Optimizing the HHTP Procotol

Gómez Roberto, Luna Felipe
ITESM-CEM
DIA-Dpto. Ciencias Comutacionales
Apdo. Postal 6-3, Modulo Servicio Postal
Atizapán de Zaragoza, 52926 Edo. de México, Mexico
rogomez@campus.cem.itesm.mx
fla@mailer.main.conacyt.mx

abstrac

Nowadays there has been an increase of the Internet community. One of the most frequently practice of the Internet community is the information search. This search is made through the World-Wide-Web (WWW) using the HTTP protocol. Users do not like to wait for their results, they tend to avoid or complain about Web pages that take a long time to display. The present work shows how we can reduce the response time by making some changes to the HTTP protocol. It is inspired from previous work on HTTP performance analysis.

Keywords: distributed systems, HTTP, WWW

1 Introduction

The WWW project merges network information technique and hypertext model to make an easy but powerful global information system. The World-Wide-Web (WWW), also known as W3 or simply Web, started as a project to provide a common place where some CERN researchers could share their ideas and knowledge referring to different projects about high-energy physics ([Lee94]). In the Web there is not centralized information, if two projects are related they make reference each other. All the references are made through a hyperlink, giving flexibility to make changes without altering the other one. The WWW world consists of documents, and links. Indexes are special documents which, rather than being read, may be searched. The result of such a search is another ("virtual") document containing links to the documents found.

A remote information server uses a simple protocol (named HTTP) to allow a *browser program* to request a keyword search. The documents are structured in a logic form or mark-up, filling all the specifications of the *HyperText Markup Language* or HTML. The information transference through web uses the HTTP protocol (*HyperText Transport Protocol*). The HTTP protocol allows us the transfer of text, hypertext, pictures and other objects with different formats.

HTTP is a client/server and stateless protocol. A client sends a request and waits for the client answer. The stateless property means that the server does not keep any information about the operations that it has been performed. The communication between client and server is based on a single connection. This feature presents some disadvantages, so we wish to change it. The original protocol design does not consider the server performance, and consequently the protocol has minimum performance. Latency and Bandwidth are the two keys for protocol performance. Latency, as measured by the RTT [1] is a measure of transaction fixed costs and it does not change as documents get bigger or smaller. Bandwidth is a measure of how many bits per second the connection can carry. Improving

[1]RTT is the time taken to sent a packet from one end of the connection to the other and back.

available bandwidth is relatively easy; all we need to do is buy a faster network. Reducing latency is somewhat harder there are only two known ways to achieve this, either change the speed of light, or move ([Sp94, G95]).

The actual needs require an improvement of the narrow wide band with high latency times, it is required that the protocol allows to perform the operation out of sequence, to control operations using priorities, data transmission and efficiently handle of bits, etc. We also have to consider security, authentication and authorization to allow the business Internet transactions.

The present work suggests some improvements to the actual HTTP protocol in order to reduce the time that a user waits the requested document, and to improve server performance.

This document is organized as follows. Next section shows the HTTP protocol features, describing its problems related with response time and server performance. Section 3 presents our proposal, pointing out the modifications proposed to the HTTP protocol. Section 4 describes the protocols implementation and test performed on it. In the last section we draw our conclusions and propose some future work.

2 The HTTP Protocol

The HTTP Protocol version 1.0.0, ([FrFiLe95, Sp95, FrSpJa98]) works in two scenarios; on the first one the client connects directly with the server, on the second one there is one or more intermediaries between client and server.

In the basic scenario, without errors, the HTTP works as follows. A client contacts a server and establishes a connection with it. Once the connection is established the client sends an information request. The server gets the request, searches the information and transfers it to the client. At the end the server closes the connection. Request and responses are expressed in a simple ASCII format. The request and the response are sent and displayed by a graphic interface called *browser*.

In order to establish the connection the client makes a TCP-IP connection to the host using a domain name or IP number, and the port number given in the address. HTTP currently runs over TCP, but could run over any connection- oriented service. The client request is sent with a first line containing the method to apply to the object requested, the identifier of the object, and the protocol version in use, followed by further information encoded in a MIME message, (*Multipurpose Internet Mail Extensions*). This message contains client information and some information related with the request.

