BANDWIDTH OPTIMIZATION USING CONTENT ALIASING OF PROXY SERVER

A Project
Presented
to the Faculty of
California State University, Chico

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
© Khyati Desai 2009
Summer 2009
BANDWIDTH OPTIMIZATION USING CONTENT ALIASING OF
PROXY SERVER

A Project

by

Khyati Desai

Summer 2009

APPROVED BY THE INTERIM DEAN OF THE SCHOOL OF
GRADUATE, INTERNATIONAL, AND INTERDISCIPLINARY STUDIES:

_________________________________
Mark J. Morlock, Ph.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

Abdel-Moaty M. Fayek
Graduate Coordinator

Seung-Bae Im, Ph.D., Chair

Orlando Madrigal, Ph.D.
PUBLICATION RIGHTS

No portion of this project may be reprinted or reproduced in any manner unacceptable to the usual copyright restrictions without the written permission of the author.
ACKNOWLEDGMENTS

Apart from the hard work and efforts I have put in, the successful completion of this project depends highly on the support, guidance, and motivation from a number of people. I would like to take this opportunity to extend my gratitude to all those people who have been a source of great support to me in the successful completion of this project.

I am highly indebted to Dr. Seung Bae Im and Dr. Orlando Madrigal for their tremendous support, guidance, and help. Without their guidance, this project could not have been materialized. I am grateful to Professor Abdel Fayek Moaty for his guidance and for showing me the correct approach. He has been a great advisor, educator, and comrade.

To conclude, I would like to thank my parents who supported me in my pursuit of graduate studies at California State University, Chico. Without them, I would not have achieved my goal of earning a Master’s degree.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Publication Rights</th>
<th>iii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>Abstract</td>
<td>viii</td>
</tr>
</tbody>
</table>

## CHAPTER

### I. Introduction

<table>
<thead>
<tr>
<th>Purpose</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope of the Project</td>
<td>5</td>
</tr>
<tr>
<td>Important Concepts</td>
<td>5</td>
</tr>
</tbody>
</table>

### II. Literature Review

<table>
<thead>
<tr>
<th>Overview of Web Caching</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Caching Systems</td>
<td>11</td>
</tr>
<tr>
<td>Caching Architectures</td>
<td>12</td>
</tr>
<tr>
<td>Web Caching Algorithms</td>
<td>12</td>
</tr>
<tr>
<td>MD5 Algorithm</td>
<td>15</td>
</tr>
<tr>
<td>Decoding MD5 Hash</td>
<td>18</td>
</tr>
<tr>
<td>Static Caching</td>
<td>18</td>
</tr>
</tbody>
</table>

### III. Design and Implementation

<table>
<thead>
<tr>
<th>Terminology</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware and Software Requirements</td>
<td>24</td>
</tr>
<tr>
<td>Our Solution</td>
<td>24</td>
</tr>
<tr>
<td>Algorithm for Content Aliasing Avoidance</td>
<td>30</td>
</tr>
<tr>
<td>Use Case Scenario</td>
<td>32</td>
</tr>
<tr>
<td>Activity Diagram</td>
<td>32</td>
</tr>
<tr>
<td>Sequence Diagram</td>
<td>35</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>PAGE</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Deployment Diagram</td>
<td>35</td>
</tr>
<tr>
<td>Results of Statistical Analysis</td>
<td>35</td>
</tr>
<tr>
<td>IV. Testing</td>
<td>40</td>
</tr>
<tr>
<td>Test Plan and Results</td>
<td>40</td>
</tr>
<tr>
<td>V. Future Scope and Conclusions</td>
<td>43</td>
</tr>
<tr>
<td>Future Advancement</td>
<td>45</td>
</tr>
<tr>
<td>Conclusion</td>
<td>46</td>
</tr>
<tr>
<td>References</td>
<td>47</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Proxy Server Setup</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td><a href="http://www.yahoo.com">www.yahoo.com</a> Web Page</td>
<td>3</td>
</tr>
<tr>
<td>3.</td>
<td><a href="http://www.yahoo.co.in">www.yahoo.co.in</a> – Web Page of <a href="http://www.yahoo.com">www.yahoo.com</a> in India Domain</td>
<td>4</td>
</tr>
<tr>
<td>4.</td>
<td>The Concept of Web Caching</td>
<td>7</td>
</tr>
<tr>
<td>5.</td>
<td>Class Diagram of Our Solution</td>
<td>25</td>
</tr>
<tr>
<td>6.</td>
<td>Collaboration Diagram of Our Solution</td>
<td>26</td>
</tr>
<tr>
<td>7.</td>
<td>Graphical User Interface (GUI) Component of Our Solution</td>
<td>27</td>
</tr>
<tr>
<td>8.</td>
<td>Graphical User Interface (GUI) Component of Our Solution with a Cache Hit</td>
<td>28</td>
</tr>
<tr>
<td>9.</td>
<td>Graphical User Interface (GUI) Component of Our Solution with a Cache Miss</td>
<td>29</td>
</tr>
<tr>
<td>10.</td>
<td>Flowchart of Our Solution</td>
<td>31</td>
</tr>
<tr>
<td>11.</td>
<td>Use case Scenario of Our Solution</td>
<td>33</td>
</tr>
<tr>
<td>12.</td>
<td>Activity Diagram of Our Solution</td>
<td>34</td>
</tr>
<tr>
<td>13.</td>
<td>Sequence Diagram of Our Solution</td>
<td>36</td>
</tr>
<tr>
<td>14.</td>
<td>Deployment Diagram of Our Solution</td>
<td>37</td>
</tr>
<tr>
<td>15.</td>
<td>MD5 and MRU Statistics for Sites Visited by a User</td>
<td>38</td>
</tr>
<tr>
<td>16.</td>
<td>HTML Code of the Site (<a href="http://www.google.com">www.google.com</a>) Visited by a User</td>
<td>38</td>
</tr>
<tr>
<td>17.</td>
<td>HTML Code of the Site (<a href="http://www.cricinfo.com">www.cricinfo.com</a>) Visited by a User</td>
<td>39</td>
</tr>
</tbody>
</table>
ABSTRACT

BANDWIDTH OPTIMIZATION USING CONTENT ALIASING OF PROXY SERVER

by

© Khyati Desai 2009

Master of Science in Computer Science

California State University, Chico

Summer 2009

Network traffic has increased tremendously due to the rise in the number of Internet users. Increase in network traffic affects several other aspects of the network such as reduced network bandwidth, increased latency, and higher response time for users. In order to design techniques that optimize network bandwidth, reduce latency, and the response time for users, it is very important to understand the characteristics of web traffic.

