CLIENT-SIDE PAGE ELEMENT WEB CACHING

A Writing Project
Presented to
The Faculty of the Department of Computer Science
San José State University

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

by
Ramya Karri
December 2008

ABSTRACT

CLIENT-SIDE PAGE ELEMENT WEB CACHING

by Ramya Karri

When a user explores different web pages in a given website, the website typically sends the entire requested page even if only a portion of the page was different to the current page. That is, two pages on a given website might share elements on the page like search bar etc., but this information is retransmitted. Most of the end-user response time is spent on the front-end while downloading all the components in the page such as images, style-sheets, scripts, and Flash. The aim of my project is to reduce the same using a new form of web page caching at the client-side, that caches the most common parts of the web pages in the website, such as logo, images, and search tab, and reuses them in the further web pages reducing the transmission data. To explore the advantage of having client-side caching and determine the effect on the response time I made the server as fast as possible using Squids: optimizing web delivery, front-end accelerator for a web server, to enhance the traffic from inside network to the internet. I also performed some benchmarks to measure the reduction in the end user response time.

Table of Contents

1. Introduction 1
2. Deliverable 1: Course Listing Website 4
3. Deliverable 2: Load Balancing using Squid 10
4. Deliverable 3: Determine the Advantage of Caching Text 18
5. Deliverable 4: Observe Firefox Caching Mechanism 23
6. Summary 27
7. Future Work 29
References 30

List of Figures

Figure 1: Course Listing Home Page 5
Figure 2: Course Listings Page 5
Figure 3: Edit Zoom 6
Figure 4: Edit Course Details Page 6
Figure 5: Success page 6
Figure 6: Add Year Page 7
Figure 7: Year Present Message 7
Figure 8: Directory Structure 8
Figure 9: Directory Structure of Squid 12
Figure 10: Squid Running 13
Figure 11: Configure Browser with Squid 14
Figure 12: Yslow installed 19
Figure 13: Directory Structure of Apache 20
Figure 14: Find Total Download Time 20
Figure 15: Caching experiment Result Graph 21
Figure 16: Yslow installed 24
Figure 17: Apache Directoty Structure 24

1. Introduction

This report provides comprehensive details on the work done in CS297, Preparation for Writing Project or Thesis class, for the project titled, "Client-side Page Element Web Caching." The purpose is to prepare a written documentation of all the work done in this project.

Due to tremendous increase in the use of internet, there is a lot of ongoing research in the field of web page caching both at the server and client-side. Even then, 80% of end-user response time is spent on the front-end while downloading all the components in the page such as images, style sheets, scripts, Flash, etc ("Best Practices for Speeding Up Your Web Site," 2008). When the user explores web pages of the same website, even when the pages have common portions with the current page, the website transmits the entire requested page to the user.

While I was doing my research on caching, I found a patent (Hawes, 2000) that describes an apparatus and methodology for loading web pages based on caching and non-cacheable portions. After discussing this idea with my advisor, Dr. Chris Pollett, we came up with an idea to cache the most common parts and reuse them in the future requests.

In this project, I am working on inventing an apparatus and methodology for a new form of web page caching at the client-side. When user explorers a website the common portions of a website are cached so as to reuse them in the future requests. The motive behind is to reduce the response time at least by 20%. In order to explore the advantage of having client-side caching and determine the effect on the response time, I

considered Squid to enhance the traffic from local area network to the internet and make the server as fast as possible.

I realized that the CS297 project is a good opportunity to explore the various technologies. Therefore, I categorized my work into four deliverables: (1) Learn PHP and CakePHP, (2) Learn Squid, (3) Determine the Advantage of Caching Text, and (4) Observe Firefox Caching Mechanism.

Learn PHP and CakePHP mainly involves exploring the features of PHP and CakePHP; to get hands-on experience with the technologies, I created a Course listing website that displays the courses offered by the computer science department in a particular semester and year. Deliverable on Squid includes understanding Squid and configuring to balance the load between two servers. Deliverable 3, "Determine the Advantage of Caching Text," mainly involves conducting experiments using web pages to understand the advantage of caching text and determine how much reduction in the response time can be obtained. The primary focus of deliverable 4, "Observe Firefox Caching Mechanism," is to understand the Firefox caching technique.

The report is mainly divided into seven sections: Introduction, Deliverable 1: Course Listing Website, Deliverable 2: Load Balancing using Squids, Deliverable 3: Determine the Advantage of Caching Text , Deliverable 4: Observe Firefox Caching Mechanism , Summary, and Future Work. The Introduction mainly addresses the purpose, plan followed, and scope of this report. It also describes the project motivation and gives an overview of the project. Deliverable 1 to Deliverable 4 are further divided into four subsections. Each subsection describes the Motivation, Goal, Implementation and Results, and Remarks of the deliverable. The Summary recapitulates all the work

done in the project and finally the Future Work outlines the work to be done in spring 2009.

