Design of a Remote Controlled Caching Proxy System: Architecture, Algorithm and Implementation

Khaled E. A. NEGM and MARYAM A. AL-ALY
Etisalat College of Engineering
Sharjah, POB 980
UAE

Abstract: - A caching proxy server acts as an invisible intermediary between browsing clients and the internet servers. In the case of a Web cache, cacheable objects are always separate and are always read in their entity with no pre-fetching. In the present study we present a novel design for a system to remote control an array of proxy system. The administrator can monitor and configure the caching array system from any normal client computational facility. The current system emphasizes on the fact that the administrator can change the configuration of the system with no need either to start the system or alter the clients' statuses. This is achieved by implementing the concurrency control system under the system hierarchy. The primarily local testing of the system shows promising results to get implemented on large scale of enterprise systems..

Key-Words: - : proxy, cache, web caching, least recently used (LRU), proxy array selector

1 Introduction

The World Wide Web has become the general environment for information sharing. The increasing usage of the WWW results in the network congestions and the slow response. The unparalleled growth of the Internet in terms of total bytes transferred among hosts; coupled with the sudden dominance of the HTTP protocol; suggest much can be leveraged through Web caching technology [1-4].

User-perceived retrieval latencies in the World Wide Web can be improved by preloading a local cache with resources likely to be accessed. A user requesting content that can be served by the cache is able to avoid the delays inherent in the Web, such as congested networks and slow servers. The difficulty, then, is to determine what content to prefetch into the cache.

Caching proxies are usually installed where many clients are accessing the Internet through a single path. A proxy keeps a copy of objects (most of the time) of Web documents and FTP downloads that passed through it. When a client requests an object from the network, the proxy may be able to serve a local copy of that object instead of fetching the object from its original source. By serving a local copy of an object, a proxy saves bandwidth on outbound network connections and potentially reduces response time. There have been many works regarding the proxy caching algorithms to achieve such goals and there are several cache implementations available from software and appliance vendors as well as the Squid open source cache software, which is designed to run on Unix-like systems only [5].

The Remote Controlled Caching Proxy Server is a novel approach that aims to design a caching proxy server that can be configured remotely over the Internet and can run on any machine under any operating systems. The system is capable to support an array system of proxy servers and it is a caching protocol independent [7-9]. This makes it independent on the vendor, operating system and the caching protocol in operation.

The purpose of the current work is to achieve the following tasks: *First*, facilitate a remote administration of the proxy server while running from a "normal client" computing facility (as an administrator) through a user-friendly graphical user interface (GUI) while serving transparently several clients concurrently. *Second*, enhance overall performance by using cache [10-12]. *Third*, use Java technology for portability between operating systems.

2 System Overview

The current system under study is designed as an object oriented structure and implemented in Java. The system is structured in a modular form to guarantee portability, manageability, maintainability and to perform best efficiency. The system utilizes the client-server paradigm where each connection is passed to a separate thread that is handled exclusively. Each module has its own

design, implementation and process control.
The system is composed of two main parts: the *first* part is the "main application", to run the proxy engine. The *second* part is the "remote administration". The second part depends on built-in data types of Java to simplify the development process and simplify the codes [13-15].
The administrator can perform all of the proxy management processes remotely from any computing facility.

