

P-3119



WEB CACHE REPLACEMENT USING REPLACEMENT ALGORITHM



A PROJECT REPORT

Submitted by

Vinoth.R

71206104060

Dhinesh.O.B

71206104011

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

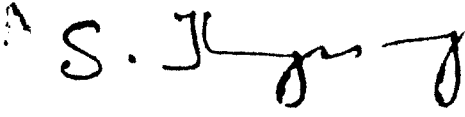
ANNA UNIVERSITY: CHENNAI- 600 025

APRIL 2010

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “**WEB CACHE REPLACEMENT USING RANDOMISED ALGORITHM**” is the bonafide work of “**VINOTH.R**” and “**DHINESH.O.B**” who carried out the project under my supervision.



SIGNATURE

Dr. S. Thangasamy

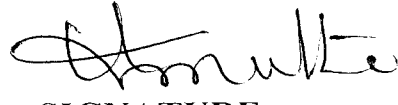
DEAN

Department of Computer Science

Kumaraguru College of Technology,

Chinnavedampatti Post,

Coimbatore – 641606



SIGNATURE

Mrs. Amutha Venkatesh

Asst. PROFESSOR

Department of Computer Science

Kumaraguru College of Technology,

Chinnavedampatti Post,

Coimbatore – 641606

The candidates with University Register Nos. **71206104060** and **71206104011** were examined by us in the project viva-voce examination held on **..16.04.2010**



INTERNAL EXAMINER



EXTERNAL EXAMINER

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made this possible and whose constant guidance and encouragement crowns all efforts with success.

We are extremely grateful to **Dr.S.Ramachandran**, Principal, Kumaraguru College of Technology for having given us this opportunity to embark on this project.

We express our sincere and heartfelt thanks to **Dr. S. Thangasamy**, Dean, Department of Computer Science and Engineering, for his kind guidance and support.

We would like to express our sincere thanks to our project coordinator **Mrs. P. Devaki**, for her valuable guidance during the course of the project.

We would also like to thank our class advisor **Mrs. R. Kalaiselvi**, for her constant support and guidance.

We would like to thank our guide **Mrs.Amudha Venkatesh**, without whose motivation and guidance we would not have been able to embark on a project of this magnitude. We express our sincere thanks for her valuable guidance, benevolent attitude and constant encouragement.

We reciprocate the kindness shown to us by the staff members of our college, people at home and our beloved friends who have contributed in the form of ideas, constructive criticism and encouragements for the successful

ABSTRACT

Web Cache Replacement Using Randomized Algorithm

Web caching is the emerging technology in web and in web caching if the client is requesting a page from server it will fetch from the server and will give response to the server. According to the locations where objects are cached, Web caching technology can be classified into three categories, i.e., client's browser caching, client-side proxy caching, and server-side proxy caching.

In client's browser caching, Web objects are cached in the client's local disk. If the user accesses the same object more than once in a short time, the browser can fetch the object directly from the local disk, eliminating the repeated network latency. However, users are likely to access many sites, each for a short period of time. Thus, the hit ratios of per-user caches tend to be low.

In client side proxy caching, objects are cached in the proxy near the clients to avoid repeated round-trip delays between the clients and the origin Web servers. To effectively utilize the limited capacity of the proxy cache, several cache replacement algorithms are proposed to maximize the delay savings obtained from cache hits. Such advanced caching algorithms differ from the conventional ones (e.g., LRU or LFU algorithms) in their consideration of size, fetching delay, reference rate, invalidation cost, and invalidation frequency of a Web object. Incorporating these parameters into their designs, these cache replacement algorithms show significant performance improvement over the conventional ones. In addition, cooperative caching architectures, proposed in enable the participating proxies to share their cache content with one another. Since each participating proxy can seek for a remote cache hit from other participating proxy's cache, the overall hit ratio can be further improved.

In server-side Web caching and content distribution network (CDN) are recently attracting an increasing amount of attention. It is noted that, as the Web traffic grows

Server-side Web caching, which distributes routes the user requests to the proper server-side proxies, it is able to release the Web server's load. Server side proxy caching will shorten the user perceived response time.

TABLE OF CONTENTS

| CHAPTER NO. | TITLE | PAGE NO. |
|--------------------|--------------------------------|-----------------|
| | ABSTRACT | iv |
| | LIST OF FIGURES | ix |
| 1. | INTRODUCTION | |
| | 1.1 Web Caching | 1 |
| | 1.2 Proxy Caching | 1 |
| | 1.2.1 Anonymous | 3 |
| | 1.2.2 Transparent | 4 |
| | 1.3 Web Prefetching | 4 |
| | 1.4 Client Side Proxy | 5 |
| | 1.5 Problem Description | 5 |
| | 1.6 Solution to the Problem | 6 |
| 2 | LITREATURE REVIEW | |
| | 2.1 Caching Infrastructure | 7 |
| | 2.1.1 Cern Httpd | 8 |
| | 2.1.2 Hensa | 8 |
| | 2.1.3 Harvest | 8 |
| | 2.1.4 Web Caching with Proxies | 9 |
| | 2.2 Cache Deployment Options | 9 |

| | | |
|-------|-----------------------------------|----|
| 2.2.2 | Strategic-Oriented Deployment | 9 |
| 2.2.3 | Consumer Oriented Deployment | 10 |
| 2.3 | Combining Caching and Prefetching | 10 |
| 2.4 | Cache Replacement Policies | 11 |
| 2.5 | Document Replacement Algorithms | 12 |
| 2.5.1 | Least-Recently Used (LRU) | 12 |
| 2.5.2 | Least-Frequently Used (LFU) | 13 |
| 2.5.3 | LRU-Threshold | 14 |
| 2.5.4 | Log (size) +LRU | 14 |

3

DESIGN AND ARCHITECTURE

| | | |
|-----|--|----|
| 3.1 | System Model | 15 |
| 3.2 | Creation of Proxy Server Configuration | 15 |
| 3.3 | Connection of Proxy Server to Internet | 17 |
| 3.4 | Replacement Algorithm Design | 17 |