In short, the report gives a clear idea of the project, the motivation behind it, detailed description of the deliverables produced, and the findings of the experiments. The plan followed to write this report is to describe each and every step followed in accomplishing the project and provide the audience with necessary details to conduct the experiments themselves.

2. Deliverable 1: Course Listing Website

Motivation

The main objective of this deliverable is to learn PHP and CakePHP. PHP is a widely-used general-purpose scripting language that is specially suited for Web development and can be embedded into HTML ("PHP," 2008). Learning PHP helps writing scripts for caching sections of web pages at the client-side. CakePHP is a rapid development framework for PHP and provides an extensible architecture for developing, maintaining, and deploying applications ("CakePHP,"2008). CakePHP offers a model view controller platform wherein the pages of the application can be organized in very systematic manner. To get hands-on experience of the two technologies, the deliverable scope was extended to creating a web application.

Goal

The goal of developing the Course Listing website is to incorporate the basic database operations such as insert and update. It is a collection of web pages, which helps in maintaining the computer science department course information and displays the list of courses offered in a particular year and semester.

Implementation and Results

The website consists of three web pages: Home Page, Edit Course Details Page, and Add Year Page. The methodology followed is agile software development, where things are done in small increments with minimal planning rather than long-term planning. The softwares needed to develop the application are Apache HTTP Server, PHP, CakePHP, and MySQL Database.

Following is the Home Page,

Figure 1: Course Listing Home Page

It has a combo box where the user can select the year and semester, on clicking the Show courses button, the courses offered by the computer science department in the selected year and semester are displayed. Following is the screen that displays the course listing in fall 2008,

Figure 2: Course Listings Page

The edit option in the Course Listing Page can be used to edit the course details.

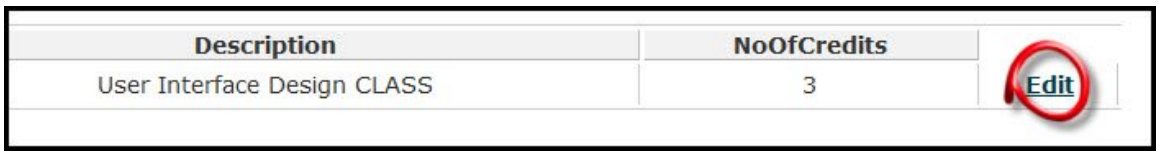

Figure 3: Edit Zoom

Following is the Edit Course Details Page.

Figure 4: Edit Course Details Page

On clicking the Edit class details button, the course details get updated in the database and the page gets redirected to the main page with a success message as shown in the below snapshot,

Figure 5: Success page

In addition to the above pages there is an Add Year Page, where the user can add a new year to the year table in the database.

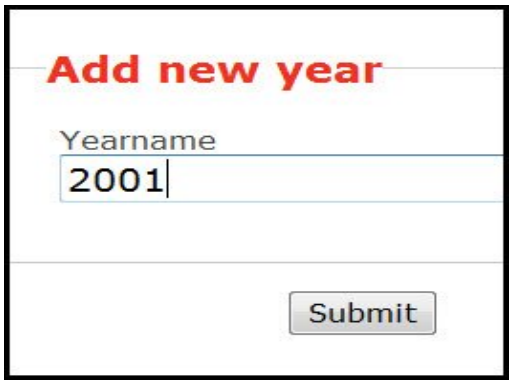

Figure 6: Add Year Page

If the user attempts to add an existing year, following message will be displayed

Figure 7: Year Present Message

Following tasks were completed to create the website in CakePHP,

1. Installed Apache HTTP Server2.2
2. Installed PHP 4.4.9
3. Installed CakePHP 1.1.19.6305
4. Installed MySQL database
5. Configured Apache and PHP, Apache and CakePHP, CakePHP and MySQL
6. Created a test folder (My Project Folder) in the webroot folder of CakePHP

