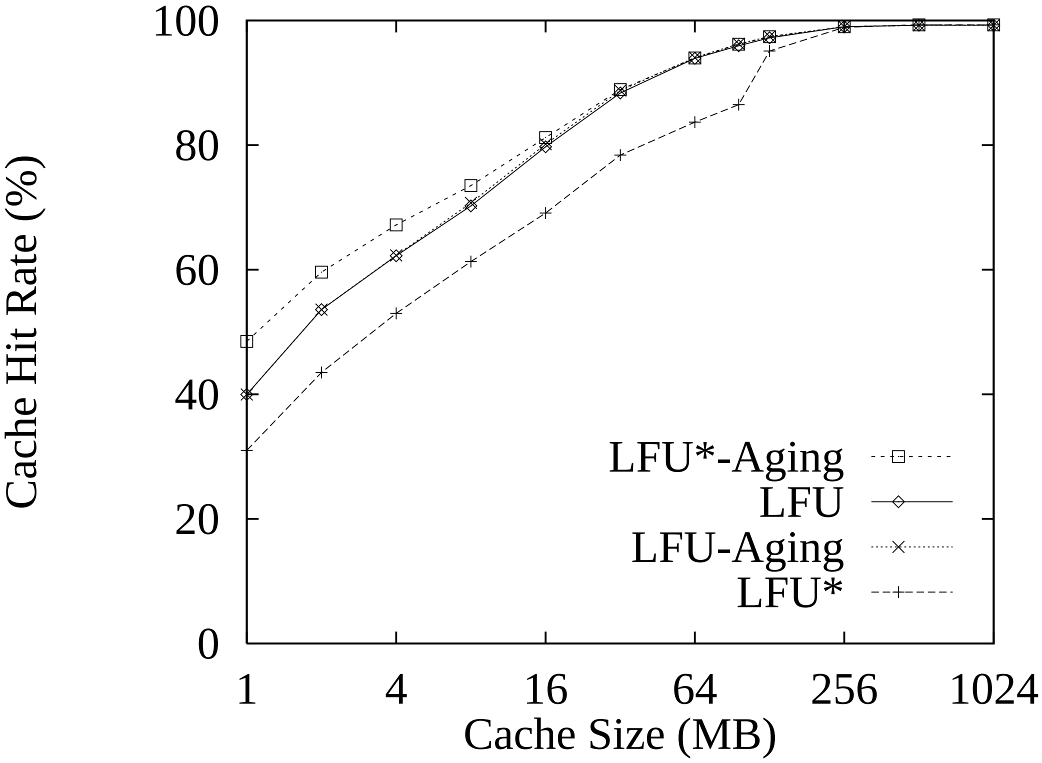


(a)

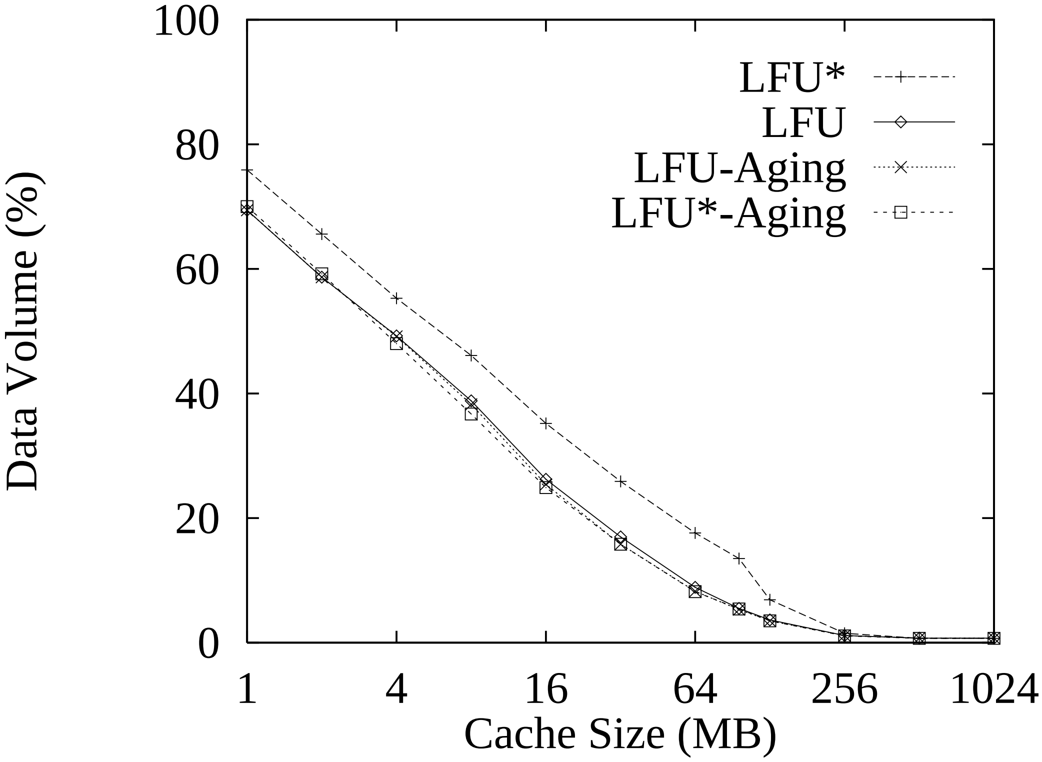


(b)

Figure 6.4: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (NASA Data)



(a)



(b)

Figure 6.5: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (ClarkNet Data)

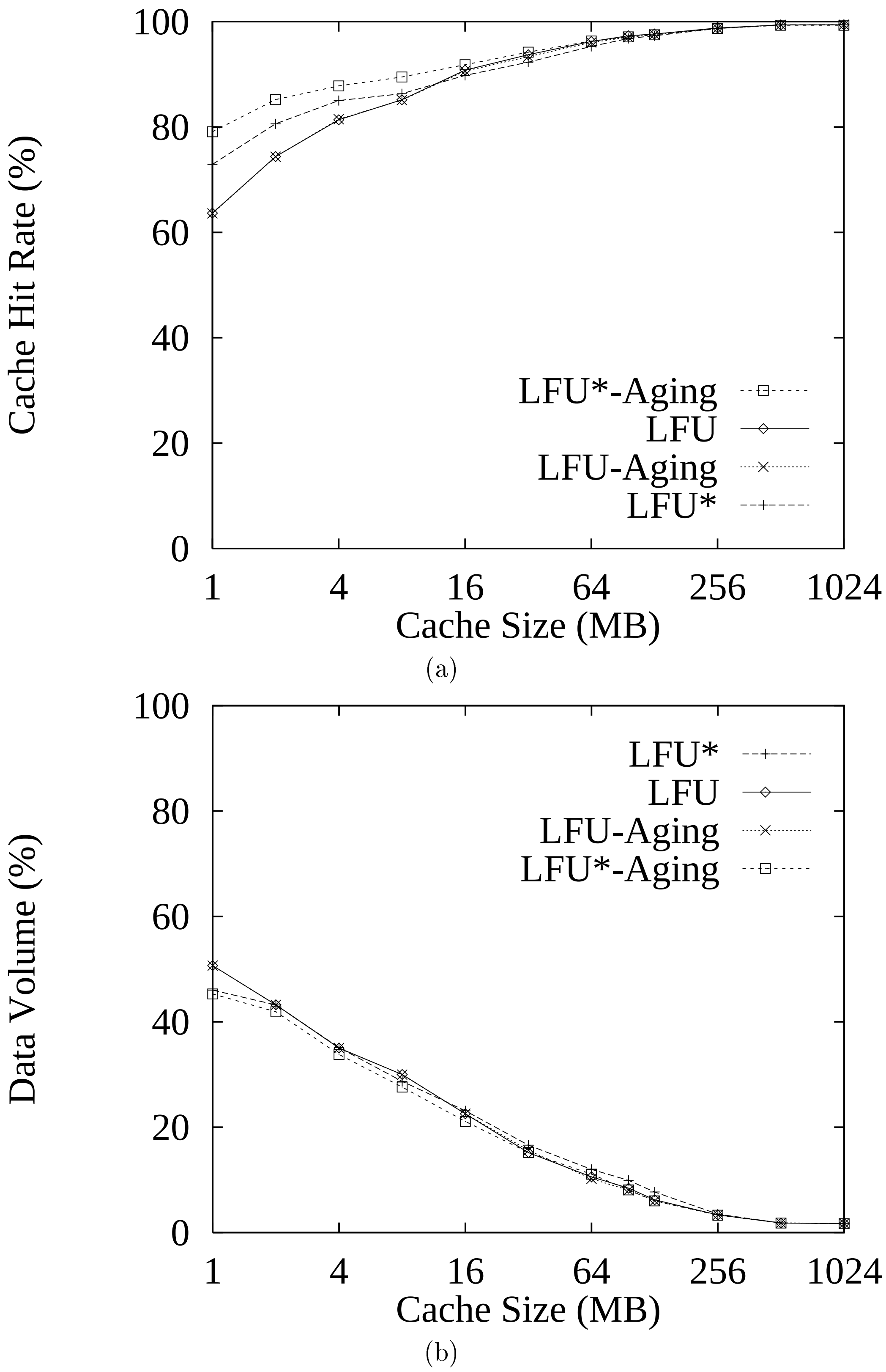


Figure 6.6: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

6.2 Determining Aging Parameters A_{max} and M_{refs}

Three of the cache replacement strategies under study, LFU-Aging, LFU*-Aging, and FBR, use an aging policy [59], as described in Section 5.3.3.2. This aging policy requires values for two parameters: A_{max} and M_{refs}. Recall from Section 5.3.3.2 that A_{max} is the maximum average age of files in the cache. Once this maximum is reached, the reference counts of all files in the cache are decreased by a factor of two. M_{refs} is the maximum reference count that one file can acquire.

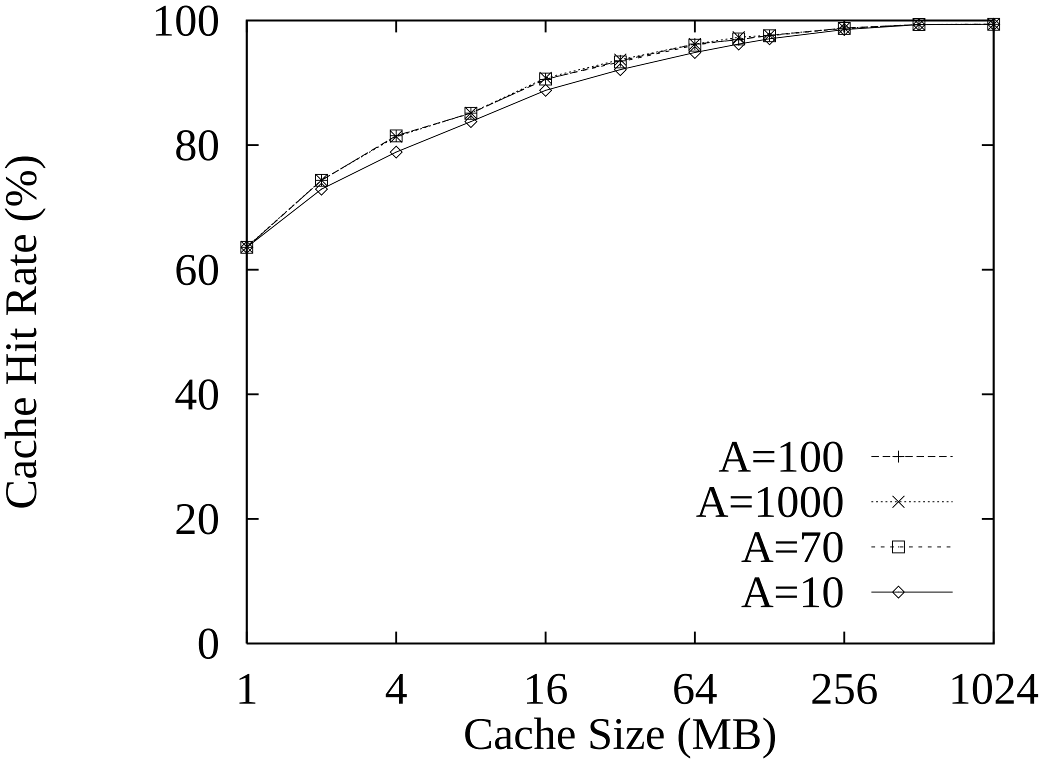
In order to simplify the overall study, different values for these parameters were tested at the start of the study to see how they affect the performance of the file cache. The values that were determined in this testing phase are used in all subsequent experiments that involve the LFU-Aging policy, the LFU*-Aging policy, or the FBR policy.

Figure 6.7 shows the effects of different A_{max} values on the performance of a file cache using the LFU-Aging replacement policy for the NCSA dataset. For all of these experiments, $M_{refs} = NoLimit$. Figure 6.7 shows that the value of A_{max} has little impact on the performance of the file cache. Across all of the data sets, the setting of A_{max} had minimal effect on file cache performance. Thus, the remainder of the graphs are not included.

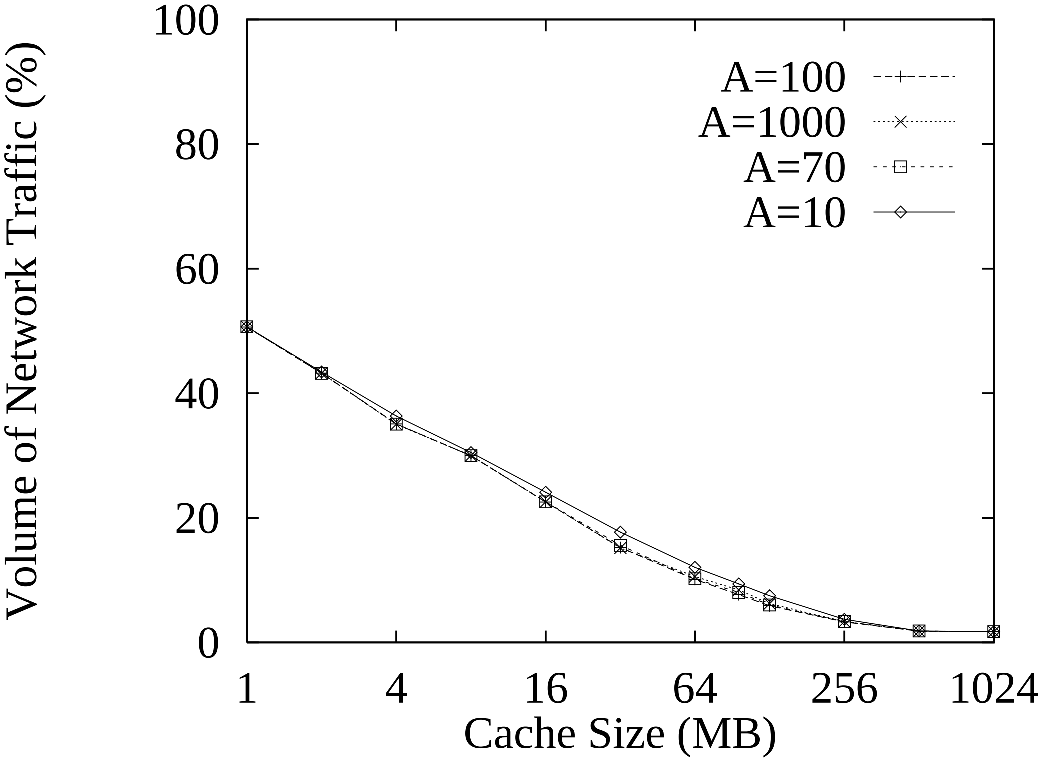
Figures 6.8 to 6.13 show the effects of different values of A_{max} on the performance of a file cache using the LFU*-Aging replacement policy, (as well as $M_{refs} = NoLimit$, no threshold policy and a single shared partition) across the six data sets. In general, the largest value of A_{max} tested for the reference count limit ($A_{max} = 1000$) provided the worst performance, and lower values (e.g., $A_{max} = 10$) provided the best performance. The performance results are most sensitive to the A_{max} setting in small caches; the A_{max} setting is almost completely irrelevant for large caches. For the busier servers (NASA, ClarkNet and NCSA) the performance differences between all of the A_{max} settings tested are quite small. It is interesting to note that the relationship between A_{max} and cache hit rate is not strictly monotonic. For example, in Figure 6.10, $A_{max} = 10$ performs well, $A_{max} = 30$ performs the best, $A_{max} = 100$ performs slightly worse than $A_{max} = 10$, while $A_{max} = 1000$ performs significantly worse than all of the other values tested. Iterative tests were performed to determine the "best" A_{max} value to use for each trace. These values are given in Table 6.1.

Across all six data sets, a value for A_{max} in the range $10 \leq A_{max} \leq 100$ appears to result in the best overall cache performance. There does not always appear to be one exact value for A_{max} that achieves both the highest hit rate and the lowest data volume. For example, Figure 6.12(a) shows that an A_{max} value of 30 results in a higher cache hit rate than an A_{max} value of 10, while Figure 6.12(b) shows that an A_{max} value of 10 achieves a lower data volume than does an A_{max} value of 30.

Table 6.1 shows that there is some relationship between the value of A_{max} and the workload of the server; servers with lighter workloads (e.g., Waterloo and Calgary) need smaller values for A_{max}, while busier servers (e.g., NCSA) need slightly larger values for A_{max} (although still in the $10 \leq A_{max} \leq 100$ range).

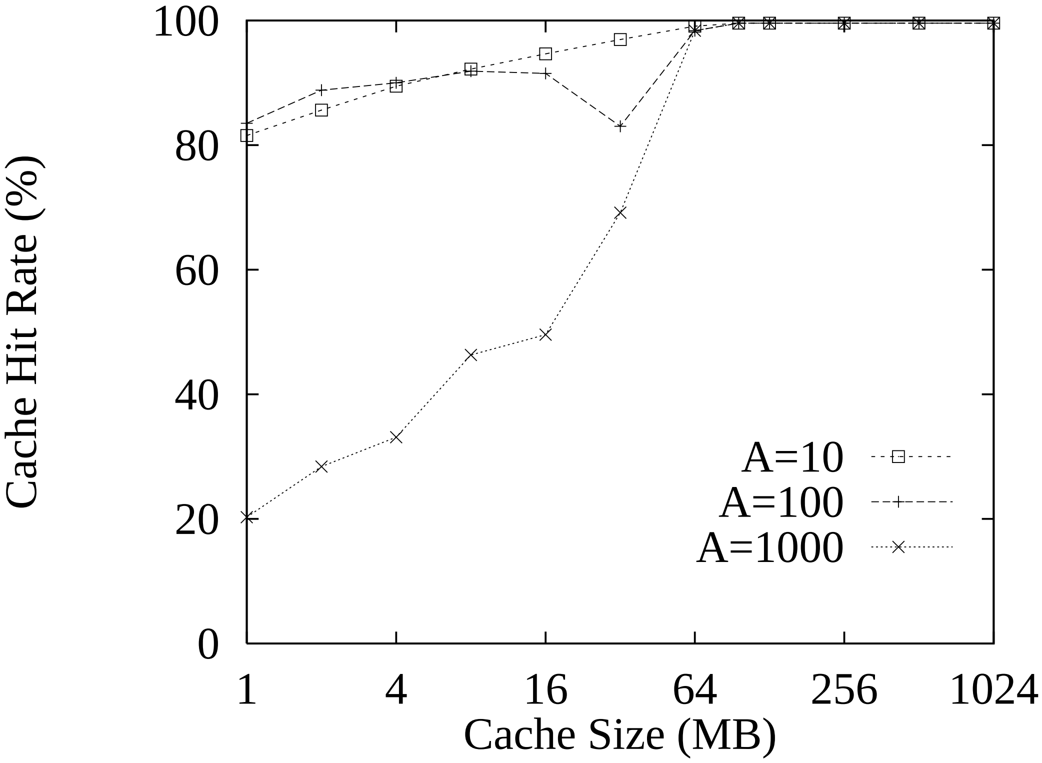


(a)

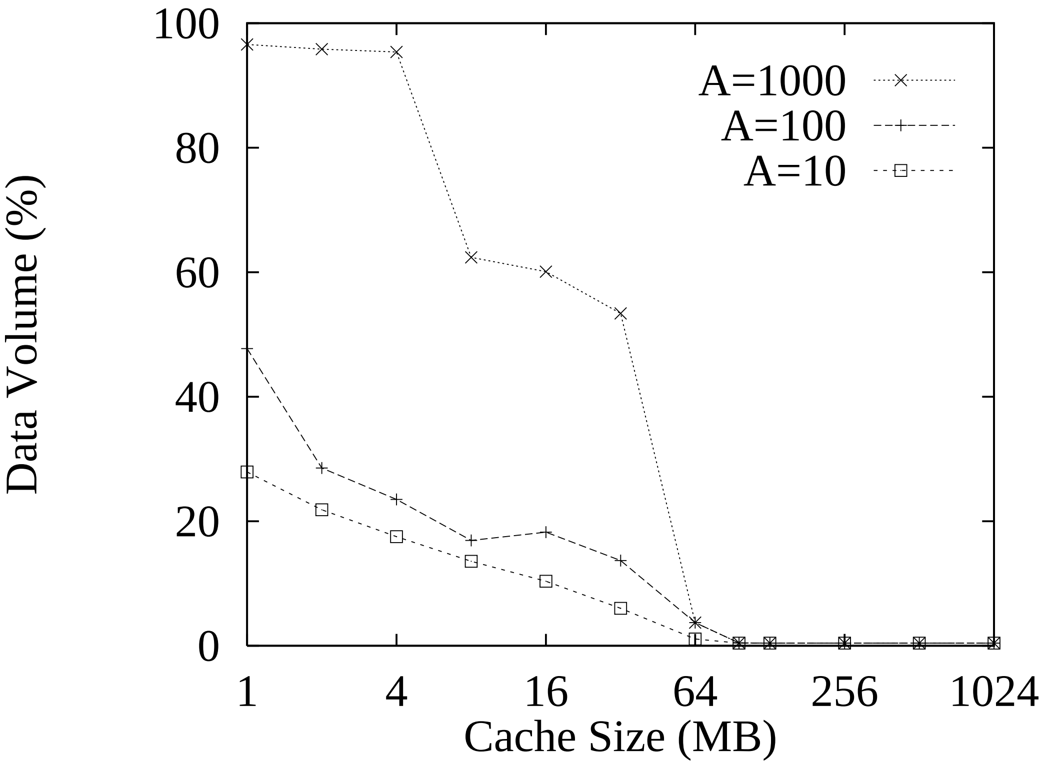


(b)

Figure 6.7: Effects of A_{max} Parameter on LFU-Aging Policy: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