4

TESTING

| | | |
|-----|-----------------------|----|
| 4.1 | Introduction | 19 |
| 4.2 | Source Code Testing | 19 |
| 4.3 | Specification Testing | 19 |
| 4.4 | Module Level Testing | 19 |
| 4.5 | Unit Testing | 20 |
| 4.6 | Integration Testing | 20 |
| 4.7 | Validation Testing | 20 |

| | | |
|--|----------------------|----|
| | 4.9 Security Testing | 21 |
|--|----------------------|----|

5 CONCLUSION AND FUTURE ENHANCEMENT

6 APPENDIX

| | | |
|--|-----------------|----|
| | 6.1 Sample Code | 23 |
|--|-----------------|----|

| | | |
|--|-----------------------------|----|
| | 6.1.1 Proxy Server Creation | 23 |
|--|-----------------------------|----|

| | | |
|--|---------------------------------|----|
| | 6.1.2 Proxy Connection Creation | 27 |
|--|---------------------------------|----|

| | | |
|--|---------------------------------------|----|
| | 6.1.3 Graphical User Interface Design | 31 |
|--|---------------------------------------|----|

| | | |
|--|----------------------------|----|
| | 6.2 Output and Screenshots | 34 |
|--|----------------------------|----|

| | | |
|----------|-------------------|-----------|
| 7 | REFERENCES | 39 |
|----------|-------------------|-----------|

LIST OF FIGURES

| CHAPTER NO. | TITLE | PAGE NO. |
|--------------------|---------------------------|-----------------|
| 1 | System Model | 16 |
| 2 | Prefetching the web cache | 35 |
| 3 | Specifying the Server | 36 |
| 4 | Log File Creation | 36 |
| 5 | Initiate the Server | 37 |
| 6 | Access Log | 38 |
| 7 | Error Log | 38 |

CHAPTER 1

INTRODUCTION

World Wide Web is the hypermedia-based system for browsing Internet sites. It is named the Web because it is made of many sites linked together; users can travel from one site to another by clicking on hyperlinks or "The World Wide Web is the universe of network-accessible information, an embodiment of human knowledge."

The recent increase in popularity of the World Wide Web has led to a considerable increase in the amount of traffic over the Internet. The Web has now become one of the primary bottlenecks to network performance. When a user requests objects, which are connected to a server on a slow network link, there is considerable latency, which can be noticed at the client end. Transferring the object over the network lead to increase in the level of traffic. Increase in traffic will reduce the bandwidth for competing requests and increase latencies for other users. In order to reduce access latencies, it is desirable to store copies of popular objects closer to the user. Consequently, Web Caching has become an increasingly important topic. After a significant amount of research to reduce the noticeable response time perceived by users, it is found that Web caching and Web Prefetching are two important techniques to this end.

1.1 Web Caching

Web caching is the emerging technology in web and in web caching if the client is requesting a page from server it will fetch from the server and will give response to the server. According to the locations where objects are cached, Web caching technology can be classified into

three categories, i.e., client's browser caching, client-side proxy caching, and server-side proxy caching.

In client's browser caching, Web objects are cached in the client's local disk. If the user accesses the same object more than once in a short time, the browser can fetch the object directly from the local disk, eliminating the repeated network latency. However, users are likely to access many sites, each for a short period of time. Thus, the hit ratios of per-user caches tend to be low.

In client side proxy caching, objects are cached in the proxy near the clients to avoid repeated round-trip delays between the clients and the origin Web servers. To effectively utilize the limited capacity of the proxy cache, several cache replacement algorithms are proposed to maximize the delay savings obtained from cache hits. Such advanced caching algorithms differ from the conventional ones (e.g., LRU or LFU algorithms) in their consideration of size, fetching delay, reference rate, invalidation cost, and invalidation frequency of a Web object. Incorporating these parameters into their designs, these cache replacement algorithms show significant performance improvement over the conventional ones. In addition, cooperative caching architectures, proposed in enable the participating proxies to share their cache content with one another. Since each participating proxy can seek for a remote cache hit from other participating proxy's cache, the overall hit ratio can be further improved.

In server-side Web caching and content distribution network (CDN) are recently attracting an increasing amount of attention. It is noted that, as the Web traffic grows exponentially, overloaded Web servers become the sources of the prolonged response

time. Server-side Web caching, which distributes routes the user requests to the proper server-side proxies, it is able to release the Web server's load. Server side proxy caching will shorten the user perceived response time.

1.2 Proxy Caching

Caching can be implemented at various points in the network. The best method among this is to have a cache in the Web server itself. Further, it is increasingly common for a university or corporation to implement specialized servers in the network called Caching Proxies. Such proxies' act as agents on behalf of the client in order to locate a cached copy of a object if possible. There are different types of proxy server based on FTP, HTTP, and SMTP and so on. They are FTP Proxy Server which relays and caches FTP Traffic. HTTP Proxy Server which has one way request to retrieve Web Pages and Socks Proxy Server is the newer protocol to allow relaying of far more different types of data, whether TCP or UDP. NAT Proxy Server which works differently from other servers, it allows the redirection of all packets without a program having to support a Proxy Server. SSL Proxy Server which is an extension to the HTTP Proxy Server which allows relaying of TCP data similar to a Socks Proxy Server.

Furthermore, a Proxy Server can be split into another two Categories:

- Anonymous
- Transparent.

1.2.1 Anonymous

An Anonymous Proxy Server blocks the remote Computer from knowing the identity of the Computer using the Proxy Server to make requests. Anonymous Proxy Servers can further be broken down into two more categories, Elite and Disguised. An Elite Proxy Server is not identifiable to the remote computer as a Proxy in any way.

1.2.2 Transparent

A Transparent Proxy Server tells the remote Computer the IP Address of the Computer. This provides no privacy. A Disguised Proxy Server gives the remote computer enough information to let it know that it is a Proxy, however it still does not give away the IP of the Computer it is relaying information for.

1.3 Web Prefetching

In Web prefetching scheme the proxy itself will give the response to the clients if the web page requested is present in the proxy itself. Several algorithms based on Markov models and Web mining techniques are proposed to derive prefetching rules from the server's access log.

