Auto-Mirroring Support in Apache

Naveen Prakash and Rose M.-G. Shie
Department of Computer Science
University of Wisconsin at Madison
email: {naveen,mei-guei}@cs.wisc.edu

Abstract:

In this paper, we introduce a new mechanism for maintaining a good response time to the user inspite of increase in load at the web server. We have extended Apache, a http server available in the public domain, to implement the design. The overloading of site is often random. When a site get overloaded because of some unpredictable event, it can seek service of 'rental server' to act as 'mirror server' for it. The rental server will be able to accept request for mirroring support and process user requests for documents of hot spot server. The documents at the rental servers should be consistent with those at the original server. Besides giving the implementation for auto-mirroring support for static documents, we have also submitted a design for such support for dynamic documents.

1 Introduction

The overloading of site is often random. At present when a site gets overloaded it starts refusing any new connection from the client. This results in multiple tries from the client and increase in the response time to the user agent. Some of the sites which have high hit rates, maintain a mirror image of their documents at different geographic location. When a user requests a document from the main site, it gives a list of its mirror sites for the user to choose from. This solution to heavy hits is static. It does not adapt to the changing load of the server. The mirror sites are always there irrespective of the server load.

In this project, we propose a mechanism to deal dynamically with such unpredictable overloading of the system. When a site gets overloaded it can adapt to it, by seeking service of some other sites to get its documents mirrored. Apache is a http server available in the public domain and we have added modules, thus it can be made to work either as a rental server or the hot server. Specifically, the rental server will be able to accept request for mirroring support and process user requests for the documents of the hot spot server. The documents at the rental servers should be consistent with those at the original server. The static documents do not create much problems, but dealing with dynamic documents like SSI involves some tricks.

2 Three Entities

There are three entities involved in the exchange of documents: Rental Server, Hot Server or Client Server and User Agent like browsers. Rental Server are the sites that act as mirror sites on request. We added a module to the Apache, thus on loading the server it can be configured to act as a rental server. The Hot Server (or the Client Server) knows that they can get overloaded sometime in future and may require service of different rental sites to redirect clients. This again was implemented by adding a module to Apache. For the experiment we used Netscape Browser. Any User Agent which understands HTTP protocol could have been used. Figure 1 shows our design to achieve auto mirroring issues.

2.1 Rental Server

We see a situation, when there are many rental sites in the network, which an overloaded site can contact at time of needs. This server can serve multiple hot server at a time. It maintains information about the different hot server it is currently mirroring in its internal data structures. This helps in identifying the hot server that redirected clients, and in servicing different kinds of request from the hot server. The information may include resources being used by different hot servers, so as to charge them and do proper cleaning work when asked to do so. It provides three different kinds of interface to the hot server: Seek_Service, Don't_Need_Service and Update. The Hot Server sends request for mirroring of its documents. The Rental Server may either accept or deny the request. On acceptance, it will send a token to the Hot Server and wait for the file transfer. The token is used by the redirected client to identify the Hot Server from which it was redirected. On receiving a request from the Hot Server that it no longer needs its service, it does proper cleaning work. Also it may receive periodic update messages form different Hot Servers it is currently serving. In this case it must replace the old copy with the updated version. For the dynamic documents it may have to contact the Hot Server for any resources that is not available to it. This mechanism of reverse calling, from the rental server to hot server, has been called Callback. The idea has been borrowed from different Distributed File Systems. For the rental server, there is no difference between a redirected request and a direct one from some user agent.

