Duplicate Transfer Problem inside a Proxy's Cache

Darius-Vasile Bufnea
Department of Computer Science
"Babeş-Bolyai" University
Mihail Kogalniceanu nr. 1,
Cluj-Napoca, Romania
bufny@cs.ubbcluj.ro

Florian Mircea Boian
florin@cs.ubbcluj.ro

Abstract

This paper presents an overview of the current web client - web proxy - web server mechanism and takes a deep look into one of its main disadvantages: the replication, in the proxy's cache, of web objects having different URL but the same content. This problem is known as the "Duplicate Transfer" problem and is mainly caused by the current mode of indexing web objects based on their URL, which is used as a primary key in the cache repository. We present in this paper a statistical analysis based on real traffic measurements, which shows that more than 10% of a proxy's cache consists of replicated objects, grabbed from the Internet in a useless manner and stored redundantly at least twice. These results urge the development of a scalable real-life solution to the duplicate transfer problem: some solutions have been previously proposed, but never deployed on a large scale in Internet.

1. Introduction

The dominant traffic types in the current Internet fall in two main categories that take approximately equal shares of the global Internet traffic: the web traffic and the peer to peer (P2P) traffic. Although the latest one has taken a serious advantage beginning with the late 90s, more recently the web traffic also claimed its share and today it exceeds any other form of Internet traffic, including P2P traffic. This come-back situation is mainly due to the following two aspects:

a) The legal battle against the illegal aspect of sharing multimedia content files transported over peer to peer protocols;

b) The migration of software development from desktop and standalone applications towards web-based applications.

Considering the main share of the web traffic, many enterprise networks currently implement various mechanisms to control and reduce this type of traffic. Usually, in an enterprise network having hundreds or thousands of clients (enterprise users), these clients access the Internet through an intermediary (middleware) server called a proxy server, such as Squid [2]. This architecture presents a series of advantages, both from the enterprise and the user's perspective. From the enterprise's perspective a web-proxy server usage can reduce the public IP space required to address the clients - these clients can be addressed using private IP addresses [5]. Also, a proxy server is generally extremely configurable and allows definition of complex access lists (ACL) that can filter the web traffic based on different characteristics such as: client address, URL, protocol, content type or date and time.

From the client's perspective, a web client that accesses the Internet through a corporate proxy server can benefit from the web objects already in cache that other clients have previously requested. This will result in fast object delivery to the client (i.e. higher navigation speed), less traffic from the Internet towards the corporate network (i.e. less consumed bandwidth, higher bandwidth available to other clients/applications) and lower web server load (as less objects are to be served to clients).

This already existing proxy-based web architecture pleases everybody: from the corporate user to the corporate staff and web servers' maintainers. However, there are certain situations, very often encountered, when the proxy server will retrieve from the Internet an already in cache object. This is caused by the fact that an object is identified by its URL (Uniform Resource Locator) rather than by its content. We will reveal later in this paper such a situation encountered in our experience of operating the Campus Intranet.

This paper advocates and urges the introduction of a new web-based mechanism for serving and indexing web objects by their content, instead of the current URL location based

method. This new serving mechanism must be implemented in both servers implied in the web delivery process: the web server and the proxy server. This mechanism is transparent to the end-user - the only client side aspect of the proposed mechanism is a faster delivery of certain web objects. There are no modifications or additions necessary on the client side, a fact that might ensure an easier and faster deployment process of the presented mechanism.

This paper is structured as follows. First, we will present the Duplicate Transfer Problem as part of a working scenario often encountered in today's enterprise networks. The disadvantages of the current state of facts in the browser - proxy server - web server architecture will be also discussed, revealing the urgent need to adapt this architecture to a new model where objects are indexed by their content. Proposed mechanisms for achieving this model will be described in the next section of the paper. We will continue by presenting a statistical analysis in order to demonstrate the impact and benefits of the new model. The paper ends with conclusions and future work.

2. Duplicate Transfer Problem

Let $C = \{C_1, C_2, \ldots, C_n\}$ be a set of web clients located in the same LAN or in the same enterprise Intranet. These clients access web resources located in the Internet through a corporate HTTP proxy server. We will denote this proxy server by S. An object O located at URL_O requested by a client C_i, object that has not been previously requested by another client, will be stored after retrieval in the web proxy's cache. Because in the proxy's cache the objects are indexed and identified by their URL rather than by their content, when a second client C_j will request a content identical object P ($O = P$) located at a different URL, URL_P ($URL_O \neq URL_P$), the object P will also be retrieved from the Internet, even if an object with a similar content already exists in the proxy's cache. This second retrieval process is a waste from the client C_j's spent time point of view or from the consumed bandwidth point of view.