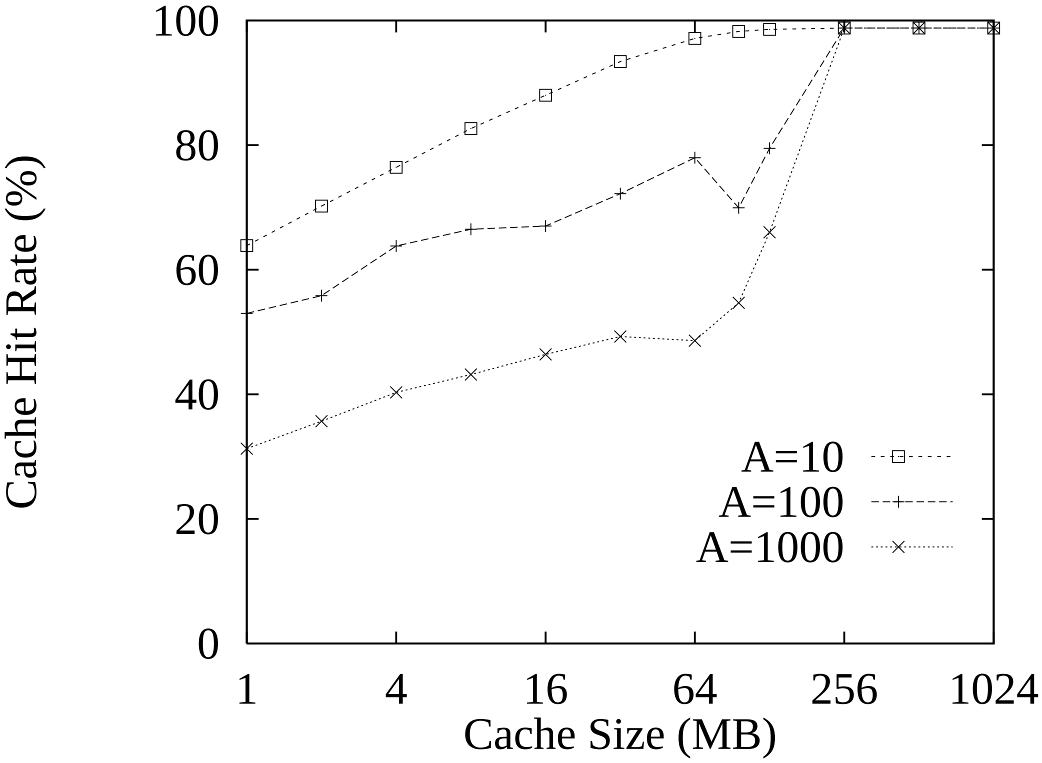


(a)

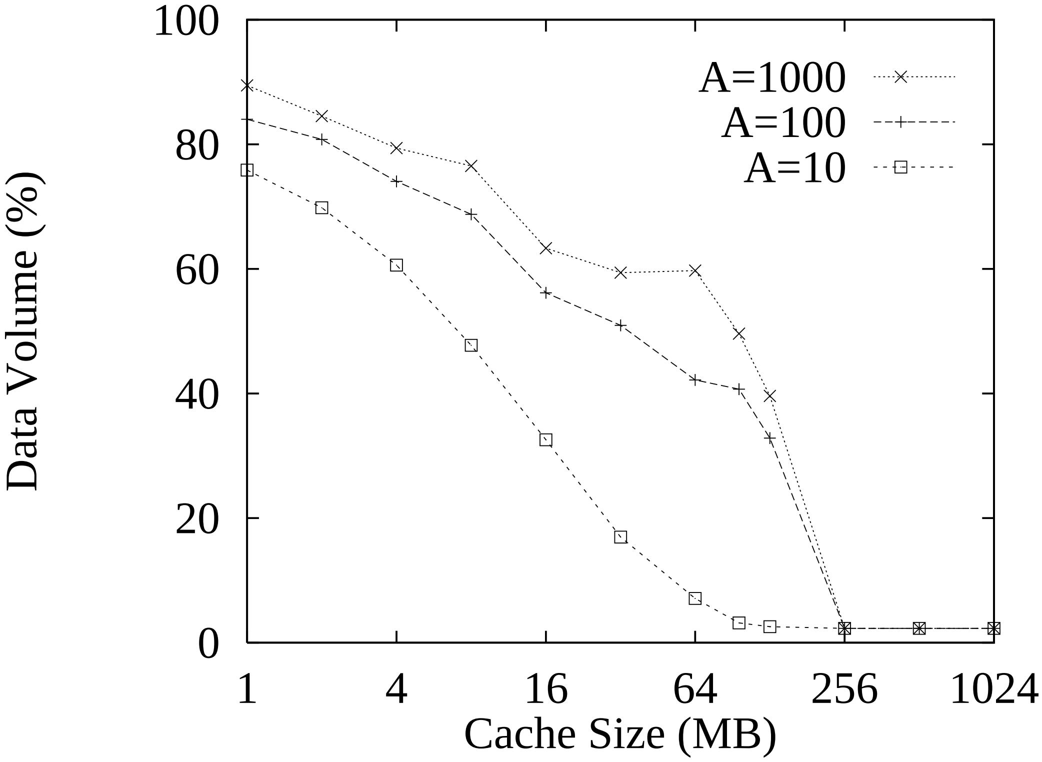


(b)

Figure 6.8: Effects of A_{max} Parameter on LFU*-Aging Policy: (a) Cache Hit Rate (b) Data Volume (Waterloo Data)



(a)



(b)

Figure 6.9: Effects of A_{max} Parameter on LFU*-Aging Policy: (a) Cache Hit Rate (b) Data Volume (Calgary Data)

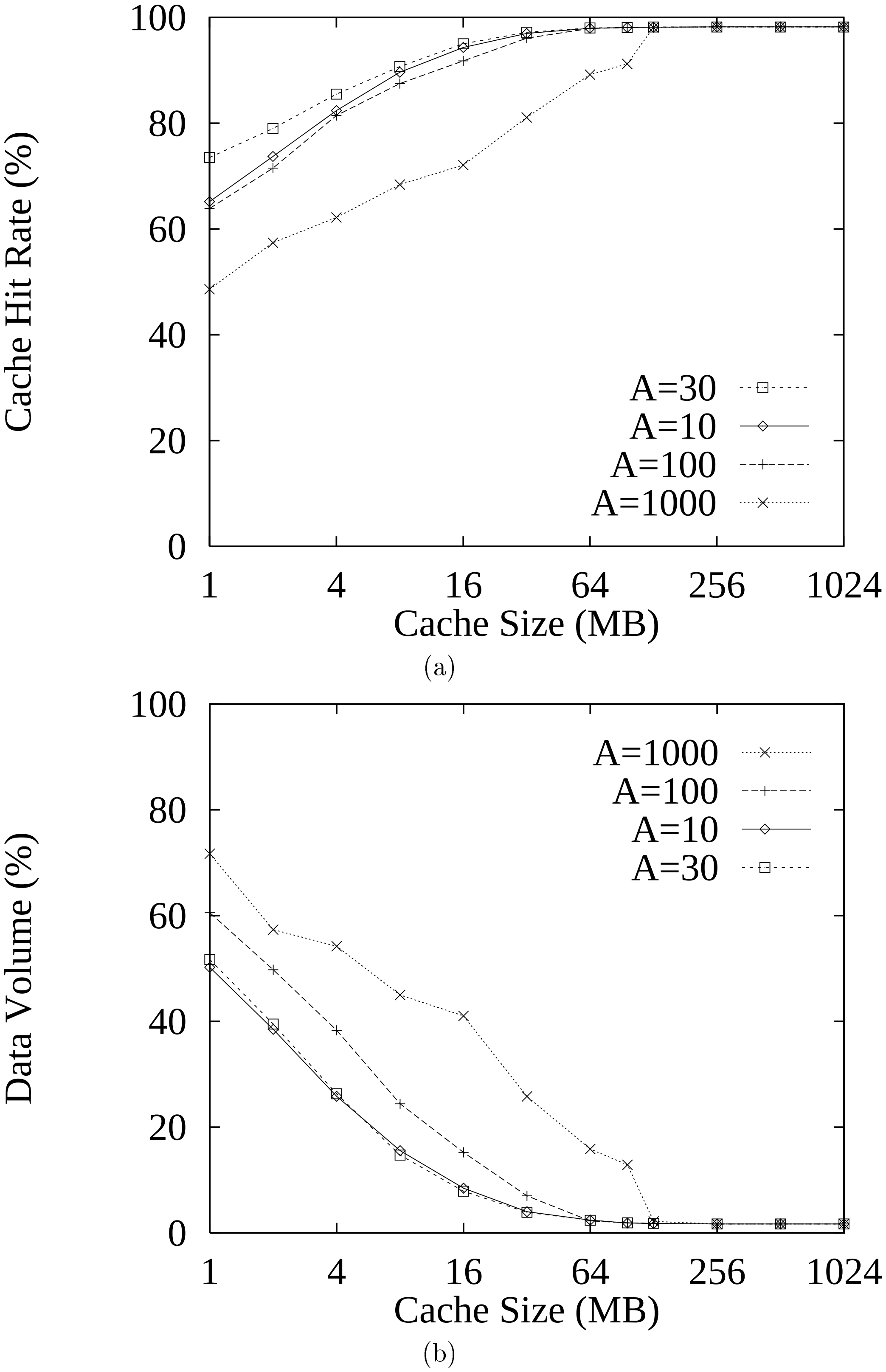
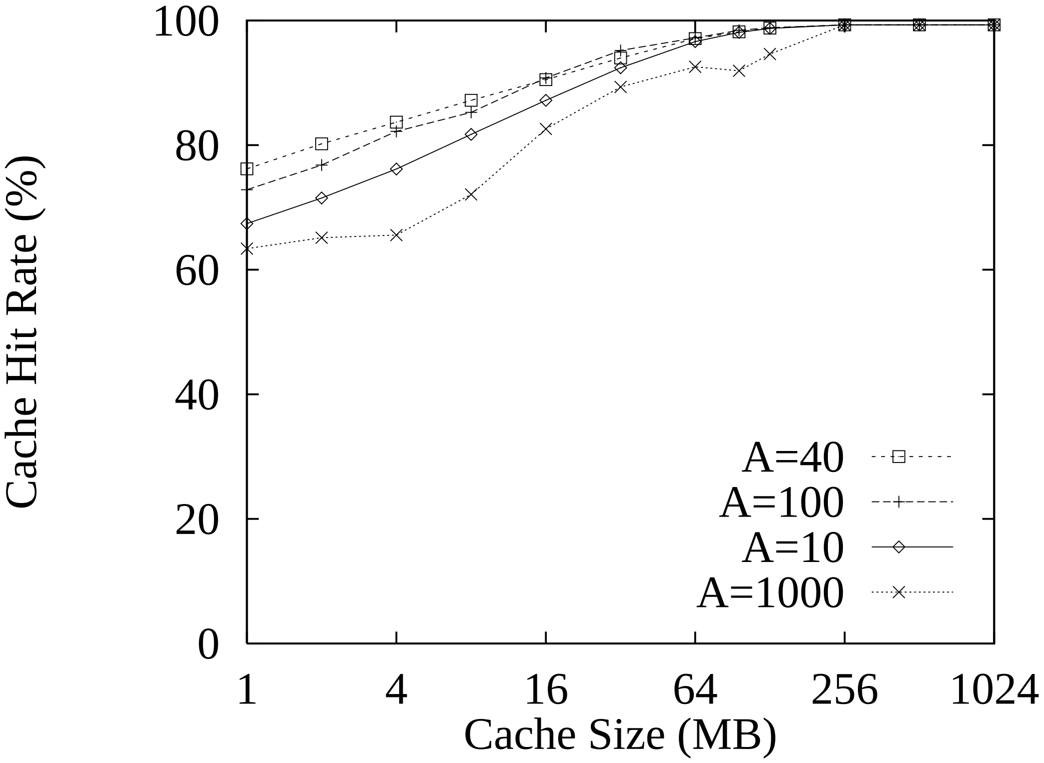
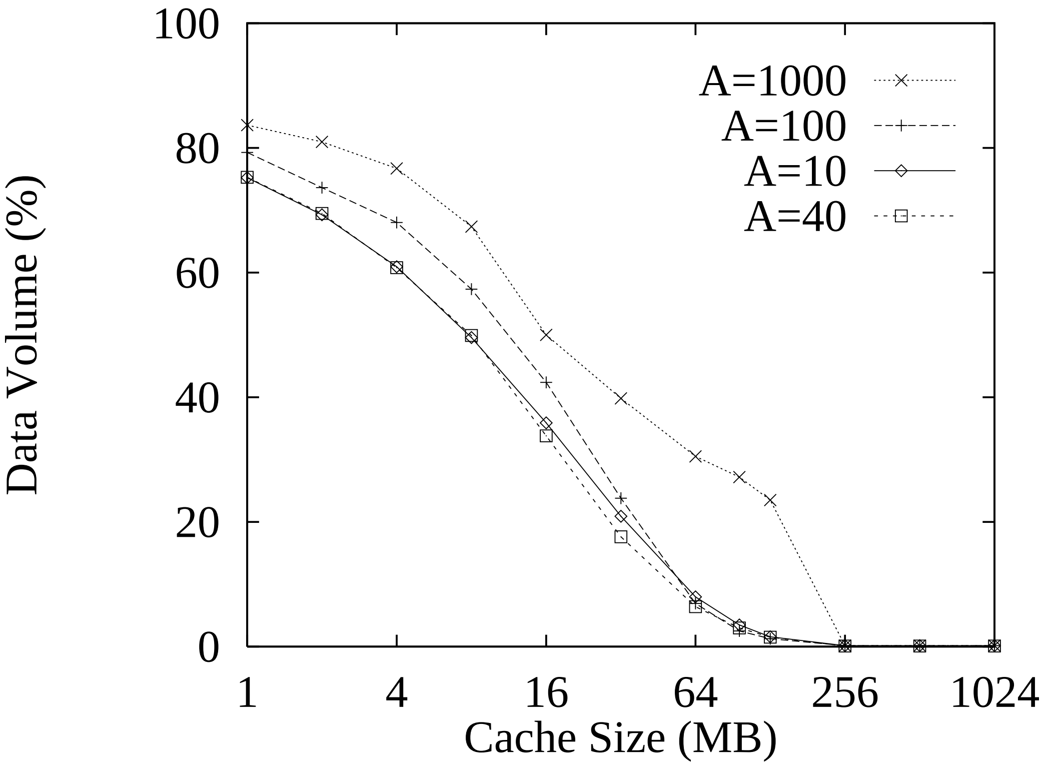


Figure 6.10: Effects of A_{max} Parameter on LFU*-Aging Policy: (a) Cache Hit Rate (b) Data Volume (Saskatchewan Data)



(a)



(b)

Figure 6.11: Effects of A_{max} Parameter on LFU*-Aging Policy: (a) Cache Hit Rate (b) Data Volume (NASA Data)

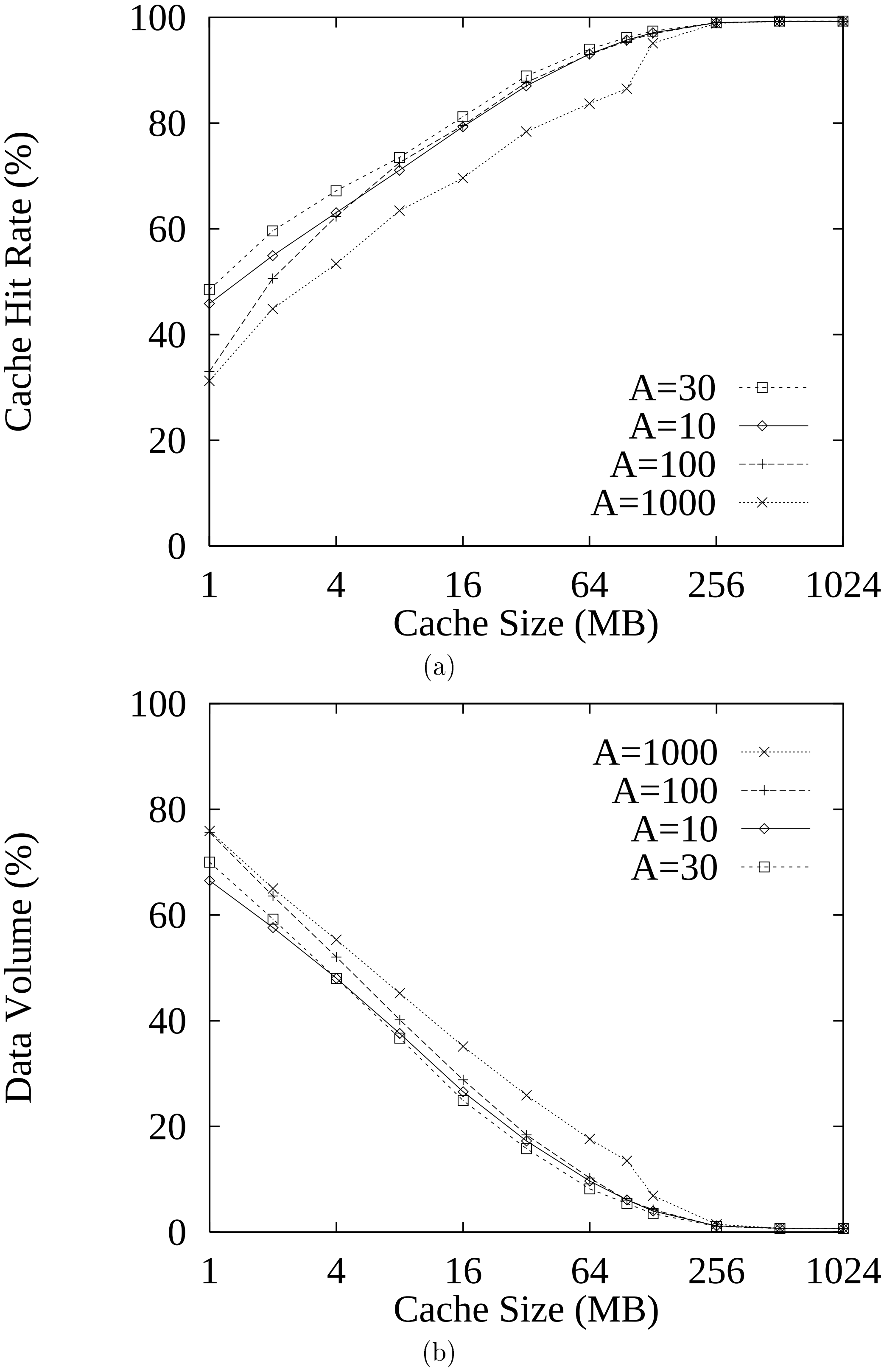
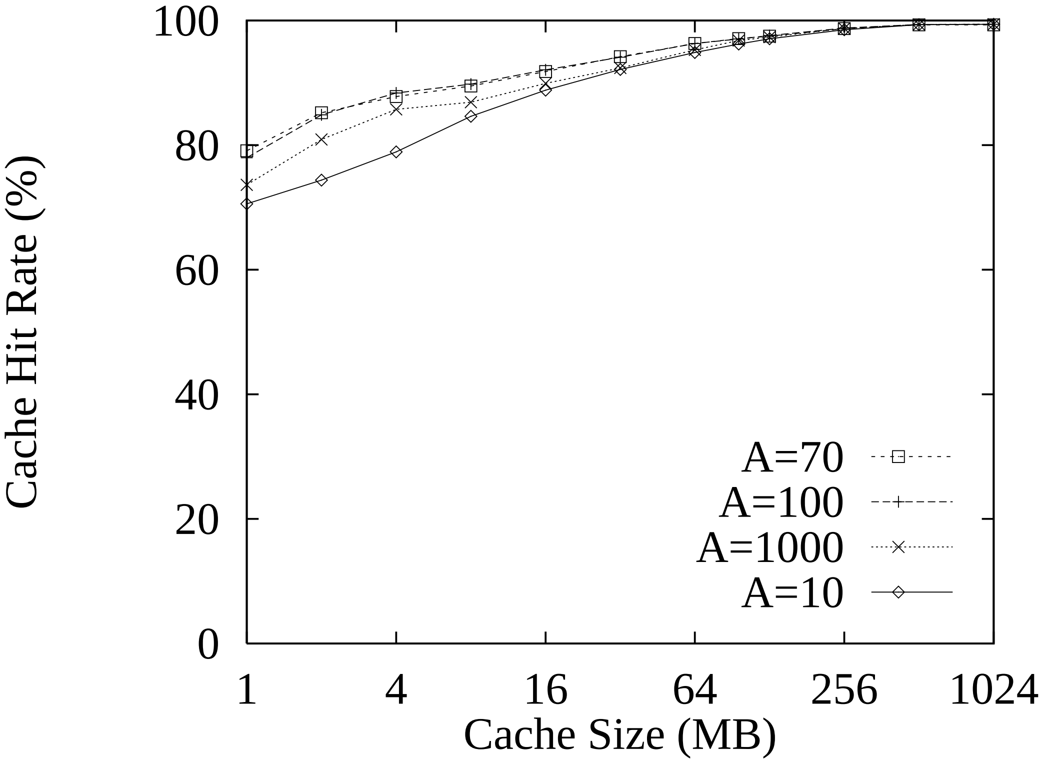
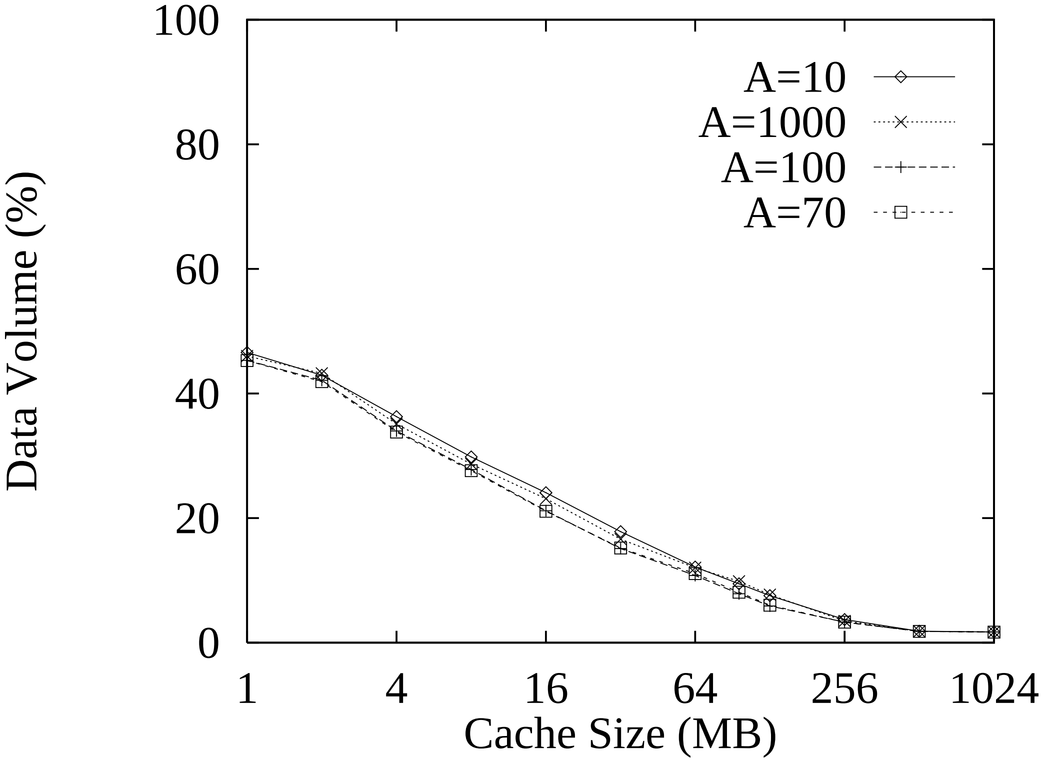


Figure 6.12: Effects of A_{max} Parameter on LFU*-Aging Policy: (a) Cache Hit Rate (b) Data Volume (ClarkNet Data)



(a)



(b)

Figure 6.13: Effects of A_{max} Parameter on LFU*-Aging Policy: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

Table 6.1: Value of A_{max} Chosen (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
A_{max}	10	10	30	40	30	70

Figure 6.14 shows the effect of the M_{refs} parameter on the performance of a file cache using the LFU-Aging replacement policy, for the NCSA data set only. For these experiments, $A_{max} = 70$ was used. Figure 6.14 shows that the choice of M_{refs} has little effect on the performance of the file cache.