The Directory Structure can be viewed as follows

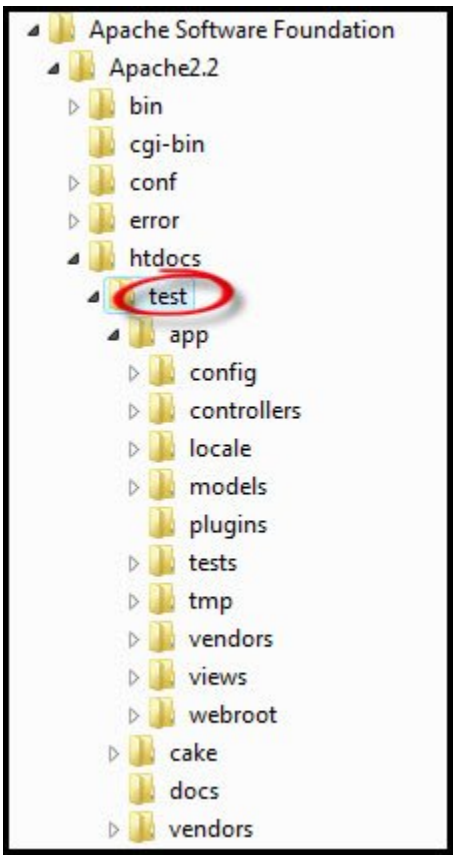

Figure 8: Directory Structure

7. Created two tables, years and course information.
8. Created models for the tables.
9. Created a controller for managing the class details. It consists of the following functions, view : For the main page, gotcoursedetails : To get the information about the courses, and edit : To handle the Edit Course Details page request.
10. Created a year controller for handling the Add Year request.
11. Created views to display the data.

Remarks

I conducted several manual tests to make sure that website is working properly. I found CakePHP to be a very helpful platform. Usually when developers create logic using a programming language like Java, they know which files belong to the view, model, and controller, but all the files are put together under a single folder, whereas in CakePHP there are corresponding folders for the model, view, and controller. So when a change is required, developers can go to the respective folders instead of searching in a single folder.

3. Deliverable 2: Load Balancing using Squid

Motivation

To explore the advantage of having cache at the client, I considered making the server as fast as possible using Squid: optimizing web delivery, front-end accelerator for a web server, to enhance the traffic from local area network to the Internet. A proxy server is a server that services the requests of its clients by forwarding requests to other servers ("Proxy Server,"2008). Some proxy servers store the most commonly requested web pages in their cache, called Caching proxy server; when user requests a web page, the proxy servers serve the request from their cache instead of forwarding them to the original server.

Squid is a caching proxy server used by hundreds of internet providers world-wide to provide their users with the best possible web access. It optimizes the data flow between client and server to improve performance and caches frequently-used content to save bandwidth ("Squid: Optimizing Web Delivery," 2008).

Goal

The main goal of the deliverable is to learn Squid and configure it to balance load between two servers. Squid support only round robin balancing i.e. pages are served in a round robin fashion between the two servers.

Implementation and Results

The softwares needed to conduct this experiment are Apache HTTP Server installed in both the servers, SquidNT, and Browser ex: Firefox or Internet Explorer

configured to work with Squid proxy server. Following have been done to perform the load balancing,

1. Installed SquidNT on Windows XP laptop (client)
2. Configured Internet Explorer to work with Squid proxy server
3. Setup a LAN at home with two servers (one machine is Windows XP and other Windows Vista Laptop) and a client
4. Stored ten html pages in each of the servers
5. Configured SquidNT to perform load balancing between the two servers
6. Tested the load balancing by requesting pages from the client browser

A detailed explanation of the above steps is given below.

Following steps have to be followed to install SquidNT on laptop having Windows XP operating system,

1. Download zip version of SquidNT from Squid official website
2. Unzip the Squid zip file to C:\ drive
3. Open a command line window (cmd.exe), and change to the directory you installed Squid to. E.g. cd \squid
4. Install the Squid service by running the following: C:\squid>sbin\squid.exe
5. Setup the default config files by copying the template configuration files in C:\squid. Copy the following three files to C:\squid\etc. squid.conf.default to C:\squid\etc\squid.conf mime.conf.default to C:\squid\etc\mime.conf cachemgr.conf.default to C:\squid\etc\squid.conf

6. create the Squid cache directories by running the following command,

   C:\squid>sbin\squid-z

The Directory Structure can be viewed as follows

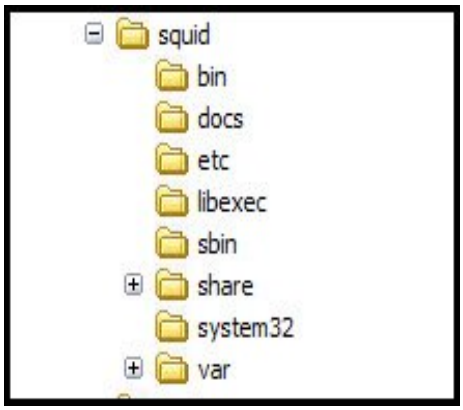

Figure 9: Directory Structure of Squid

The following screen illustrates that SquidNT is running.