Researchers have focused on the problem of aliasing in the proxy server caches for a long time. Aliasing in proxy server caches occurs when the same content is stored in cache multiple times. This storing of the same content in cache multiple times happens when multiple requests are made to multiple, but different, websites that contain the same content. Websites that store the same content are called mirrors. Mirroring
increases efficiency but at the cost of consuming storage space in cache. In most cases, the amount of storage space consumed by aliasing in cache is enormous.

Techniques such as abstracting common content from websites across domains and mirroring, can be used to utilize the problem of content aliasing for the benefit of proxy server caches. In contrast, in this project, we devise a technique wherein we check the cache contents for redundancy, in efforts to reduce network traffic and latency, but also, to indirectly optimize the use of network bandwidth. Our technique not only finds a solution to the problem of redundancy in aliasing in proxy server caches, but is also a contribution to the open source community.
CHAPTER I

INTRODUCTION

Purpose

In the field of web server management, researchers have focused on *aliasing* in proxy server caches for a long time. Web caching consists of storing frequently referred objects on a caching server instead of the original server, so that web servers can make better use of network bandwidth, reduce the workload on servers, and improve the response time for users [1]. Aliasing means giving multiple names to the same thing. Aliasing in proxy server caches occurs when the same content is stored in cache multiple times. A proxy server acts as a mediator between the original server and the clients. The proxy server setup can be seen in Figure 1.

On the World Wide Web, aliasing commonly occurs when a client makes two requests, and both the requests have the same payload. Currently, browsers perform cache lookups using Uniform Resource Locators (URLs) as identifiers. Aliasing causes repetitive data transfers even when the current request has already been cached under a different URL.

Websites that contain the same content are called mirrors. Mirrors are redundancy mechanisms built into the web space to serve webpages faster, but they cost in terms of cache space. As the amount of web traffic increases, the efficient utilization of network bandwidth increasingly becomes more important. We need to analyze web
Fig. 1. Proxy server setup.

traffic and understand its characteristics to be able to optimize the use of network bandwidth, to reduce network latency, and to improve response time for users [2].

Commercial browsers are slowly becoming aware of aliasing in proxy server caches, and therefore, are encouraging website designers to make cache-friendly web pages that avoid aliasing. However, website designers as well as administrators do not know much about the effects of aliasing and how it causes repeated transfers of the same payload.

In this project, we propose to implement a different approach of storing contents in the cache such that browsers do not request the same content from multiple websites if this content is already cached. We propose to store contents in the cache based on their name key and content key. The name key will be the MD5 hash of the URL, while the content key will be the MD5 hash of the individual objects in the URL. A name
key will point to a set of content keys. For example, the website www.yahoo.com, as shown in Figure 2 [4], provides its users content based on regional domains such as www.yahoo.co.au, www.yahoo.co.ca, and www.yahoo.co.in (as shown in Figure 3) [4].

Fig. 2. www.yahoo.com web page.


Most of the content on these regional domains’ websites, including, groups, mails, messenger, geocities, is the same. Using data aliasing, we can cache the common content, but retrieve the unique (not common across regional domains) content from the original server. In this setup, a request will comprise of common content from the cache and unique content from the original server. We will also use mirroring along with
content aliasing to keep the cache consistent and updated with the contents of the original server. In order to fulfill requests, the cache will require a constant connection with the original server. Mirroring refers to a special case of aliasing in which replicas of pages or entire sites are deliberately made available through different URLs [3].

When a browser requests a web page, we will check both name keys and content keys, and therefore, will avoid the problem of duplicate contents in the cache. We expect our solution to not only solve the problem of better use of network bandwidth, reduce network latency, and improve response time for users, but also be a contributor to the open source community.
Scope of Project

Web caching is defined as caching of web pages in order to optimize the use of bandwidth, reduce the load on servers and perceived lag. In simple words, it means storing the frequently referred objects on a caching server instead of an original sever. These objects range from the objects that have been recently acquired from the original website to the ones which are already stored in the cache. In this context, we will be devising a method wherein the cache contents will be checked for redundancy. Along with the cached URL, the cache contents will also be checked. This will help in reducing the latency and reduce network traffic, which indirectly will optimize the bandwidth. The basic goal of this project is to optimize the bandwidth usage by using content aliasing. The implementation will also reduce latency and network traffic.

Important Concepts

In this section, we introduce important concepts, such as, mirroring, web caching, and duplicate suppression, that we use throughout this project.

Mirroring

Mirroring is defined as keeping multiple copies of the content of a Web site or Web pages on different servers using different domain names. A mirrored site is typically an exact replica of the original site and is updated frequently to ensure that it reflects the updates made to the content of the original site. The main purpose of mirroring is to build in redundancy and ensure high availability. Mirrored sites also help make access faster when the original site is geographical distant; for example, a popular web site in Germany may have a mirror site in the United States (U.S.) to reduce the response time.
for its U.S. users. In some cases, an original site that does not have a high speed connection to the Internet, for example a small business, may arrange to mirror its contents at a larger site with a faster connection and in closer proximity to its users [5].

Shivkumar and Garcia-Molina investigated mirroring in a large crawler data set and reported that in the WebTV client trace far more aliasing happens than expected. In fact, they reported that 36% of reply bodies are accessible through more than one URL [3]. Similarly, Bharat et al. surveyed techniques for identifying mirrors on the Internet [6]. Bharat and Broder investigated mirroring in a large crawler data set and reported that roughly 10% of popular hosts are mirrored to some extent [6]. Broder et al. considered approximate mirroring or “syntactic similarity” [6]. Although they introduce sophisticated measures of document similarity, they report that most “clusters” of similar documents in a large crawler data set contain only identical documents.