Prefetching rule is an implication of the form $o_1, o_2, o_3, o_4, \dots, o_i \rightarrow o_{i+1}$ which mean that if $o_1, o_2, o_3, \dots, o_i$ have been referenced in a client's precedent requests, the object o_{i+1} will also be referenced in the client's subsequent requests with confidence c in prefetching rule. The confidence c of the prefetching rule is the conditional probability of $P(o_{i+1} | o_1, o_2, o_3, \dots, o_i)$ where

$P(o_1, o_2, \dots, o_i)$ is the probability that the sequence o_1, o_2, \dots, o_i is contained in an access sequence an object referenced in user access sequence. the object which was referenced is called “implied object” only if the prefetching rule $o_1, o_2, o_3, o_4, \dots, o_i, o_{i+1}$ is triggered by some client A and the client A have already referenced the object o_1, o_2, \dots, o_i in the precedent request. Otherwise the object o_{i+1} is called “non-implied object”.

1.4 Client Side Proxy

In client side proxy caching, caches are kept close to clients and this will reduce overall backbone traffic considerably. Client side proxy caching can improve user perceptions about network performance in two ways. First, when serving clients locally, caches hide wide area network latencies. On a local cache miss, the original content provider will serve client requests. Second, temporary unavailability of the network can be hidden from users, thus making the network appear to be more reliable.

1.5 Problem Description

The proxy server should be properly designed. If the proxy server provided rich information, a Web server may deliberately send all possible prefetching hints with various levels of confidences to the proxy. Without any control, a proxy will prefetch every implied object into its cache, despite that the confidences of some prefetching rules may be low. In this case, a significant portion of the cache content will be replaced because a proxy may concurrently serve a large amount of client requests and each of these requests may trigger certain

prefetching rules. As a result, the state of the cache content will become unstable and the cache-hit ratio will drop sharply. On the contrary, if the prefetching control is over strict, a proxy will tend to discard some beneficial hints provided by the Web server, thus whittling down the advantage of Web prefetching.

Another problem in Cache replacement algorithms data structure were used which requires priority queue for implementation and Data structure needs to be constantly updated even when there is no eviction and we cannot view the data present in cache since the data are stored using heap structure and they can be connected to only certain number of clients.

1.6 Solution to the Problem

An innovative cache replacement algorithm (i.e.) randomized algorithm is proposed. Randomized algorithm combines the benefit of both utility, based schemes and RR schemes and it avoids the need for data structures. The utility function assigns to each page a value based on recentness of use and frequency of use, size of page, cost of fetching and RR scheme would replace the least recently used web documents. These data will be evacuated only when it crosses the expiry time

To reduce the latency time and to increase the memory capacity and processing power the proxy server is designed in which data and images are stored separately. The proxy server can be connected to number of clients.

CHAPTER 2

LITRETURE REVIEW

2.1 Caching Infrastructures

There have been several projects aimed at building single software packages or complete infrastructures supporting caching in the Web.

2.1.1 Cern Httpd

This method is first widely available and it is still popular. This software gained widespread acceptance and served as a reference implementation to many caching proxies developed later on. Technically, however, it is inadequate, especially when large numbers of requests have to be handled. This is mostly due to its simple mapping of URLs to cache file system names, resulting in expensive lookup operations on every request. Moreover, a new, full-fledged UNIX process is created for every request, leading to huge memory and central processing unit (CPU) loads on high access rates. On one hand, this makes sense given the (high) probability that the original requesting client will try again sometime later. On the other hand, this continued loading binds resources urgently needed on the proxy, especially during the busiest access times of day. Finally, the conceptual flaw that caused this software to be of no use in cooperative caching schemes. CERN caches can only be configured to use one cache higher up in a thus strict hierarchical tree of caches.

Moreover, it could not gracefully deal with the failure of such parent cache.

2.1.2 Hensa

Hensa used Lagoon, a rather immature proxy, for a short time before switching to the Cern proxy software. As soon as many people started using the HENSA caching service, the limitations of the software that was present in CERN HTTPD became apparent. A commercial caching proxy from Netscape Communications was installed on a set of six machines sharing their load. This is achieved by making use of several domain name server (DNS) resource record entries for the one Internet name of the HENSA caching proxy that are returned round-robin to clients resolving the cache's name to IP addresses. The HENSA approach of a centrally administered caching service subsequently faced several barriers inherent to a nondistributed approach. Each of them had to be dealt with by migrating to new software or hardware to keep up with demand.

2.1.3 Harvest

The most sophisticated software caching in on the resulted from the Harvest. It sharply contrasts in concept from the HENSA approach by enabling administrators to span a tree of cooperating caches over wide distances. Harvest caches can be arranged in parent- or sibling relationships, with each cache contributing to the overall data set in a generally autonomous way. Whenever a caching proxy cannot serve a requested document from its local cache, it sends messages to its parents and siblings querying whether they hold the document. These messages are sent in parallel,

and the first peer returning a positive acknowledgment is asked to deliver the document in question.

2.1.4 Web Caching with Proxies

After a serious research in caching technologies it was found that Web caching with proxies is the efficient technology. Web proxy will be between the server and client and will serve for web page request.

2.2 Cache Deployment Options

There are three main cache deployment choices:

- Near the content consumer (consumer-oriented)
- Near the content provider (provider-oriented)
- At strategic points in the network, based on user access patterns and network topology.

2.2.1 Provider-Oriented Deployment

In provider oriented deployment method caches positioned near or maintained by the content provider, as in reverse proxy and push caching, improve access to a logical set of content. This type of cache deployment can be critical to delay-sensitive content such as audio or video. Positioning caches near or on behalf of the content provider allows the provider to improve the scalability and availability of content, but is obviously only useful for that specific provider. Any other content provider must do the same thing.

2.2.2 Strategic Point Oriented Deployment

In strategic point oriented deployment method the dynamic deployment of caches at network choke points, is a strategy embraced by the adaptive caching approach. Although it would seem to provide the most flexible type of cache coverage, it is still a work in progress and, to the best of the authors' knowledge, there have not been any performance studies demonstrating its benefits. The dynamic deployment technique also raises important questions about the administrative control of these caches, such as what impact network boundaries would have on cache mesh formation.

2.2.3 Consumer-Oriented Deployment

Positioning caches near the client, as in client side proxy caching has the advantage of leveraging one or more caches to a user community. If those users tend to access the same kind of content, this placement strategy improves response time by being able to serve requests locally. Thus this technology is widely used in recent trends and used in this project model

2.3 Combining Caching and Prefetching

Prefetching and caching are two known approaches for improving the performance of file systems. Although they have been studied extensively, most studies on prefetching have been conducted in the absence of caching or for a fixed caching strategy. After the invention of complication in individual prefetching technology (i.e.) prefetching file blocks into a cache can be harmful even if the blocks will be accessed in the near future. This is because a

cache block needs to be reserved for the block being prefetched at the time the Prefetch is initiated. The reservation of a cache block requires performing a cache block replacement earlier than it would otherwise have been done. Making the decision earlier may hurt performance because new and possibly better replacement opportunities open up as the program proceeds and hence combining caching and prefetching is essential.