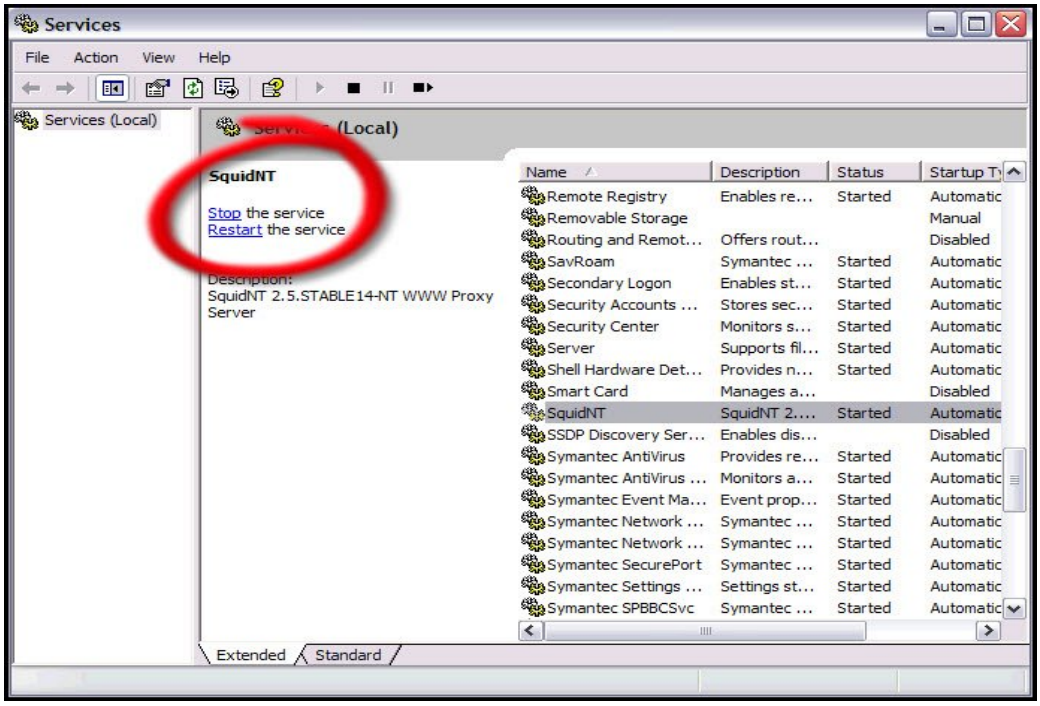

Figure 10: Squid Running

Squid NT service can be started from the Services Control Panel applet (Control Panel >Administrative Tools >Services).

The browser can be configured with Squid as follows, Open Internet Explorer > Tools > Internet Options > Connections > LAN Settings > Check the box for Use proxy server for your LAN > Enter localhost for Address and 3128 for Port (default port on which Squid runs).