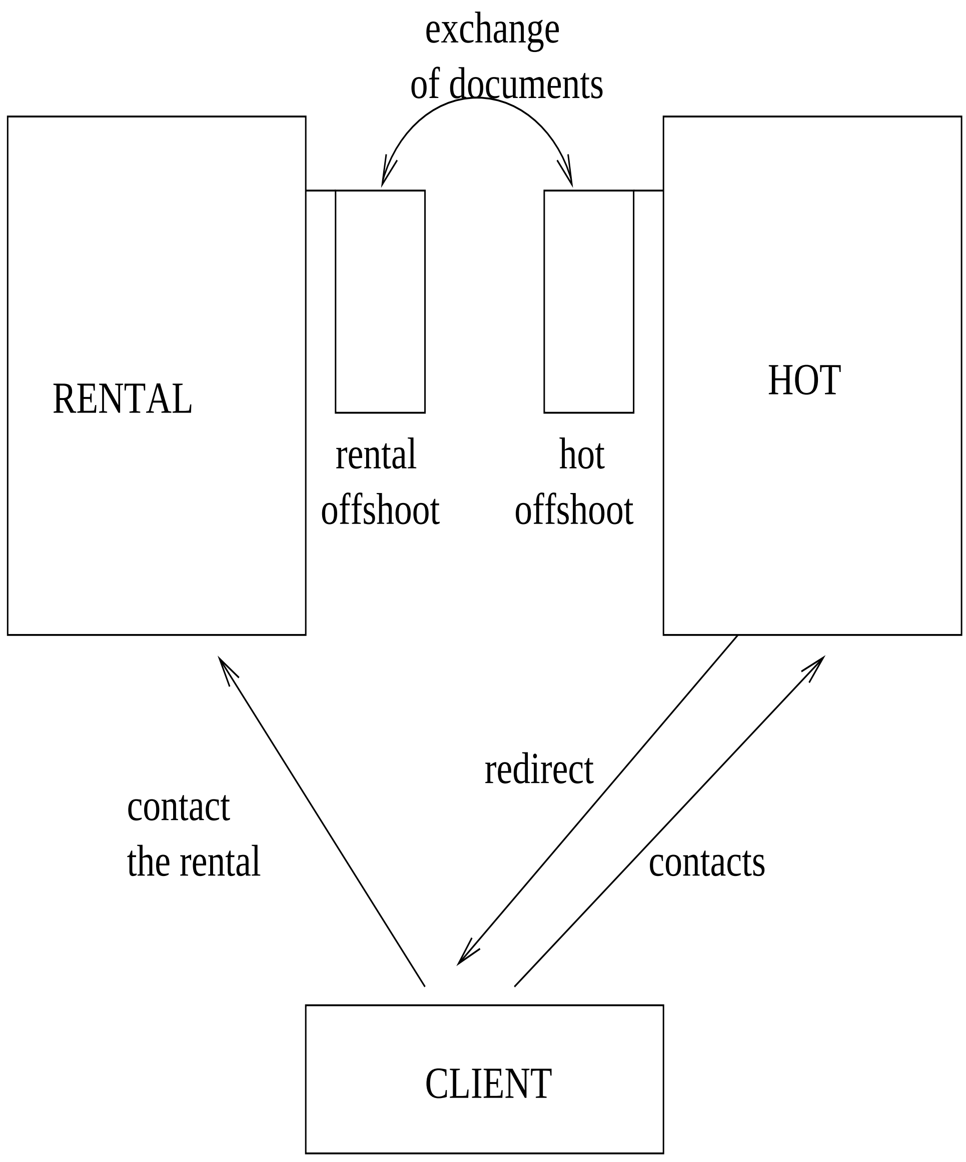

Figure 1: Three entities in our implementation.

2.2 Hot Server or Client Server

The entity that seeks the service of rental servers, when overloaded, is called a hot spot server, or hot server in short. The hot server has a list of rental sites in some priority, that it can contact at time of overloading. The number of rental site it chooses depends on its load. On decrease in load, it can inform the rental server that it no longer requires its service. This adapts gracefully to the changing load at the system. There are two threshold defined. If load is more than the upper threshold, the next rental server in the priority is contacted, and if it is less than the lower threshold, Don't_Need_Service message is sent to the least priority rental server among the list of rental server currently mirroring the documents. The values for two threshold should be chosen so as to prevent oscillations. If there is any modification to local files, it can send update message containing the new version of the file to all the rental sites that are mirroring it.