2.4 Cache Replacement Policies

One of the key complications in implementing cache replacement policies for Web objects is that the objects to be cached are not necessarily of homogeneous size. For example, if two objects are accessed with equal frequency, the hit ratio is maximized when the replacement policy is biased towards the smaller object. This is because it is possible to store a larger number of objects of smaller size.

In addition to non homogeneous object sizes, there are several other special features of the Web, which need to be considered. First, the hit ratio may not be the best possible measure for evaluating the quality of a Web caching algorithm. For example, the transfer time cost for transferring a large object is more than that for a small object, though the relationship is typically not straightforward. It will depend on the distance of the object from the Web server. Furthermore, Web objects will typically have expiration times. So, when considering which objects to replace when a new object enters a Web cache.



We must consider not only the relative frequency, but also factors such as object sizes, transfer time savings, and expiration times. It may not always be favorable to insert an object into the cache, because it may lower the probability of a hit to the cache.

However, maximizing the cache hit ratio alone does not guarantee the best client response time in the Web environment. In addition to maximizing the cache hit ratio, a cache replacement algorithm for Web documents should also minimize the cost of cache misses, i.e., the delays caused by fetching documents not found in the cache. Clearly, the documents, which took a long time to fetch, should be preferentially retained in the cache. For example, consider a proxy cache at Northwestern University. The cache replacement algorithm at the proxy found two possible candidates for replacement. Both documents have the same size and are referenced with the same rate, but one document originates from the University of Chicago while the other is from Seoul National University. The cache replacement algorithm should select for replacement the document from the University of Chicago and retain the document from Seoul National University because upon a cache miss the former can be fetched much faster than the latter.

2.5 Existing Document Replacement Algorithms

2.5.1 Least-Recently-Used (LRU)

In the standard least recently used (LRU) caching algorithm for equal sized objects we maintain a list of the objects in the cache, which is ordered, based on the time of last access. In

particular, the most recently accessed object is at the top of the list, while the least recently accessed object is at the bottom. When a new object comes in and the cache is full, one object in the cache must be pruned in order to make room for the newly accessed object. The object chosen is the one which was least recently used. Clearly the LRU policy needs to be extended to handle objects of varying sizes.

LRU treats all documents equally, without considering the document size, type or network distance. It also ignores frequency information, thus an often-requested document will not be kept if it is not requested for a short period so LRU does not perform well in web caches. A scan, stream of multiple documents accessed only once, can force all the popular documents out of an LRU-based cache.

2.5.2 Least-Frequently-Used (LFU)

In least-frequently used method evicts the document, which is accessed least frequently. Least Frequently Used also has some disadvantages; the important problem with this method is cache pollution which means that a document that was formerly popular, but no longer is, will stay in the cache until new documents become more popular than the old one which was used. This can take a long time, and during this time, part of the cache is wasted. It assumes that probabilities are constant, but in practical it is not so. It also includes problem with implementation. Ideally, an implementation of the algorithm would keep a frequency counter for documents not in the cache as well as those present. On the scale of

performance of the algorithm diminishes. Even if only the counters for the documents in the cache are kept, counters have to be updated continuously, even when no document needs to be replaced, incurring considerable overhead. It is also necessary to keep the documents sorted by frequency to be able to make rapid decisions at replacement time. This method has no notion of document size or cost of retrieval.

2.5.3 LRU-Threshold

LRU-Threshold method is the same as LRU, except documents larger than a certain threshold size are never cached. this method also has some disadvantages, it causes cache pollution and it includes implementation problem. The counters present in the cache for documents need to be updated continually even if no document needs to be replaced.

2.5.4 Log(Size)+LRU

Log (size) +LRU method evicts the document who has the largest $\log(\text{size})$ and is the least recently used document among all documents with the same $\log(\text{size})$. $\log(\text{size})$ +LRU method include problem with implementation and the counters needs to be constantly updated even when there is no eviction.

CHAPTER 3

DESIGN AND ARCHITECTURE

Module 1: Creation of Proxy server configuration

Module 2: connection of proxy server with internet

Module 3: Replacement algorithm design

Module 4: web caching and prefetching

3.1 System Model

The proxy is located near the Web clients in order to avoid repeated round-trip delays between the clients and the origin Web servers. The origin Web server in this model is an enhanced Web server which employs a prediction engine and this will derive prefetching rules from the server's access log periodically. These derived rules are assumed to be frequent. That is, only rules with supports larger than the minimum support are derived and provided by Web servers. The derived prefetching rules are stored in the prefetching rule depository of the Web server.

As shown in Fig. 3.1, the proxy serves the requests sent from the Web clients. In the case that a cache miss occurs, the proxy will forward the request to the origin Web server for resolution. Upon receiving the request, the origin server will log this request into record, fetch the requested object form the Web object depository, and check the prefetching rule depository at the same time. If this request triggers some prefetching rules in the prefetching rule depository, the objects implied by these prefetching rules and their corresponding confidences will be piggybacked to the responding message as hints and returned to the proxy. After the proxy receives the response with

the hints piggybacked from the origin Web server, the proxy will first send the requested object back to the client and then determine whether it is worth caching the piggybacked implied objects in the proxy. here the cache replacement algorithm is devised for the integration of Web caching and Web prefetching techniques. if cache hit is found (i.e., the client's request can be satisfied directly with the proxy's local cache), we assume that the proxy will still communicate with the origin Web server to obtain the prefetching hints related to that request after the proxy has sent the response to the client. As such, we are able to investigate each request the prefetching hints from the origin Web server to ensure that the discovered prefetching hints are always up-to date.