The object can be identified using a *Uniform Resource Locator* (URL) defining the address of an object. The method field in HTTP indicates the method to be performed on the object identified by the URL.

Exemples of common request methods are GET, HEAD and POST. The first one is used to obtain the requested object. HEAD is designed to browser's update. POST creates a new object linked to the specified object; it can also be used to transfer a new object to the server.

The response from the server consists of HTTP version being used by the server, the coded results of the attempt to understand and satisfy the request, and an explanation for a human reader, except where noted for particular status codes. The headers on returned objects are listed as objects headers, as well as any MIME conforming headers. Additional information may follow, in the format of a MIME message body. The significance of the data depends on the status code.

In the second scenario, the protocol is affected by the existence of many intermediary servers named Proxies. These Proxies are cache servers of web resources. They were designed to reduce the delivery response time, and consequently increase the server performance.

2.1 HTTP Protocol Performance

There are many studies about HTTP performance ([K95a, FL95, Sp94]), and others that presents some possible solutions ([H95, K95]). In [Sp94] Simon E. Sperno shows one of the first performance problems studies in servers that use HTTP protocols. The paper presents the factors that influence in the response time. One of the Sperno's conclusions shows that a client access follows the next steps: the client requests a document and the images included on it, when he gets the entire document there is a high probability that he uses a hyperlink of an object residing in the same server. This causes some traffic on the network lines, in the next paragraph we explain the reason of this.

TCP establishes connections via a three-way handshake. The client sends a connection request, the server responds, and the client acknowledge the response. The client can send data along with the acknowledgement. Since the client must wait for the server to send its connection response, this procedure sets a lower bound on the two RTTs transaction time. TCP determines the best rate to use through a process called *Slow Start*. The slow-start approach requires the TCP sender to open its "congestion window" gradually, doubling the number of packets each round-triptime. TCP does not reach full throughput until the effective window size is at least the product of the round-trip delay and the available network bandwidth. This process affects the performance, since for pages larger than 1K size, it will add at least one RTT to the total transaction time. Longer documents may experience several more slow start induced delays. Another problem in using TCP is the use of $Time\_Wait$ value. This is the time that server must keep the object once the connection has been closed. A high value reduces the available resources at the server.

Venkata, Padmanabhan and Mogul ([PaMo94]) studied the performance problem from a latency point of view. They agree in the fact that the delays of requesting an HTML object depends on the network latency, and suggest for avoid it to eliminate the use of round trips. They also observed that the slow-start process mentioned above is good for congestion avoidance but bad for short-connection completion latency. There is a transfer reduction in the case of transmissions greater than 500 Kbytes, that is the case of documents including images. To minimize these problems they suggest two things, the first one consists in to keep a connection in order to processes different requests, and the second one is to use a pipelining method to do it.

Keeping the connection reduce the extra charge in the server produced for creating a new connection every time a request arrives to the server, it also minimizes the delays produced by the TCP slow-star process.

In HTTP protocol the client interacts with the server in a stop-and-wait fashion. It sends a request for an image only after having received the data for the previous one. The pipelining proposed by the authors consists in recollect all images with the same request. In order to accomplish this, two new methods are introduced in the server: GETALL and GETLIST. The first one specifies that the server should return a HTML document and all its images residing on that server. The second method allows a client to request a set of documents or images from the server. The pipelining solution reduces the latency more than 50% when images are transfered from a remote server with a size of 2Kb, with 45Kb images the latency is reduced 22%. We can conclude that there is great benefit in the transmission of small images, if a medium number of them are transmitted.

One of the most complete performance studies is presented by Mogul in [M95]. He consideres the pipelining solution, just described, to design a *persistent connection* protocol (named P-HTTP). He makes a simulation using data obtained from real applications but, instead of using a HTTP server, he uses a simulator to model the relevant behavior of a HTTP server.