It might receive messages from the rental server for some resources required for sending dynamic documents to the clients. In such case it must send the required objects to the rental server. When a http request comes to the hot server from a client, it checks to see if there are any mirror sites for the requested document. If yes, it redirects the client to anyone of those site with the token given by the rental server to this hot server. Only the GET requests are redirected. The HTTP/1.1 specifically mentions that the user agent must not automatically redirect the requests other than GET or HEAD, unless it can be confirmed by the user. Some existing HTTP/1.0 user agents will erroneously change a POST request into a GET request, when automatically redirecting it after receiving a 302 status code.

2.3 Client or User agent

The client on getting a redirect message from a hot server, must contacts the rental server, without the end user knowing about it.

3 Implementation

For implementing our project, we chose Apache as it was available in the public domain. Furthermore most of the current http servers in the network are either a clone of it or its descendants. The modular structure and its extensive documentation helped in increasing its functionality. In the beginning we started with extending HTTP/1.1 so that the rental and the hot server can communicate. This proved to be a major constraint and then we modified our design by forking an offshoot process in both the rental and the hot server. This helped us in choosing our own protocol for exchange of information among these two, independent of HTTP. The rental offshoot waits for request from hot servers on a pre-assigned TCP Port different from the standard HTTP port. When a request comes from a hot server, whose documents it is already mirroring, it checks for the token and and takes appropriate action on the subdirectory corresponding to this hot server. If it is the first request, then it creates a new subdirectory corresponding to the hot server and returns a token that identifies this newly created directory.

There were many choices for the generating a unique token. It could have been a monotonically increasing sequence number. The unique token could either be associated with each file at different hot server, that is a function of the hot server and the file, or we could have a unique token for each hot server. The former case was more general, but required more processing at the hot server before redirection. The advantage was that it did not require copying of the directory structure of the hot server at the rental site. Also this would lead to selective mirroring of documents, only the really hot ones getting copied to the rental sites. The hot server must keep the association between its files, its mirror sites and token returned by those sites for the files. This token should be appended to the Location field in the HTTP response, along with the rental server address. The other option was to create a subdirectory corresponding to every hot server mirrored. The name could be same as the IP address of the hot server, and this can be used as a token by the client to identify the site from which it got redirected. This required copying the directory structure at the rental site, but no processing was required before redirection. We experimented with both case, and in the absence of a proper measure to measure the 'heat' of a document, settled for the latter case. This also reduced the processing time at the hot server for each client's request.

For automatic redirection, we used the Location field in the HTTP response with the returned status code of 302 implying temporary relocation of the requested document. The url in the location field contained the address of the rental server with the file path appended to it. The file path was same as the one in the client request, with only the hot server's IP address appended to it. This helped in the lookup of files at the rental server.

The offshoot of the Hot Server periodically measures the load of the system. For the load measurement, we had many metrics to choose from. One could have measured the number of disk IO's. But since in cases where only a small subset of document is really hot, the whole file could have been found in the cache for servicing the large number of request for it.

So we chose to measure the CPU utilization in the last time interval. If the utilization is more than the upper threshold, the subprocess would send a signal to the main process. The signal handler of the main process will check to see if there are still some rental site left whose service it can seek. If yes it will contact the one with the maximum priority among the sites yet to be contacted. If the subprocess finds that load has decreased below the lower threshold, it will send another kind of signal to the main process. The signal handler for this signal would check if there are rental servers which are mirroring its document, and if yes, it would contact the one with least priority to ask it do the cleaning work at its site. Since we chose the later option of having a token per hot server, this step was simplified.

3.1 HTTP headers

3.1.1 HTTP Request

```
GET /index.html HTTP/1.1
Host: romadur.cs.wisc.edu
```

3.1.2 HTTP Response (under normal circumstances)

```
HTTP/1.1 200 OK
Date: ...
Server: Apache/1.3.3 (Unix)
Last-Modified: ...
Content-Length: ...
Connection: close
Content-Type: text/html
```

3.1.3 HTTP Response (when overloaded)

```
HTTP/1.1 302 Found
Date: ...
Server: Apache/1.3.3 (Unix)
Location: http://128.105.119.109/128.105.183.19/index.html
Connection: close
```

```
Content-Type: text/html
```

Address of the rental server is 128.105.119.109, while the address of the hot server is 128.105.183.19. The requested file in the above example is index.html.

