LetSurf - A Distributed Caching System

Mangesh Bedekar, Prateek Gupta and Suvrajyoti Chatterjee
BITS Pilani, K. K. Birla, Goa Campus, NH-17B, Bypass Road, Zuarinagar, Goa, India,
mangesh.bedekar@gmail.com, mailtoprateek@gmail.com, suvrajyoti.c@gmail.com

Abstract - We propose a system wherein each computer (connected through high speed LAN) will cache web objects as the user browses through the Internet and make that cache available to the other users (connected to the system). This will exponentially enhance Internet experience of the users in terms of download and surfing speeds. This will also help in reducing the load on the main Internet bandwidth. This will also help in reducing the load on the proxy server, decentralized filtering. Also it facilitates collaborative n social web surfing by allowing the users to rank, comment, suggest similar pages, etc. We will also include small application which will allow the user to examine his browsing patterns.
Keywords- Web Caching, Web Proxy, Distributed Caches, Collaborative Surfing

Introduction

The individual's importance as an active member of the internet grows every day. Starting off as passive surfer of data, users today contribute to forums, collaborate on projects and lead active cyber lives on Facebook. Though we have come a long way, we still lack the ability to communicate and utilize our local communities as we do in real life. The ability to trust and utilize our geographic peers opens up endless possibilities and would be our first step at creating James Cameron's fabled living planet in avatar, where all our personal computers are part of an immense organism, processing, storing and retrieving information at scales unimaginable. As part of this idea, we utilized the power of distributed storage to create an application which accomplishes a simple but essential task-making the internet as fast as it should be. Going further, we have developed proof of concept modules for a system which allows users to explore community preferences and utilize social opinion by storing and retrieving user ratings and suggestions about websites.

Distributed Caching

LetSurf emphasizes on maintaining a local cache on each user's computer which can be used to serve another user when requested. Whenever a user requests for a resource, the resource will be fetched from the LAN (if available on any client), otherwise the resource will be fetched from the Internet. The system is implemented on a two tier model which has a Server Module running on Central Node & Client Module running user's computer. The client module will use some user defined amount of hard disk space & processing power on the user's computer.

Collaborative Surfing

The system allows the user to view and contribute to the social opinion about a particular web resource by viewing other's comment & posting their own. Each web resource will be treated as object which will have attributed like rank, user's comments, etc. These objects will be stored in a database and fetched as and when required.

Embedding Application

We can embed applications in the user's browser window which help the user to examine his browsing pattern, etc.

Security Implementation

Some of the security issues that cropped up during development of the system were Man in the Middle (MITM) attacks, intentional or unintentional cache corruption.

System Architecture

The system runs on a two tier model:
• Client Modules
• Central Server Module
The basic structure of both modules is given below:

Client Module Structure

The module provides the following services:
• Connection Initiation Ability to connect to a remote server socket
• Connection Acceptance Ability to accept connections under certain conditions, as stipulated by protocol
• Caching Data accessed is cached at the user's computer. The data is stored locally in the cache and is maintained with a URL-filename map, which allows the system to retrieve data for a specific URL. Data in the cache is maintained till an expiry date as set in the meta-data, beyond which it is deleted. If the cache size limit is crossed, older data is

purged. The corresponding entries in the central server are also cleaned. All html standard cache directives are obeyed.
• Administration The user has a control over when the system is active, or can set options regarding what should be cached.

Central Server Structure
The central server needs to deliver fast replies to multiple queries.
The central server must provide an ability to add an entry when a search request returns a null set. Also, it must provide for an entry deletion service which allows modules to delete entries in case of connection failures and cache purges, among others. Storage of large data sets is carried out in systems such as the Bigtable distributed storage system for Google, etc. We hope to implement these in further versions.

Scalability:
Single machine server may be enough for smaller network. Distributed Database server will be a scalable option. Each user computer or device carries a module which provides basic services needed for our system. All requests and replies by and to the browser are sent to through this module. When a request from a browser is made (User Y), the module contacts a central server, which maintains a URL-IP map in a database, and requests address identifier of a computer which has the particular resource in its cache. If another computer X is found, the cached copy of that piece of information in X is returned to the user Y. If no computer with that particular resource is found, then the request is directed to the internet for a fresh fetch of data. An entry for this particular resource is made in the central server so that the local cached copy of this resource is available to other users from now on. A table is also maintained which contains the list of online users' IP's. When a user goes offline, the user is removed from the table of online users. In case a match is found in the URL-IP map and the user is offline, then the request is not redirected to the user.

Module - Server Communication
Module server communication for search or deletion queries may be carried out using a simple protocol implemented over the established TCP connection.
Connections for retrieval of cached data should be done through a reliable connection to prevent data corruption. Thus, HTTP over TCP would be the preferred protocol for such a transfer. The system is scalable to large networks through the use of usernames, instead of a direct IPs, for references within the system. Also, setting usernames and passwords allows for an important layer of encryption for each user, which may be implemented through an SSL connection between the central server and each client module. Such scalability will be implemented at a later stage.

Implementation Details
Both Client module and Central server module shared a common set of functionalities which are provided by a general proxy server. Hence a proxy server was used as a common base with separate modifications for each module.

Base Proxy:
Functionalities provided are as follows:
• Connection Initiation
• Connection Acceptance
• Caching
• Indexing
• Proxy Chaining
• Script Injection

Modification for Client Module:
The base proxy was modified to differentiate between requests coming from the user's browser & request coming from the Central Server. The client module redirects the requests selectively based on the decision made in the above statement.