2.1 The Main Proxy Application

The main proxy consists of a portable Java proxy, including threading, configuration and caching modules. These components are sufficient to accomplish all the tasks if there are no network restrictions to block the proxy requests. The main proxy module receives an HTTP proxy request and return the data associated with the request on behalf of the host site. In case of an HTTP error message, it propagates as if no proxy is present to the requesting client from one of the caches present within the caching array [16]. If the requested object has been previously cached to the direct caching server attached to the client system, the object will be returned. In this case an update will be performed to register an update of the requested object by date, time, size, number of requests, and type, etc. (including all the characteristics flags given to the object for future usage) [17].
If the object is not present within the cache server attached to the client system, the request is forwarded to a main proxy director (caching director) leading to the other proxy that do have the object existed within its storage domain. The feature of forwarding requests to another web proxy, along with caching behavior, enables one to use a hierarchy of caching proxy servers, which reduces the latency for the requested object [18-20]. If the requested object could not be found throughout the cache array, the request is finally forwarded to the web server, which delivers the object back through the chain of proxies, enables each of them to cache the replay for future use.
If this is not the case, i.e., the object is not currently present in any of the caches within the array, the request is forwarded to the internet and once on getting a reply, the previous step will be performed accordingly. This is of course in parallel to processing the client request to browse the object. All theses steps are performed in transparent, parallel, with no latency sensed of the normal browsing client from any computing system served by this array.

The proxy application contains the following components: `daemon`, `cache`, `configuration`, `administration`, `proxy`, `httpRequestHdr` and `httpReplyHdr`. In the following we will state the functionality of each component of the main proxy architecture.

`Daemon:` This is the main thread that launches the application and never dies. At its start it performs a general initialization such as creating the `config`, `cache` and `admin` objects as well as creating a special socket of type `ServerSocket` in java. This in addition to it enters an endless loop of listening to the main socket including for each request it creates a new thread called a proxy thread.
`proxy`: This component handles all communication between client and proxy and between proxy and web servers. It posses all the fundamental proxy function as disused before. Once it finishes its thread gets terminated.
`cache`: This component encapsulates all the details of caching from the other components in the application (i.e., `DocumentInfo`). When it is executed to cache an object, first, it generates a file name out of the URL of the object and enumerates the hash table that has two fields. The *first* field is the filename (the `key`) and date (the `value`). If the file name is found in the cache it will be returned it to the proxy thread; otherwise, it will return a status message indicating 'not cached' and off course will be cached.
`config`: When the administrator changes proxy parameters and sends it to main application, the applet will forward the object to the proxy and proxy gets it. Consequently, the proxy alters its behavior. To support both the proxy and the applet to process the job of getting and setting parameters; this component is designed. It will appear in both the applet and proxy code and handle all get/set methods involved with the configurable parameters.
`admin`: This thread handles all communications with remote administrator. It creates a socket and listens on a special admin port.
`httpRequestHdr:` This class is used to help proxy thread using HTTP protocol. It creates the http header fields upon sending.
`httpReplyHdr`: This class receives the http header fields upon arrival. For example, if the proxy fails to forward a request because the web server could not be reached, it generates an HTML web page with a proper message and constructs an

HTTP message to send back to client, informing him of the error and giving him the correct headers of this error.

The cache handles `DocumentInfo` objects that correspond to files in the proxy director local cache directory and contain information about their creation date, size, and number of times they have been referred to a `DocumentInfo` object is created and inserted into the cache after the successful receipt and saving of the desired document.

In this case of a newly received documents are always copied to into the cache (they replace older versions if such exist). Since storage space is bounded, when there is no room to copy a new document, cache cleanup must be performed. In our application, the upper bound to the size of the cache directory is determined by the High Water Mark parameter. The current size of the cache directory is indicated by the Current Water Mark parameter. Upon cleanup, documents are removed until the size of the cache directory is below the Low Water Mark parameter. The High and Low water marks are customizable. The algorithm that chooses documents for removal is called the removal algorithm. The algorithm that we chose to implement is the Least Recently Used (LRU) algorithm [21-22]. The LRU algorithm is used when the cache is full and a file needs to be deleted.

2.2 The Remote Administration System

The *second* part of the system is the remote control part. The part is implemented as a Java applet to enable the proxy administrator to remotely manage the proxy from his or her machine via the web browser. The remote administration contains the following components: `administrator`, `configuration`, and `AuthenticationDialog`.