3.2 File Exchange Protocol

As for the file transfer between Hot Server and Rental Server, the concept is similar to Trivial File Trnasfer Protocol. Rental Server is waiting for any requests from Client Servers. When a Client Server requires service, for example, it seeks rental sides for acting as mirror sites, the connection is established. The Client Server sends "new" request, thus the Rental Server validates the client by checking if this client has requested before. If it has no record of this client, the Rental Server issues a grant message which contains token for identifing this client back. Client Server then send a message to indicate if the transfer is "continue" or "finish" after getting a grant message. The Rental Server sent ACK message to client and waiting for further data transfer unless the client's message is "finish". After that, the Client Server is responsible for locating which file it is going to send, if necessary, it has to start a file walk within a directory. The message order from the client is filename first, thereafter a sequence of 8k chunks of file contents, and the "eof" message if the file reaches the end. The Rental Server is in charge of creating the file within this client's file repository on disk. It will send ACK message after getting a filename message and eof message, but no ACK back with each 8K chunk data packet. By designing this, we make sure the start and end of the file transfer has been successfully done, without much worry of individual packet exchange since this conecction is based on TCP implemention. The Client Server may have other files within the requested directory need to be exchanged, thus it send "continue" message to the Rental Server, the same procedure above repeats. Once all files of requested service from the Client Server have been copied into the Rental side, the Client Server close the connection after giving a "finish" message to the rental side. The illustration of the protocol is shown in Figure 2.

3.3 Consistency

As is the case with any distributed system, where the files are copied at multiple sites, we had to deal with the consistency issues. But unlike most of them, consistency was not a problem in our case. As there was only one