Figures 6.15 to 6.20 show the effect of the M_{refs} parameter on the performance of a file cache using LFU*-Aging replacement, (as well as A_{Max} as given in Table 6.1, no threshold, one shared partition) across the six data sets. Three different settings are used: 100, 1000, and ∞ (no limit).

The setting of this parameter has minimal effect on the performance of the file cache. For example, Figure 6.18(a) shows that for the NASA data, setting $M_{ref} = 100$ results in a slight increase in the cache hit rate, while Figure 6.18(b) shows that having no limit on the reference count results in a decrease in the data volume. Since the difference in all cases is quite small, no limit was used for M_{refs} in the remaining simulations.

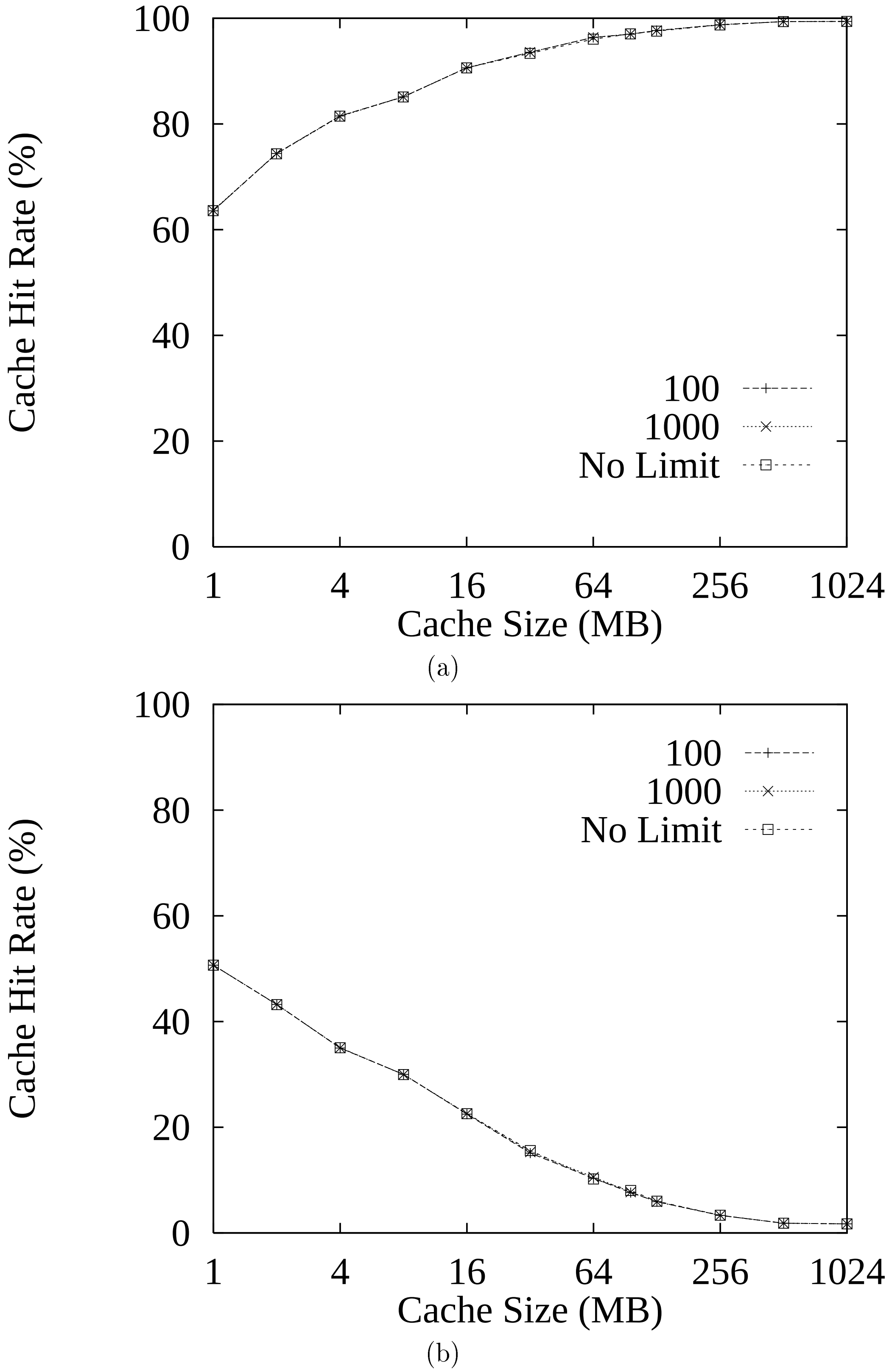


Figure 6.14: Effects of M_{refs} Parameter on Performance of LFU-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

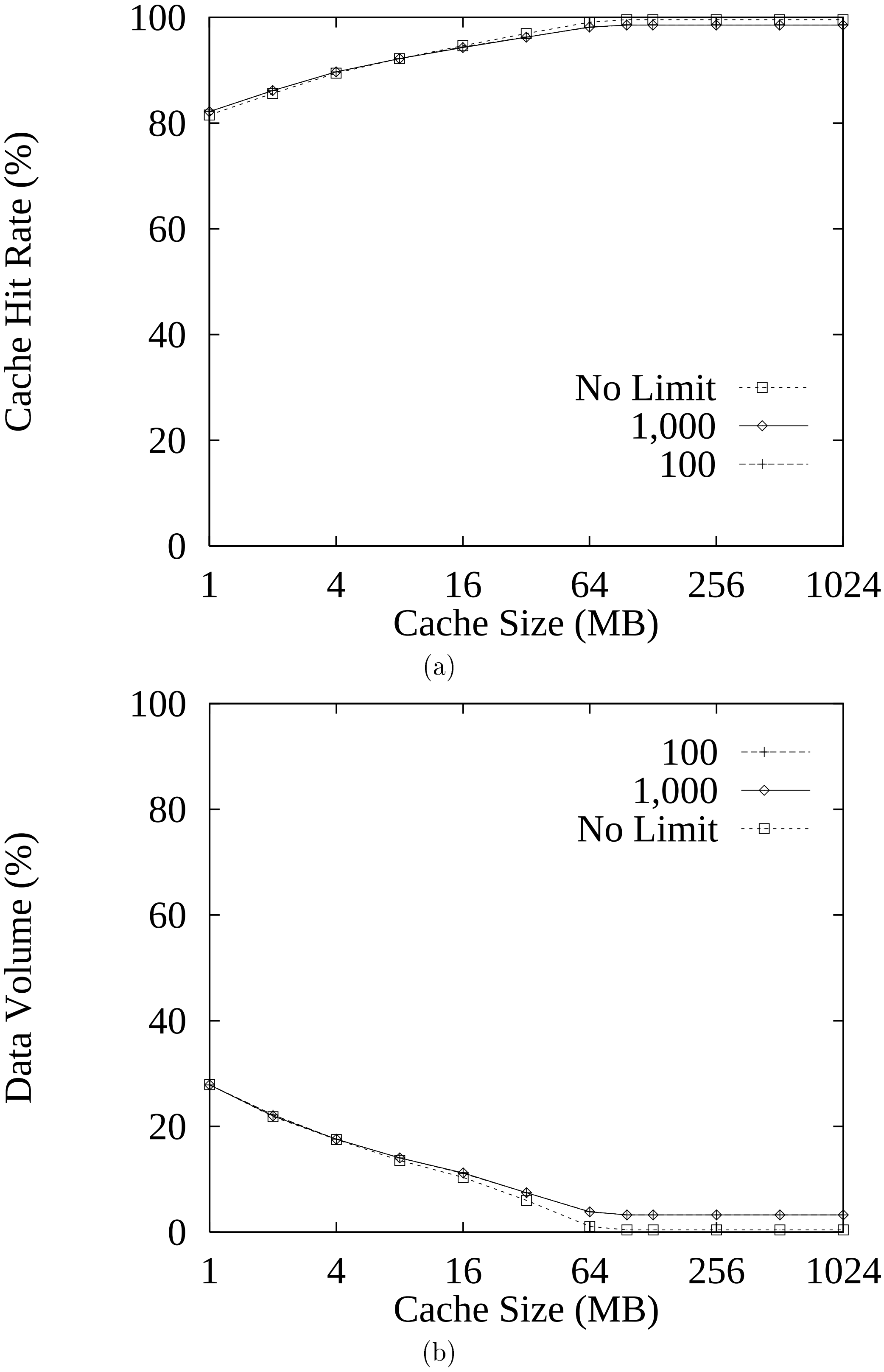
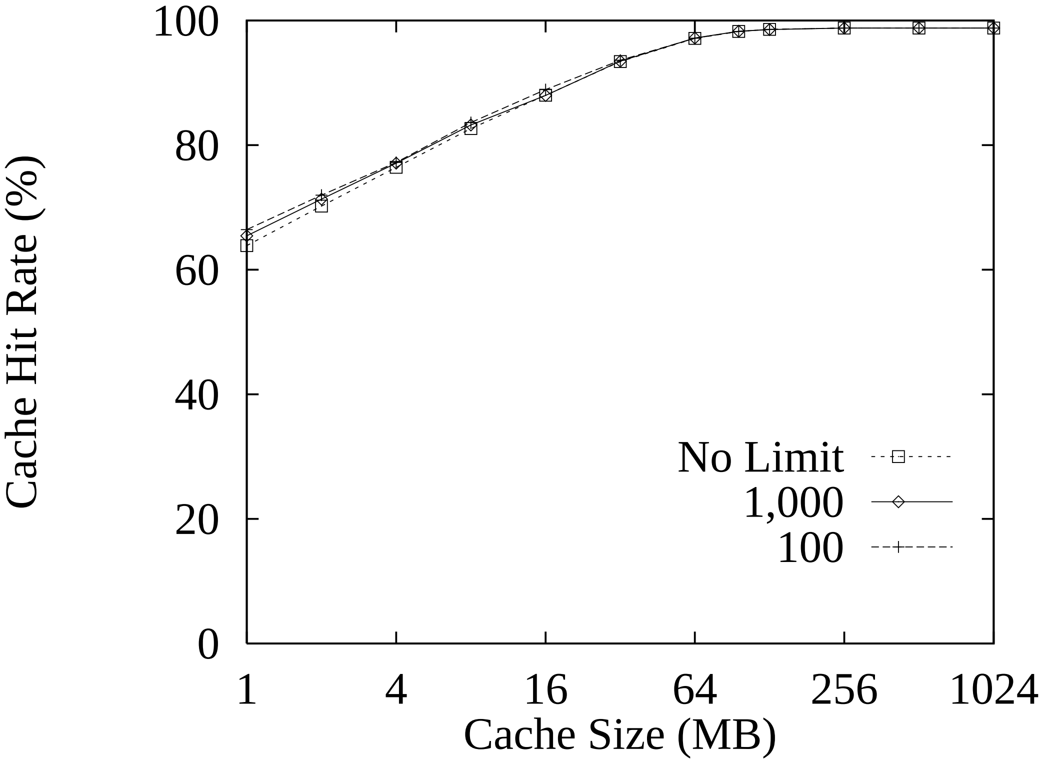
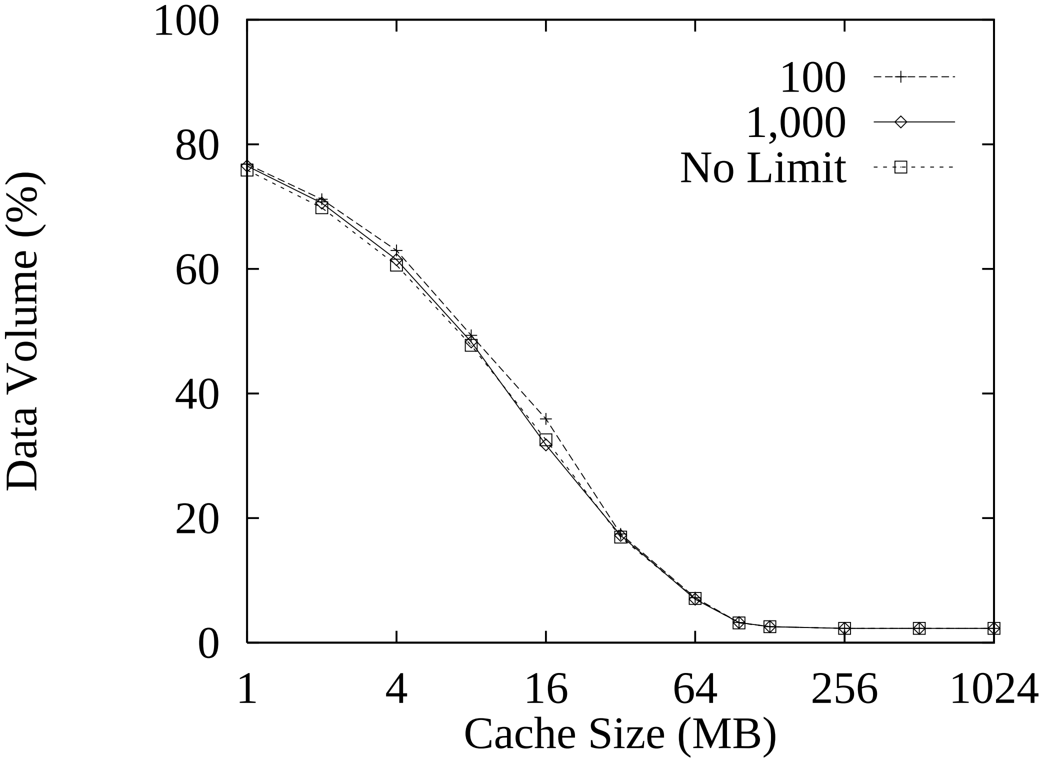


Figure 6.15: Effects of M_{refs} Parameter on Performance of LFU*-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (Waterloo Data)



(a)



(b)

Figure 6.16: Effects of M_{refs} Parameter on Performance of LFU*-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (Calgary Data)

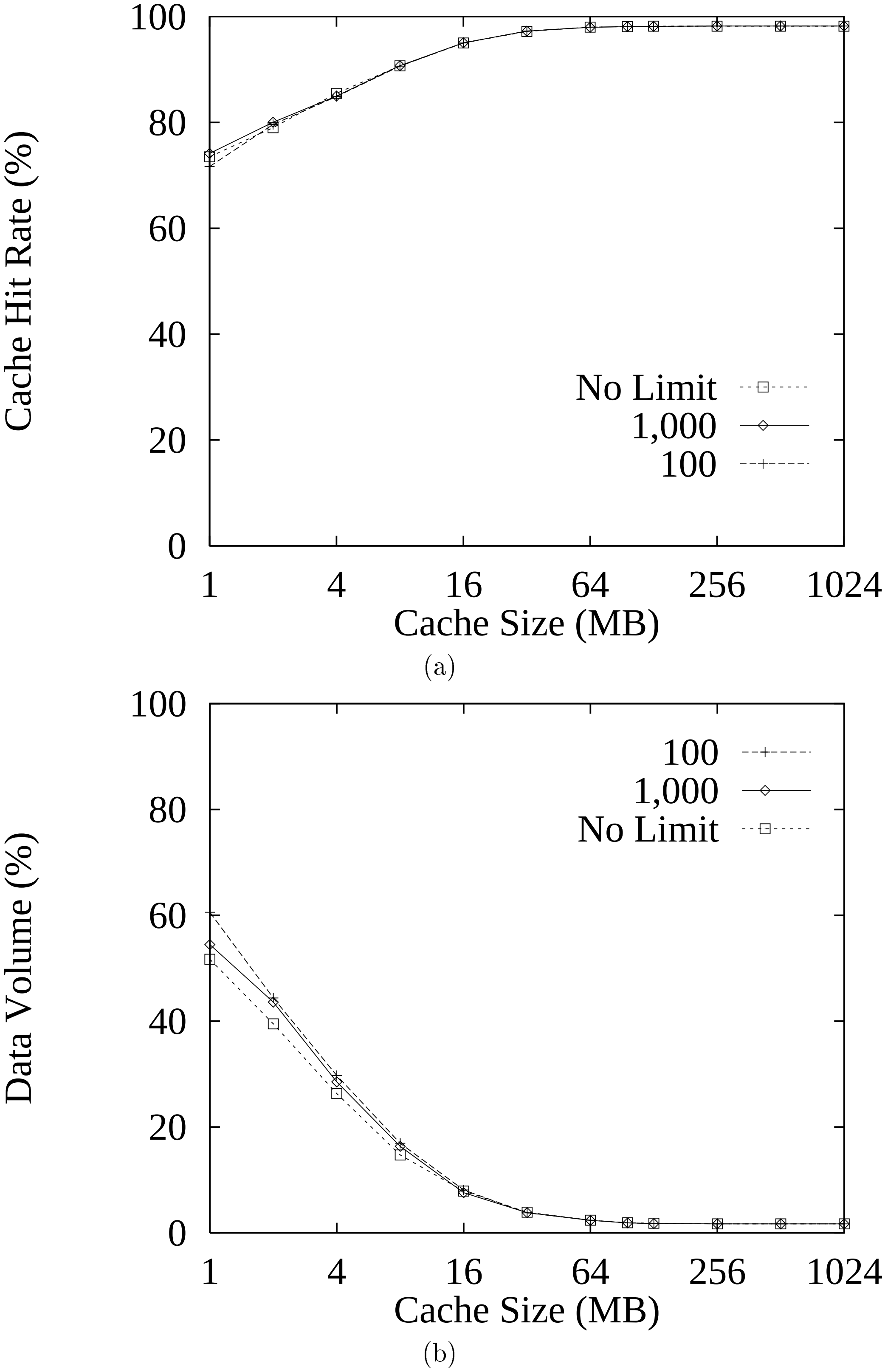


Figure 6.17: Effects of M_{refs} Parameter on Performance of LFU*-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (Saskatchewan Data)

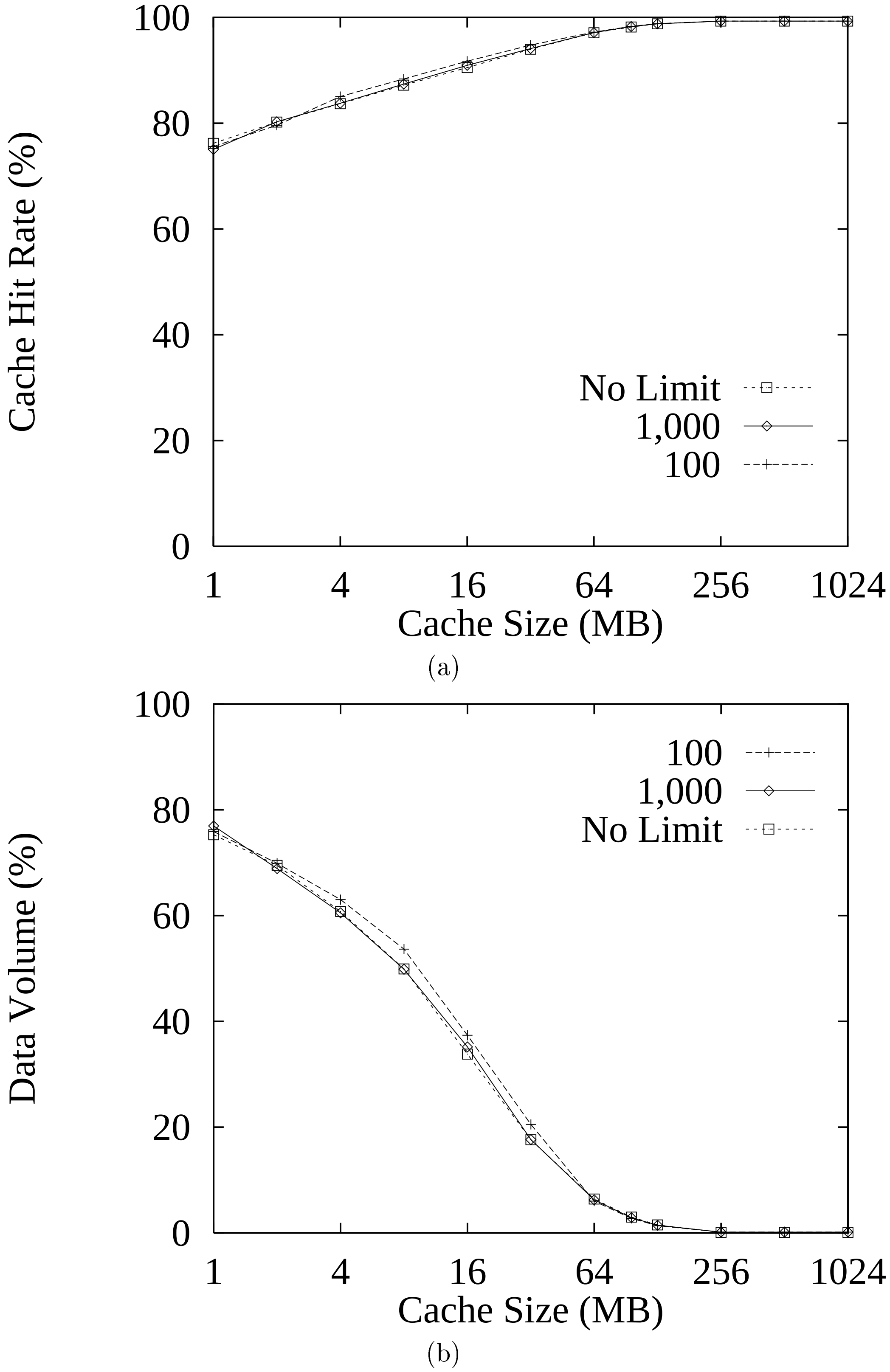


Figure 6.18: Effects of M_{refs} Parameter on Performance of LFU*-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (NASA Data)