This inefficient approach is used by the most popular proxy server in the Internet - Squid. Squid indexes and stores web objects by their URL - in fact it uses the MD5 checksum of their URLs as the primary key for storing, indexing and retrieving objects from its cache.

A real scenario that covers the above formal description follows. Our Computer Science Department hosts a series of Linux workstations labs. Periodically, all these Linux workstations execute an automatic update process, which implies retrieval of any available updates from the Internet. An update is a web object located in a specific web repository. For load balancing reasons, the updates repository is replicated in multiple mirrors, each mirror being accessed with a different URL. This fact implies that the same updates, located in repositories having different URLs, are treated by the proxy server as different objects, even if they have the same content. If one of the Linux workstations is retrieving first the updates from a certain mirror M_1, these updates are cached by our department proxy server. Another Linux workstation that will setup the update process at a future moment of time, might be assigned to download these updates from a different mirror M_2, having a different URL from the first one. Even if the updates are the same, the proxy server will treat these updates as different objects and will retrieve them again from the Internet and will also cache them - i.e. will cache an object twice or several times even if their content is the same.

For a better understanding of this situation we depicted it in figure 1.

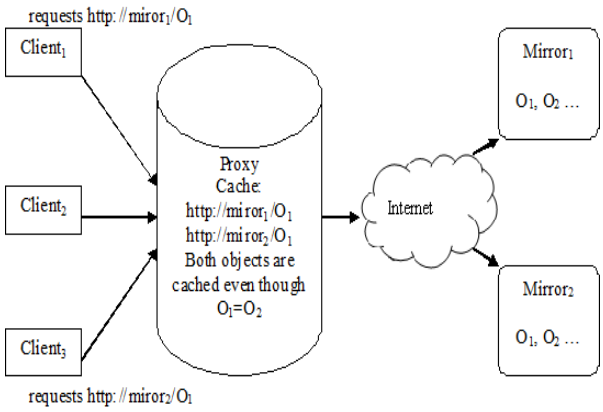

Figure 1. Duplicate Transfer Problem

The disadvantages of this behavior are obvious. First of all, any web client, starting with the second one, that accesses a certain update, might wait for the update being retrieved from the Internet even if the update is already in the proxy server's cache, located in a location that is faster to access. Secondly, multiple retrievals of the same objects lead to wasted precious network resources such as bandwidth. And finally, there is a waste of storage space at the proxy server level, because objects having the same content are cached more than once.

An alternative solution to this situation is to set up a repository of updates (a local mirror) located in our department's Intranet. Forcing the Linux workstations to retrieve the updates from this local repository will save download time (all the clients access a very fast, proximity located update repository) and bandwidth (all the updates are downloaded only once, specifically when our mirror is synchronizing with other Internet mirrors). However, this solution did not please us. Setting up and maintaining a local repos-

itory might be an expensive process (from the human resources point of view). Also, this addresses the problem of our specific scenario, but does not solve the general problem of having the same object being retrieved and cached by an HTTP proxy server more than once.

3. Solutions to the Duplicate Transfer Problem

The solution of the above problem is identifying and indexing web objects at the level of a proxy server by their MD5 checksum [6]. This mode of identifying and indexing web objects is not to replace the classical one where web objects are identified and indexed by their source URL. It may be used as an alternative method for maintaining web objects at the level of a proxy server, especially web objects of a considerable size.

In order to maintain web objects at a proxy server by their MD5 checksum, certain requirements have to be fulfilled by the web server where the web objects reside. The web server must notify a client requesting a certain web object of the object's checksum prior to the object's content delivery to that client. The justification for this step is that, once a client (i.e. a middleware proxy server) receives an object's checksum, it might not be interested anymore in that object's content because it locates the object by its checksum in his own cache.