**Web Caching**

Although the number of servers in response to the popularity of web browsing has increased, it is not sufficient – a number of servers are unable to meet a large number of user requests. Web caching, a technique that stores frequently used objects on local servers instead of on original servers, can help websites reduce network latency and improve user response time. Typically, when a user requests contents from a website, the content must travel through several servers before reaching the user who requested it. With web caching, the original content travels to the cache once; subsequently, the content is served by the cache itself. Figure 4 presents a basic web caching system. Objects stored by web caching could have been existed on the server for a long time or could have been acquired from the original server recently. Website administrators use
several types of web caching techniques, such as, active caching, adaptive caching, transparent caching, push caching, proxy caching, and reverse proxy caching.

One of the most important issues that affect web caching is how the size of the cached document affects memory usage and network bandwidth utilization. These two factors are in contradiction with each other. For example, let us assume that two documents with significant difference in their sizes, but nearly equivalent values, are cached. At the time of eviction, assume that the larger document is evicted instead of the smaller document. Evicting the smaller document frees up only a small amount of memory, but uses less network bandwidth to obtain the document again when requested by the user. On the other hand, evicting the larger document frees up a large amount of
memory, but uses more network resources to obtain the document again. Most common web caching algorithms in use include greedy dual algorithm, greedy dual-size algorithm, randomizing proxy cache algorithms, and the lowest relative value algorithm.

**Duplicate Suppression**

Many web pages with different URLs are duplicates of each other. If the cache detects the duplicate existence of a requested page, it will reduce network cost, if the page is served from cache. Techniques like data compression, caching, and content simplification can be used to avoid sending unnecessary bytes over a network. There are several reasons due to which hit ratios cannot be improved significantly. One of the reasons are things like shopping carts, stocks, and results to queries, electronic commerce, cannot be cached.

Due to large network size, many pages will not be referenced multiple times by any one cache, meaning, that the probability with which the kth page will be referenced is 1/k. Re-references follow a distribution similar to Zipf’s law [7, 8]. Williams et al. observed that the simple hit ratio is always higher than the weighted hit ratio because number of references to the smaller resources is more than those to larger resources. In contrast, Breslau et al. reported that there is no strong correlation between the frequency with which a page is accessed and the size of the document, even though the average size of the resources is smaller.

The utility of web pages can be extended using several techniques including delta encoding, prefetching, and partial transfers. Dougliis et al. observed that the frequency with which a web page changes varies from months to hours or days [9]. Pages that change more frequently than this frequency cannot be cached. In such cases, delta
encoding can be used in which the server sends only the difference between the current and the cached versions. Prefetching is a technique in which future requests are predicted based on past requests, and the data that is most likely to be requested again is prefetched to reduce latency. However, the techniques discussed above do not take into account the result of the fact as to what happens if identical responses are generated by two different resources. In such cases, a cache entry for either one of them can be used to fulfill the requests, and therefore, duplicate suppression must be used.

Douglis et al. reported that 18% of full body responses recorded at a corporate firewall resulted in a new instance of a particular resource which is identical to at least one other instance of a different resource [9]. Mogul reviewed a variety of end-to-end duplicate suppression schemes involving hints supplied by the origin servers to clients and by clients to caches, and reported [10]. Santos and Wetherall as well as Spring and Wetherall described a general protocol independent network layer technique that eliminates redundant traffic by caching packet payloads and transmits their digests to reduce redundant transfers [11].
CHAPTER II

LITERATURE REVIEW

Overview of Web Caching

As the popularity of the Internet grows, so does the dominance of the use of Hyper Text Transfer Protocol (HTTP). Internet users commonly face overloaded servers, network congestion, network latency, and higher response times. Currently, the World Wide Web (WWW) is the most used means of sharing documents and getting information about various topics ranging from education, research, news, recipes, etc. As more and more information becomes available over the web, the demand for more network bandwidth is likely to surpass supply in the near future. The attractiveness of WWW to its users will be compromised if it gets more and more congested, and users have to wait for a long time for web pages to load. The above problem may be solved by either increasing the network bandwidth or by using web caching. Increasing network bandwidth may cause lots of technical problems, not to mention that it is expensive [30], [31].

One web caching technique is to use large caches; another is to use efficient cache management techniques. Both these web caching techniques are limited in effects because most of the documents are accessed only once. Additionally, these techniques are costly. Some of the problems associated with the cache management techniques are that the time it takes to transfer files varies, and that due to the use of HTTP, it is only possible to transfer files in their entirety over the web – it is not possible to transfer
certain parts and portions of files. The basic concept of caching involves storing web pages contents on local servers that are closer to their users, so that when the user requests the same web page again, it is server from the local server and not the origin server. Web proxy caching, therefore, reduces user response time as well as usage of network bandwidth. Web proxy caching improves network performance in three ways – it reduces network latency and amount of web traffic as well as the transit cost for access providers. Web proxy caching reduces network latency because the proxy server is located closer to users. Web proxy caching reduces the amount of web traffic because documents do not need to travel through the network – they are retrieved from the cache instead of the origin server. Finally, web proxy caching reduces transit costs for access providers because access providers are not involved anywhere in cache hits [32], [33].

Web Caching Systems

In web caching systems, documents may be cached at proxies, clients, or servers. When a user requests a page, the request first goes to the user’s own browser. If the user’s browser does not have a copy of the requested page, the request is forwarded to the proxy server. If the request cannot be fulfilled by the proxy server also, then the request is forwarded to the main server. Several factors, including cache architecture (hybrid, distributed or hierarchical), proxy placement, caching contents, cache placement (or replacement) need to be considered to make the cache system work correctly. Low network latency, robustness, absence of single point of failure, transparency, fast response, higher availability, scalability, adaptability to various network architectures,
simplicity, and ease of implementation are some of the most desirable features of web proxy caching [18], [20].

Caching Architectures

Caching architectures may be hierarchical, distributed, or hybrid. In hierarchical caching architecture, the cache is placed at different levels like bottom, regional, institutional and national. In distributed caching architecture, there are no intermediate cache levels - traffic flows through less congested network levels. Hybrid caching architecture is a combination of hierarchical and distributed architecture. Experts are still debating as to which architecture has optimal performance.