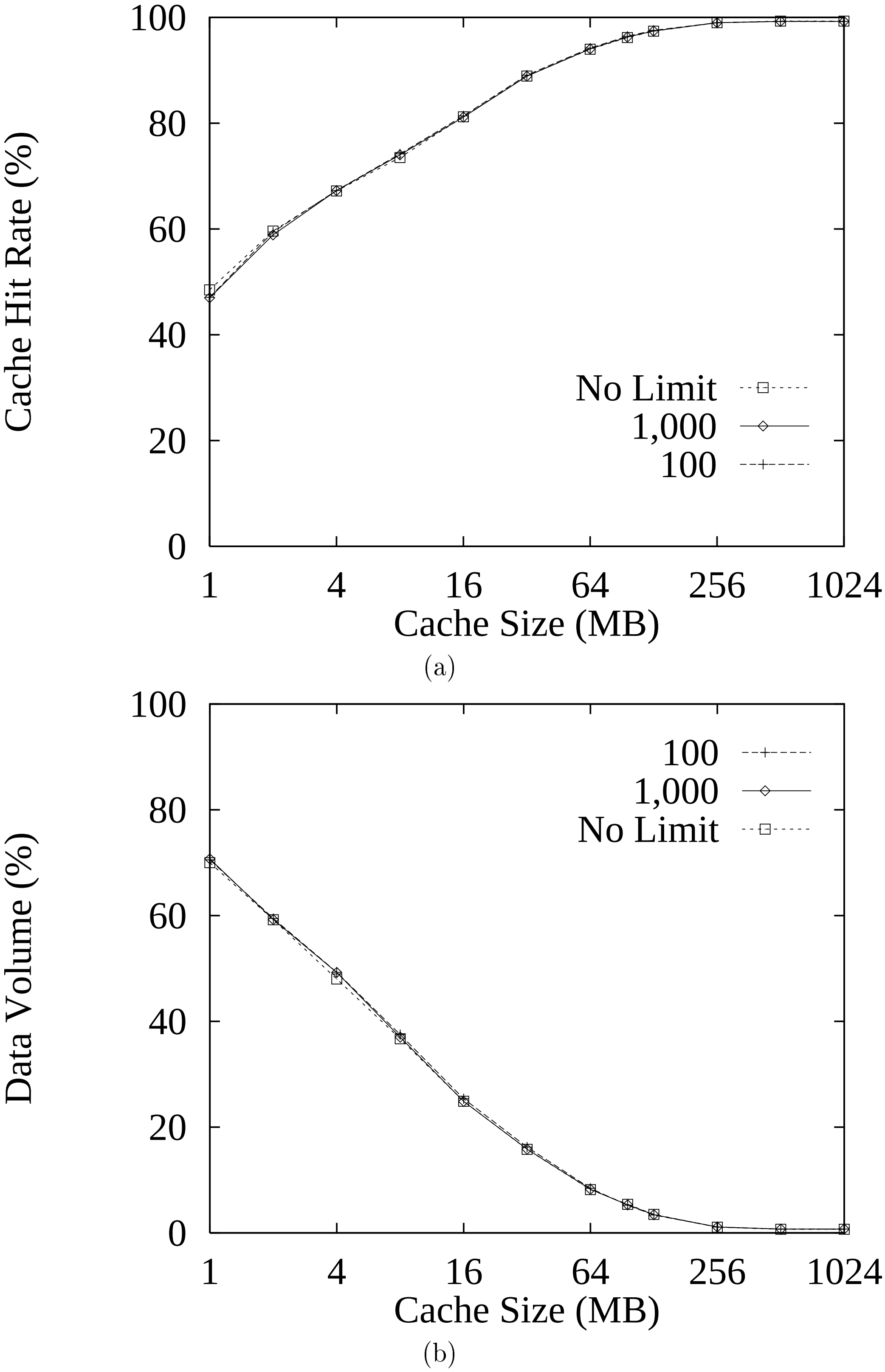
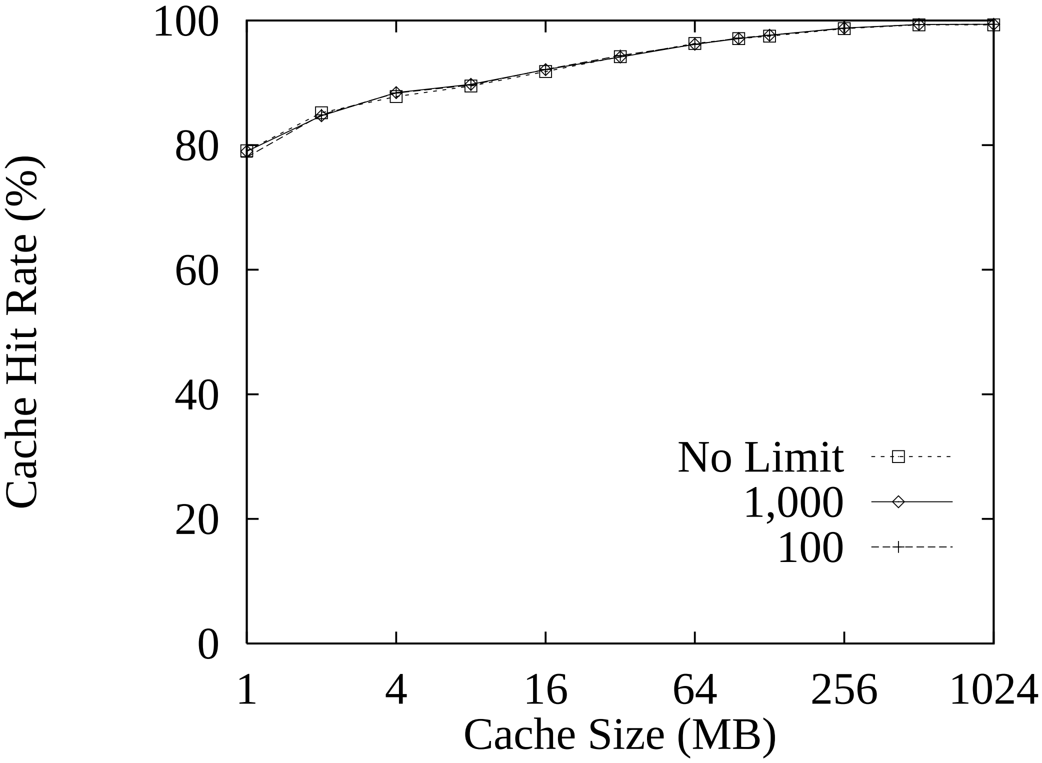
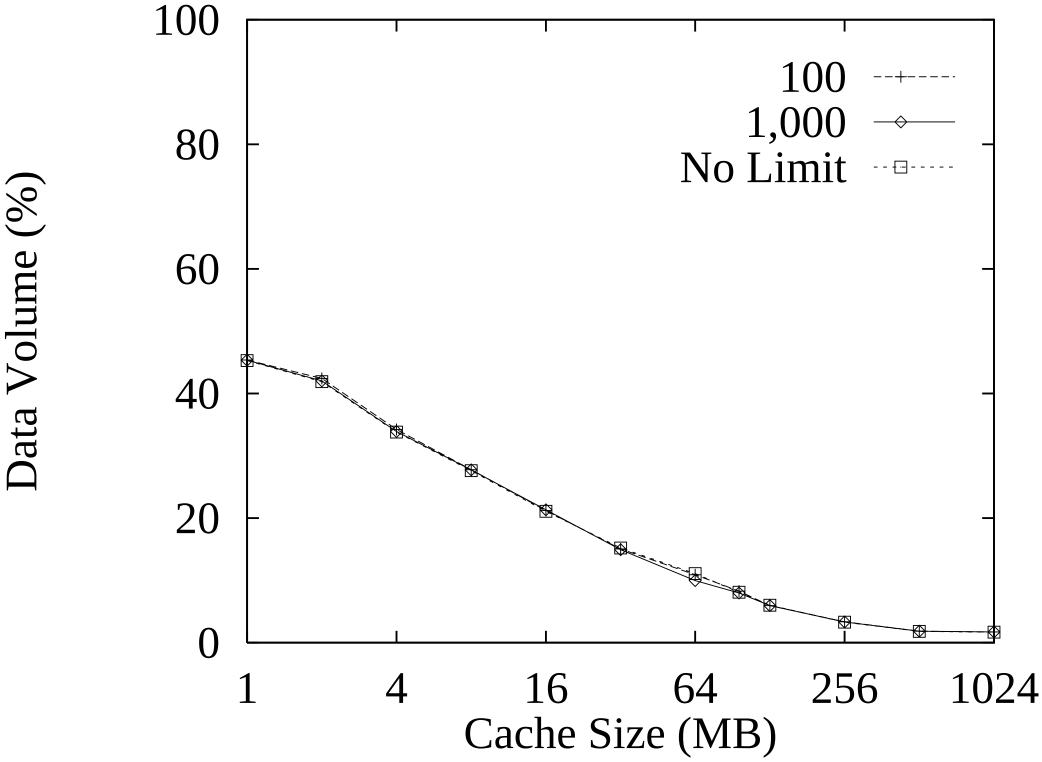


Figure 6.19: Effects of M_{refs} Parameter on Performance of LFU*-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (ClarkNet Data)



(a)



(b)

Figure 6.20: Effects of M_{refs} Parameter on Performance of LFU*-Aging Replacement Policy: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

6.3 Effects of Cache Replacement Policies

With the parameter setting preliminaries completed, the main experiments were then performed. The first group of experiments evaluated the effects of different replacement policies on the performance of a file cache. The four policies tested were LFU-Aging, LRU, FBR, and LFU*-Aging. The first three of these replacement policies are discussed in Section 5.3.3.2. The LFU*-Aging policy is described in Section 6.1. The replacement policy that achieves the highest cache hit rate and the lowest data volume is considered to be the best policy. Although LFU-Aging did not perform as well as LFU*-Aging (see Section 6.1), it is included in this Section to see how the standard LFU policy (with aging) compares with the LRU and FBR policies.

Figures 6.21 to 6.26 show the results of the replacement policy experiments. As is to be expected, the cache hit rate increases for all policies as the cache size increases, and the data volume correspondingly decreases.

Performance differences between the policies are most evident on small caches (when good replacement decisions are important) and negligible on large cache sizes (when very few replacements occur). The relative order is quite consistent across the range of cache sizes considered. In general, the modified LFU algorithm with aging (LFU*-Aging) provides the best performance, while the LRU policy, traditionally one of the "best" caching policies in computer systems performance studies [16, 57, 70], performs the worst. The FBR policy, which combines the notions of LRU (in the New Section) and LFU (in the Old Section) generally provides performance between that of LFU-Aging and LRU (better than LRU, but not as good as LFU-Aging or LFU*-Aging). These results indicate that the extra implementation overhead of FBR caching is not likely to be worthwhile for Web server caching. LFU*-Aging is a simpler policy to implement, and consistently provides better performance.

Since the LFU*-Aging policy always performs as well or better than LFU-Aging, the LFU-Aging policy is not included in the remaining experiments.

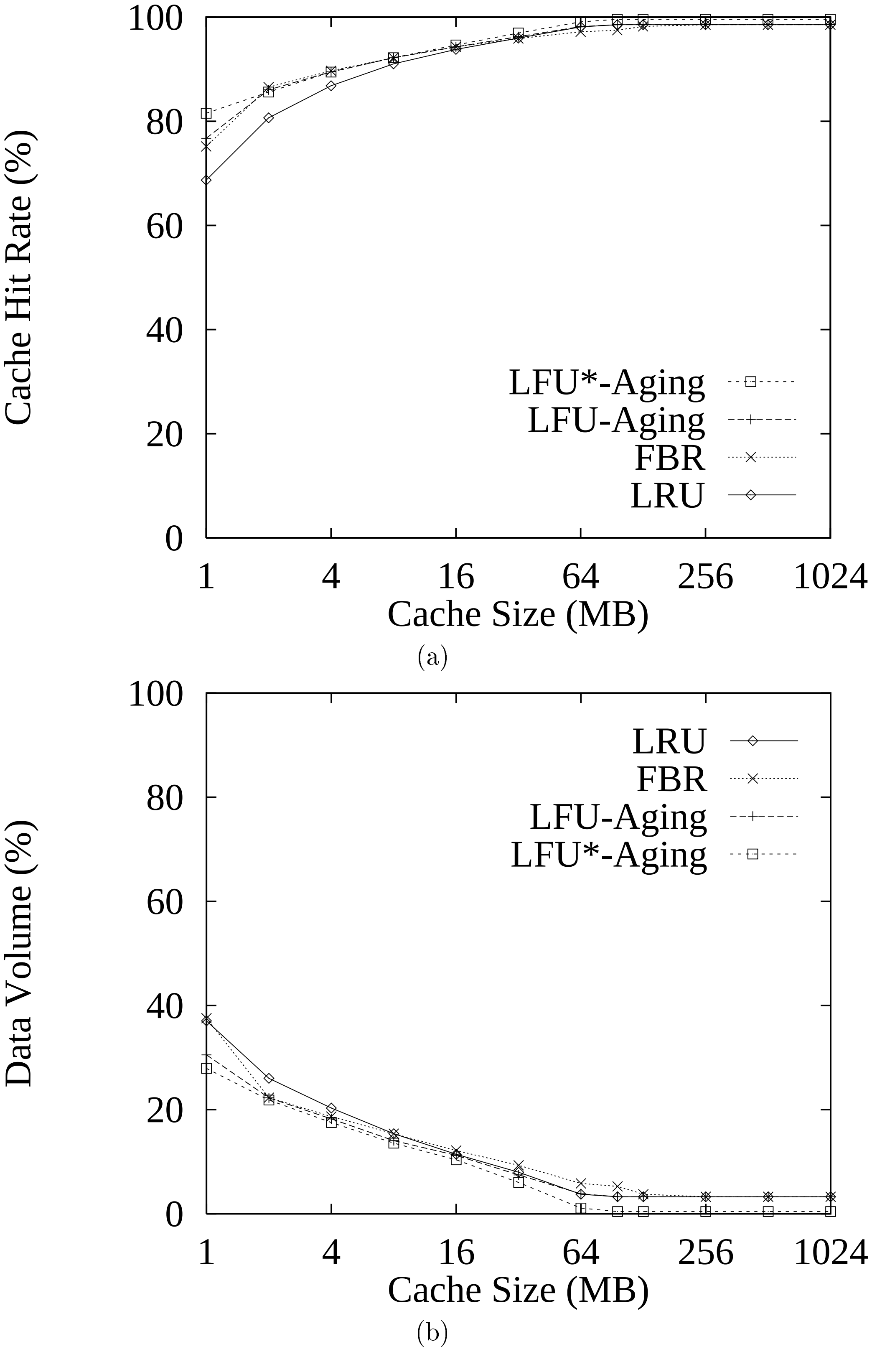


Figure 6.21: Comparison of Cache Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Waterloo Data)

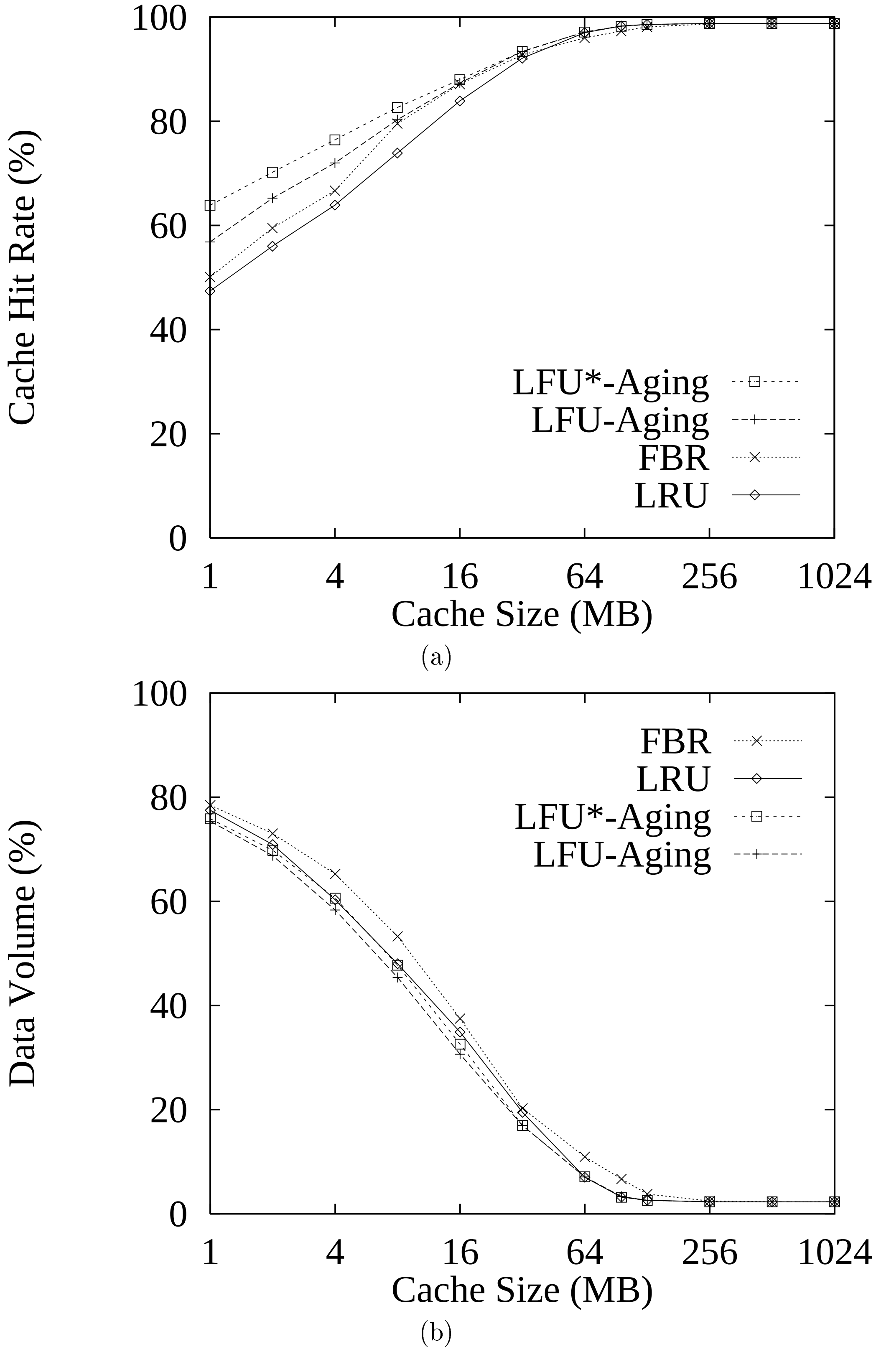
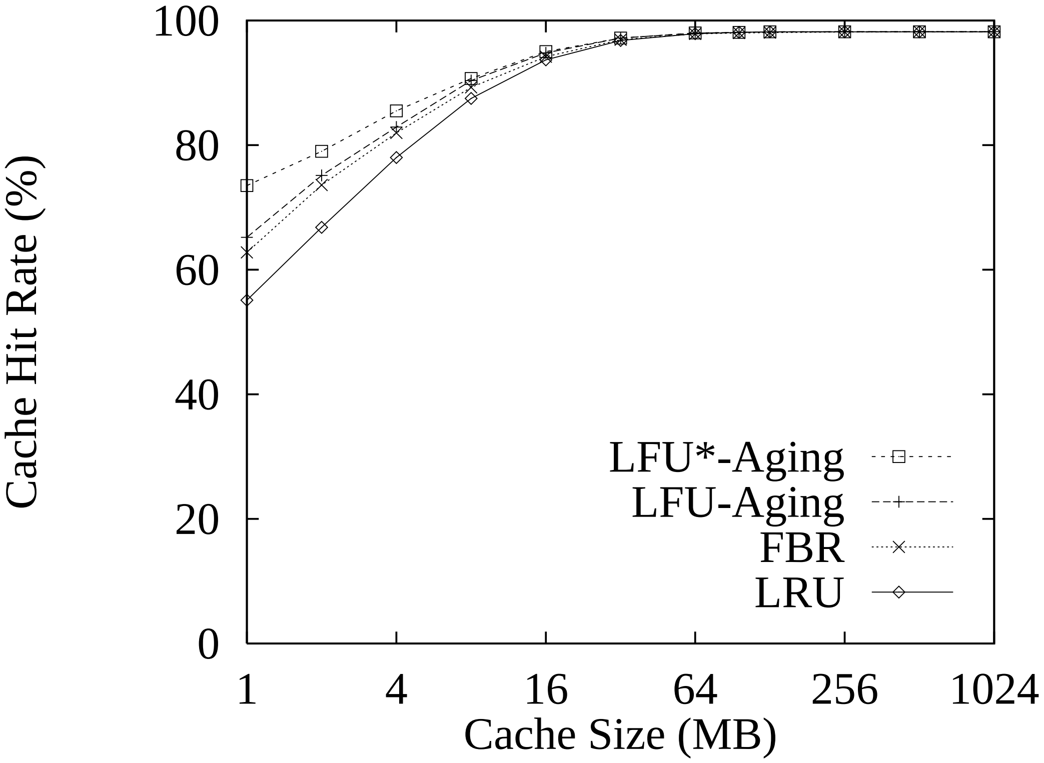
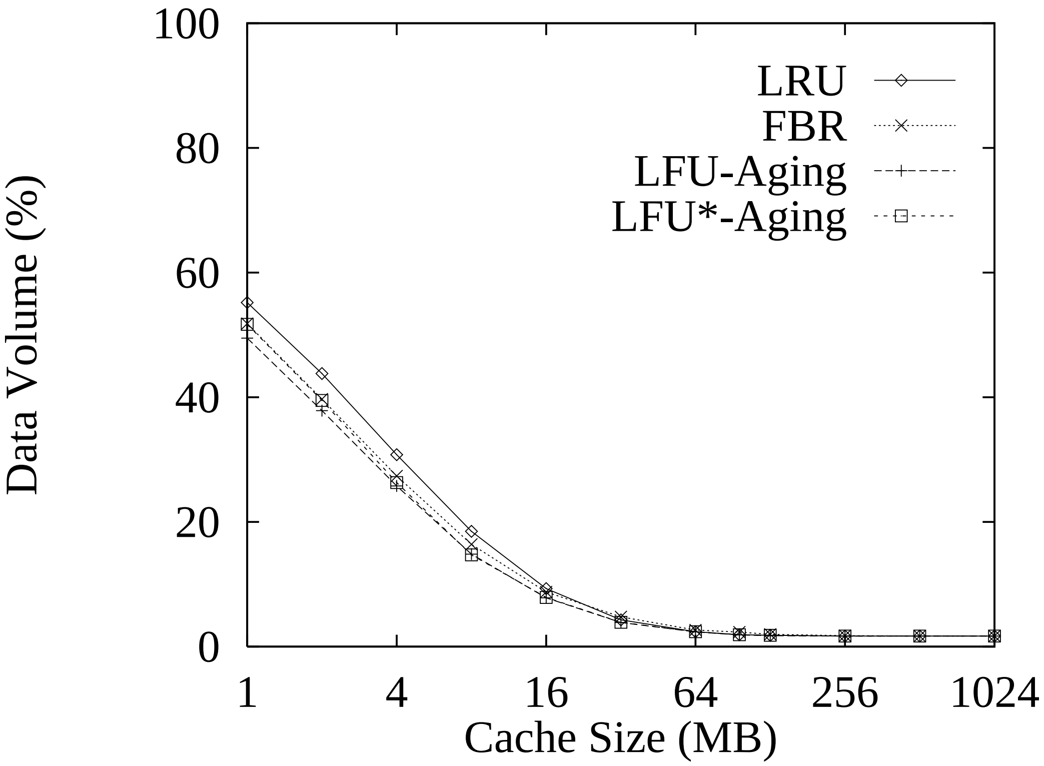


Figure 6.22: Comparison of Cache Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Calgary Data)



(a)



(b)

Figure 6.23: Comparison of Cache Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Saskatchewan Data)