The delivery of the MD5 checksum to the client, if it is available at the level of the web server, might be an implicit or an explicit process. In the first case, the web server might notify a web client, even though this might not be interested, about the checksum of the web object that will be served next by using the `Content-MD5` HTTP header as described in [4]. For example, the Apache web server is able to deliver such an HTTP header by using the `ContentDigest on` directive in its `httpd.conf` configuration file [1]. Unfortunately, this is an Apache only feature, the other dominant web server in Internet - i.e. IIS - lacks such a feature. However, the web server might notify the client about the checksum only when the client explicitly requests it, by using a different HTTP request (besides the HTTP request for the web object itself). Such a preliminary request may be invoked using the `HEAD` HTTP command.

In order to deliver the checksum the interested clients, this checksum must be available at the web server's level. There are multiple approaches for storing and computing the checksum for a web object:

- The checksum is computed by the web server "on the fly", when a client sends the first request for a web object. While the web object content is read from the file system to be delivered to the client, the web server can also compute the content's checksum. This is a run-once process, because the checksum information can be stored in the web server memory cache and may be subsequently delivered to any other client that might request that web object again.

- The checksum is pre computed and is located in a file in the same web space (web folder) as the web objects. This is a very common situation, when for extremely large file offered for download, the client can also download a file containing the checksum of the large file, information that might be helpful in verifying the download. For example, a file available for download called Fedora-8-i386.iso might be accompanied in the same web folder by a file called Fedora-8-i386.iso.md5 containing the MD5 checksum of the Fedora-8-i386.iso file.

4. Statistical Analysis

In order to analyze the Squid cache, we developed a software tool [3] that retrieves for each cache object its content and some of its properties such as: size, URL and, most important, its MD5 checksum. We will briefly present next the results of this analysis.

By object cont:

Total objects in cache	653529
Unique objects	572586
Useless (redundant) objects	80943
Percent of useless objects	12.38 %

By object occupied space:

Cache size in bytes	10662828959 (9.93 GBytes)
Space occupied by unique objects	9548218120
Useless cache space occupied by useless objects	1114610839
Percent of useless cache	10.45 %

Other statistics:

Useless unique objects (appear at least twice in cache)	39243
Minimum duplications count	2
Maximum duplications count	1028
Size in bytes of the object having maximum number of occurrences	575
Average number of duplicate occurrences	3.06
Minimum size in bytes of a duplicate object	332
Maximum size in bytes of a duplicate object	4140152
Number of occurrences of the duplicate object having maximum size	3

The results of our experiment show that, from a proxy server perspective, approximately 10 to 12 % of the cache size consist of duplicate objects that are retrieved from the Internet and stored in the proxys cache at least twice in a useless manner.

In these situations, a duplicate transfer aware proxy server might reduce an enterprise consumed bandwidth by at least 10 %. This is not the single advantage, the proxy clients might also benefit with a higher cache-hit ratio from already in cache objects previously grabbed from the Internet for other clients.

5. Conclusions and Future Work

We advocate in this paper the need of a new method for indexing and storing objects in a proxy's cache by their content and their MD5 checksum, approach that is perfectly interoperable with the current one that supposes web objects management by their URL.

The advantages brought by such a mechanism are obvious: fast objects delivery to a client in certain situations, bandwidth savings from the corporate perspective, and lower load from the server point of view. From the above exposed experiment, we conclude that such a mechanism may reduce the storage space and the download traffic of a proxy server with at least 10 percent.

The Squid proxy server development projects list includes the Duplicate Transfer Detection project. Unfortunately, this project is listed as stale project, not being actively developed. The Duplicate Transfer Detection project was developed for Squid version 2.4STABLE7, a version which is five year old. We are currently working to port the DTD project to the latest version of Squid proxy server, i.e. version 3.0.

Because the MD5 message digest algorithm presents some collision related security issues, future work may also imply the implementation of a similar mechanism using other message digest algorithm such as SHA1.

References

[1] *Apache Core Features*. http://httpd.apache.org/docs/1.3/mod/core.html.

[2] *Squid: Optimising Web Delivery*. http://www.squid-cache.org.

[3] D. Bufnea. A tool for md5 checksum retrieval of squid cache objects. *http://www.cs.ubbcluj.ro/~bufny/*.

[4] J. Myers and M. Rose. The content-md5 header field. *RFC 1864*, October 1995.

[5] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. *RFC 1918*, February 1996.

[6] R. Rivest. The md5 message-digest algorithm. *RFC 1321*, April 1992.