Web Caching Algorithms

A perfect caching algorithm would be able to serve user’s requests immediately without reloading the requested data. However, this is only possible if future data requests are known a-priori, and all the requested data is stored in the cache. One of the biggest problems with this scenario is that the size of cache memory is limited. The most important factor that affects the effectiveness of caches is the placement or replacement algorithm, which yields the \(ht\) ratio. There are a number of different cache replacement algorithms, such as, greedy dual, greedy dual-size. Commonly, a majority of cache replacement algorithms work by first intercepting the data that is served from the cache. If the requested data is not cached, cache replacement algorithms are not needed anymore, and the request is furnished from origin server [7], [54].

If the requested data does not fit in the cache, a cache replacement algorithm must evict data from the cache. The strategy used to evict data from cache, also known as
replacement scheme or replacement policy, is an important consideration in the selection of cache replacement algorithm. Replacement schemes play an important role in the efficiency of the cache. Ideally, a replacement scheme should evict data that is not accessed anymore or data that is easy to reload. Various web-caching algorithms, such as greedy dual, greedy dual-size, randomizing proxy caching, lowest retail value, least recently used, least frequently used, and most recently used, are used for effective web caching. We discuss these algorithms briefly in the upcoming paragraphs [54].

The Greedy Dual Algorithm

The greedy dual algorithm, proposed by Young, maintains an evaluation value $H$ associated with each page $H(p)$. The algorithm also maintains the cost, $C(p)$, associated with caching the document. Whenever a document needs to be replaced, a document with smaller $H(p)$ value is evicted. Once a document is evicted, the $H$ value of the documents that remain in the cache is reduced by the $H$ value of the evicted document. In contrast, the $H$ value of a document that is requested again is reset to $C(p)$ [12].

The Greedy Dual-Size Algorithm

Cao and Irany modified the greedy dual algorithm to also consider the size of the document along with its cost. Therefore, in greedy dual-size algorithm, the $H$ value of a document is calculated based on its cost as well as its size. The greedy dual algorithm, presented in the previous paragraph, does not consider the size of the document. In greedy dual-size algorithm, whenever a document is cached or reassessed, its $H$ value is reset to $c(p)/s(p)$ where $c(p)$ is the cost of the document and $s(p)$ is the size of the document. Using this algorithm improves the efficiency of memory use because when
two same-cost documents are cached, the larger-sized document is removed before the smaller-sized document [13].

One of the problems associated with the greedy dual-size algorithm is that the cost associated with maintaining data structures that keep track of deleted or requested documents is very high. A solution to this problem of high maintenance costs is to use randomizing proxy caching algorithm, which evicts documents randomly. However, this algorithm uses memory inefficiently because documents are evicted in a randomized order. Randomizing Proxy caching Algorithm do not maintain any data structures because the documents are evicted randomly. The main disadvantage of this scheme is the usefulness of the documents is not taken into consideration which means that even the most useful document can be evicted since the selection is a random process. Psounis and Prabhakar [14] proposed a new scheme which is a combination of the greedy dual and greedy dual-size web caching algorithms as well as the randomizing proxy algorithm. In this new replacement scheme a subset of cache (size K) is maintained. The subset remains empty as long as no document is deleted. But when a document needs to be deleted the subset is filled with documents that are selected randomly from the main cache. Now only the least useful document from the subset is deleted instead of all the documents.

The Lowest Retail Value Algorithm

As the name suggests, the lowest retail value algorithm (LRV) evicts documents with the lowest retail value. The retail value of a document is calculated as $V = (C/B) \times P_r$, where $C$ is the cost, $B$ is the benefit associated with the document, and $P_r$ is the probability of the document being accessed [12], [13].
The Least Recently Used Algorithm

The least recently used algorithm (LRU) evicts the least requested document. However, this algorithm is very expensive because it needs to keep track of which document was used when. For this algorithm to be efficient and accurate, it is important that only the appropriate documents are discarded [12], [13].

The Least Frequently Used Algorithm

The least frequently used algorithm (LFU) evicts documents that are used the least often [12], [13].

The Most Recently Used Algorithm

The most recently used algorithm (MRU) does exactly opposite of what LRU does – it evicts the document that was used most recently. This algorithm is very effective in scenarios where access is unpredictable [12], [13].

MD5 Algorithm

MD5, developed by Ron Rives in 1992, is a comparison cryptographic hash algorithm that succeeded the MD4 algorithm. MD5 takes an input of any length and generates an MD5 digest of fixed length (128 bits or 32 characters). Because MD5 uses the same algorithm every time, a particular data string always generates the same MD5 hash every time [15].

MD5 cryptographic hash offers several advantages over its predecessors (such as MD4) and its competitors (such as, SHA and SHA-1). One of these advantages is that MD5 is a one-way cryptographic hash. Another advantage is that MD5 can accept inputs of any length but still generates a fixed-length output. MD5 is fast, and it is highly
unlikely that two different strings can hash to the same digest. Moreover, with MD5 it is also highly unlikely that two different input strings can hash to the same digest. Furthermore, MD5 is reliable in the sense that the same input string always yields the same output digest every time [15], [29].

The MD5 algorithm is comprised of the following steps:

1. Append padding bits: MD5 pads the b-bit input provided to it. It then adds 1 bit to the end of the input. Furthermore, bits are added until the message length is congruous to 448 modulo 512 [15].

2. Append length: MD5 appends the result from step 1 with 64-bit representation of the input. The output, denoted by $M$, at this step becomes a multiple of 512 bits [53].