In the following we will define the functionality of each module within this part:

`admin` thread: This thread handles all communications with remote administrator. When the administrator accesses the proxy, the proxy thread traps the event and sends it back to get displayed at the admin web page with the admin applet. The applet starts by presenting an authentication dialog box and waits for the administrator to enter the appropriate credentials needed for this process, e.g, ID, and password. Then it sends these credentials to the proxy application (to the admin port) and thus the admin thread at the main application can copy that request without interfering the handling of all other clients' requests. The admin thread in the main application processes the password and sends back an authentication acceptance to the applet, on the admin port, an answer (Ack/Nack).

`config`: This component is same as `config` component in proxy application.

`configDialog`: This component displays the configuration dialog box. This allows the administrator to allocate the current configuration and change it.

`AuthenticationDialog`: This component displays the dialog box with message "wrong authentication" if the administrator enters incorrect credentials.

This part is a platform specific GUI configuration tool allowing configurations and options to be changed on the remote proxy server on-a-fly using java applet implementations. This human user interface will accept a user-friendly input and forward it to the main proxy director for updates. If the administrator wants to change proxy configuration, he or she can access any of the proxy servers present in the array by requesting the proxy machine via its authenticated address and credentials, i.e, IP address, MAC address, digital certificate, SSL3, VPN connection or cross site certification. In our case we started by a very simple step, that is the IP address or the machine URL address window in the browser plus the suffix '/admin' or based on any person/machine authentication mechanism on the implemented within the system.

The browser considers this to be valid client request, and forwards it on to the proxy or the proxy array. When the proxy monitors this request, it compares it to the IP address of the host machine on which it runs. If they do not match the proxy forwards this as normal request on to the main proxy. But if they do match, the proxy assumes that an administrator is requesting to login. The module responses by sending back a Java applet which request an authentication processes from the administrator which is implemented in the `AuthenticationDialogue`. Once it the administrator inputs the credits, the processes reaches the proxy application control program. Once passing the authentication session, the proxy replies by an acknowledgement (ACK) to the applet, and the applet responses by presenting all the parameters and status of the proxy. This enables the administrator to alter parameters, thus

change the proxy behavior and sending the new parameters to the proxy. The status of the changes will be confirmed and used to display the current proxy status and parameters. This will ensure that the indicated configuration is truly the configuration being executed.

3 Concurrency Control and Proxy Hierarchy Advantages

Several data structures are common to all running threads, and must be protected to ensure their consistency. The cache, which handles `DocumentInfo` objects that correspond to files in the cache directory, features a two-level locking scheme that enables concurrent reading, writing and checking the existence of documents in the cache, while retaining the consistency of the structure. A thread that wishes to insert a document into the cache must perform the following actions:

a. Save the appropriate file to the cache directory.
b. Acquire a lock on the cache.
c. Insert a corresponding `DocumentInfo` object into the cache.
d. Release the lock on the cache.

For a thread that wishes to read a cached document must perform the following actions:

a. Acquire a lock on the cache.
b. Retrieve the appropriate `DocumentInfo` object from the cache.
c. Release the lock on the cache.
d. Acquire a lock on the retrieved `DocumentInfo` object.
e. Read the corresponding file.
f. Release the lock on the `DocumentInfo` object.

For a thread that wishes to check if a document exists in cache must:

a. Acquire a lock on the cache.
b. query the cache.
c. Release the lock on the cache.

This is a novel scheme that will afford several properties and enhancements, the *first*, are that, several files may be read and written to the cache directory concurrently. Concurrent reading is enabled as different threads may hold locks on different `DocumentInfo` objects simultaneously and read the corresponding files. Note that when a specific `DocumentInfo` object is held by one thread for reading, other threads that need the corresponding document will wait in queue for it to be released. Concurrent writing is enabled as no locking is involved during the writing of the files to the cache directory. Note that a corresponding `DocumentInfo` object is copied into the cache only after the completion of the writing of the file, and therefore documents may be read from the cache only after they have been successfully written to the disk.