A typical system model is shown below

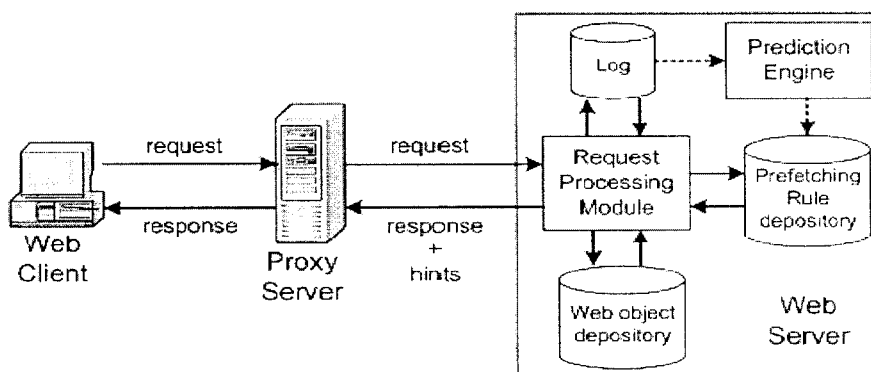


Fig 1. Module for integrating web caching and prefetching

3.2 Creation of Proxy Server Configuration

In this module proxy server is configured in which the images are stored separately. This is done by initializing the Boolean function for images. the port number and maximum number of connection possible with the proxy server are initialized and during the running mode of the server, the server is started and we check whether the ip address is present. If so the server socket is created with port number and maximum number of connection and if ip address is not present then we get the new ip address for it and this process is done by setting the timeout period.

Graphical user interface design is done in this phase. This includes the menu items like start server and stop server and viewing cache and the graphical user interface contains help menu item and client can also view that administrator is currently connected to which person.

3.3 Connection of Proxy Server with Internet

The proxy server is connected with the internet by creating HTTP configuration and proxy cache pool. Proxy ip address and port number is connected with the network and if the cache is enabled then that cache is used.

3.4 Replacement Algorithm Design

Replacement algorithm designing includes two iteration models. in the first iteration step the N-documents is randomly picked from the cache and among that N-documents the least useful document is evicted and then next M least useful

document is retained and in subsequent iteration the $N-M$ documents is randomly picked from cache And it is Appended to the M previously retained documents and among the N -samples the least useful document is evicted and M next least useful document are retained.

CHAPTER 4

TESTING

4.1 INTRODUCTION

After finishing the development of any computer based system the next complicated time consuming process is system testing. During the time of testing only the development company can know that, how far the user requirements have been met out, and so on.

Following are the some of the testing methods applied to this effective project:

4.2 SOURCE CODE TESTING

This examines the logic of the system. If we are getting the output that is required by the user, then we can say that the logic is perfect.

4.3 SPECIFICATION TESTING

We can set with, what program should do and how it should perform under various condition. This testing is a comparative study of evolution of system performance and system requirements.

4.4 MODULE LEVEL TESTING

In this the error will be found at each individual module, it encourages the programmer to find and rectify the errors without affecting the other modules.

4.5 UNIT TESTING

Unit testing focuses on verifying the effort on the smallest unit of software-module. The local data structure is examined to ensure that the data stored temporarily maintains its integrity during all steps in the algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.

4.6 INTEGRATION TESTING

Data can be tested across an interface. One module can have an inadvertent, adverse effect on the other. **Integration testing** is a systematic technique for constructing a program structure while conducting tests to uncover errors associated with interring.

4.7 VALIDATION TESTING

It begins after the integration testing is successfully assembled. Validation succeeds when the software functions in a manner that can be reasonably accepted by the client. In this the majority of the validation is done during the data entry operation where there is a maximum possibility of entering wrong data. Other validation will be performed in all process where correct details and data should be entered to get the required results.

4.8 RECOVERY TESTING

Recovery Testing is a system that forces the software to fail in variety of ways and verifies that the recovery is properly performed. If

recovery is automatic, re-initialization, and data recovery are each evaluated for correctness.

4.9 SECURITY TESTING

Security testing attempts to verify that protection mechanism built into system will in fact protect it from improper penetration. The tester may attempt to acquire password through external clerical means, may attack the system with custom software design to break down any defenses to others, and may purposely cause errors.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENT

In the first phase of the project, proxy server was configured. The proxy server, which was designed in first phase, doesn't use the data structure to store the data in the cache and hence the server is capable of accessing to number of clients and the administrator can view the data's present in the cache.

The work to be proposed in the next phase is to connect this proxy server created to connect it with the Internet later the cache replacement policies should be done to reduce the response time of the user and then to test and implement the proxy server created.

CHAPTER 6

APPENDICES

6.1 Sample Code

6.1.1 Proxy Server Creation

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.InetAddress;
import java.net.ServerSocket;
```

```
public class ProxyServer
    implements Runnable
```

```
{
```

```
//initializing proxy server with port number and maximum
connection
```

```
ProxyServer(Object p, String ip, int port, boolean saveimgs)
```

```
{
```

```
    server_port = 80;
```

```
    max_connections = 8;
```

```
    server_running = false;
```

```
    server_exit = false;
```

```
    console = p;
```

```
    server_ip = ip;
```

```
    server_port = port;
```

```
    this.saveimgs = saveimgs;
```

```
    myThread = new Thread(this);
```

```
    myThread.setDaemon(true);
```



```

        myThread.start();
    }

    ProxyServer(Object p, int port)
    {
        this(p, "127.0.0.1", port, false);
    }

    public boolean isServerRunning()
    {
        return server_running;
    }

    //initializing proxy server to start
    public synchronized boolean startServer()
    {
        if(server_running)
            return true;
        try
        {
            if(server_ip == null || server_ip.equals("")) ||
server_ip.equals("*"))
                ss = new ServerSocket(server_port, max_connections);
            else
                ss=newServerSocket(server_port,max_connections,InetAddress.getBy
Name(server_ip));
            ss.setSoTimeout(1000);

```

```

    }
    catch(IOException e)
    {
        ss = null;
        return false;
    }
    System.out.println("Server Started on " + server_ip + ", port: "
+ server_port);
    server_running = true;
    return true;
}
//initializing proxy server to stop
public synchronized boolean stopServer()
{
    server_running = false;
    try
    {
        ss.close();
    }
    catch(IOException _ex) { }
    ss = null;
    return true;
}

public void serverExit()
{

```

```

    }
//proxy server in running phase
    public void run()
    {
        while(!server_exit)
            if(server_running)
                try
                {
                    java.net.Socket s = ss.accept();
                    new ProxyConnection(console, this, s, saveimgs);
                    ((ServerInterface)console).updateHTTPCounter();
                }
                catch(IOException _ex) { }
            else
                try
                {
                    Thread.sleep(500L);
                }
                catch(InterruptedException _ex) { }
    }

```