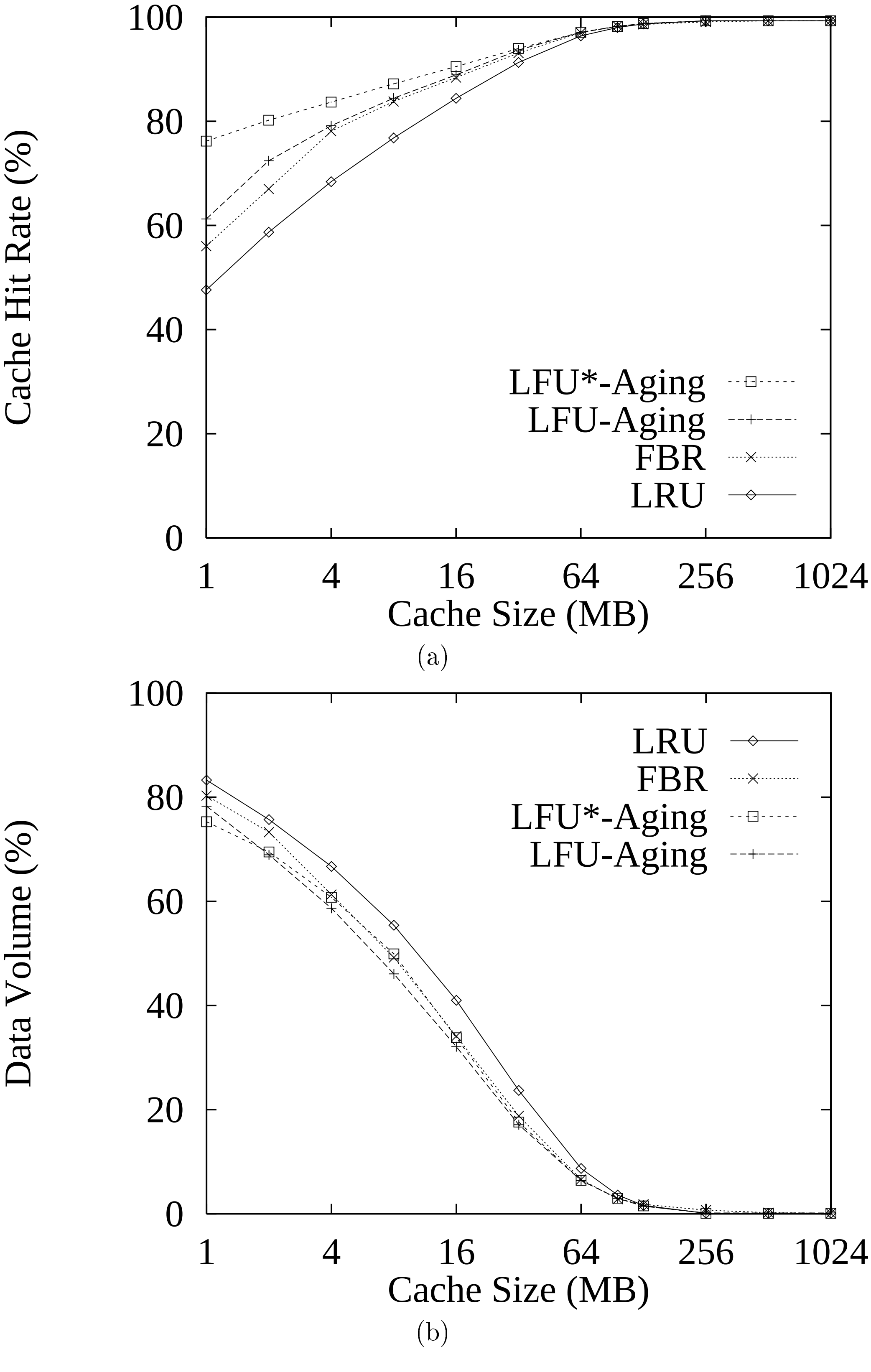
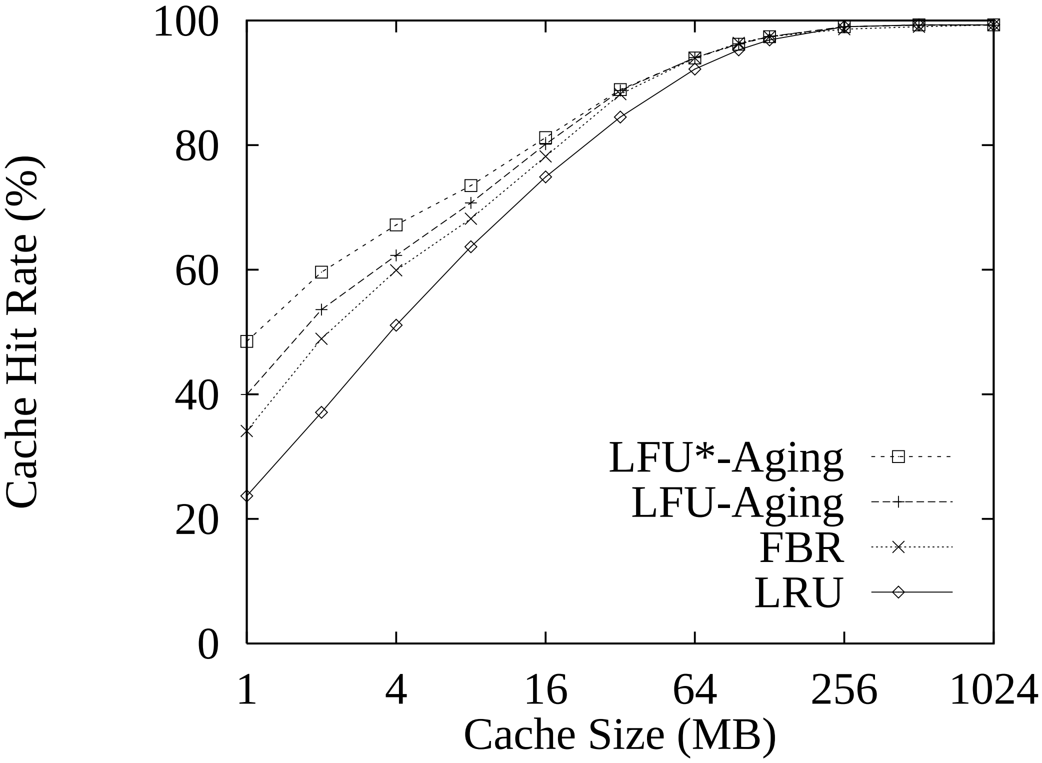
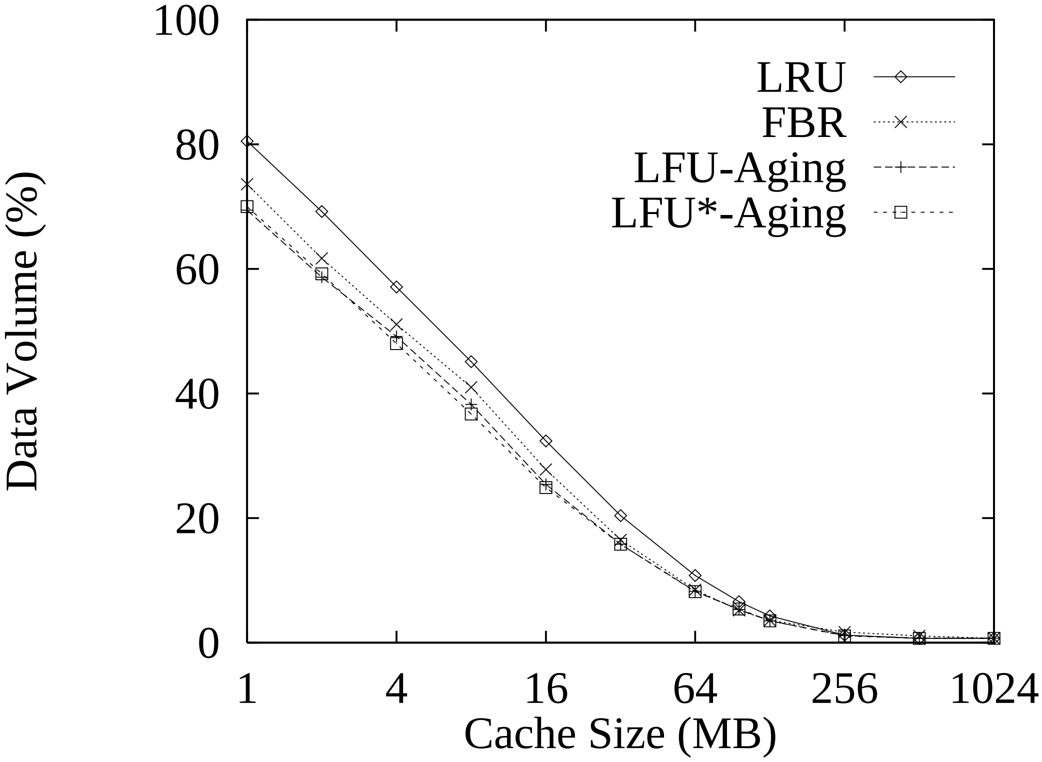


Figure 6.24: Comparison of Cache Replacement Policies: (a) Cache Hit Rate (b) Data Volume (NASA Data)

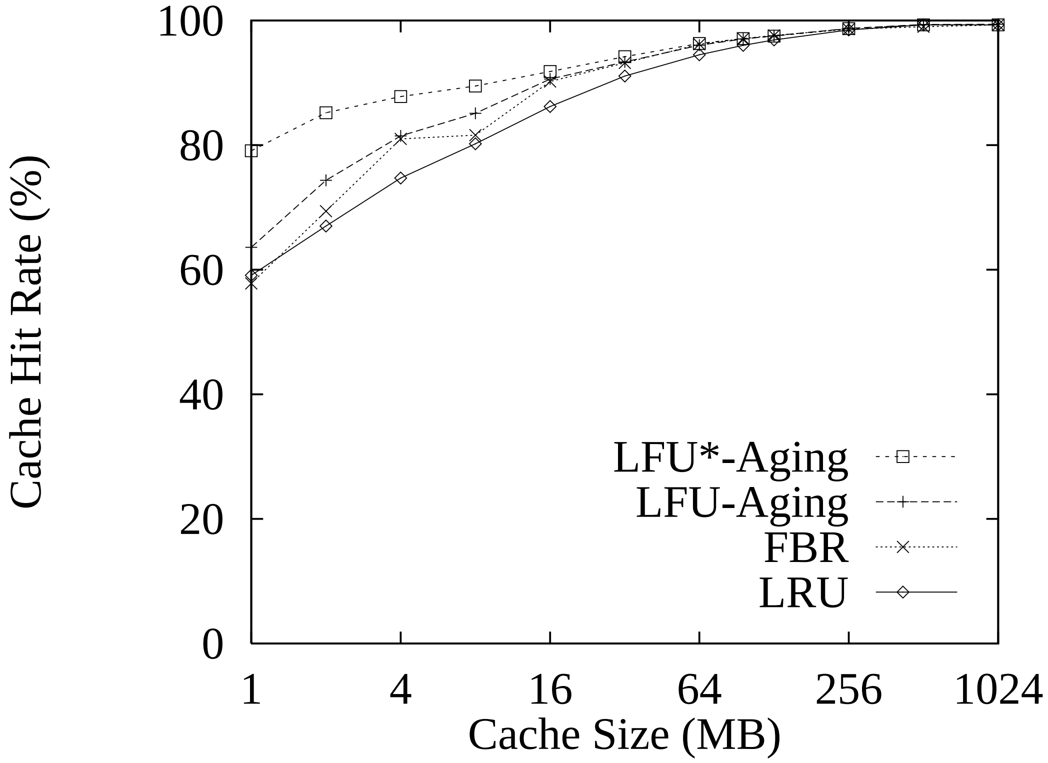


(a)

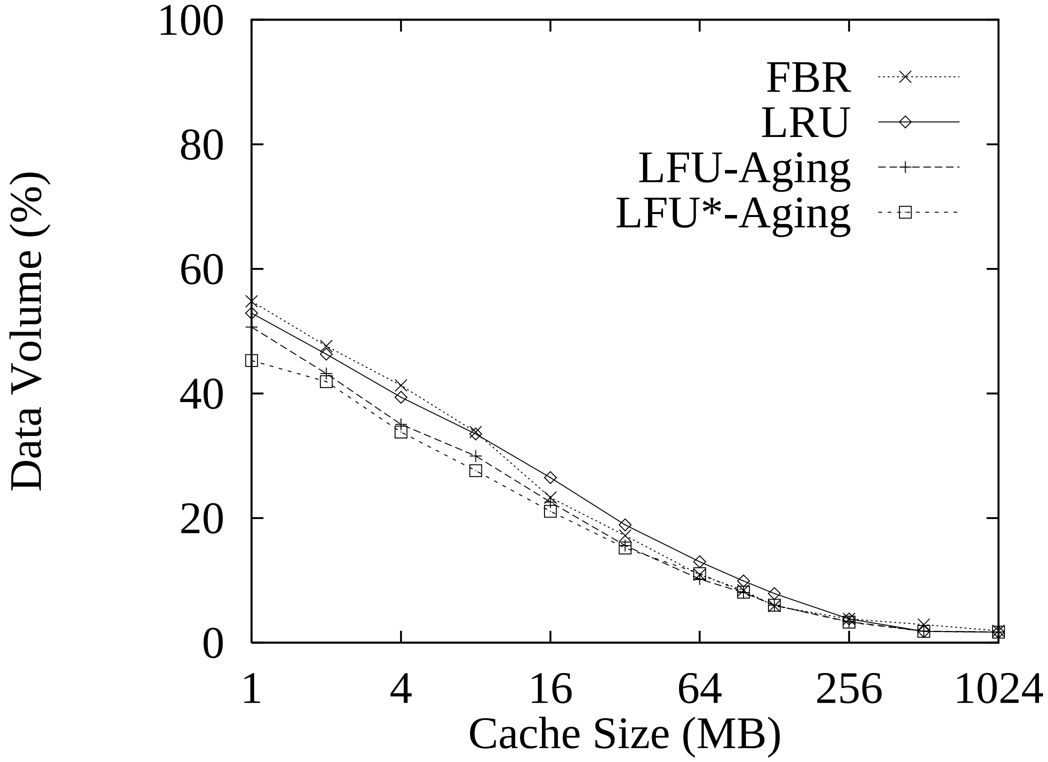


(b)

Figure 6.25: Comparison of Cache Replacement Policies: (a) Cache Hit Rate (b) Data Volume (ClarkNet Data)



(a)



(b)

Figure 6.26: Comparison of Cache Replacement Policies: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

6.4 Effects of a Threshold Policy

The next group of experiments studied the effects of a threshold policy on the performance of a file cache. The threshold policy places a constraint on the sizes of documents allowed into the cache. Three of the replacement policies (LFU*-Aging, LRU, and FBR) are used in combination with the threshold policies.

Four different levels are tested for the threshold policy. The levels are: (i) no restriction on document size (i.e., no threshold policy; the results for this level are the same as those in Section 6.3); (ii) an upper limit on document size; (iii) a lower limit on document size; and (iv) both an upper and a lower limit on file size.

The results for the threshold experiments are shown in Figures 6.27 to 6.32. In each of these figures, graphs (a) and (b) show the results for a file cache using LRU replacement and the various threshold policies, graphs (c) and (d) show the results when LFU*-Aging replacement is used in combination with the threshold policies, and graphs (e) and (f) show the results when the threshold policies are used in conjunction with the FBR replacement policy. The key in each graph shows the range of document sizes permitted in the cache for the given policy. Since the LFU*-Aging policy performs the best when no threshold policy is used, the LFU*-Aging curve is included in each graph for ease of comparison.

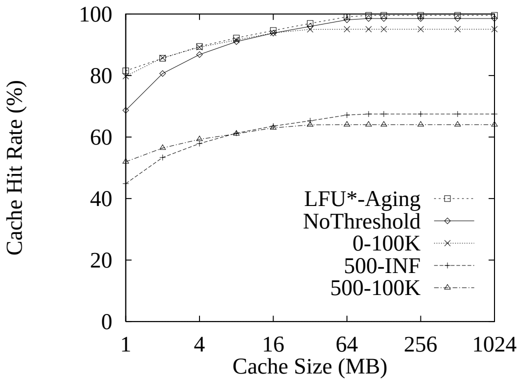
For each data set, an upper limit on document size of 100,000 bytes was used. On average, 95.4% of the documents on the six servers were less than 100,000 bytes in size. The results for this threshold policy are labelled 0-100K on each of the graphs.

Across all of the data sets, the "upper limit only" policy improved the hit rate for small cache sizes, compared to a similar cache with the same replacement policy but no threshold policy. For the LFU*-Aging replacement policy, the performance improvement in cache hit rate was most noticeable in the Calgary dataset (Figure 6.28), but quite marginal in the other datasets. For the LRU and FBR replacement policies, the performance improvements in cache hit rate by having an upper threshold instead of no threshold were slightly greater than for LFU*-Aging, but still resulted in cache hit rates below that achievable by LFU*-Aging with no threshold (with the exception of the Calgary data set).

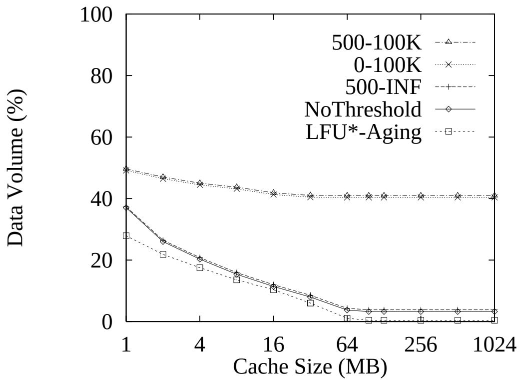
In all cases, using an upper limit threshold policy results in an increase in the data volume. This increase occurs because files that are larger than the maximum cacheable size are responsible for a significant portion of the data volume. This policy thus prevents the cache from minimizing data volume.

A single value (500 bytes) is tested as a lower limit for the threshold policy. Across the six servers, approximately 14.8% of the files were smaller than 500 bytes in size. The results for these experiments are shown in Figures 6.27 to 6.32 as well, and are labelled 500-INF on each of the graphs. As was expected, the cache hit rate worsened for all data sets, since many small, frequently requested files are now excluded from the cache. The intent of this threshold policy is to cache only large files, which are responsible for more of the data traffic. However, the results show that a decrease in the data volume is not achieved. As with the upper limit threshold policy, using a lower limit prevents the cache from reaching the maximum achievable hit rate or the minimum achievable data volume.

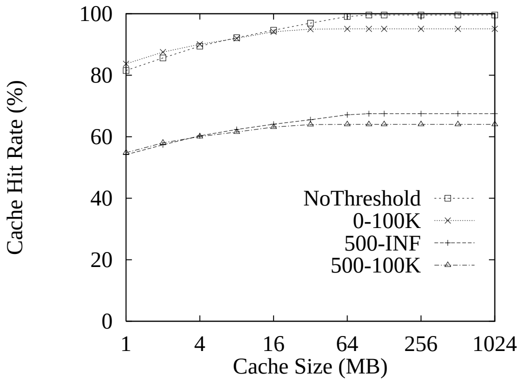
The last set of experiments involving the threshold policy places both an upper and a lower limit on the size of files that may be added to the cache. These experiments used an upper limit of 100,000 bytes and a lower limit of 500 bytes. The results are labelled as 500-100K on Figures 6.27 to 6.32. This policy achieves a slightly better hit rate than using only a lower limit, but does not achieve as high a hit rate as a policy that implements only an upper limit. However, using both an upper and a lower restriction on file size results in a significant increase in the data volume, which is clearly undesirable. Once again the threshold policy is unable to reach either the maximum achievable hit rate or the minimum achievable data volume, no matter how large the cache is.



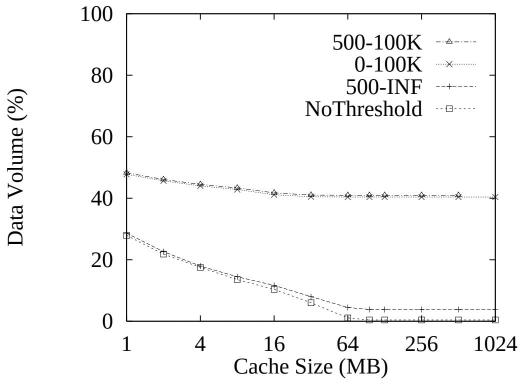
Cache Hit Rate for LRU



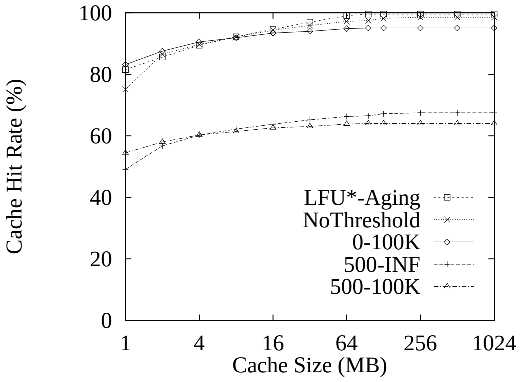
Data Volume for LRU



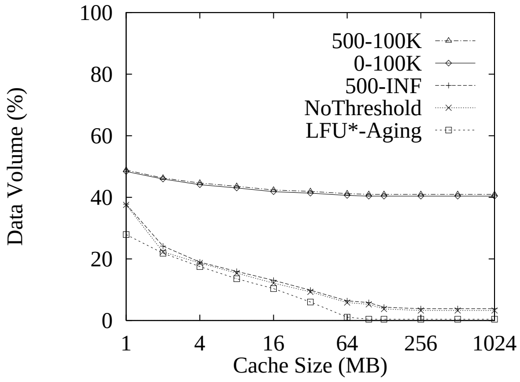
Cache Hit Rate for LFU*-Aging



Data Volume for LFU*-Aging

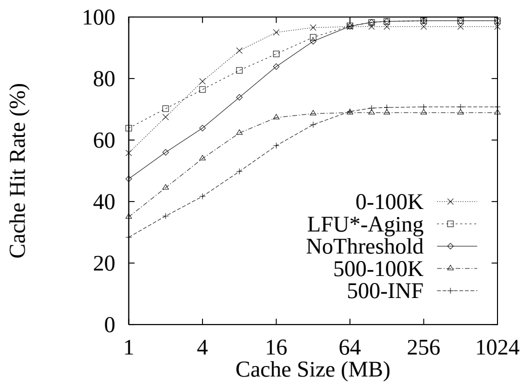


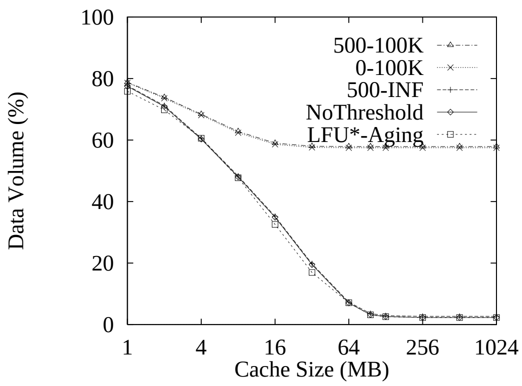
Cache Hit Rate for FBR



Data Volume for FBR

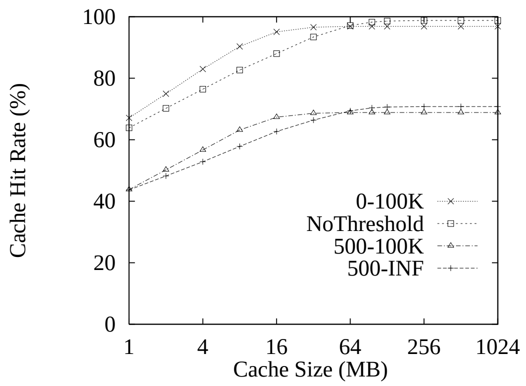
Figure 6.27: Comparison of Document Size Threshold Policies (Waterloo Data)