The *second* is the consistency of the cache is kept during cleanup. The cleanup procedure is invoked when an attempt to insert a new document into the cache fails due to lack of room. At this point, the cache is held by the thread that attempted the coping, and thus other threads may not access the cache until cleanup is done. Nevertheless, some cached documents may be in the midst of their reading. However, their corresponding `DocumentInfo` objects are locked, and if the cleanup procedure, which must acquire a lock over the `DocumentInfo` object before removing the document, wishes to delete one of them, it will wait on it until reading is done and the lock is released.

Third, that several other structures are common to all threads, namely the servers statistics database, the list of servers that are accessible without proxy, the configuration object that stores all the parameters of the application and the components of the graphical interface. Synchronization of threads accessing these objects is achieved simply by requiring the acquisition of a lock on the object before using its services, and releasing the lock when done. Threads hold the discussed objects for negligible intervals of time as they only offer simple operations and they all reside in main memory; thus, the scheme will not affect parallelism of the application to any significant degree.

On of the advantages of the current implementation, is the usage of a hierarchical proxy system. In which the administrator can access each proxy uniquely and alter its behavior without affecting other proxies. In case of a single proxy X, only one administrator can control it at a given time, thus protecting proxy X from being handled by different administrators at the same time.

All clients' requests plus the administrator remote configuration are done in parallel. This facilitates the application to be multithreaded. For instance, in a certain time frame, the proxy can handle a request from client A, a request from client B, two requests from client C (this could be happen because a modern browsers use threads too) and

Table 1: Remote Proxy Cache Control Algorithm

```
Read Http Request from clientIf request is from administrator then
       Send web page with appletElse
       If URL is accessible then
          If web page is cached then Get page from cache and send it to client
          Else
   Open socket to proxy or web server

          Send URL
          Send Http reply to client
       If web page is cacheable then Cache the web page
       End if
   End if
End if
```

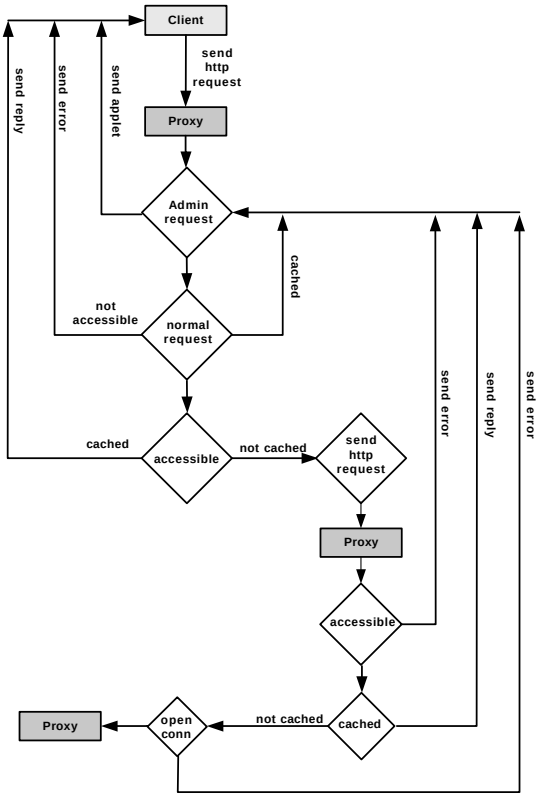

Figure 1: Block Diagram for the Flow control of the System Configuration and Functionality

communication with a remote applet running on the administrator machine. None of these users would notice the difference. The control algorithm is shown in table 1. The block diagram of the system along with the control flow of data is shown in figure 1.

4 Summary and Conclusion

In this work we present a novel design for a remote control of caching proxy arrays. The advantage of this is that it can be configured remotely by its administrator over the Internet from ant normal computing facility. The system contains two main parts: a main application for the proxy engine, and an applet for remote administration. The design of the system is an object-oriented and multithreaded implementation using Java. This will ease the control of the system as a platform independent function. One of the advantages of the current system is that it allows concurrent administrators to configure the system in parallel with affecting each other or affecting the clients' operations. Such functionality was achieved by using a two-level-lock algorithm.