3. Buffer initialization: MD5 uses four 32-bit registers (A, B, C, and D) in the derivation of the final 128-bit message digest. These four registers are initialized in the following manner and since bytes with the lower value are stored first the values in the register will look like below:

   $A = x”67452301”$
   $B = x”efcdaba89”$
   $C = x”98badcfe”$
   $D = x”10325476”$

4. Calculate digest: The algorithm divides message $M$ from step 2 into 512-bit blocks. The algorithm processes each 512-bit block separately. Let $X_i$ denote the $i^{th}$ block of $M$. The algorithm begins with the processing of the lowest 512-bits of $M$ i.e. $X_0$ followed by $X_1, X_2$, and so on, until all blocks of $M$ are processed. There are a total of 4 rounds, with 16
steps in each round. Therefore, MD5 takes 64 steps in all. The algorithm proceeds in the following manner:

Values of the 4 registers (A, B, C, and D) in step 3 are stored into temporary variables: AA, BB, CC and DD. The algorithm then performs the following operation on all the blocks:

\[
D = A + ((D + \text{Func} (A, B, C) + X_j[k] + T[i] \llq s) \\
D \leftarrow C, A \leftarrow D, B \leftarrow A, C \leftarrow B
\]

In these calculations, addition is additions modulo \(-2^{32}\). However, Func (A, B, C) is different in each round. In the first round, F (A, B, C) = \((0 \leq i \leq 15)\). In the second round, G (A, B, C) = \((16 \leq i \leq 31)\). In the third round, H (A, B, C) = \((32 \leq i \leq 47)\). Finally, in the fourth round, it is I (A,B,C) = \((48 \leq i \leq 63)\).

In these functions,

\[
F (A,B,C) = (A \land B) \lor (\lnot A \land C), \\
G (A, B, C) = (A \land C) \lor (B \land \lnot C), \\
H (A, B, C) = A \oplus B \oplus C, \text{ and} \\
I (A, B, C) = B \oplus (A \lor \lnot C), \text{ where}
\]

V is the bitwise-or operation,

\(\lnot\) is the bitwise complement,

\(\oplus\) is the exclusive–or operation,

\(\land\) is the bitwise-and operation.

Values of temporary variables (AA, BB, CC, DD) are added to (A, B, C, D), respectively; i.e.

\[
A = A + AA \\
B = B + BB
\]
\[ C = C + CC \]
\[ D = D + DD. \]

When MD5 finishes processing all the blocks, the message digest is available in the four registers A, B, C, and D. The first byte of the digest is the lower-order byte of D. The last byte is the higher-order byte of A [15], [29].

Decoding MD5 Hash

MD5 does not offer an efficient or easy method of getting the original string that was provided as an input from the digest that was computed for it. Brute force attack is one of the most effective methods of reverse engineering an MD5 digest. Using brute force attack means performing hundreds and hundreds of permutations and combinations is time consuming and processor intensive. Therefore, MD5 is considered to be quite safe from reverse engineering. MD5 is strong in the sense that there is a very small likelihood \((1/16^{32})\) that two different input values can evaluate to the same hash value [29], [50], [53].

Static Caching

Static caching, a new approach to web caching, uses yesterday’s log to predict today’s traffic. In contrast to dynamic caching, where cached documents are updated more than once a day, static caching is performed only once. Static caching is simpler to use, and inflicts very low CPU overhead. Moreover, it considerably improves cache performance by using cache compression techniques, meaning, that stable web pages can be compressed to reduce their size. This compression indirectly frees up space in the cache. Cache server performance typically relies on two major metrics: byte hit ratio and hit ratio. Hit ratio is the
percentage of all accesses that are fulfilled by the data in the cache; while, byte hit ratio is the hit rate with respect to the total number of bytes in the cache [16], [35], [48].

The main goals of static caching are to predict future traffic and to estimate the value of the document. The algorithm, based on server logs, predicts future documents using the fact-finding approach. While making predictions, the algorithm uses both cached and non-cached documents. Dynamic documents and large-sized documents cannot be cached because dynamic documents change constantly, and because the cost associated with caching these documents is high. Once the algorithm calculates future traffic, it estimates the value of each document. It then caches documents with higher estimated values. The most important goal of the algorithm is to give a high hit ratio.

Static caching algorithm predicts future traffic based on the belief that future will be most likely similar to the past. However, prediction times may vary, therefore, this algorithm assumes a 24-hour period. Once the algorithm is done predicting future traffic, it proceeds to estimate the value of the predicted documents. The value of the documents is the ratio of number of references to the document and the size of the document.

Given the background from the previous paragraphs, we can now discuss the specifics of the static caching algorithm. The static caching algorithm defines a fixed set of URLs by analyzing the logs of previous periods. It then calculates the value of the unique URL. Depending on the value, URLs are arranged in the descending order, and the URL with the highest value (on the top) is selected. This set of URLs is known as the working set. When a user requests a document, and the document is present in the working set, the request is fulfilled from the cache. Otherwise, the user request is fulfilled from the origin server [35], [45], [48].
Static caching algorithms are simple and easy to implement. The main disadvantage of static caching algorithms is that they can get costly when a new URL needs to be inserted into the existing URL set. Static caching algorithms cannot be used in situations where cache contents keep on changing constantly. In such a situation, static caching algorithms can be used along with other algorithms like LFU and LRU. Static caching helps reduce user response time as well as increase the efficiency of network bandwidth [35], [45], [48].
CHAPTER III

DESIGN AND IMPLEMENTATION

Current web caching algorithms suffer from the problems of inefficient use of network bandwidth, increased latency, and poor response time for users. In this chapter, we present our web caching algorithm that uses content aliasing to address all these problems. Our solution is made up of four components, and we discuss each of these components as well as a use case scenario in the upcoming sections. But first we introduce terminology and the software and hardware requirements of our solution.

Terminology

In this section, we introduce and explain terms that we use throughout the project. These terms include web caching, mirroring, content aliasing, and payload among several others.

- **Web caching:** Web caching essentially means storing frequently referred objects on a local server (also known as a caching server) in addition to storing them on the origin server [17].

- **Mirroring:** “Mirroring typically refers to a special case of aliasing in which replicas of pages or entire sites are deliberately made available through several different URLs” [18], [19].
Content Aliasing: Many web pages that belong to a particular institution or organization objects or content embedded in them that are common to several documents. By keeping track of such common content we can effectively reduce latency and enhance the overall usage of communication channels. [18], [19]

Payload: Payload is referred to as the data carried in the body of an HTTP response packet. In our solution, we consider only full-body responses. In context of server response, payload refers to the matter of interest: it could be the header information in message digests or the contents of a web page in case of a complete web page download. [18], [19]

Payload Hit: A cache reference that returns from cache storage, instead of the origin server, the exact payload that the origin server would return at the time of access. Payload hit always returns payloads from cache instead of the origin server regardless of whether or not messages are exchanged with the origin server. A revalidation request followed by a “not modified” response qualifies as a payload hit because it avoids payload transfer. [18], [19]

Payload Miss: “A reference for which the correct payload is not obtained from cache storage, but rather is obtained from elsewhere, including, the origin server or a remote intermediate cache, is referred to as payload miss” [18], [19].