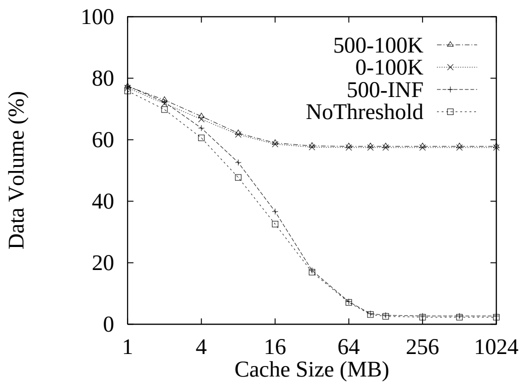




Cache Hit Rate for LRU

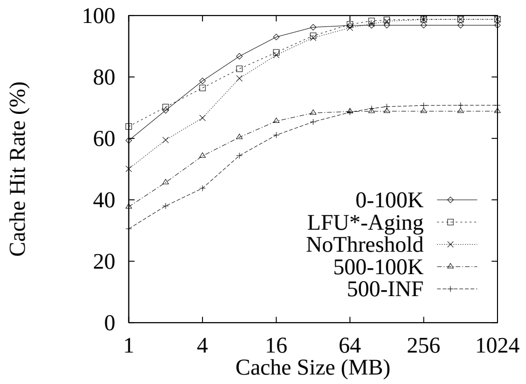
Data Volume for LRU

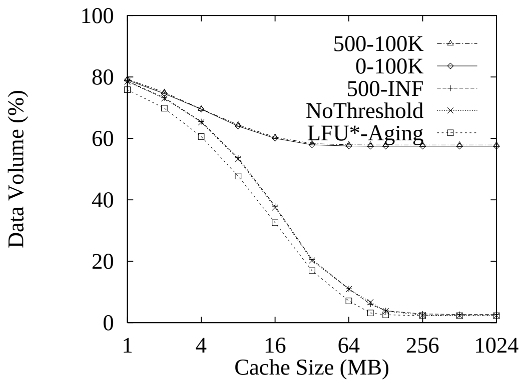




Cache Hit Rate for LFU*-Aging

Data Volume for LFU*-Aging

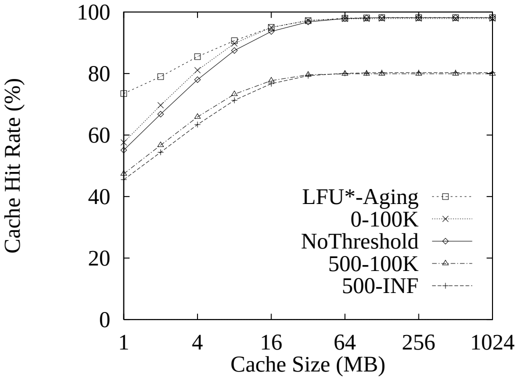




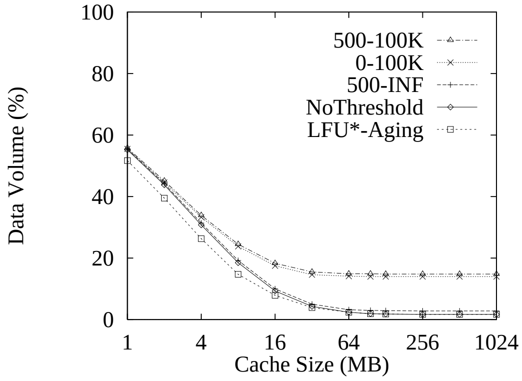
Cache Hit Rate for FBR

Data Volume for FBR

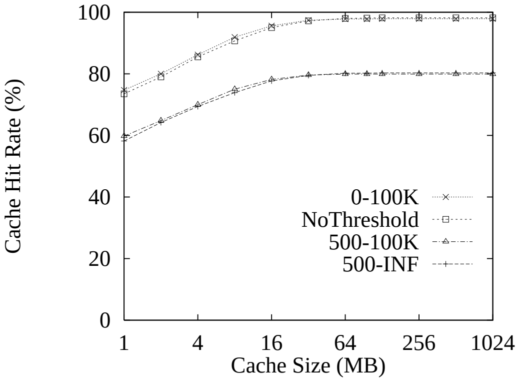
Figure 6.28: Comparison of Document Size Threshold Policies (Calgary Data)



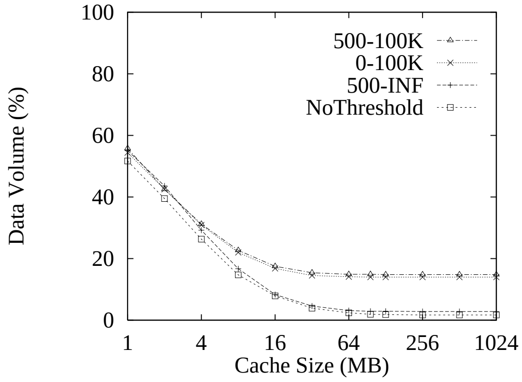
Cache Hit Rate for LRU



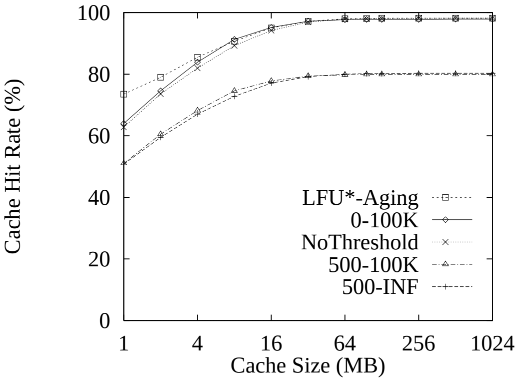
Data Volume for LRU



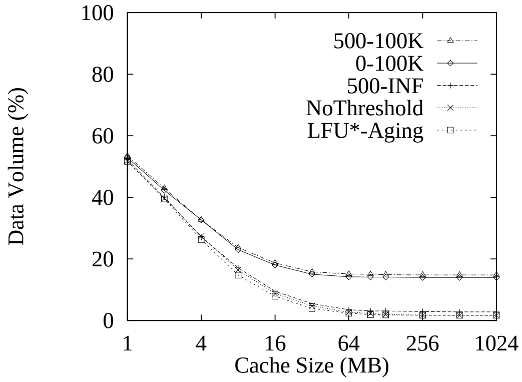
Cache Hit Rate for LFU*-Aging



Data Volume for LFU*-Aging



Cache Hit Rate for FBR



Data Volume for FBR

Figure 6.29: Comparison of Document Size Threshold Policies (Saskatchewan Data)

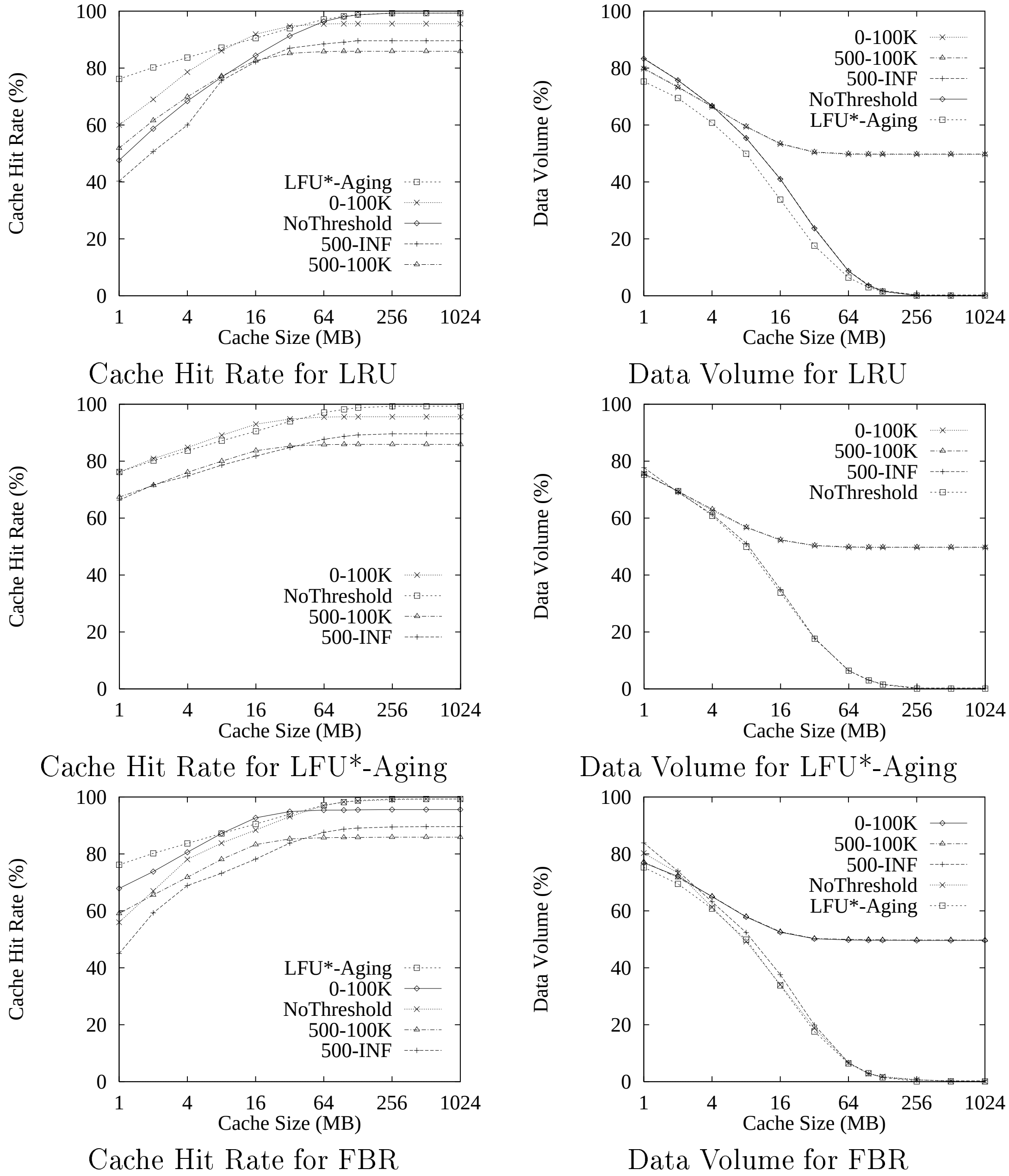


Figure 6.30: Comparison of Document Size Threshold Policies (NASA Data)

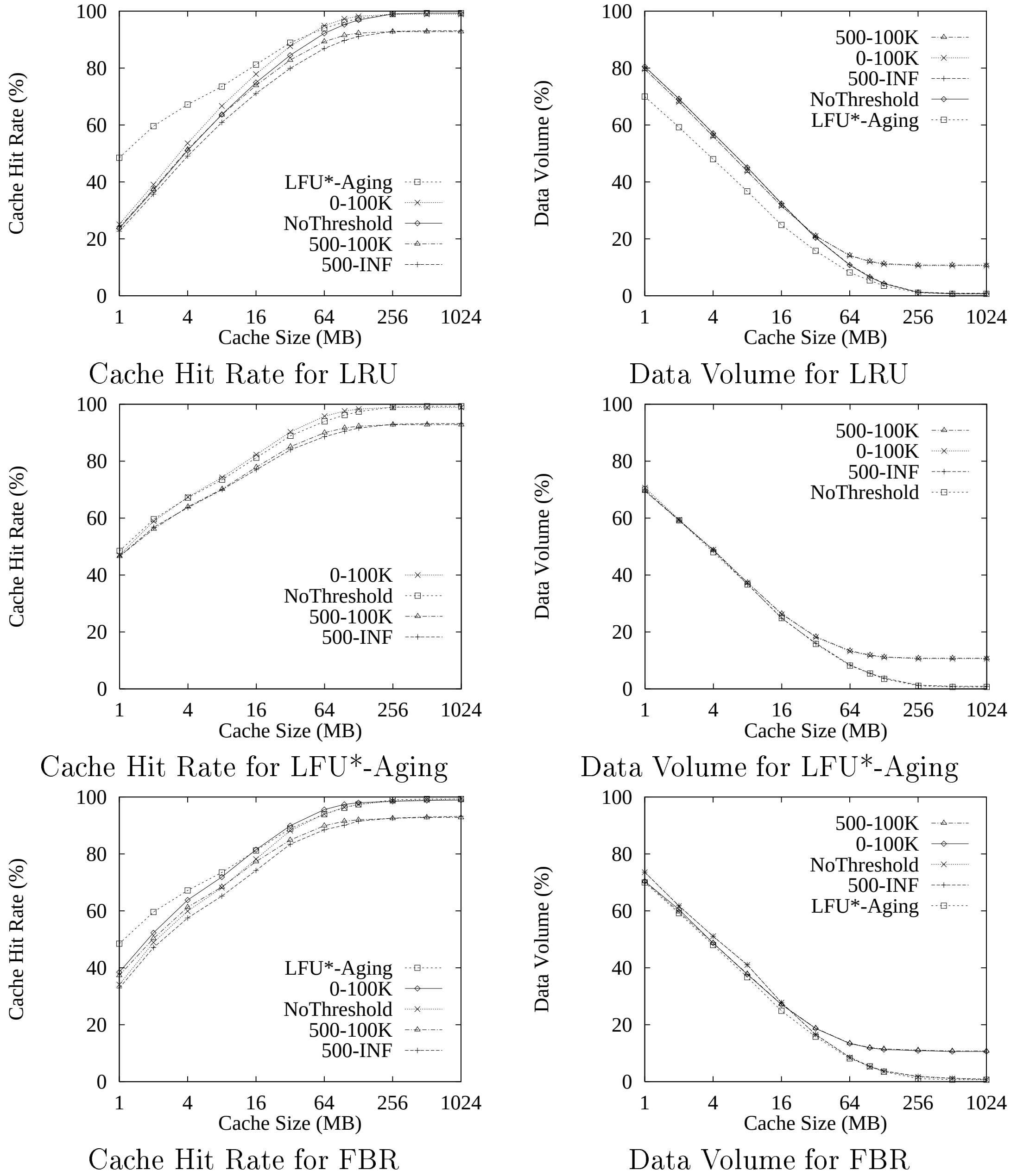


Figure 6.31: Comparison of Document Size Threshold Policies (ClarkNet Data)

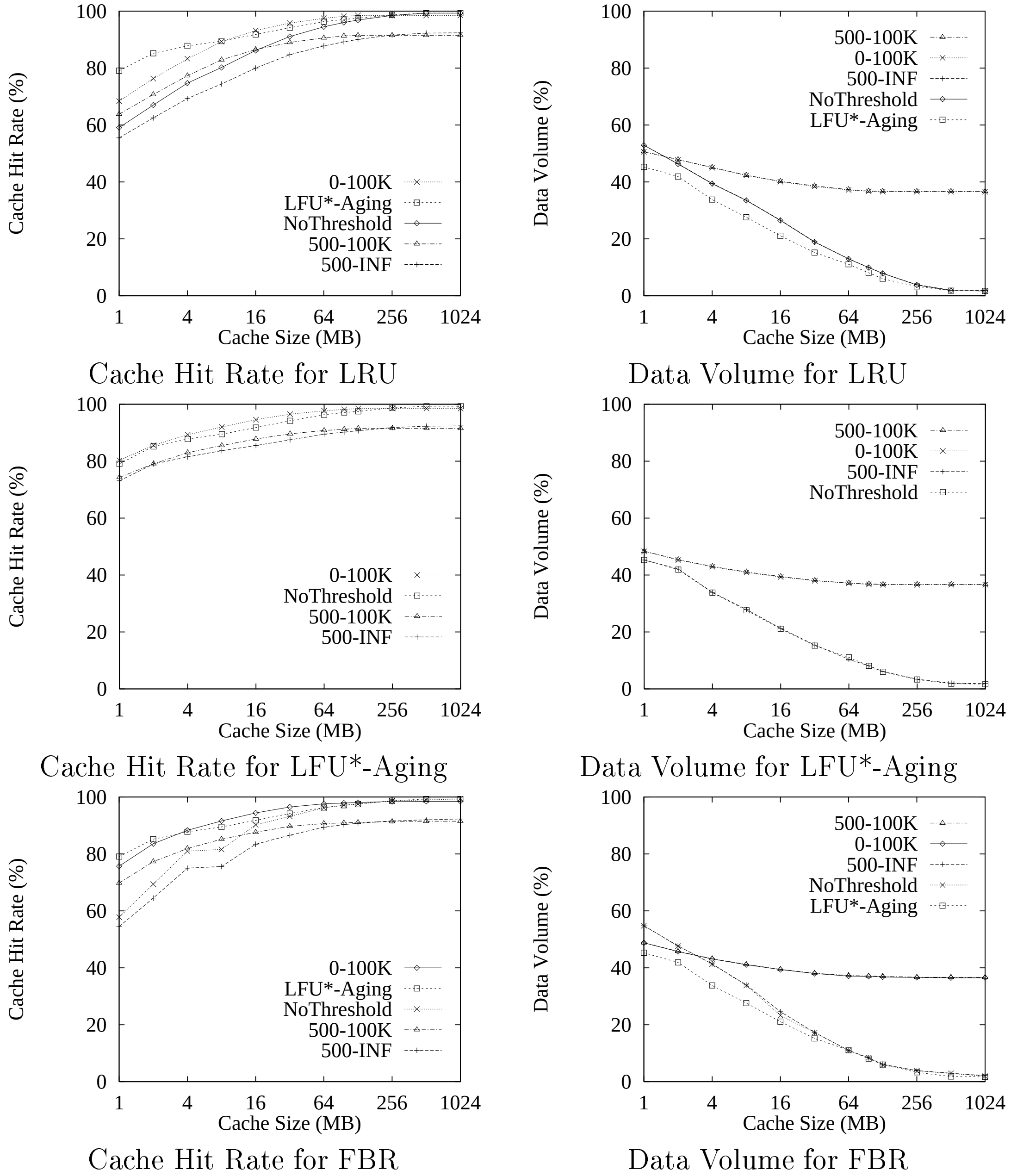


Figure 6.32: Comparison of Document Size Threshold Policies (NCSA Data)

Table 6.2: Partition Values Used (All Data Set)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
HTML (%)	39	47	56	31	20	51
Images (%)	50	50	37	64	78	48
Other (%)	11	3	7	5	2	1

6.5 Effects of a Partitioning Policy

The final group of experiments attempts to determine if a partitioning approach to file caching could improve the performance of the file cache. Partitioning is tested with all 12 cache sizes under study, as well as the LFU*-Aging, LRU, and FBR replacement policies.

The control case is when no partitioning occurs (i.e., the cache space can be shared by all file types). The results for this approach are the same as those discussed in Section 6.3.

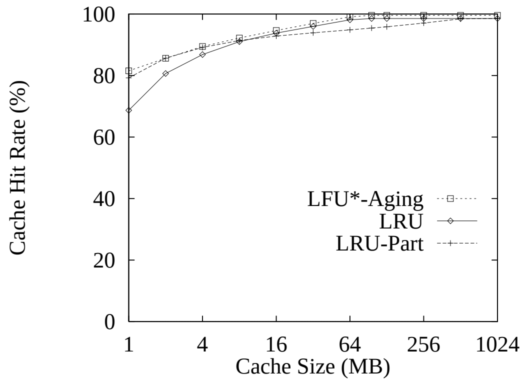
The second case under study uses three partitions: one for HTML documents only, one for Image documents only, and the third for any other document types that are requested. Initially, a number of fixed partition sizes were studied (e.g., HTML received 40% of the available cache space, Images 40% and other documents the remaining 20%). However, none of the tested cases were able to improve the performance of the file cache (when compared to the results for the LFU*-Aging policy in Section 6.3). A second attempt used partition sizes that are data set dependent. More specifically, the partition sizes were set to correspond to the percentage of references at each site to HTML and Image documents. Table 6.2 gives the settings that were used. Note that the "Other" category refers to all documents that are not HTML or Image files.

Figures 6.33 to 6.38 show the results for the partitioning policy, which used the parameters listed in Table 6.2. In general, partitioning seems to improve the hit rate slightly for small caches using either the LRU or the FBR replacement policies. This is likely occurring because the partitions are protecting the small HTML and Image files from being replaced by other types of documents. Partitioning appears to have little positive impact on the LFU*-Aging cache.

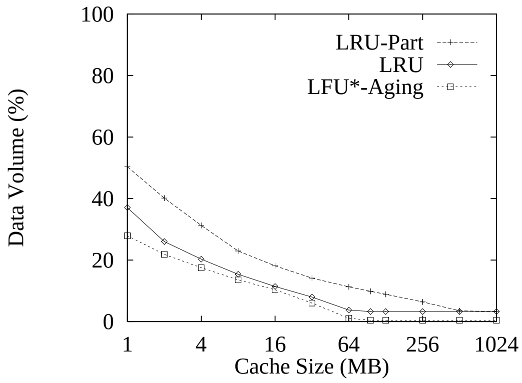
The partitioning policy is almost always worse in terms of data volume. The likely explanation for this is the "other" file types are responsible for a significant portion of the data volume. Since the "other" partition is quite small for all data sets, the partitioning policy (for the LRU and LFU*-Aging replacement policies) is not able to minimize the data volume. Across a number of the data sets (Waterloo, shown in Figure 6.33(f), Calgary, shown in Figure 6.34(f), NASA, shown in Figure 6.36(f), and NCSA, shown in Figure 6.38(f)), the FBR policy had an unexpected outcome for the cache performance in terms of data volume. In these graphs, the partitioning policy curves are not monotonically decreasing, as one would expect to see. Instead, these curves have more of a "wave" shape. One possible explanation for this is that when the cache is small, the FBR is able to minimize the data volume for the HTML and Image files. As the cache size increases, so does the "other" partition. This means that larger and larger (even if they are infrequently requested) files can be added to this partition, which could negatively impact the performance of the "other" partition. This results in an increase in data volume, rather than a decrease, as the cache size increases. This phenomenon is not seen with the LRU policy, and is not as pronounced with the LFU*-Aging policy, because of the different replacement choices made.

A third case, involving seven partitions (HTML, Image, Sound, Video, Dynamic, Formatted and other document types), was also tested. This scenario also performed worse than the LFU*-Aging policy with no threshold or partitioning policy.

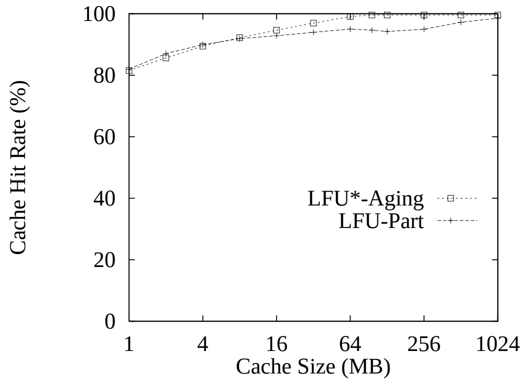
Since using a partitioning policy does not result in consistent performance improvements, and because it is difficult to determine the partition sizes to use, there does not appear to be an advantage in using this approach.