```

String server_ip;
int server_port;
int max_connections;
private Thread myThread;
private ServerSocket ss;
private boolean server_running;

```

```
private boolean server_exit;
Object console;
static final String localhost = "127.0.0.1";
static final boolean debug_mode = false;
boolean saveimgs;
}
```

6.1.2 Proxy Connection Creation

```
import java.io.Serializable;
import java.util.Date;

public class ProxyCache
    implements Serializable
{

    ProxyCache()
    {
        url = "";
        filename = createName();
    }

    public boolean isExpired(Date d)
    {
        return expiration.after(d);
    }

    private String createName()
```

```
{
    char file_char[] = new char[10];
    for(int i = 0; i < 10; i++)
    {
        int location = (int)(Math.random() * 1000D) %
alphabets.length;
        file_char[i] = alphabets[location];
    }

    return new String(file_char) + ".cache";
}

public String getName()
{
    return filename;
}

public void setExpiration(Date d)
{
    expiration = d;
}

public void setExpiration(long d)
{
    expiration = new Date(d);
}
```

```
public Date getExpiration()
{
    return expiration;
}
```

```
public void setURL(String newurl)
{
    url = newurl;
}
```

```
public String getURL()
{
    return url;
}
```

```
public void setHeader(String h)
{
    header = h.getBytes();
}
```

```
public void setHeader(byte h[])
{
    header = h;
}
```

```
public void setContent(String c)
{

```

```
        content = c.getBytes();
    }

    public void setContent(byte c[])
    {
        content = c;
    }

    public byte[] getHeader()
    {
        return header;
    }

    public byte[] getContent()
    {
        return content;
    }

    public boolean matchURL(String target)
    {
        return url.equals(target);
    }

    String url;
    Date expiration;
    String filename;
    byte header[];
```

```

byte content[];
static char alphabets[] = {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
    'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
    'u', 'v', 'w', 'x', 'y', 'z', '1', '2', '3', '4',
    '5', '6', '7', '8', '9', '0'
};
}

```

6.1.3 Graphical User Interface Design

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class Web extends JFrame
{
    JMenuBar mb;
    JMenu Furher;
    JMenuItem
Initiateserver,Haltserver,Lucidhoard,Lucidsimilies,Depart;
    JMenu Scrutiny;
    JMenuItem
Serverretort,Clientappeal,Beginpotraitviewer,Beginhoardviewer;
    JMenu Launch;
    JMenuItem Serverconfiguration;
    JMenu Lendahand;

```



```

        JMenuItem Assist,Concern;
        String msg;
public Web()
{
//setTitle("WEBCACHING AND WEBPREFETCHING");
mb=new JMenuBar();
setJMenuBar(mb);
        Furher=new JMenu("Furher");
        Scrutiny=new JMenu("Scrutiny");
        Launch=new JMenu("Launch");
        Lendahand=new JMenu("Lendahand");
        mb.add(Furher);
        mb.add(Scrutiny);
        mb.add(Launch);
        mb.add(Lendahand);

        Initiateserver=new JMenuItem("Initiateserver");
        Haltserver=new JMenuItem("Haltserver");
        Lucidhoard=new JMenuItem("Lucidhoard");
        Lucidsimilies=new JMenuItem("Lucidsimilies");
        Depart=new JMenuItem("Depart");
        Serverretort=new JMenuItem("Serverretort");
        Clientappeal=new JMenuItem("Clientappeal");
        Beginpotraitviewer=new
JMenuItem("Beginpotraitviewer");
        Beginhoardviewer=new
JMenuItem("Beginhoardviewer");

```

```

        Serverconfiguration=new
JMenuItem("Serverconfiguration");
        Assist=new JMenuItem("Assist");
        Concern=new JMenuItem("Concern");
        Furher.add(Initiateserver);
        Furher.add(Haltserver);
        Furher.add(Lucidhoard);
        Furher.add(Lucidsimilies);
        Furher.add(Depart);
        Scrutiny.add(Serverretort);
        Scrutiny.add(Clientappeal);
        Scrutiny.add(Beginpotraitviewer);
        Scrutiny.add(Beginhoardviewer);
        Launch.add(Serverconfiguration);
        Lendahand.add(Assist);
        Lendahand.add(Concern);
        JMenu Systemlogs;
        Systemlogs=new JMenu("Systemlogs");
        Scrutiny.add(Systemlogs);
        Systemlogs.add("Admittancemonitor");
        Systemlogs.add("Blundermonitor");

    }
    class WH extends WindowAdapter
    {
        public void windowClosing(WindowEvent e)

```

```

    {
        System.exit(0);
    }
}

public static void main(String args[])
{
    Web fra=new Web();
        Toolkit tk=fra.getToolkit();
        fra.setIconImage(tk.getImage("sb.jpg"));

    fra.setSize(800,600);

        fra.setVisible(true);
    }
}

```

6.2 Output and Screenshots

The final implementation includes the process of connecting the proxy server with the client and the server will fetch the documents from web and the response time of the user reduced is shown

The function initiate server and halt server and lucid hoard(i.e)viewing the cache and lucid smiles(i.e) viewing smiles and depart function is implemented in server side can be shown as follows

In scrutiny the function server retort and client appeal and portrait viewing and hoard viewing and admittance monitor functions is implemented in server side can be shown an as follows