Aliasing: “Aliasing occurs in Web transactions when requests containing different URLs elicit replies containing identical data payloads” [18], [19].

Aliased Payload: “In the context of a given trace of Web transactions, a payload is aliased if it is accessed via more than one URL in the entire trace” [18], [19].

Response Time: Response time is calculated as the time difference between the Transport Control Packet (TCP) packet containing the first byte of HTTP request and the TCP packet containing the last byte of the corresponding HTTP response. [21]

Time-to-live: Time-to-live (TTL) is defined as the difference between web object freshness lifetime and its age. TTL describes the validity of cached objects. Many cacheable objects are subject to changes during their lifetime. Once an object is modified, all copies cached elsewhere become stale unless these copies are updated. When the
object is retrieved from the origin server, TTL is initialized to a value, which reflects the maximum tolerance of discrepancy. This value decreases with time if the object is cached elsewhere. When the cached object is accessed, it is considered valid only if the remaining TTL is still positive, otherwise, the origin server validates the object [20].

- **Cache**: The term cache refers to an HTTP proxy that caches requests. A cache is a collection of duplicate original data which is stored elsewhere or is computed earlier in cases where the original data is expensive (usually in terms of access time) to fetch from the origin server. Once data is stored in cache, future requests for this data can be fulfilled from the cached copy rather than refetching it from the original web server. [21]

- **Object**: An object is a generic term that may represent a document, an image, or any other type of data available on the Web. URLs identify objects unambiguously, and can refer to data available from HTTP, File Transfer Protocol (FTP), Gopher, and other types of servers. Because, currently, much of Web content is comprised of images, audio, video and binary files, the term objects is preferred to the terms documents or pages. [22]

- **Cache Hit**: “A cache hit means a valid copy of the requested object exists in a cache” [23].

- **Cache Miss**: “A cache miss means that the object does not exist in the cache, or its cached copy is no longer valid” [23].

- **Origin Server**: “An origin server is the authoritative source for an object; in case of URLs, the origin server is the hostname part of the URL” [23].

- **Fresh, Stale, Refresh**: These terms refer to the status of cached objects. If an object is fresh then it will be returned as a cache hit. If an object is stale, then proxy servers, such as, Squid will refresh it by including an IP Multimedia Subsystem (IMS) request header and forwarding the request on toward the origin server. [23]

- **Proxy Server**: A proxy server is a computer network that acts as a mediator between the client and other network services. A client connects to a proxy server, and the proxy server connects to the server. The proxy server provides the resource either by connecting to the specified server or by serving it from a cache. In some cases, the proxy may alter the client's request or the server's response for various
purposes, such as reformating web pages for the cell phones, deploying proxies to intercept computer viruses and hostile content served from remote Web Pages. [24]

Hardware and Software Requirements

Our solution requires the computer processor to be at least a Pentium 4 1.7 GHz with 128MB primary memory and 20GB secondary memory. Our solution also requires basic I/O devices such as mouse, keyboard, and a monitor. We require that the computer system be equipped with a local area network (LAN) card or a modem, with a CAT 5 UTP cable. Our solution can run either in Windows or Linux environment, and requires the 1.5 version of the Java Development Kit (JDK).

Our Solution

Our solution, that optimizes network bandwidth utilization, reduces network latency, and improves user response time, is comprised of four components: the graphical user interface component, the caching component, and the page replacement component and MDH Hashing component. Figure 5, our solution’s class diagram, shows the relationship among the four components of our solution, and the functions associated with these components. The user, through the request() function, requests a page from the browser. The Browser performs fetchpage() to get the page requested by the user from the server or cache. The Browser checks the availability of the page in cache with the check cache() function. If the page is available in the cache, then the browser displays the page; and if the page is not available in the cache the browser requests the page from the server and displays the page using the result() function. The cache then stores the page, depending on whether or not space is available in the
cache. If sufficient storage space is not available, then our solution uses the
replacement() function to evict one or more pages that are no longer needed, and
saves the new page.

In the collaboration diagram of our solution, as shown in Figure 6, the user
requests a URL from the browser. The browser, in turn, requests the page from the cache.
The cache checks the availability of the requested page. If the page is available in the
Fig. 6. Collaboration diagram of our solution.

cache (a cache HIT), the page is returned to the browser, and the browser displays the requested page for the user. However, if the requested page is not available in cache (a cache MISS), the user request is forwarded to the server. The server returns the page to the browser, and the browser displays the requested page to the user. After fulfilling user request, the page is saved in the cache memory if sufficient space is available in memory. If cache lacks the storage space to store this page, then, the page replacement module is called upon to evict another page and to make space for this page to be stored. The
browser requests for the analyst component for analysis, the analyst component returns the results.

Graphical User Interface Component

Our solutions’ graphical user interface (GUI) component is a customized browser, as shown in Figure 7. It will provide all the functionality that a normal browser provides. It contains an address bar, and can support ActiveX objects. When a user pushes the “GO” button, the browser is expected to establish a TCP/IP connection with the server. Figure 8 demonstrates our graphical user interface with a cache hit, and Figure 9 with a cache miss.

Fig. 7. Graphical user interface (GUI) component of our solution.
Caching Component

Caching is a vital component of our solution. Our solution creates a cache memory. This cache memory, represented as a folder on the file system, acts as a local server, and stores cached web pages. Caching, according to our solution, can be done in two ways: in main memory or on hard disk. Caching into main memory falls into the category of temporary caching, while caching on the hard disk is more permanent in nature. Pages cached temporarily in main memory are lost when the user closes the browser. On the other hand, pages cached on the user’s hard disk are lost only when the user deletes them.
Page Replacement Component

In our solution, we use the MRU algorithm to evict objects from cache to make space for more objects. MRU is one of the most efficient algorithms.