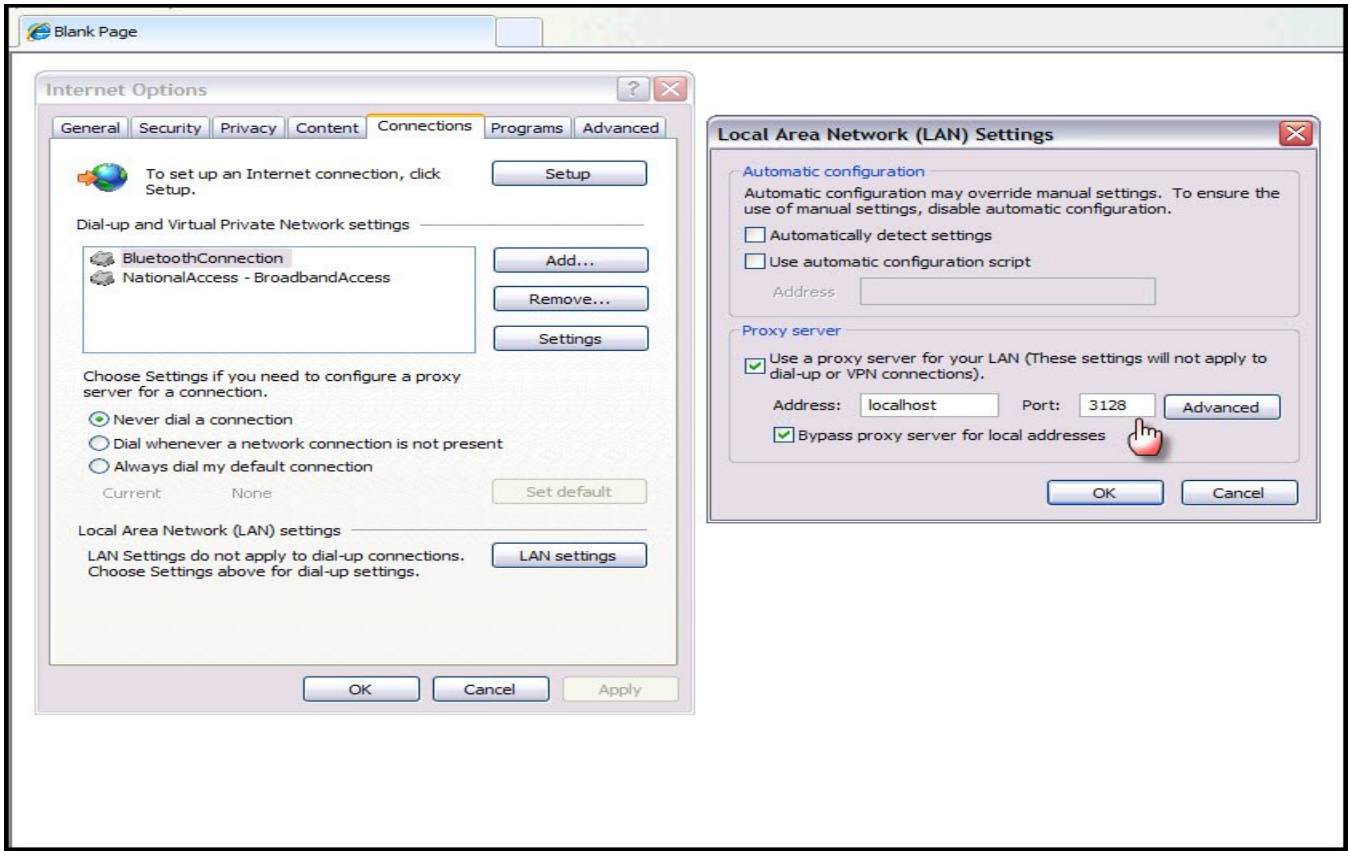

Figure 11: Configure Browser with Squid

Try to ping the systems from other systems to verify that systems are in LAN. If the other system is not responding to the ping, turn off the windows or Norton firewall in that system and try again.

In order to configure Squid to balance load between two servers, modify the Squid.conf file and add additional lines as follows:

cache_peer ip.of.server1 parent 80 0 no-query round-robin

acl sites_server_1 dstdomain www.mysite.com

http_access allow sites_server_1

cache_peer ip.of.server2 parent 80 0 no-query round-robin

acl sites_server_2 dstdomain www.mysite.com

http_access allow sites_server_2

These lines specify that the requests are sent to the two servers in round robin fashion. Before requesting pages from the servers, make sure that Apache is running on both the servers and Squid is running on the client.

Results can be seen by opening C:\squid\var\log\access.log, Squid has successfully done the load balancing between the two servers using round robin algorithm. Consider the following lines, taken from access.log. These lines indicate a request to the server for 9605.html

TCP_MISS/200 18847 GET http://www.mysite.com/9605.html - NONE/- text/html
TCP_MISS/404 471 GET http://www.mysite.com/useit_style.css - ROUNDROBIN_PARENT/ip.of.server1 text/html
TCP_MISS/404 469 GET http://www.mysite.com/images/arrow_yellow.gif - ROUNDROBIN_PARENT/ip.of.server2 text/html
TCP_MISS/404 479 GET http://www.mysite.com/20021223_02_mistake.gif - ROUNDROBIN_PARENT/ip.of.server1 text/html
TCP_MISS/404 469 GET http://www.mysite.com/20021223_05_mistake.gif - ROUNDROBIN_PARENT/ip.of.server2 text/html
TCP_MISS/404 479 GET http://www.mysite.com/20021223_01_mistake.gif - ROUNDROBIN_PARENT/ip.of.server1text/html

Following are some of the observations

1. Squid has successfully balanced the load between the servers even when it tried to serve a request for a single web page

2. Squid returns a TCP_MISS/404 error, when it accesses an images or css files not stored on the server
3. Once Squid gets the requested page from the server, it stores it in its cache and the future requests of the same page are served from its cache

   Following lines show that the request for page 9605.html is served from its cache as we got a TCP_HIT,

   TCP_HIT/200 18855 GET http://www.mysite.com/9605.html - NONE/- text/html

   TCP_NEGATIVE_HIT/404 479 GET http://www.mysite.com/useit_style.css - NONE/- text/html

   TCP_NEGATIVE_HIT/404 477 GET

   http://www.mysite.com/images/arrow_yellow.gif - NONE/- text/html

   TCP_NEGATIVE_HIT/404 487 GET

   http://www.mysite.com/20021223_02_mistake.gif - NONE/- text/html

   TCP_NEGATIVE_HIT/404 477 GET

   http://www.mysite.com/20021223_05_mistake.gif - NONE/- text/html

   TCP_NEGATIVE_HIT/404 487 GET

   http://www.mysite.com/20021223_01_mistake.gif - NONE/- text/html
4. Squid returns a TCP_NEGATIVE_HIT/404 error, when it tries to retrieve an image/css/page that gave a TCP_MISS/404 on request from the server
5. If the image or css is present in the server, then Squid returns a TCP_IMS_HIT for the page

   This can be seen in the below example

   TCP_MISS/200 433 GET http://www.mysite.com/Hello.html -

ROUNDROBIN_PARENT/ip.of.server1 text/html

TCP_MISS/200 2756 GET http://www.mysite.com/apache_pb22.gif - NONE/-image/gif

TCP_IMS_HIT/304 238 GET http://www.mysite.com/Hello.html - NONE/-text/html

Squid has cached both the text and image, therefore it returned a TCP_IMS_HIT, as the copy exists in its cache.