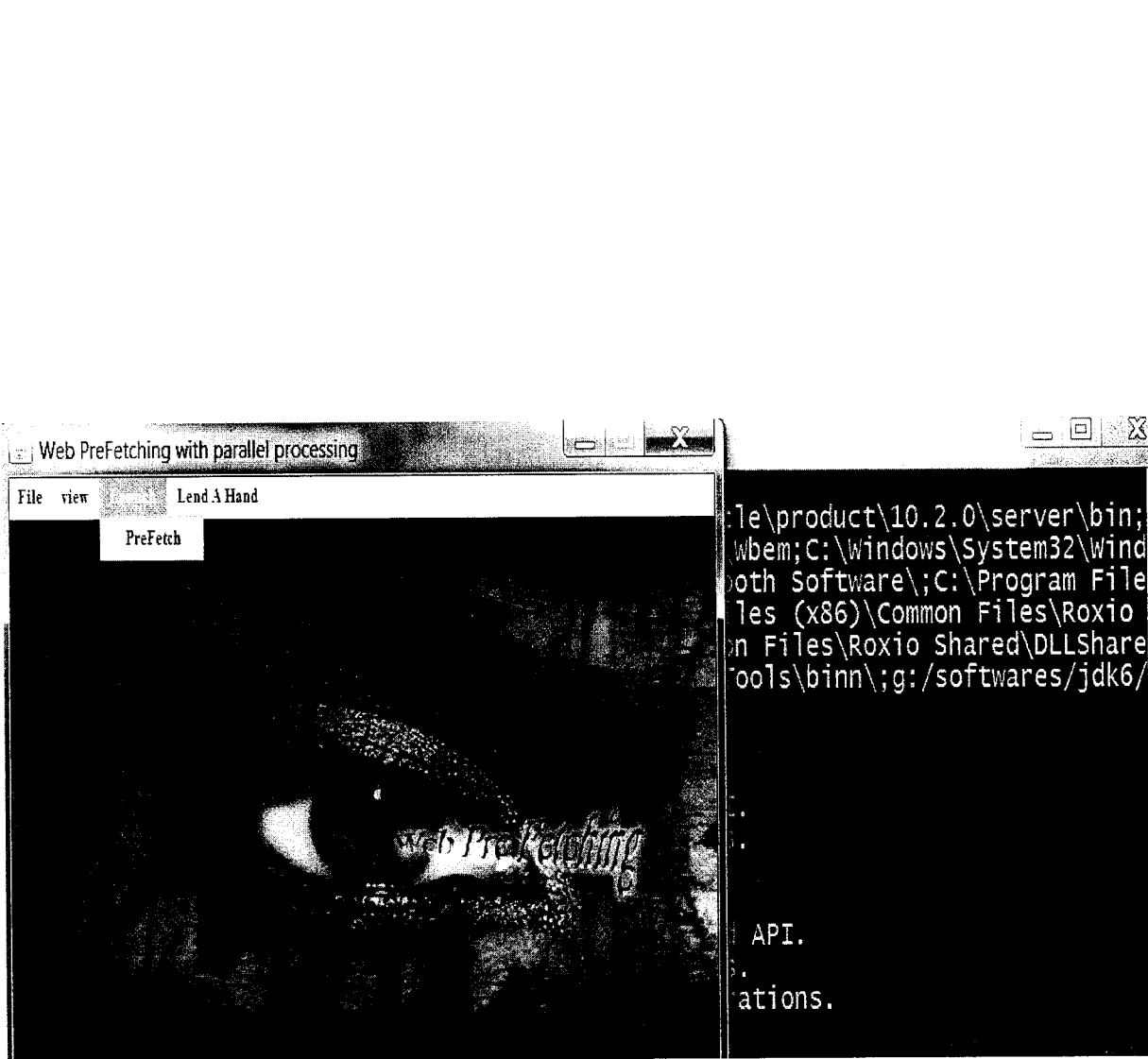


Fig 2.Prefetching the Web Cache

Server Configuration

Network | **Logging**

TCP/IP

IP : 192.168.100.7 Proxy Server :

PORT : 2008 Proxy Port :

Note: leave proxy ip/port blank if not using proxy.

Connections

Receive TimeOut : 300 Cached URLs : 3000

Send TimeOut : 30 Cache Expires (hrs) : 1

Number of Threads : 100 Show Printable Contents ?

Save Cancel

Fig 3.Specifying the Server

Server Configuration

Network | **Logging**

Logging

Access Log : /logs/access.log

Error Log : /logs/error.log

Logging Level : Minimal
 Normal
 Detail

Save downloaded images ?