The simulation shows that persisitent-connectiion can greatly reduce the response time, server overheads, and network overheads of HTTP protocol. P-HTTP should also help an overload server flow-control its clients. It also shows that increasing the maximum number of active connections will increases user's perceived performance, at the expense of somewhat increased server resource demands. The feasibily of P-HTTP depens on the availability of robust client and server implementations.

3 HTTPf v1.0 protocol description

Our work proposes a new implementation that improves the HTTP protocol performance in latency time. In order to accomplish this we propose the creation of a server and a client browser that communicate each other using a modified HTTP protocol. This new protocol will be named HTTPf v1.0.

The server process is divided in several parallel tasks. These tasks are performed by a distributed system composed of multiple servers. In this way, the server system will be distributed, this means that its components will work in a distributed way (distributing the work between different machines). In the client this new design is based on the use of multithreads. This will require the parallel use of multiple TCP connections between client and server, inside the same HTTP protocol.

The protocol is modified in order to provide information to the server, about the number of images contained inside a HTML document. In this way the server sends the entire document, without waiting that the client sends the request (specially images) of every document component.

The final design incorporates some ideas already tested in commercial products. One of these ideas is the use of a *request broker*, which decides how the client will be treated and thread creation, one for each client. Moreover, the server can be executed on different computer architectures (and several plataforms Unix, MS-DOS, Windows, etc). Only one of these computers will be considered the front end CPU and will hold the *Web Request Broker*. The IP address of the front-end computer will be known by the public, in this way the clients will contact this computer to send their requests. On the other hand, this provides a computer cluster server that is seen as one logical unit.

The client consists of a graphic interface and is executed in any environment that can be supported by the Java Runtime Environment. It uses the already established connection with the server (i.e. the *Keep Alive* concept introduced in the protocol HTTP version 1.1, [FiBe95, KrMoKr98, FrGeSm97]) and finds what kind of document has been sent by the server. If a document is a text with HTML format, it is examined to determinate how many and which images are referenced. Once the image list is conformed, these are requested to the server, without using any protocol and in a parallel way.

A physical server (different of the front-end server) treats the clients. The *Web Request Broker* is executed in this server, and decides which computer is going to treat the request. The requests are treated in a *round-robin* way.

The implementation uses TCP sockets. It is not possible for a computer to take data from another computer if they are not connected. So, the connection between the final server and the client is different from the initial one (between the front server and the client).

3.1 HTTPf v1.0 protocol steps

The client initiates the protocol when it establishes the communication with the server. The client connects with the server using TCP/IP sockets at the port number 24827 (port number are selected in an arbitrary way).

The client creates a thread named *GetReponse*, which waits for the server's response. It will receive the information directly from the server that takes care of him.

Once the client has created the thread and the connection has been established (the port number is provided by the operating system), it creates and sends the complete message request. This message includes the URL, the client IP address and the socket number, which receives the server response.

Meanwhile, the *WebReqBroker* in the server receives the HTTPf request message and closes the client/server connection once the message arrives. Then, it decides which remote CPU will take care of the request, and forwards the request message by intermediary of the *Java Remote Method Invocation* (RMI) to the selected CPU. Once the message has been sent, and the connection between the selected server and the client is made, the *WebReqBroker* throws out anything related with the request.

We have to mention that the remote CPU has a method named *ListenToBroker*, which receives the HTTPf message request sent by the *WebReqBroker* and creates a new thread, called *ClientProcessThread*. In the *ListenToBroker* method we found a function similar to *WebReqBroker*. They resemble because both receive requests, and create a thread, which attend the request. This means that both can treat several clients in parallel.

Next, *ClientProcessThread* process the HTTPf request and gets, from local file system, the desired URI. This will be part of the response. Besides, if the document has an HTML format, two threads are created which wait for a client connection.

Now *ClientProcessThread* initiates a connection with the client using a port number included in the HTTPf request message. Once the communication is established, the server sends the HTTPf response message. This message respects the same HTTP message format: response state, response headers and the requested information. The only difference is that in this case the *MicroServer's* socket numbers are also included.