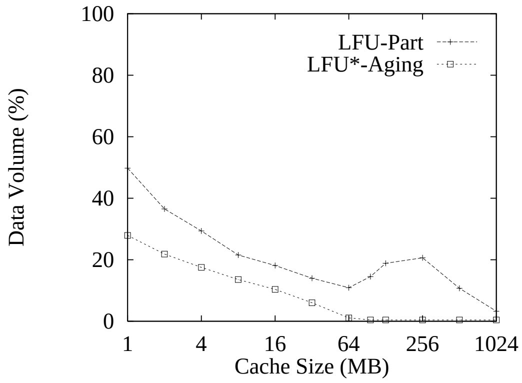
Cache Hit Rate for LRU



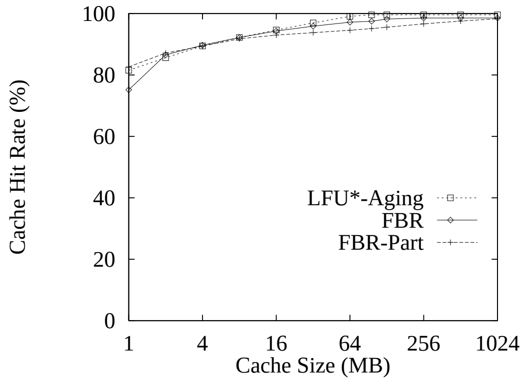
Data Volume for LRU



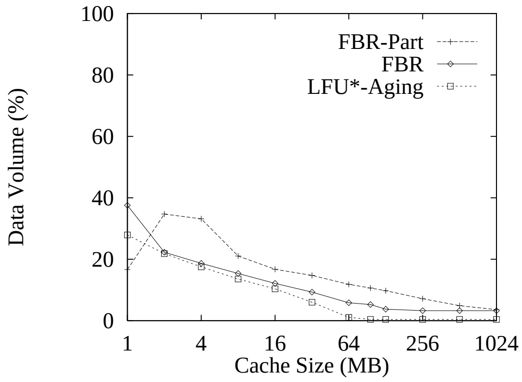
Cache Hit Rate for LFU*-Aging



Data Volume for LFU*-Aging

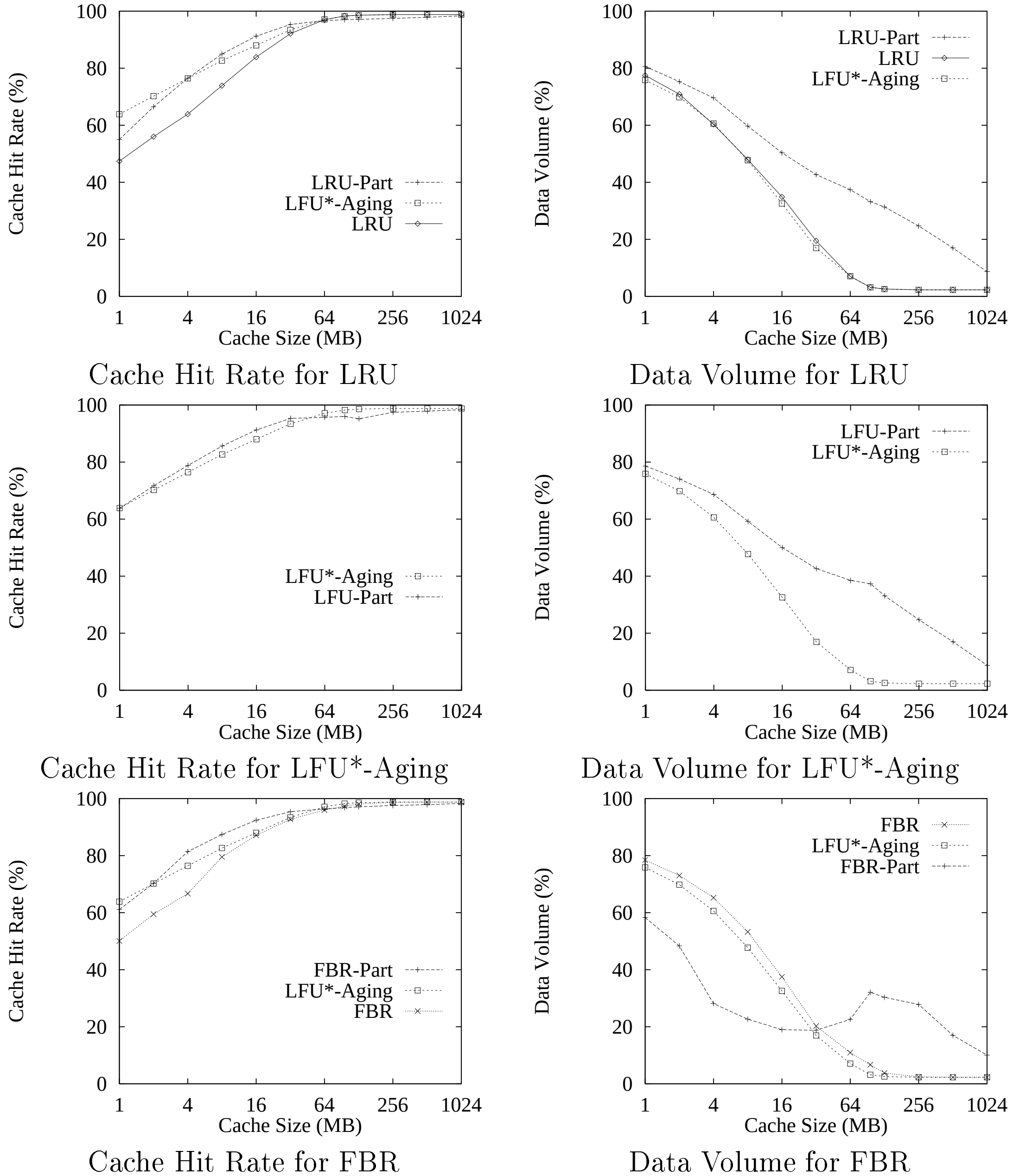


Cache Hit Rate for FBR



Data Volume for FBR

Figure 6.33: Comparison of Partitioning Policies (Waterloo Data)



Cache Hit Rate for LRU

Data Volume for LRU

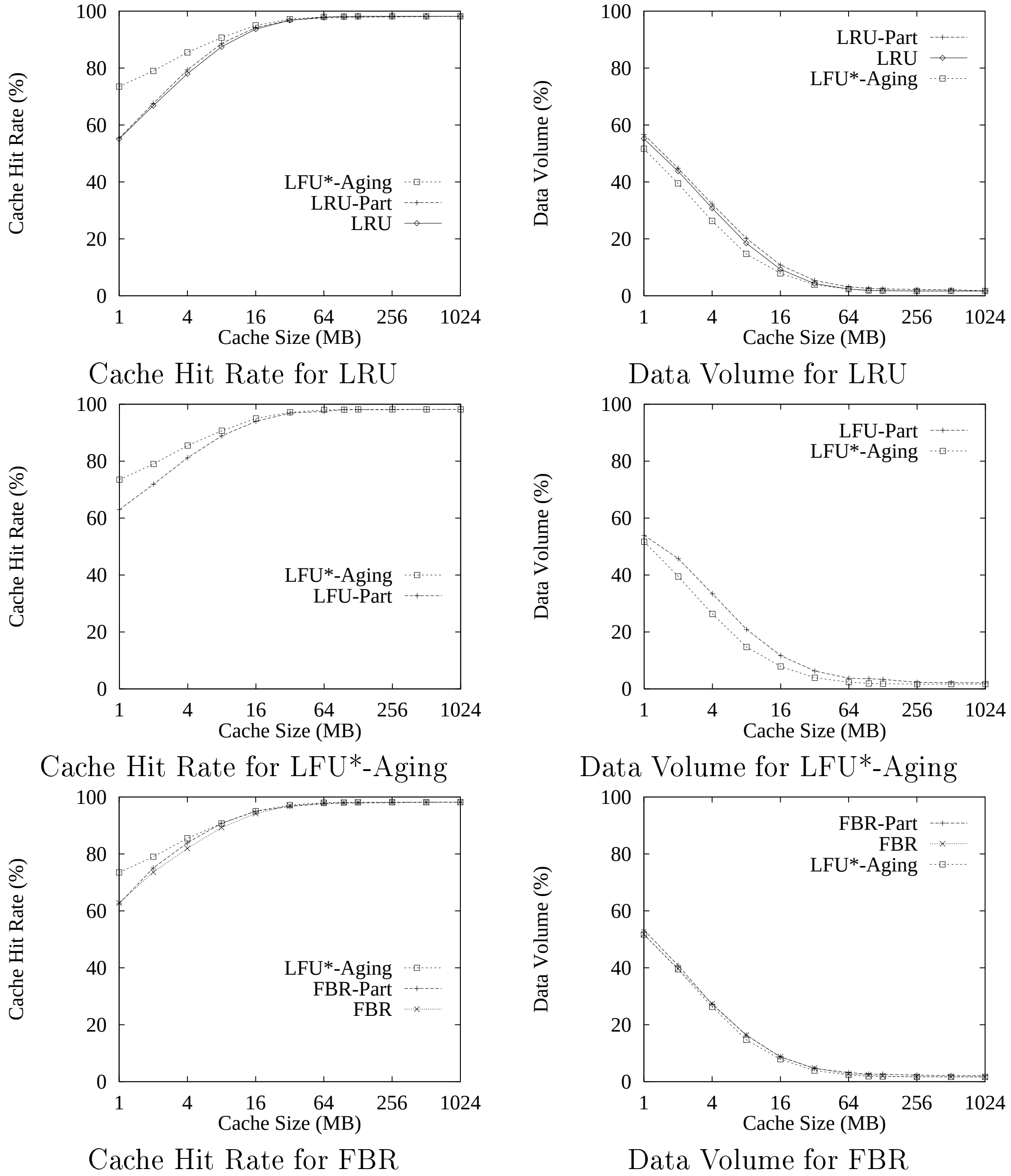
Cache Hit Rate for LFU*-Aging

Data Volume for LFU*-Aging

Cache Hit Rate for FBR

Data Volume for FBR

Figure 6.34: Comparison of Partitioning Policies (Calgary Data)



Cache Hit Rate for LRU

Data Volume for LRU

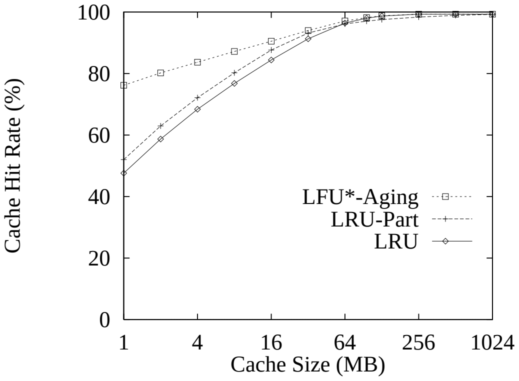
Cache Hit Rate for LFU*-Aging

Data Volume for LFU*-Aging

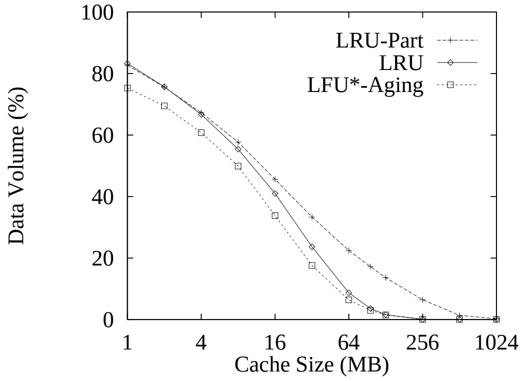
Cache Hit Rate for FBR

Data Volume for FBR

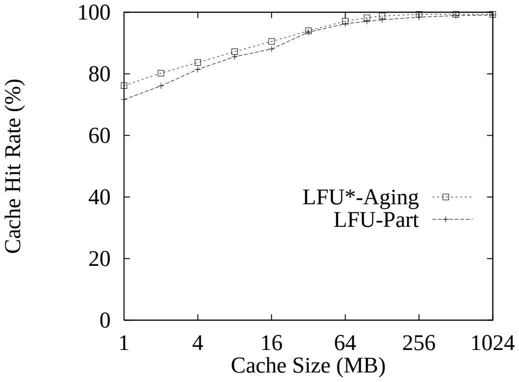
Figure 6.35: Comparison of Partitioning Policies (Saskatchewan Data)



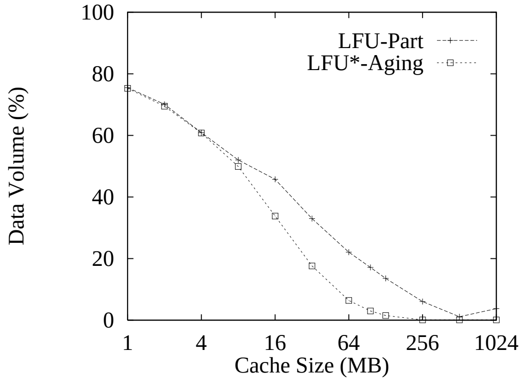
Cache Hit Rate for LRU



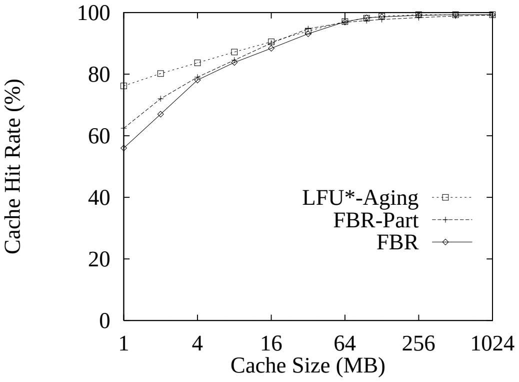
Data Volume for LRU



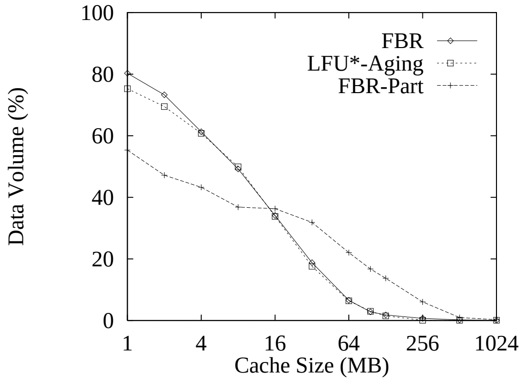
Cache Hit Rate for LFU*-Aging



Data Volume for LFU*-Aging

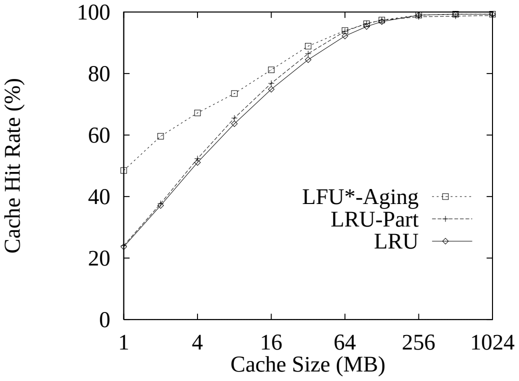


Cache Hit Rate for FBR

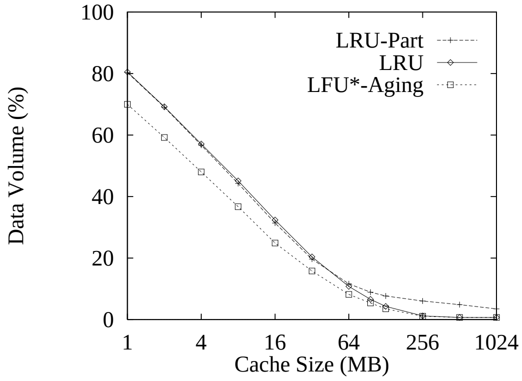


Data Volume for FBR

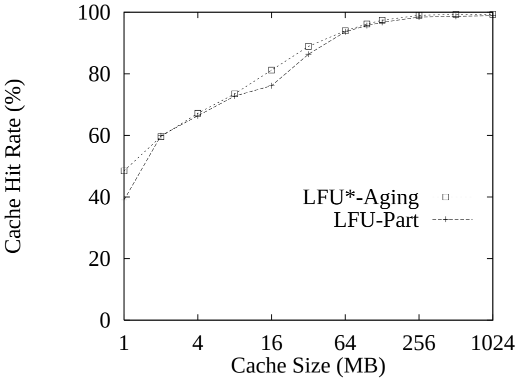
Figure 6.36: Comparison of Partitioning Policies (NASA Data)



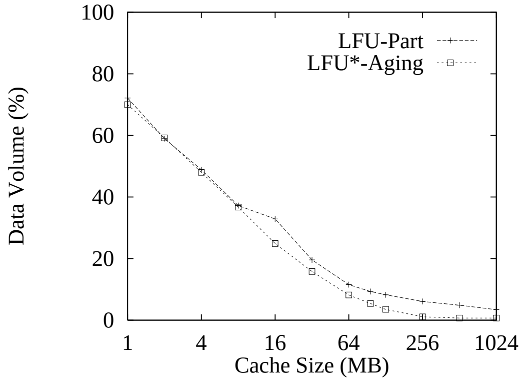
Cache Hit Rate for LRU



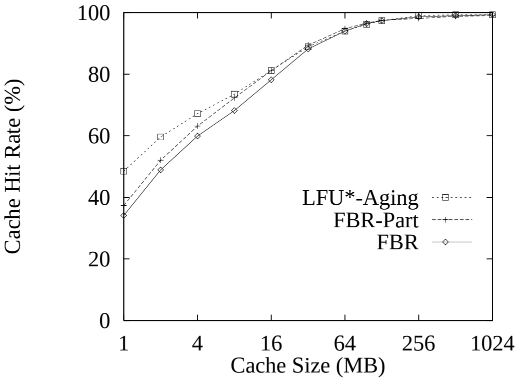
Data Volume for LRU



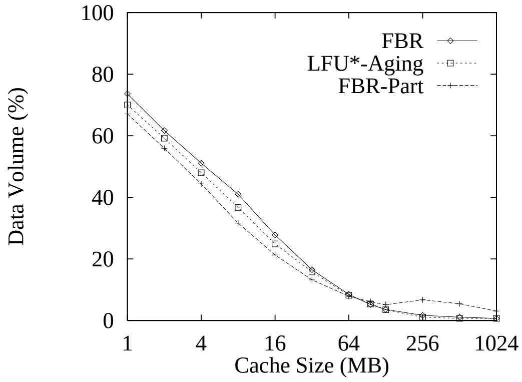
Cache Hit Rate for LFU*-Aging



Data Volume for LFU*-Aging

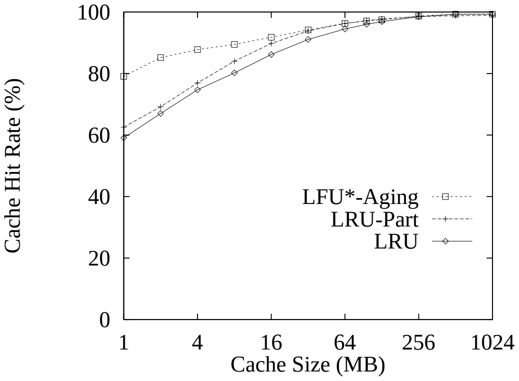


Cache Hit Rate for FBR

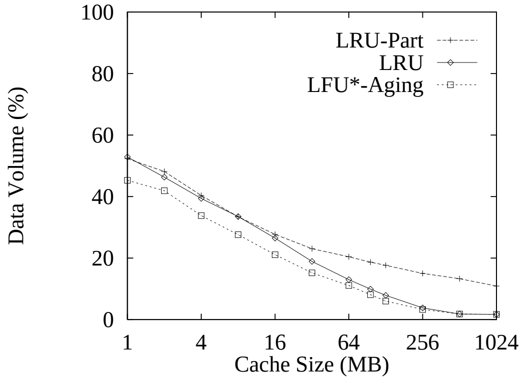


Data Volume for FBR

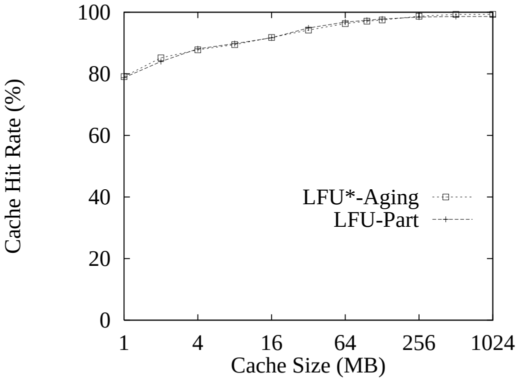
Figure 6.37: Comparison of Partitioning Policies (ClarkNet Data)



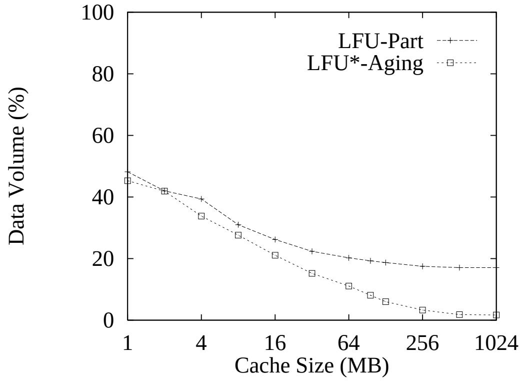
Cache Hit Rate for LRU



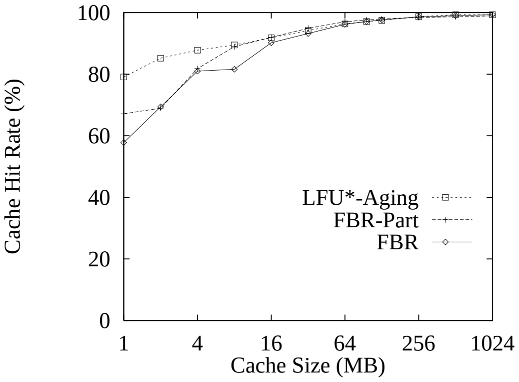
Data Volume for LRU



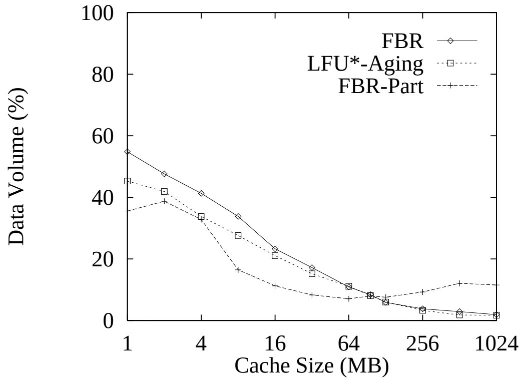
Cache Hit Rate for LFU*-Aging



Data Volume for LFU*-Aging



Cache Hit Rate for FBR



Data Volume for FBR

Figure 6.38: Comparison of Partitioning Policies (NCSA Data)

6.6 Summary of Results

This simulation study found that for a Web server, a file cache using the LFU*-Aging replacement policy performs better (i.e., has a higher cache hit rate and a lower data volume) than file caches using LFU, LRU or FBR replacement policies. This study also found that a threshold policy that places an upper limit on document size can improve the hit rate of a file cache, but increases the data volume. Finally, this study found that in some cases it is possible to improve file cache performance with the use of a partitioning policy, but that it is difficult to determine the partition sizes to use. If the partitions are not set correctly, the performance of the file cache is negatively impacted.