Modification for Server Module:

Database integration
A MySQL server is storing all the necessary data. A persistent connection to the database made as soon as the server module starts.

Search function
When a client requests a particular URL, the server runs a SQL search query on the MySQL database through the connection object. The database server traverses the IP – URL table to find a match and return the IP of the client if match is found (returns null, if match not found). The Persistent Connection object with the MySQL database to issue a query through a method of the object.

Database Updating:
INSERT & UPDATE SQL queries are used to modify the IP – URL table whenever a client requests for a web resource.

Database Maintenance:
Whenever the client loses some data (in case of cache corruption or data expiry or cache purging), the central proxy server updates it's IP – URL table accordingly.

Redirection scheme:
On an incoming request, the central server runs the Search function to retrieve the IP of the client module having the particular resource.

- If match is found, the central server chains itself to that particular client module proxy which fetches the data from its cache.
- If match is not found, the central server redirects the request to the Internet for a fresh fetch of data.

In either cases, the IP-URL table is updated.

Security Issues
In a distributed caching system, one of the major concerns regarding implementation is the security aspect. Caches may store private data, and sensitive private browsing information. Our implementation of the distributed caching mechanism aims to maintain the integrity of the user's data.

MITM attacks
The man-in-the-middle attack (often abbreviated MITM), or bucket-brigade attack, or sometimes Janus attack, is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. MITM attacks are generally carried out by using various transport layer and lower levels hacks like ARP poisoning and port stealing.

MITM in the LetSurf system:
There is a possibility of MITM in the LetSurf system, as explained by the example given below:
A social networking website, let us say http://www.sns.com is used by two users A and B.
The login page of http://www.sns.com passes the form data in POST format.

In the post format, the requested URI is the same, and the data is passed to the URI within the content of the request. If A logs into http://www.sns.com from IP 10.4.1.46.

Table 1-URL Map maintained in the server

URL	*IP*
http://www.sns.com/login.php	*10.4.1.46*

Now when user B attempts to log in. The central server will check the requested URL (http://www.sns.com/login.php) and redirect the request to get re-routed from the 10.4.1.46 IP. At this stage all private data of user B passes through user A's computer and user A may eavesdrop to find out sensitive information about user B's account.

Our solution to prevent MITM attacks:
Since POST requests may share the same URL for different content type, we want to prevent any un-authorised eyes to view the private information passed in the content of the post requests. We will be detecting the request method at the central servers (GET or POST) and will be storing only the GET requests at the URL-IP table. This will ensure that no private data is viewed by an un-authorised 3rd party. We are also on implementing a two way SSL type encryption, which will use asymmetric and symmetric key encryption to encrypt the requests and the responses between the client and the server.

Proxy Cache poisoning
The cached data of a user's browsing patterns is stored in the user's own hard disk and consequently he may have access to it. Data corruption may occur possibly due to erroneous handling of the cached files by the users. Data may also be tampered by the user to change important aspects of a page (For eg. form actions which may be changed in a way to force the user to submit private information on a rogue server). To deal with cache poisoning we are implementing a symmetric encryption with corruption detection.

Symmetric Key Encryption:
The encryption key is trivially related to the decryption key, in that they may be identical or there is a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link.

Symmetric v/s Asymmetric Key Encryption:
Unlike symmetric algorithms, asymmetric key algorithms use a different key for encryption than for decryption. i.e. a user knowing the encryption key of an asymmetric algorithm can encrypt messages, but cannot derive the decryption key and hence cannot decrypt messages encrypted with that particular key.

Why Symmetric Key Encryption?
Since one of our primary objectives is to increase the internet speeds, decryption speed plays a major role in selecting the algorithms involved in the process.
Symmetric-key algorithms are generally much less computationally intensive than asymmetric key algorithms. In practice, asymmetric key algorithms are typically hundreds to thousands times slower than symmetric key algorithms.

Solution to data corruption problem:
We will maintain a table of respective filenames and their MD5 checksum of all cached files to prevent cache poisoning. MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. For example, file servers often provide a pre-computed MD5 checksum for the files, so that a user can compare the checksum of the downloaded file to it. In cryptography, MD5 (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit hash value. An MD5 hash is typically expressed as a 32-digit hexadecimal number. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. The MD5 algorithm is designed to be quite fast on 32-bit machines.So, if a proxy detects that the MD5 checksum of that file is not matching the stored MD5 checksum which is stored in the hash tables. It will detect data corruption and redirect the end user to retrieve the URL directly from the internet. The proxy will delete the corrupt file and inform the central server to modify their IP URL map tables.

References

Proxy:

[1] Thomas Keir (2006). *Beginning Ubuntu Linux: From Novice to Professional. Apress.*

HTTP Standards:

[1] *RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1,* R. Fielding, J. Getty, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee (June 1999)

Rabbit Proxy v3.2:

[1] http://rabbit-proxy.sourceforge.net/.
[2] http://rabbit-proxy.sourceforge.net/faq.shtml
[3] http://rabbit-proxy.sourceforge.net/RabbIT3-bin.tar.gz
[4] http://rabbit-proxy.sourceforge.net/RabbIT3-src.tar.gz
[5] http://www.nada.kth.se/projects/prup98/web_proxy/doc/.

MD5:

[1] Berson, Thomas A. (1992). "*Differential Cryptanalysis Mod 232 with Applications to MD5". EUROCRYPT*. pp. 71–80
[2] Bert den Boer; Antoon Bosselaers (1993). Collisions for the Compression Function of MD5. Berlin ; London: Springer. pp. 293–304.