Hash Component

MD5 hashing plays a very crucial role in the implementation of our solution. We use MD5 to generate 32-bit hash value of a web page URL. We call this hash of a web page URL the name key. We also hash the contents of a web page, which we call the content key. When a page is requested, we compare the name as well as the content of the requested URL. This is very useful in case of mirrored sites in comparing both the content and URL so that duplicate contents are not cached. Since in mirrored sites, although the URL’s are different, the contents are more or less the same like www.yahoo.com, www.yahoo.co.in, and so on, as we discussed in Chapter I.
Algorithm for Content Aliasing Avoidance

To implement content aliasing, we first get the URL of the page requested by the user. We retrieve this through the I/O module. We then send this URL to the hash module so that we can calculate the name key and the content key of the URL requested by the user. In the next step, we look up in the hash table to check for the presence of the name key we generated in the previous step. These steps are illustrated in Figure 10. If this lookup returns a positive result, then we know that the object requested by the user already exists in the cache. In this case when the object already exists in the cache, we first get the entry associated with this name key. Our second step is to also retrieve the expiration time, time-stamp, associated with this entry. We then check whether the page is expired. If the page is expired then we send an IMS request to the origin server to check if the contents of the object have been refreshed or modified since the last time the page was cached. We then check whether the page is modified or not.

If the requested object has indeed been modified since it was cached, the origin server replies with a “get refreshed page” response. However, if the object has not been modified since the last time it was cached, then the origin server responds with a “not modified” reply. When the origin server responds with “get refreshed page,” a new object key is created and saved. If the origin server responds with “not modified” response, then the page is served from the cache itself.

If this lookup returns a negative result, then we know that the object requested by the user does not exist in the cache. The request for the object is then made to the origin server, using the MD5 hash component, a name key and a content key is generated.
Fig. 10. Flowchart of our solution.
This entry is then stored in a hash table. A stored entry, associated with the newly-created entry, is created. An object key is then created to store the object on the disk. We check, subsequently, whether or not the disk location corresponding to the object key created above is preoccupied. If the location is indeed preoccupied, then, using the page replacement component, we either discard or overwrite a previous page; and store the new page, while fulfilling the user’s request at the same time.

Use Case Scenario

Figure 11 presents a use case scenario of our solution. First, a user requests a web page by providing a URL to the browser. The browser, in turn, forwards the request to the cache. The cache checks for the availability of the page. If the page is available (in case of cache HIT), the cache returns the page to the browser, and the browser returns the page to the user, thereby, fulfilling the user’s request.

If, however, the requested page is not available in the cache (in case of a cache MISS), the request is forwarded to the server. The server returns the page to the browser, and the browser displays the page as requested by the user. The requested page is then saved in cache memory so that next time when the user requests this page, it is fulfilled from the cache itself. If sufficient memory is not available in cache, then the page replacement component is invoked. Additionally, when the browser requests for analysis to the analyst, then the analyst returns the results.

Activity Diagram

Figure 12 presents an activity diagram corresponding to our solution. When a user requests a URL through the browser, the browser requests the page from the cache.
Fig. 11. Use case scenario of our solution.

If the page is available (i.e., a cache HIT), the page is returned to the browser, and the browser displays the page. If the page is not available in cache (i.e., a cache MISS), the request is send to the server. The server returns the page to the browser, and the browser displays the page to fulfill user request. The page is then saved in cache.
memory if sufficient storage space is available. If the cache does not have sufficient space available, then using the page replacement component, we make space for the page in cache. The browser requests for analysis from the analysts, and the analysts return the results.
Sequence Diagram

We present a sequence diagram of our solution in Figure 13. When a user requests a URL through the browser, the browser requests the page from the cache. If the page is requested available (i.e. cache HIT) the page is returned to the browser, and the browser displays the page. If the page is not available in cache (i.e., a cache MISS), the request is send to the server. The server returns the page to the browser, and the browser displays the page to fulfill user request. The page is then saved in cache memory if sufficient storage space is available. If the cache does not have sufficient space available, then, using the page replacement module, we make space for the page in cache. The browser requests for analysis from the analysts, and the analysts return the results.

Deployment Diagram

We present the interactions among the three components of our solution in a deployment diagram, as shown in Figure 14. The first component is a client whose internal component is a web browser. The second component is the cache, which is a database that stores cached web pages. The third is an application server which is also a web browser.

Results of Statistical Analysis

In this section, we present results of statistics analysis, as shown in Figure 15, 16, and 17. Fig. 15 shows the MD5 statistics for the sites visited by the user. Figure 16 represents the HTML code of the visited site. In this figure, for example, the user visits www.google.com. Figure 17 shows similar statistics as shown in Figure 16, but for the site www.cricinfo.com.
Fig. 13. Sequence diagram of our solution.
Fig. 14. Deployment diagram of our solution.
Fig. 15. MD5 and MRU statistics for sites visited by a user.

Fig. 16. HTML code of the site (www.google.com) visited by a user.
Fig. 17. HTML code of the site (www.cricinfo.com) visited by a user.
CHAPTER IV

TESTING

Testing is a must for any software development project because it discovers errors and defects in software that cannot be otherwise discovered due to limitations such as human skills, limited dataset, or shortage of time. We test our solution thoroughly to ensure that it is correct, complete, and error-free. Testing helps determine the quality of our solution. Our object is to uncover hidden errors, and to develop a solution that is largely devoid of flaws and errors. We used several techniques, such as, unit testing, alpha testing, beta testing, white box testing, black box testing, smoke testing, sanity testing, integration testing, and performance testing, to test our solution.

Test Plan and Results

A comprehensive test plan describes in detail the steps involved in software testing, including the features and functions that need to be tested, and the criteria for success and failure. A good test plan also describes testing schedule and the resources needed for testing. It is important to establish a good acceptance criterion for each function and feature that is tested. A good acceptance criterion addresses a number of areas such as the function and performance of the system, stress testing the system, testing the system for usability, as well as the security of the software [25].
Our test plan consists of various types of testing, including, unit testing, alpha testing, beta testing, white box testing, black box testing, smoke testing, sanity testing, integration testing, and performance testing. We expose each module of our software to several types of testing, and report our results here. Unit testing focuses on the smallest unit of software: a module. It tests module interfaces to ensure that information flows correctly in and out of a module [26].