Remarks

Squid is a very useful caching proxy server. It successfully balanced the load between the two servers. It can be used in major companies to cache the most frequently visited pages to serve the requests faster and can also be configured to prevent employees from visiting few websites.

4. Deliverable 3: Determine the Advantage of Caching Text

Motivation

The main objective of this deliverable is to determine the advantage of caching text. The very idea of this project is to cache sections of page and reduce the total download time. Before I start working on the apparatus of the new form of web caching, I considered doing few experiments to understand how much gain in download time can be obtained.

Goal

The main goal of this deliverable is to conduct an experiment using web pages; few pages having images and few only with text. The total download time of all the pages will be recorded and the values will be compared for pages with images versus those without images.

Implementation and Results

In order to accomplish the goal, I manually created 200 web pages, 100 pages with an image and 100 pages with only text. The 100 pages with image have been classified as set of ten pages each. All pages with image are 100 Kbytes in size; each set consists of same image, but images sizes from set 1 to set 10 ranges from 10k to 100kbytes. The 100 pages with only text have been classified as set of ten pages with size ranging from 110k to 200kbytes. All the pages belonging to the same set have different text.

The softwares needed to conduct this experiment are

1. Firebug
2. Yslow
3. Apache server

Following are the steps followed to accomplish the goal.

1. Installed Firebug from https://addons.mozilla.org/en-US/Firefox/addon/1843
2. Install Yslow, after installing Firebug, from http://developer.yahoo.com/Yslow/
3. Verified that Yslow and Firebug got installed properly.

   If properly installed at the bottom of the Firefox browser we can see the icons as shown below

Figure 12: Yslow installed

4. Installed Apache Http Server
5. Gathered the information necessary from the internet and created the web pages necessary for the experiment
6. Stored all the pages in the htdocs folder of Apache server

   The directory structure of Apache is shown below

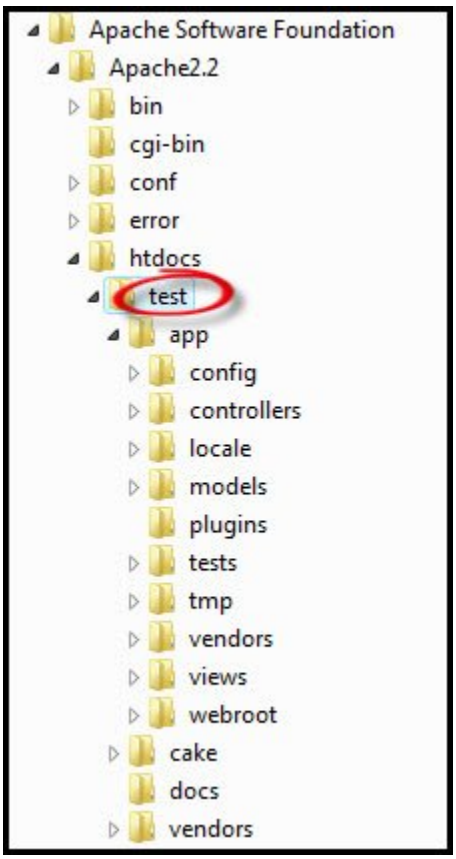

Figure 13: Directory Structure of Apache

7. Request pages from the browser and record the total download time using Yslow.

   The values can be found as shown below,

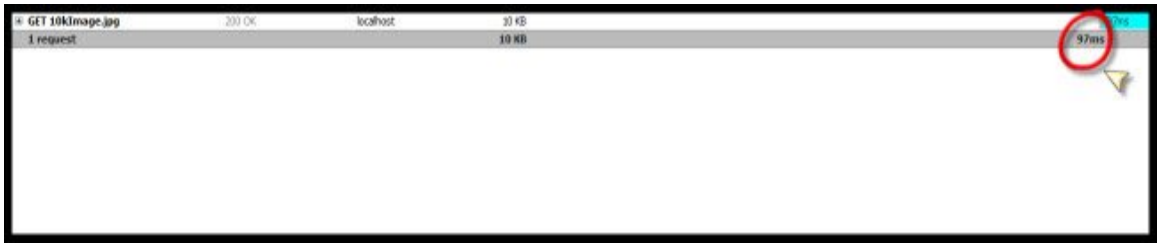

Figure 14: Find Total Download Time

8. Found the total download time for each set and calculated the mean

9. Conducted the same experiment making a system in United States as server and system in India as client. Recorded the total download time for each set.
10. Created a graph (shown below) with values obtained from experiment conducted only in Unites States.