It is important to note that the client/server connection has not been closed, for this reason client's requests will be made through this connection.

However, if the document received by the client is a HTML text, the client asks for the images, otherwise the treatment for that request is considered as finish.

Once the client has received the document it determinates how many images are included on it. Then the client creates the same number of threads, named *GetImage* each one. These threads, in parallel, request all the images to the *MicroServers.* There is not special protocol used for this request, the client only sends the image name and once the image is received the communication channels are closed.

We do not have to forget that there are only two channels to obtain all images. Both channels are used in the following way: *GetImage* requests an image, receives and store it; next the thread asks for a new image and repeats the same steps until the image list is empty. For this manipulation the pipelining concept is used to send all the images names, one after the other, and to receive them in the same way.

Finally there are some importants implementation aspects to be considered. The client/server channel uses a reception buffer of 4096 bytes, all the received messages are comprised using the zlib algorithm, and connections in the process disable Nagle algorithm.

4 Proofs and implementations

To prove the proposed protocol we implemented a global server using tree computers. The first one was a HP9000 with HP-UX 10.20 as operating system, 64Mb in RAM and a 10Mbps Ethernet card. The second one was a PC with Linux Red Hat 5.2, 32 Mb in RAM, Intel Pentium II at 450Mhz and a 10 Mbps Ethernet card. The last one was a PC with a Windows NT 4.0 operating system 256Mb in RAM, Intel Pentium II at 450Mhz and a 10Mbps Ethernet card.

As we use Java Language ([Ru97, O95, SiNg95, Ka96]) to implement the protocol, all the computers use the *Java Runtime Environment* to execute the system.

The two PCs are located in a LAN network at the Puebla City, and the HP equipment is in another LAN network at Mexico City (the distance between these two cities is about 120 kms). The two LANs are communicated by a fiber optic channel E1 (2.04 Mbps). It is interesting to know that the servers have different functionality proposes.

In the client size, we used two computers. One was the Windows NT server mentioned above and the other one was a PC with Windows 95 as operating system, 16Mb in RAM, Intel at 233Mhz and a 10Mbps network card. Both clients were connected to the same LAN that include Windows NT and Linux servers. The *Java Runtime Environment* was also used for the execution of the client process.

We use four pages in this proof. The first one, for simplicity named P1.html, is 827 bytes long and includes 5 images. Taking in account the images and the text the document is 130,091 bytes long. This page was used to prove the protocol with a page without user text and medium size images. The second page, called p2.html, contains 13,428 bytes and has 21 images. The total size of the page is 76,619 bytes. This is an example of an HTML page that contains thumb size images. The third one, p3.html, is 12,378 bytes long and includes 6 images (total size 51,221 bytes). This is a typical web page that we can found at Internet, it does not have much text and is composed of medium size images. The last page, p4.html, has 21,171 bytes and 9 images, making a 36,433 bytes size page. This document represents the web pages with several pages of small size.

More than 137 requests were made to the global server so we can say that each server provided 1/3 of all responses, and that every client contributed with half of the requests. Some servers provided the response in a comfortable time for the user and the others made it slower. The clients presented short responses times.

The document size varies between 382,976 bytes and 36,191 bytes, (105,963 in average). The reception time for the complete documents is 14.67 seconds as maximum and 0.001 as minimum (1.37 seconds in average).

5 Conclusions and future work

We presented a modification of HTTP protocol in order to reduce the response time and to improve server performance.

From a user point of view the time average obtained in the proof (1.37 seconds) is a very good time, compared with the actual response time of any page at Internet. However, we observed that the equipment features affect the response time. In the case of Windows 95 PC, it required 50 seconds to display the image, once it has been received. The reason of this behavior is the browser requirements, 18 MB in RAM as minimal. It will be of interest to measure the response time of clients connected to Internet with modems.

There are many proposals for future works. The implementation of a load balancing for the request assignment in the server could be interesting. The implementation of cache mechanism in the client is also to be considered. The amelioration of the pipelining mechanism can help in the performance. The employment of only one header for different kinds of message could help to reduce the network traffic. The proposed protocol can be used in a multiprocessor machine instead of a distributed system.