6.7 Summary

This chapter has presented a performance study of a file cache for a Web server. A trace-driven simulation was used to study the effects of the replacement policy used by the file cache, as well as the effects of a threshold policy (which restricts which documents can be added to the cache based on the size of the document) and a partitioning policy (which maintains separate partitions within the cache for files of a particular type).

The simulation results show that the choice of replacement policy has the greatest impact on the performance of the file cache. Across the six data sets studied, the LFU*-Aging policy achieved the highest cache hit rate and the lowest data volume. The results also show that a threshold policy that implements an upper limit on file size can have some positive effects on file cache performance, while the partitioning policy does not appear to be as useful.

The results from this study show that a properly designed file cache can reduce data volume by at least an order of magnitude. The decision that has the largest performance impact is the choice of replacement policy. Recency-based replacement policies (e.g., LRU), which have often been the replacement policies of choice in most computer systems performance studies, are not the best choice for a file cache on a Web server. The success of simple frequency-based policies such as LFU-Aging and LFU*-Aging suggest that more complex policies such as FBR should not be chosen. While some of the ideas considered (thresholding and partitioning) can improve file cache performance, this improvement is often minimal. Furthermore, these enhancements are difficult to implement, and can have a negative impact on performance.

Chapter 7

Summary and Conclusions

This thesis has presented both a workload characterization study of Internet Web servers and a performance evaluation of file caching for Internet Web servers. This chapter concludes the thesis with a summary of the work that has been done. This chapter also lists the contributions of this thesis, and suggests areas for future research.

7.1 Summary

This thesis is divided into two parts: a workload characterization study of Internet Web servers, and a performance evaluation of file caching for Internet Web servers. Vital to both of these parts was the acquisition of Web server access logs. These access logs contain information on all the requests that the Web server received (during the time frame that the log was collected), as well as information on the server's response to each of these requests. Access logs from six different Web servers were obtained: three from academic (i.e., university) environments, two from scientific research organizations, and one from a commercial Internet provider. These access logs represent three different orders of magnitude in server activity, and two different orders of magnitude in time duration.

The first part of this thesis presents a workload characterization study for Internet Web servers. Throughout this study, emphasis was placed on finding invariants: observations that applied across all six of the data sets under study. In total, ten invariants were identified. These invariants are summarized in Table 7.1.

Table 7.1: Summary of Invariants Found in Web Server Workloads

Invariant	Name	Description
1	Success Rate	Approximately 88% of requests to the server result in the successful return of the document
2	File Types	HTML and image files account for 90-100% of requests
3	Mean Transfer Size	Mean transfer size $\leq$ 21 kilobytes
4	Distinct Requests	Among all server requests, less than 3% of the requests are for distinct files
5	One Time Referencing	Approximately one-third of the files and bytes referenced in the log are referenced only once in the log
6	Size Distribution	The file size distribution is heavy-tailed; the tail of the distribution is Pareto with $0.40 < \alpha < 0.63$
7	Concentration of References	10% of the files accessed account for 90% of server requests and 90% of the bytes transferred
8	Inter-Reference Times	File inter-reference times are exponentially distributed and independent
9	Remote Requests	Remote sites account for $\geq$ 70% of the accesses to the server, and $\geq$ 60% of the bytes transferred
10	Wide Area Usage	Web servers are accessed by 1000's of domains, with 10% of the domains accounting for $\geq$ 75% of usage

Once the workload characterization study was completed, the identified invariants were used to suggest performance enhancements for Web servers. Based on a number of the invariants, file caching appears to have the potential for greatly improving Web server performance. First, there are a large number of references to a Web server for a small number of documents (Invariant 4). Second, within this set of documents there is a concentration of references (Invariant 7), which means that only a small subset of these documents need to be cached while still satisfying a high percentage of all requests. Since the average size of the files is quite small

(Invariant 3), only a small amount of cache space would be needed to achieve this level of performance. The invariants also provide information on things that might affect the performance of a file cache. First, approximately one-third of the files and bytes accessed were accessed only once during the duration of the logs (Invariant 5). Obviously, caching any of these files would degrade the performance of the file cache. Second, the file size distribution was found to be heavy-tailed (Invariant 6). This means that while most files are quite small, there are also some very large files. This suggests that restricting which documents are cached based on document size could affect the performance of the cache. Finally, since most of the requests are for either HTML or Image documents (Invariant 2), a partitioning approach to file caching may improve performance, by protecting HTML and Image documents from replacement by (less popular) documents of other types.

The second part of this thesis presents a performance study of file caching strategies for Internet Web servers. This study addresses the issues raised in the discussion of the performance implications of the invariants. This study focuses on the performance impacts of the replacement policy, threshold policy, and partitioning policy that are chosen. The results from this study show that a properly managed reasonably sized (e.g., 32 MB) file cache can reduce the data volume transfered by the server by an order of magnitude. The cache replacement policy has the greatest impact on performance results. The study found that the LFU policy (with aging) performs better than the LRU policy (LRU has traditionally been the choice for replacement policy in many computer systems performance studies). Making only slight modifications to the LFU policy can result in further performance improvements. This study also found that a threshold policy that implements an upper limit on file size can have some positive effects on file cache performance, while the partitioning policy does not appear to be as useful.

7.2 Relevance and Contributions

Due to the tremendous growth in both the size of the World-Wide Web and the number of Web users, recent research efforts have focussed on improving the performance of the Web. Both the workload characterization and the performance evaluation performed in this thesis contribute to this effort.

The workload characterization study presented in the first part of this thesis is important, as an understanding of WWW workloads is required in order to improve Web performance. The workload characterization study given in this thesis differs from other studies that have been done in that the work in this thesis attempts to find characteristics that describes the workloads of many different Web servers, rather than just the characteristics that are specific to one server.

One of the goals of World-Wide Web research is to ensure that the Web is scalable, so that the Web can continue to grow. The use of file caching is one method of achieving this scalability. File caching can be used to reduce server loads, network loads, and response times. The performance evaluation in this thesis provides information on the effectiveness of a file cache for a Web server, and examines the effects of different cache configurations on the overall performance of the cache.

7.3 Future Research

Much research on improvements to World-Wide Web performance remains to be done. Some of this research can follow directly from the work done in this thesis.

First, a workload characterization study of Web servers over time is needed. Such information is needed in order to predict the workload of Web servers in the future. This information is needed by developers, who must design the servers up to several years in advance, and by system administrators, who must predict what resources will be needed in order to provide a certain quality of service to the users. In the future, multimedia objects (such as sound and video) will likely become more popular; as well, new developments may also change the characteristics of the workload (e.g., development of network-wide document caches [22]). Continued workload characterization studies are needed to monitor these changes.

Second, workload models for Web servers are needed, to drive simulations of Web servers, or to compare the performance of different Web servers under realistic conditions. These simulations may be used to identify performance bottlenecks in existing servers, or to aid in the development of new servers.

The performance study of file caching in this thesis looked at a number of different replacement policies, as well as a number of other policies that seemed promising. Further investigations in this area could evaluate other replacement policies or different methods of restricting which documents are added to the cache (i.e., instead of a threshold policy based on size, test an alternative scheme). As well, this study observed only the overall cache hit rate and data volume; it did not monitor the performance impact on specific types of traffic (e.g., cache hit rate for video documents). Furthermore, if Web access logs were modified to contain information on transfer times, the effects of file caching on reducing latency could be measured. This modification is very important to researchers, as latency is the metric in which Web users are most interested. Finally, the results of this study need to be confirmed through the actual implementation of a file cache on a Web server.

While this thesis has looked specifically at the performance of a file cache on a Web server, this is not the only possible location for a file cache in a Web environment. Furthermore, file caches can be used in multiple locations at the same time. Studying the effects of multi-level file caching is another topic that is left for future research.

Finally, not all of the information available in the access logs under study was used in this thesis. For example, the effects of geographically distributed file caches on the configuration of the file cache at the server could be examined. To enable other researchers to examine this or other issues of interest, a number of the access logs used in this thesis have been made available to other researchers, via the Internet Traffic Archive (ITA)(`http://town.hall.org/Archives/pub/ITA/`).[1] The C programs used in the workload characterization study and the file cache simulations are available via URL `http://www.cs.usask.ca/projects/discus/`.

[1]At the time of writing, permission had been granted to release four of the six access logs (Calgary, ClarkNet, NASA, and Saskatchewan) to the ITA.

References

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, "Caching Proxies: Limitations and Potentials", *Electronic Proceedings of the Forth World Wide Web Conference '95: The Web Revolution*, Boston, MA, December 11-14, 1995.
Available at URL: `http://www.w3.org/pub/Conferences/WWW4/Papers/155/`

[2] M. Andreessen, "NCSA Mosaic Technical Summary", National Center for Supercomputing Applications, 1993.

[3] M. Arlitt and C. Williamson, "A Synthetic Workload Model for Internet Mosaic Traffic", *Proceedings of the 1995 Summer Computer Simulation Conference* (SCSC'95), Ottawa, ON, pp. 852-857, July 1995.

[4] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants", *Proceedings of the 1996 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Philadelphia, PA, pp. 126-137, May 23-26, 1996.

[5] S. Balbach, Personal communication (e-mail), ClarkNet, September 1995.

[6] T. Berners-Lee, L. Masinter and M. McCahill, "Uniform Resource Locators", RFC 1738, December 1994.
Available at URL: `ftp://ftp.internic.net/rfc/rfc1738.txt`

[7] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen and A. Secret, "The World-Wide Web", *Communications of the ACM*, 37(8), pp. 76-82, August 1993.

[8] R. Fielding, H. Frystyk and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", work in progress, January 19, 1996.
Available at URL: `ftp://ftp.internic.net/internet-drafts/draft-ietf-http-v11-spec-01.txt`

[9] A. Bestavros, "Demand-Based Document Dissemination for the World Wide Web", Technical Report BU-CS-95-003, Boston University Computer Science Department, 1995.
Available at URL: `ftp://cs-ftp.bu.edu/techreports/95-003-web-server-dissemination.ps.Z`

[10] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya and S. Mirdad, "Application-Level Document Caching in the Internet", *Proceedings of the Second International Workshop on Services in Distributed and Networked Environments (SDNE '95)*, Whistler, BC, June 1995.
Available at URL: `ftp://cs-ftp.bu.edu/techreports/95-002-web-client-caching.ps.Z`

[11] C. Bowman, P. Danzig, U. Manber and M. Schwartz, "Scalable Internet Resource Discovery: Research Problems and Approaches", *Communications of the ACM*, 37(8), pp. 98-107, August 1994.

[12] H. Braun and K. Claffy, "Web Traffic Characterization: An Assessment of the Impact of Caching Documents from NCSA's Web Server", *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, IL, October 1994.
Available at URL: `http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/claffy/main.html`

[13] H. Braun and K. Claffy, "Post-NSFNET Statistics Collection", *Proceedings of INET '95*, pp. 577-586, June 1995.
Available at URL: `ftp://ftp.sdsc.edu/pub/sdsc/anr/papers/post-nsfnet-stats.ps.Z`

[14] R. Cáceres, P. Danzig, S. Jamin and D. Mitzel, "Characteristics of Wide-Area TCP/IP Conversations", *Proceedings of ACM SIGCOMM '91*, Zurich, Switzerland, pp. 101-112, September 1991.

[15] R. Cáceres, "Multiplexing Traffic at the Entrance to Wide-Area Networks", Ph. D. Thesis, Computer Science Division, University of California, Berkeley, December 1992.

[16] P. Cao, E. Felten, A. Karlin and K. Li, "A Study of Integrated Prefetching and Caching Strategies", *Proceedings of the 1995 SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Ottawa, ON, pp. 188-197, May 15-19, 1995.

[17] K. Claffy, "Internet Traffic Characterization", Ph.D. Thesis, Department of Computer Science, University of California, San Diego, 1994.
Available at URL: `ftp://ftp.sdsc.edu/pub/sdsc/anr/papers/kc-thesis-94.ps.Z`

[18] M. Crovella and A. Bestavros, "Explaining World Wide Web Traffic Self-Similarity", *Proceedings of the 1996 SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Philadelphia, PA, pp. 160-169, May 23-26, 1996.

[19] M. Crovella, Personal communication (e-mail), Department of Computer Science, University of Boston, May 1996.

[20] C. Cunha, A. Bestavros and M. Crovella, "Characteristics of WWW Client-Based Traces", Technical Report BU-CS-95-010, Boston University Computer Science Department, July 18, 1995.
Available at URL: `ftp://cs-ftp.bu.edu/techreports/95-010-www-client-traces.ps.Z`

[21] R. D'Angostino and M. Stephens, Editors, *Goodness-of-Fit Techniques*, Marcel Dekker Inc., New York, NY, 1986.

[22] P. Danzig, M. Schwartz and R. Hall, "A Case for Caching File Objects Inside Internetworks", *Proceedings of ACM SIGCOMM '93*, San Francisco, CA, pp. 239-248, September 1993.

[23] J. Dumoulin, Personal communication (e-mail), NASA Kennedy Space Center, September 1995.

[24] E. Fogel, Personal communication (e-mail), Department of Computing Services, University of Saskatchewan, September 1995.

[25] R. Fridman, Personal communication (e-mail), Department of Computer Science, University of Calgary, October 1995.

[26] K. Froese and R. Bunt, "The Effect of Client Caching on File Server Workloads", *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, Wailea, HI, January 1996.

[27] S. Glassman, "A Caching Relay for the World Wide Web", *First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.
Available at URL: `http://www.cern.ch/Papers/WWW94/steveg.ps`

[28] J. Gwertzman, "Autonomous Replication in Wide-Area Internetworks", B.A. Thesis, Center for Research in Computing Technology, Harvard University, Cambridge MA, April 1995.
Available at URL: `ftp://das-ftp.harvard.edu/techreports/tr-17-95.ps.gz`

[29] J. Hodge, Personal communication (e-mail), Department of Computer Science, University of Waterloo, September 1995.

[30] A. Hopmann, "HTTP Session Extension", Work In Progress, July 3, 1995. Available at URL: `http://www.ics.uci.edu/pub/ietf/draft-ieft-http-ses-ext-00.txt`

[31] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley & Sons, Inc., New York, NY, 1991.

[32] N. Johnson and S. Kotz, *Distributions in Statistics: Continuous Univariate Distributions - 1*, John Wiley & Sons, Inc., New York, NY, 1970.

[33] N. Johnson and S. Kotz, Editors-in-Chief, *Encyclopedia of Statistical Sciences, Volumes 6 and 9*, John Wiley & Sons, Inc., New York, NY, 1988.

[34] M. Kendall, "Natural Law in the Social Sciences", *Journal of the Royal Statistical Society*, Vol. 124, pp. 1-19, 1961.

[35] D. Knuth, *The Art of Computer Programming*, Addison-Wesley Publishing Company,Inc., Don Mills, ON, 1973.

[36] T. Kwan, R. McGrath, and D. Reed, "NCSA's World Wide Web Server: Design and Performance", *IEEE Computer*, Vol. 28, No. 11, pp. 68-74, November 1995.

[37] T. Kwan, R. McGrath, and D. Reed, "User Access Patterns to NCSA's Web Server", Technical Report UIUCDCS-R-95-1934, Department of Computer Science, University of Illinois, Urbana-Champaign, February 1995.

[38] D. LaLiberte and M. Shapiro, "The Path URN Specification", IETF Draft, work in progress.
Available at URL: `ftp://ftp.internic.net/internet-drafts/draft-ietf-uri-urn-path-01.txt`

[39] W. Leland, M. Taqqu, W. Willinger and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1, pp. 1-15, February 1994.

[40] A. Luotenen and K. Altis, "World-Wide Web Proxies", *First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[41] University of Kansas, Lynx software, version 2-4-2, January 12, 1996.
Available at URL: `ftp://ftp2.cc.ukans.edu/pub/lynx/`

[42] R. McGrath, "Performance of Several HTTP Daemons on an HP 735 Workstation", NCSA, April 25, 1995.
Available at URL: `http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html`

[43] R. McGrath, Personal communication (e-mail), National Center for Supercomputing Applications, January, 1996.

[44] R. McGrath, "Improving Performance of HTTP on Solaris Server Platforms", NCSA, November 17, 1995.
Available at URL: `http://www.ncsa.uiuc.edu/InformationServers/Performance/Solaris/rep3.html`

[45] R. McGrath, "Measuring the Performance of HTTP Daemons", NCSA, February 5, 1996.
Available at URL: `http://www.ncsa.uiuc.edu/InformationServers/Performance/Benchmarking/bench.html`

[46] R. McGrath, "Performance of Several Web Server Platforms", NCSA, January 22, 1996.
Available at URL: `http://www.ncsa.uiuc.edu/InformationServers/Performance/Platforms/report.html`

[47] National Center for Supercomputing Applications, Mosaic software, version 2.6, July 7, 1995.
Available at URL: `ftp://ftp.ncsa.uiuc.edu/Mosaic/`

[48] National Center for Supercomputing Applications, "KeepAlive Directive", 1996.
Available at URL:
`http://hoohoo.ncsa.uiuc.edu/docs/setup/httpd/KeepAlive.html`

[49] Netscape Communications Corporation, Netscape Navigator software, version 2.01.
Available at URL: `http://www.netscape.com/`

[50] Network Wizards, "Internet Domain Survey", July 1995.
Available at URL: `http://www.nw.com/zone/WWW/report.html`

[51] NSFNET Statistics, April 1995.
Available at URL: `ftp://nic.merit.edu/statistics/nsfnet/`

[52] K. Obraczka, P. Danzig and S. Li, "Internet Resource Discovery Services", *IEEE Computer*, Vol. 26, No. 9, pp. 8-22, September 1993.

[53] V. Padmanabhan and J. Mogul, "Improving HTTP Latency", *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, IL, October 1994.
Available at URL: `http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html`

[54] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections", *IEEE/ACM Transactions on Networking*, Vol. 2, No. 4, pp. 316-336, August 1994.

[55] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", *Proceedings of ACM SIGCOMM '94*, London, England, pp. 257-268, August 1994.

[56] V. Paxson, "Growth Trends in Wide Area TCP Connections", *IEEE Network*, Vol. 8, No. 4, pp. 8-17, July/August 1994.

[57] J. Peachey, "The Bradford-Zipf Distribution and Program Behaviour", M.Sc. Thesis, Department of Computational Science, University of Saskatchewan, 1982.

[58] J. Postel, "Transmission Control Protocol", RFC-793, September 1981. Available at URL: `ftp://ftp.internic.net/rfc/rfc793.txt`

[59] J. Robinson and M. Devarakonda, "Data Cache Management Using Frequency-Based Replacement", *Proceedings of the 1990 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Boulder, CO, pp. 134-142, May 1990.

[60] J. Sedayao, "Mosaic Will Kill My Network!", *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, Illinois, October 1994.
Available at URL: `http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/sedayao/mos_traf_paper.html`

[61] L. Slothouber, "A Model of Web Server Performance", StarNine Technologies, Inc., 1996.
Available at URL: `http://louvx.biap.com/white-papers/performance/overview.html`

[62] K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC-1737, December 1994.
Available at URL: `ftp://ftp.internic.net/rfc/rfc1737.txt`

[63] M. Spasojevic, M. Bowman and A. Spector, "Using a Wide-Area File System Within the World Wide Web", *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, IL, October 1994.
Available at URL: `http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/spasojevic/paper.html`

[64] S. Spero, "Analysis of HTTP Performance Problems", July 1994.
Available at URL: `http://sunsite.unc.edu/mdma-release/http-prob.html`

[65] S. Spero, "Progress on HTTP-NG", March 1995.
Available at URL: `http://www.w3.org/hypertext/WWW/Protocols/HTTP-NG/http-ng-status.html`

[66] Sun Microsystems Inc., "World-Wide Web Server Performance", 1996. Available at URL: `http://www.sun.com/sun-on-net/Sun.Internet.Solutions/performance/index.html`

[67] A. Tanenbaum, *Computer Networks*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1988.

[68] G. Trent and M. Sake, "WebSTONE: The First Generation in HTTP Server Benchmarking", White Paper, Silicon Graphics, February 1995. Available at URL: `http://www.sgi.com/Products/WebFORCE/WebStone/`

[69] S. Williams, M. Abrams, C. Standridge, G. Abdulla and E. Fox, "Removal Policies in Network Caches for World-Wide Web Documents", to appear in the *Proceedings of ACM SIGCOMM '96*, Stanford, CA, August 1996.

[70] D. Willick, D. Eager and R. Bunt, "Disk Cache Replacement Policies for Network Fileservers", *Proceedings of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, PA, pp. 2-11, May 25-28, 1993.

[71] W. Willinger, M. Taqqu, R. Sherman and D. Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level", *Proceedings of ACM SIGCOMM '95*, Cambridge, MA, pp. 100-113, 1995.

[72] World-Wide Web Frequently Asked Questions, April 11, 1996. Available at URL: `http://www.io.org/faq/www/index.html`

[73] G. Zipf, *Human Behaviour and the Principle of Least Effort*, Addison-Wesley Press, Inc., Cambridge, MA, 1949.

[74] W. Zhu, "Characterizing Wide Area Conversations on the Internet", M.Sc. Thesis, Department of Computational Science, University of Saskatchewan, July 1994. Available at URL: `ftp://ftp.cs.usask.ca/pub/discus/thesis-wanning.ps.Z`

Appendix A

Glossary

document: A single file available on a Web server. Throughout this thesis, the words "file" and "document" are assumed to be interchangeable.

HTML: Acronym for HyperText Markup Language. HTML is used to create hypertext documents, which allow for text and images to be included in a single file, as well as hyperlinks to other (related) documents.

HTTP: Acronym for HyperText Transfer Protocol. A network protocol that runs on top of TCP (Transmission Control Protocol), in order to provide reliable data transfer between Web clients and servers.

latency: The amount of time (i.e., delay) between the issuing of a request by a user, and the arrival of the (complete) response.

URL: Acronym for Universal Resource Locator. A naming scheme for a document, which specifies the Internet host and complete path name from which to retrieve the document.

Web browser: A program that runs on a Web client, to provide a user with a (graphical) interface to the World-Wide Web.

Web client: A computer that communicates with Web servers in order to retrieve documents that were requested by a user.

Web page: A collection of documents and/or hyperlinks. A page usually consists of a single HTML file, zero or more inline images, and zero or more links to other documents. Some pages may also include audio or video clips in addition to text and images.

Web proxy: A computer that forwards requests from Web clients to Web servers, and returns the Web server's response to the appropriate client. Web proxies are used to provide Web access, or to allow for the caching of Web documents.

Web server: A computer that stores Web documents, and sends copies of these documents to Web clients that request them.

World-Wide Web: A globally distributed application running on the Internet to allow users to access a continually increasing amount of information.