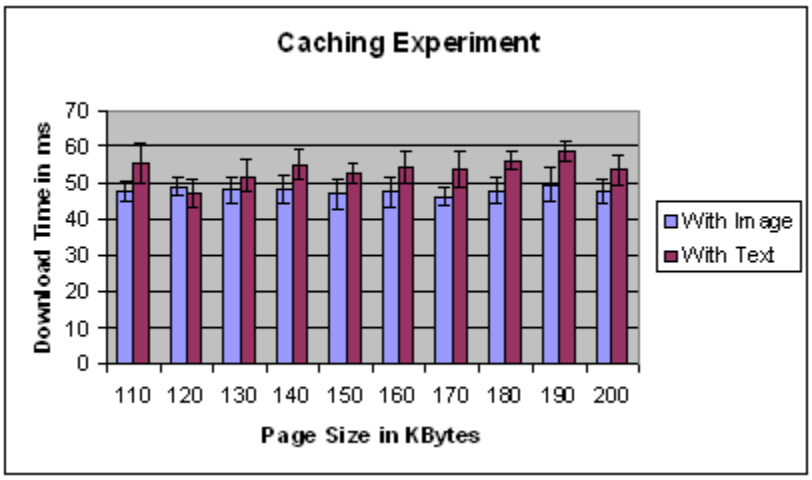

Figure 15: Caching experiment Result Graph

Following are results based on the graphs obtained in the experiment.

1. Caching the image did not have much profit when the same system is used as server and client.
2. Caching image versus using as text has a little advantage but is very negligible, as seen in the graph.
3. For the experiment conducted with client in India, there has been a large variation in the values produced; this might be due to the route the web page takes or the number of hops it has to pass to reach the destination.

Remarks

Doing this experiment, gave me a lot of insight into how much download time is taken in downloading the components of web pages. While recording the values in the experiment, I encountered few outliers i.e. few pages took more than three times the acceptable download time. Therefore I have removed the outliers, requested the pages again, and considered only those values which are in acceptable range.

I got an opportunity to learn a great tool, called Yslow. It is a very powerful Firefox plug-in that analyzes web pages. It has several views that give a detailed view of all the components in the page. If a page can be improved, Yslow also lists the specific changes to be made. Using Yslow has not only simplified but also reduced the time taken for recording the experiment results.

5. Deliverable 4: Observe Firefox Caching Mechanism

Motivation

The aim objective of this deliverable is to learn the caching techniques of Firefox. In order to write scripts that cache sections of web page, it must be supported by the current browsers such as Firefox and internet explorer. I considered researching about the already existing caching mechanisms supported by both the browsers.

Goal

The goal of this deliverable is to explore the various Firefox caching options. While exploring the various caching techniques, I learned that Firefox caches the source of the image tag, so I wanted to test if Firefox is capable of caching text, when stored as an image.

Implementation and Results

In order to accomplish the goal I divided the deliverable into two parts. First part consists of experimenting how Firefox debugs the image tag and second part works with script tag. The softwares needed for the experiment are:

1. Apache HTTP Server
2. Firebug
3. Yslow

Below are the steps followed to setup the software needed for the experiment:

1. Installed Firebug from https://addons.mozilla.org/en-US/Firefox/addon/1843
2. Install Yslow, after installing Firebug, from http://developer.yahoo.com/Yslow/

3. Verified that Yslow and Firebug got installed properly.

   If properly installed at the bottom of the Firefox browser we can see the icons as shown below

Figure 16: Yslow installed

4. Installed Apache Http Server

   The directory structure of Apache is shown below

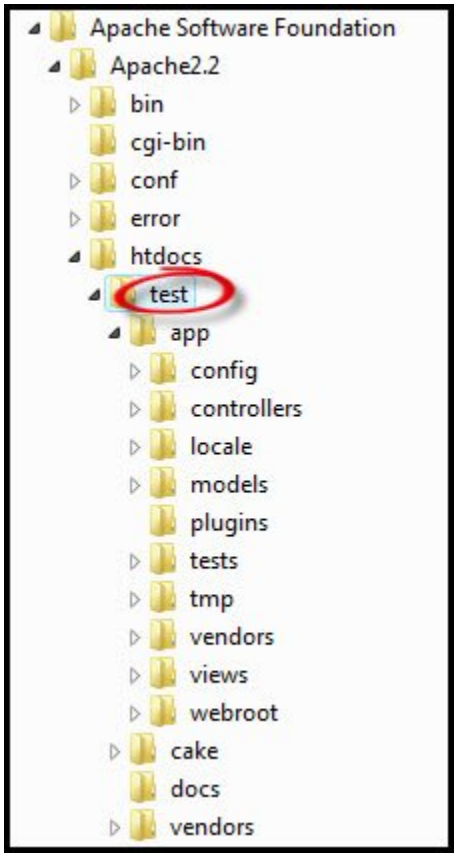

Figure 17: Apache Directoty Structure

Experiment 1: Firefox and image tag

In this experiment, my aim was to test if Firefox caches text on storing as an image.