Integration testing ensures that the software works correctly even after the various unit-tested modules are integrated together to perform more complex tasks, such as, hashing of URL and MRU Replacement component. We used integration testing to test compatibility between different modules like replacement, GUI, and MD5 Hashing. White box testing is performed to ensure that all executable paths are properly tested and that execution paths are taken based on the evaluation of logical decisions that yield Boolean results. White box testing tests all loops and their boundaries as well as verifies the validity of internal data structures [27].

Black box testing focuses on verifying the functional requirements of the software being tested. In black box testing, testers provide various inputs to the software and ensure that expected results are obtained. Usually, they check for interface errors, incorrect or missing function, performance errors, and initialization and termination conditions [28]. We performed black box testing on all the modules of our project.

Parameter testing involves testing for various system parameters, such as, performance, failover recovery, and system security [28]. We performed parameter testing on the GUI component of our solution. Alpha testing is performed in a controlled
environment by end users or customers at the developer’s site [29]. Beta testing is also performed by end users or customers, but not in a controlled environment [30].

We first tested the methods individually. During this individual testing, we found a few syntax errors in the GUI component. We fixed these errors and proceeded to verify whether or not MD5 was generating a unique hash value for each visited page. During this verification, we found out that MD5 does indeed generate a unique MD5 hash of the URL of each visited page.
CHAPTER V

FUTURE SCOPE AND CONCLUSIONS

Web caching has widely been used to improve user response time without increasing cost. As more and more web pages furnish dynamic content, web-caching at the proxy level is becoming less effective. We can reduce processing delays by using cache management policies that place database query results on the server side for multiple page requests in the cache. There are two cache management policies to determine which set of query results to cache, and how frequently the cache content be refreshed. The first policy shows how the intrinsic value of page fragments, its popularity, its size and the staleness of page fragments, and size affects caching decisions. Additionally, when whole cache replacement incurs high overheads, it may be preferable to relax refresh intervals while controlling content staleness by invalidating individual fragments. The second policy to ensure fragment invalidation does not cause cache choking. Numerical results based on various fragment value, request rate, and query processing time distributions show that both policies are robust under a wide range of operating conditions and could significantly improve server performance.

We propose a solution that ensures that fragment invalidation does not cause cache choking. Moreover, our solution improves memory usage in caches, reduces network latency, increases network bandwidth utilization, and reduces user response time. Our solution implements predictive caching as well as content aliasing avoidance in
real time application of caches. Numerical results based on various fragment values, request rates, and query processing time distributions show that both policies are robust under a wide range of operating conditions and could significantly improve server performance. Our solution, although generic in its approach, provides a unique web-based concept, which is a stand-alone application. In other words, our solution can not only be used in web caching proxy, but also is useful in any sort of generic caching, including email servers, where bulk mails put excess load on the servers. ConAA, a standalone application, can be used to maintain only one copy of bulk mail, and keep rest of the links to the same content; thereby, heavily saving server space and reducing overhead at the cost of little increase in user response time for searching.

ConAA is UNIX platform dependent because it uses system calls from php scripts. Therefore, it cannot be ported directly to a Windows’ machine. We did not enable handling of remote directories in this version: we leave this for future work. Therefore, in the current version of our solution, the identification and removal of aliasing is confined to the current working directory. When ConAA is extended to work on remote directories and sub-directories, users will be able to use it as a script to periodically check for and to remove content aliasing.

Additionally, we use the graphical analyzer (GrAn) that helps administrators analyze access log files. GrAn, therefore, helps administrators implement an optimal replacement policy according to the requirements of their organization. 'In this version, we retain the least recently created file and the rest of the copies are linked to it. It can be extended to give a graphical representation of the TTL’s and frequencies of hits of cached pages. The links that are used now are soft links. That is, once the target file is removed,
the links become orphans. Use of hard links could also be tried as an alternative and later given as an option for the administrator.

Our solution is flexible enough to use other mapping protocols such as most recently used, least recently modified, among several others. A standalone script could also be used to retrieve URL names from file content, so that it is more intuitive to understand what the content stands for. This is contrast to using Squid’s way of naming files, which is purely numbers.

Future Advancement

The topic we addressed here is open to research and can be advanced to a great extent for providing various benefits to internet users. Our solution presented in this project can be enhanced by the following:

- **Dynamic Pages**: The application can also be used for dynamic pages, which change frequently.

- **Mobile WAP**: Our application is only limited to network-connected computer systems. It can be further enhanced for mobile web application programs as well.

- **Reducing Costs**: Costs can be further reduced by implementing our solution on a large scale.

- **Making it Compatible for other Platforms**: The application is currently not compatible with Mac. In the future, it can be enhanced to work on Macs.

- **Add More Features for Statistical Analysis**: Analyses of traces from deployed proxy cache servers demonstrates that a round-trip to a remote server is the
dominant factor for response time. In the future, more such statistical analysis can be deployed to further enhance our solution.

Conclusion

The need for higher bandwidth and greater connectivity speed is becoming a necessity. Here, we have shown how web caching improves web performance in terms of reduced traffic, reduced latency, improved user response time, and optimal use of the existing bandwidth. Our idea is to implement our solution at much higher levels, so that there can be a central proxy-caching server with enormous capacity in terms of memory and storage space. This allows the top level parent could be connected to lower level proxy servers that are located in different regions; where clients can connect to local proxy servers.

We have successfully detected content aliasing using a web-based application, database queries and files system calls. We were able to remove content aliasing for the same too. We built a standalone application, which worked on URLs in the current working directory. We avoid a considerable amount of duplicate storage through our solution. It is, therefore, a very useful tool for web proxy caches. It can also be used to monitor and analyze memory usage pattern. Moreover, our solution is successfully able to keep cached pages in synchronization with the pages on the web server, checking for new pages if needed.
REFERENCES
# REFERENCES