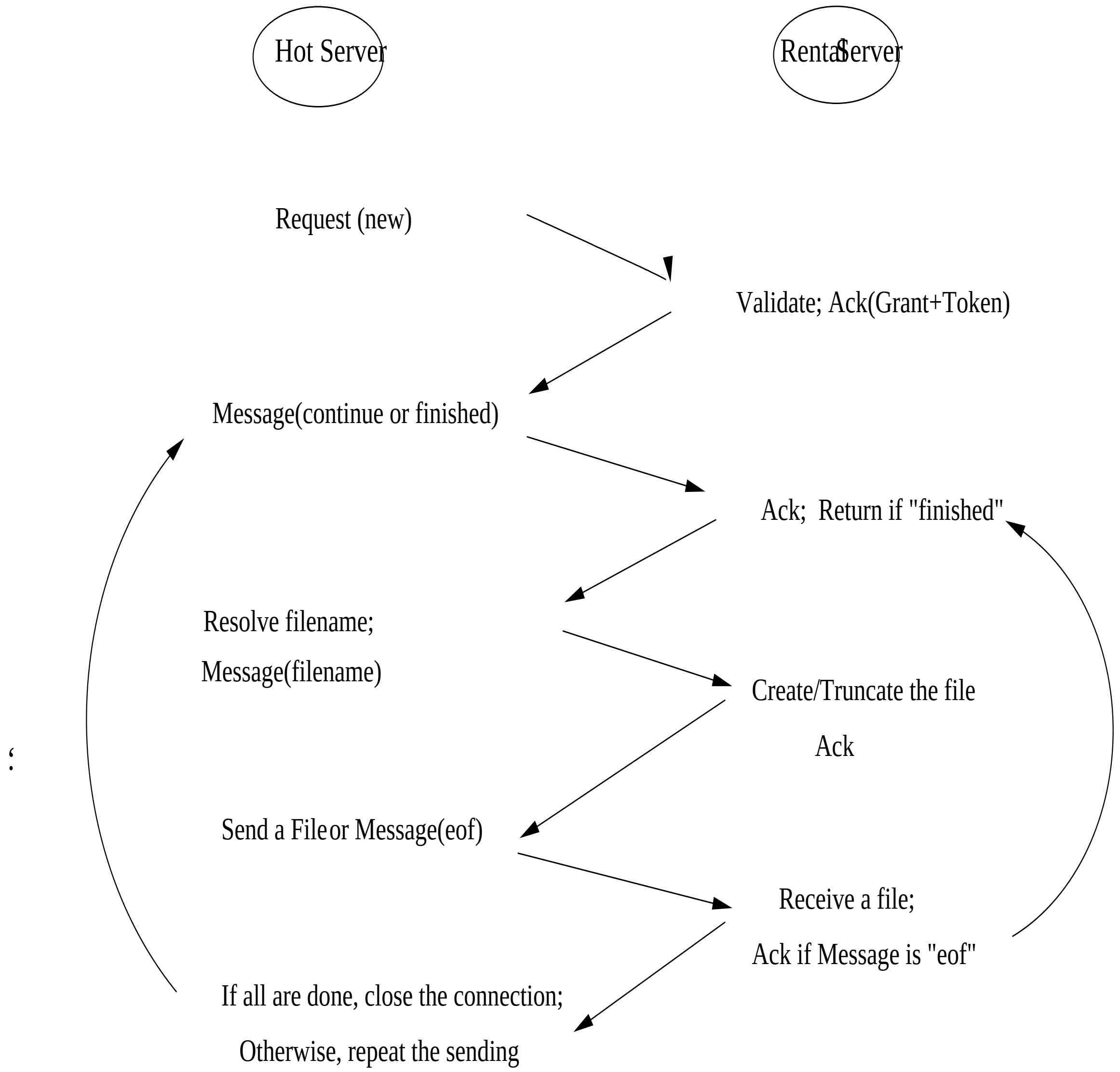

Figure 2: Illustration of File Exchange Protocol between Hot Server and Rental Server.

actual owner of the files and only this (hot site) could modify the files, we chose the write through policy. As soon as the files got modified at the hot server, an update message with the new version of the file was sent to all the mirror sites. The other option could have been to do a periodic polling and check the last modified time of the files. Besides providing a weak semantics for the consistency, it would have greatly increased the load of the already overloaded system. For implementing the former design, we were looking for support from the operating system, such that when a file of interest to the server gets modified the server must be informed about it. But in the absence of such support, we had to externally send signal to the server on modification to a file.

3.4 Dynamic Documents

There are many different ways of producing dynamic html documents depending on the customization of the client. Most of these methods require resources (executable files, etc) which may not be available at the rental site. For example if the rental site is architecturally different from the hot site, copying executables wouldn't have made any sense. The different kinds of dynamic document support includes SSI, Fixed CGI, Parameter-variable fixed CGI and Parsed-parameter variable CGI. We started working with Server Side Include (SSI). There is SSI parser at the server end which looks for SSI commands embedded in the html documents and takes actions depending on them. These embedded SSI commands may include executable which is not located at the rental site. In such cases the rental site instead of throwing error, must contact the hot server with the name of the executable along with its arguments (if any). The hot server on receiving such callback message from the rental server, must process the request and send results to the rental server, which can slice them into the html document before sending it to the client. This part of the protocol between the rental server and the hot server, is yet to be fully implemented.

4 Conclusion

Our project shows that auto-mirroring is a very clean method for maintaining the response time to clients inspite of increase in load at the server. There are many functionality that can be added, depending on the specific situation. Selective mirroring of hot documents and extensive support for dynamic documents, being among others. Also we feel that number of user-defined signal supported by OS should be more than two (SIGUSR1 and

SIGUSR2). OS should also provide a mechanism, where a process can register its files of interest, and it should be immediately notified if there are any updates to these files.

5 Acknowledgements

We sincerely thank Prof. Cao and Jussara Almeida for their valuable suggestions from time to time.

6 References

1. *Getting started with on-the-fly HTML. Dr. Clue.* http://www.netscapeworld.com/nw-05-1997/nw-05-clue.html

2. *Apache Server.*http://www.apache.org

3. RFC 2068, *Hypertext Transfer Protocol – HTTP/1.1*

4. Richard Stevens. *Unix Network Programming.* Prentice-Hall. 1996.

5. Cherkasova, Ludmila and Peter Phaal. *Session Based Admission Control: a Mechanism for Improving the Performance of an Overloaded Web Server.* Hewlett-Packard Laboratories.