Following are the steps to conduct this experiment.

1. Created a html page and saved it as an image with .jpg extension
2. Using the above image I developed a web page
3. Requested this web page using Firefox

Following are the observations

1. Firefox could not display the image.
2. It has cached the image in its cache.
3. Hence, Firefox is not checking the content of images; instead if it encounters an image tag it is simply caching the content in its cache.

Experiment 2: Firefox and Script tag

In this experiment, my aim was to understand how much reduction in response time can be obtained using scripts. So I created four web pages that share few components but have different text.

Following are the steps to conduct this experiment

1. Created four pages; each had common header, image in text, left bar, and search bar; All four pages have different text.
2. Stored these pages in htdocs folder of Apache, and requested the pages using Firefox browsers
3. Recorded the response times using Yslow.

Following are the results

Page Number	download time taken for text	download time taken for style.css	download time taken for header javascript	download time taken for header image	download time taken for left panel java script	download time taken for search bar java script	download time taken for image	Total download time taken for page
Page 1	1070 ms	29 ms	1030 ms	46 ms	71 ms	13 ms	49 ms	2300 ms
Page 2	1010 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	1010 ms
Page 3	1030 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	1030 ms
Page 4	1020 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	1020 ms

On observing the results, I found that coding effectively using java scripts will reduce the response time by a significant amount.

Remarks

Doing this experiment gave me a good understanding about Firefox caching mechanisms. I am quite surprised to see how Firefox has simply cached the source of the image, even though it was not really an image. Hopefully, it will soon be taken care by the Firefox team.

6. Summary

Before coming to this project, the only language I used to develop applications was Java. This project gave me an opportunity to learn PHP. Both of them being web application development tools, have similar features for development such as scripts, page forwarding etc, but having a platform like CakePHP for developing PHP applications makes a huge difference. Previously when I had any errors in the web application I had a tough time searching and remembering the file names as there was only a single folder but with CakePHP I was able to organize all my files in the respective model, view, and controller folders which reduced a lot of confusion.

I was also very impressed by the Squid proxy server. It was my first time working with caching proxy servers, so was amazed with the results produced from load balancing experiment I conducted. On observing the results I see that Squid has successfully made requests to both the servers in a round robin fashion. On researching more on Squid, I learnt that there are many other advanced functionalities that can be achieved, but Squid development teams have Squid binary versions available only till Windows XP operating system version. For those with latest laptops having a vista operating system, there aren't any Squid versions available.

I was also very amazed with the work done by the Yslow product. It not only gives a detailed view of all the components in the web page but also determines the download time of all the components in the page. Understanding which parts of the pages are taking more time to download will help us improve the page. Doing the fourth

deliverable gave me a lot of insight into how Firefox interprets html, which is very important when I develop the apparatus for the new caching mechanism.

7. Future Work

In 2009, I will extend the present project, and create a cache at client-side having Squid at the server-side to see how much reduction in response time can be obtained. I will create a design pattern to develop web applications, such that they have a unique id for elements on a web page; I will write a script that caches the common portions of website at the client-side and reuses these in the future requests. I will also write a light weight script that checks if the browsers javaScript is on or off. Since pages are also cached using Squid, I will run several benchmarks to measure the response time and to determine if there is any advantage in having both client-side and server-side caching. The deliverables I worked in this assignment helped me build the foundation for implementing my final project.

References

Best Practices for Speeding up Your Web Site. Retrieved September 27, 2008, from http://developer.yahoo.com/performance/rules.html

CakePHP. Retrieved November 12, 2008, from http://cakephp.org/

Cevasco, F. (2006). The CakePHP Framework: Your First Bite. Retrieved July 12, 2006, from http://www.sitepoint.com/article/application-development-cakephp

Hawes, M.K. (2000). U.S. Patent No. 6,094,662. Washington, D.C.: U.S. Patent and Trademark Office.

PHP. Retrieved November 12, 2008, from http://www.php.net/

Proxy Server, Retrieved December 8, 2008, from http://en.wikipedia.org/wiki/Proxy_server

Squid: Optimizing Web Delivery. Retrieved November 13, 2008, from http://www.squid-cache.org/