Save Cancel

Fig 4. Log File Creation

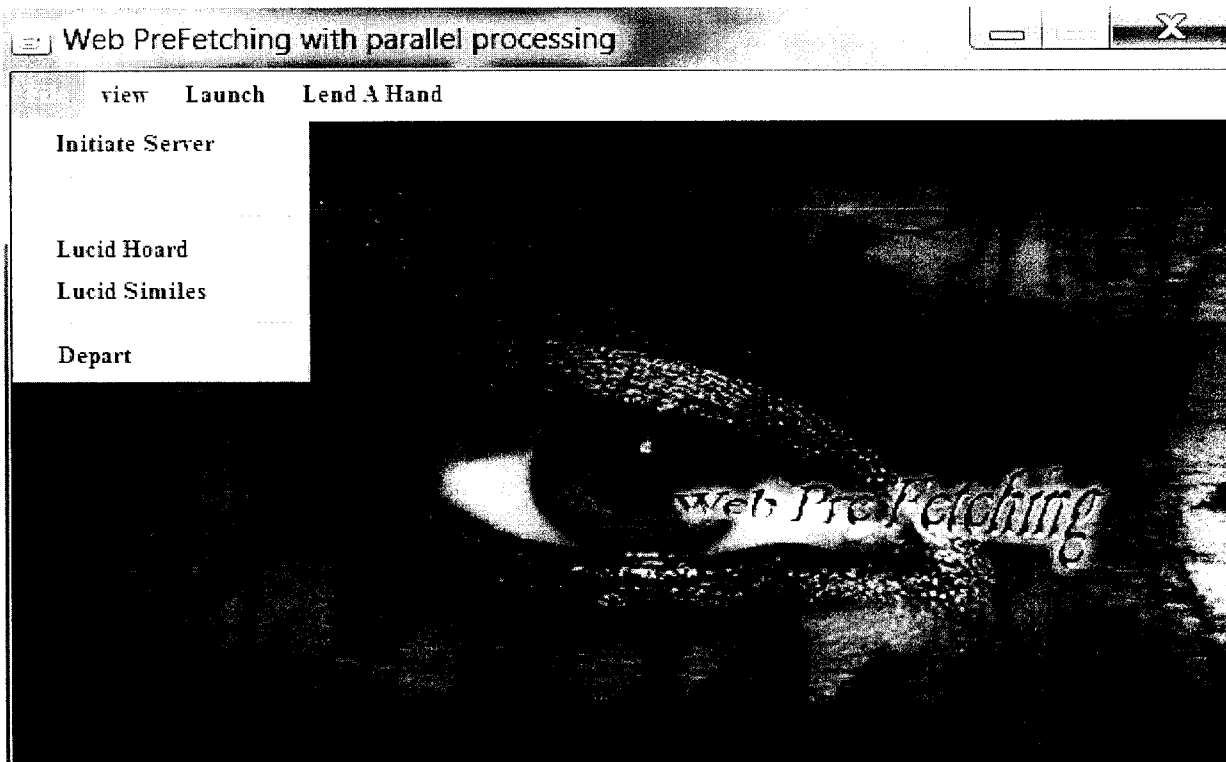
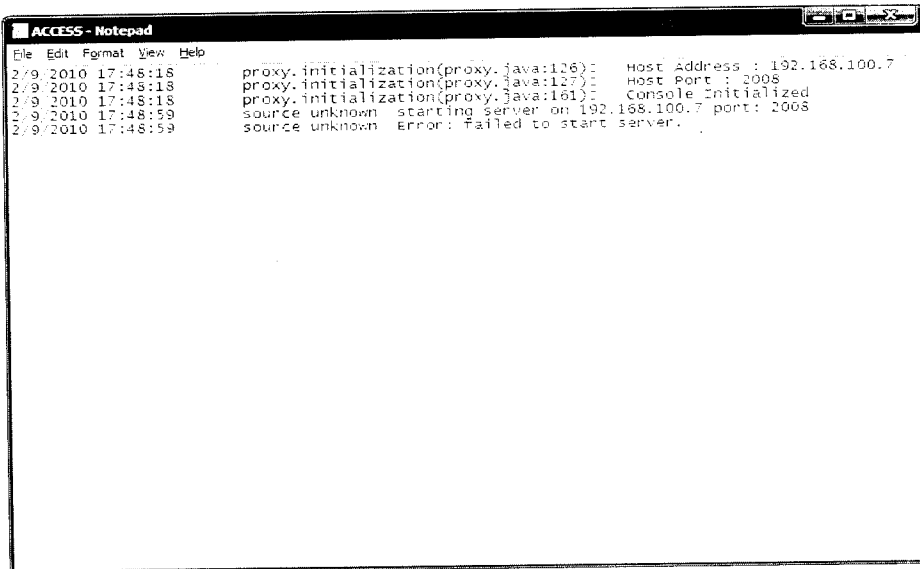
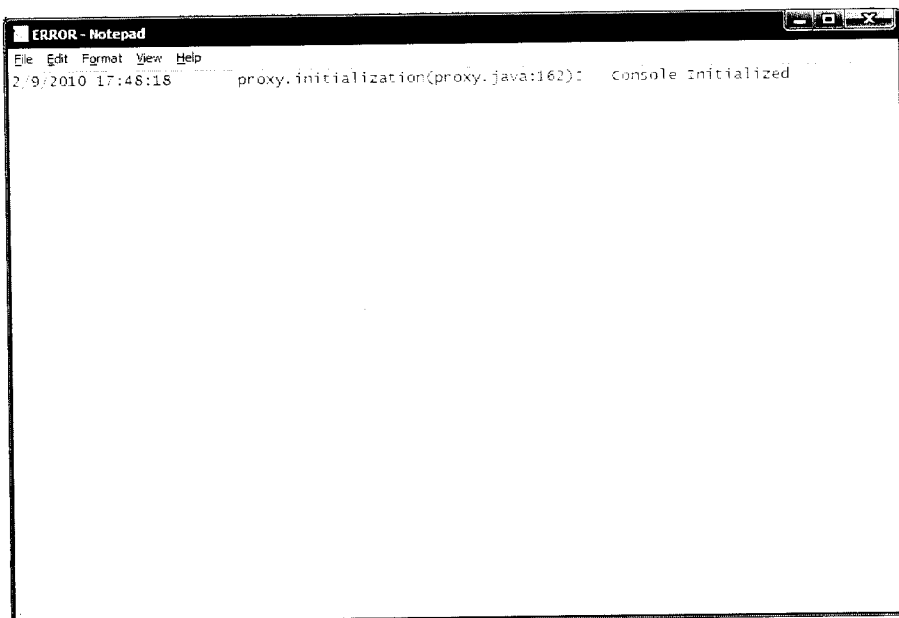


Fig 5. Initiating the Server



```
ACCESS - Notepad
File Edit Format View Help
2/9/2010 17:48:18 proxy.initialization(proxy.java:126): Host Address : 192.168.100.7
2/9/2010 17:48:18 proxy.initialization(proxy.java:127): Host Port : 2008
2/9/2010 17:48:18 proxy.initialization(proxy.java:161): Console Initialized
2/9/2010 17:48:59 source unknown starting server on 192.168.100.7 port: 2008
2/9/2010 17:48:59 source unknown Error: failed to start server.
```

Fig 6.Access Log File



```
ERROR - Notepad
File Edit Format View Help
2/9/2010 17:48:18 proxy.initialization(proxy.java:162): Console Initialized
```

Fig 7.Error Log File

CHAPTER 7

REFERENCES

Books:

[1]. Patric Naughton Herbert Shieldt, ‘ *Java 2: The Complete Reference* ’, Tata McGraw-Hill, 1999

Conference Proceedings:

[2] C. Aggarwal, J.L. Wolf, and P.-S. Yu, “Caching on the World Wide Web,” IEEE Trans. Knowledge and Data Eng., vol. 11, no. 1, pp. 94- 107, Jan. /Feb. 1999.

[3] P. Barford and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation,” Proc. 1998 ACM SIGMETRICS Int’l Conf. Measurements and Modeling of Computer Systems, 1998.

[4] G. Barish and K. Obraczka, “World Wide Web Caching: Trends and Techniques,” IEEE Comm. Magazine, Internet Technology Series, pp. 178-185, 2000.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web Caching and Zipf Like Distributions: Evidence and Implications,” Proc. IEEE INFOCOM 1999, Mar. 1999.

[6] P. Cao, E.W. Felten, A. Karlin, and K. Li, “A Study of Integrated Prefetching and Caching Strategies,” Proc. 1995 ACM SIGMETRICS

Int'l Conf. Measurements and Modeling of Computer Systems, pp.
188-197, 1995.