References

[FiBe95] Fielding, Roy; Frystyk, Henrik: Berners-Lee, Tim Hypertext Transder Protocol HTTP/1.1. Internet draft (draft-ietf-http-v11-spec-00.txt) November 1995

[FL95] Luotonen Ari; Franks,John Byte Ranges with HTTP URLs. Internet draft (draft-loutonen-http-url-byterange-00.txt), Netscape Communications Corp., Juin 1995

[FrFiLe95] Berners-Lee, Tim; Fielding, Roy; Frystyk, Henrik Hypertext Transfer Protocol HTTP/1.0 Internet draft (draft-ief-http-v10-spec-02.txt), MIT/W3C, August 1995

[FrGeSm97] Frystyk Nielsen, Henrik; Gettys, Jim; Baird-Smith, Anselm; Prud Hommeaux, Eric; Wium Lie, H Kon. Lilley, Chris Network Performance Effects of HTTP/1.1, CSS1 and PNG URL http://www.w3.org/TR/NOTE-pipelining-970624, June 1997

[FrSpJa98] Frystyk Nielsen, Henrik; Spreitzer, Mike; Janssen Bill; Gettys, Jim. HTTP-NG Overview: Problem Statement, Requierements and Solution Outline URL http://www.w3.org/Protocols/HTTP-NG/1998/draft-frystyk-httpng-overview-00. November 1998

[G95] Gettys, Jim HTTP-NG Development URL: http://sunsite.unc.edu/ses/http-ng-archive/0005.html. October 1995

[H95] Hopmann Alexander. HTTP Session Extension. Internet draft (draft-ief-hhtp-ses-ext-00.txt), ResNovaSoftware, Inc., July 1995

[K95] Kristol, David M. A Proposed Extension Mechanism for HTTP Internet draft (draft-kristol-http-extensions-00.txt) AT&T Bell Laboratories, Janury 1995

[K95a] Kristol, David M. Proposed HTTP State-Info Mechanism. Internet draft (draft-kristol-http-state-info-01.txt), AT&T Bell Laboratories, September 1995

[Ka96] Kadel, Rich Generic client-server classes URL http://www.javaworld.com/javaworld/jw-09-1996/jw-09-client-server.html. September 1996

[KrMoKr98] Krishnamruthy, Balachander; C. Mogul Jeffrey; M. Kristol, David. Key Differences between HTTP/1.0 and HTTP/1.1 Document no. HAg1g3000-981207-01TM AT&T Technical Memoramdum, December 1998

[Lee94] Berners-Lee, Tim Cailliau, Robert; Luotonen, Ari; Nielsen, Henrik; Secret, Arthur The World-Wide Web. Communications of the ACM, August 1994

[M95] Mogul, Jeffrey C. The Case for Persistent-Connection HTTP, Digital Western Research Laboratory Research Report 95/4. Digital Equipment Corporation, May 1995

[O95] Ousterhout John. The relationship between TCL/TK and Java. URL http://www.smli.com/research/tcl/java.html. October 1995

[PaMo94] Padmanabhan, Venkata N.; Mogul, Jeffrey C. Improving HTTP Latency. Proceedings Second WWW Conference 94: Mosaic and the Web. Chicago Illinois. Octuber 1994

[Ru97] Rusty Harold, Elliote Java Network Programming, O'Reilly & Associates, Inc. 1997

[SiNg95] K. Sigghal, Sandeep; Q. Nguyen, Binh; Redpath, Richard; Fraenkel, Michael; Nguyen, Jimmy Building High-Performance Applications and Services in Java: An experimental study IBM T.J. Watson Research Center - IBM Software Solutions

[Sp94] Spero, Simo E. Analysis od HTTP Performance Problems URL http://sunsite.unc.edu/ release/http-prob.html, July 1994

[Sp95] Spero, Simon E. Progress on HTTP-NG URL http://www.w3.org/hypertext/WWW/Protocols/HTTP-NG/http-bg-status.html, 1995