The primarily testing of the system is performed in a simulated environment and showed proper functionality. Also it is tested with a local environment (loop back) and showed promising results.

Within these testing the server is able to cache the documents, handle multiple clients' requests at the same time and forward the request to other proxies or to web servers. The main goal of this project is to allow remote administration of the proxy server, while running, from a client machine through a user-friendly graphical user interface (GUI) via web browser [23-25].

Future Work

Currently the system is under full review and testing in a real environment. Also some other improvements are in progress for the `authenticationDialogue` to afford full secure functionality in relation to the sensitivity of data processed [26-27].

References

[1] D. Wessels, K. Claffy, Internet Cache Protocol (ICP), version 2, RFC 2186, 1997.

[2] L. Fan, et. al., Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol, Proc. of SIGCOMM'98, 1998, pp. 254-265.

[3] O. Sptscheck et. al., Optimizing TCP Forwarder Performance, IEEE/ACM Transactions on Networking, Vol. 8, 2000, pp. 146-157.
[4] B. Davison, A Web Caching Primer, IEEE Internet Computing, Vol. 5, 2001, pp. 38-45.
[5] Squid Web Proxy, http://www.squid-cache.org/.
[6] V. Duvvuri, Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web, Master's thesis, Univ. of Massachusetts, 1999.
[7] Khaled E. A. Negm, Distributed Proxy Cache Cluster Optimization Simulation System, WSEAS Transactions on Computers, Vol. 3, 2004, pp. 1161-1166.
[8] Khaled E. A. Negm, CARP Compliant Proxy Enforcer Frame Work, Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI 2003), 2003, pp. 118-124, 2003.
[9] Khaled E. A. Negm and Maryam A. Al-Aly, Design and Implementation of An Intelligent Proxy Server, Proc. of the 14th International Conference on Control Systems and Computer Science- CSCS14- 2003, pp 329-333, 2003.
[10] B. Davison and B. Wu, Implementing a web proxy evaluation architecture, In Proceedings of the 30th International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems (CMG), 2004.
[11] J. Almeida, V. A. F. Almeida, and D. J. Yates. Measuring the behavior of a World Wide Web server, Proceedings of the Seventh Conference on High Performance Networking (HPN), 1997, pp. 57–72.
[12] J. Almeida and P. Cao. Measuring proxy performance with the Wisconsin Proxy Benchmark. *Computer Networks and ISDN Systems*, Vol. 30, 1998, pp. 2179–2192.
[13] S. Freund, and J. Mitchell, A Type System for the Java Bytecode Language and Verifier, J. of Automated Reasoning, Vol. 30, 2003, pp. 271–321.
[14] G. Bigliardi, and C. Laneve, A type system for JVM threads, in Workshop on Types in Compilation, 2000.
[15] G. Czajkowski, and T. von Eicken, JRes: A resource accounting interface for Java, Proceedings of the ACM Conference on Object Oriented Languages and Systems, 1998, pp. 21–35.
[16] C. Zhang, X. Zhang, Y. Yan, Two fast and high- associativity cache Schemes, In: IEEE Micro, Vol. 17, 1997, pp. 40-49.
[17] R. Wooster and M. Abrams, Proxy Caching that Estimates Page Load Delays, WWW6, Computer Networks and ISDN Systems, Vol. 29, 1997, pp. 1497-1505.
[18] P. Rodriguez, et. al, Analysis of web caching architectures: hierarchical and distributed caching, IEEE/ACM Transactions on Networking (TON), Vol. 9, 2001, pp. 404-418.
[19] H. Che, et. al, Hierarchical Web Caching Systems: Modeling, Design and Experimental Results, IEEE Journal on Selected Areas in Communications, Vol. 20, 2002, pp. 1305-1314.
[20] A. Fiat, R, Karp, M. Luby, L. McGeoch, D. Sleator, N. Yong, Competitive paging algorithms, Journal of Algorithms, Vol. 12, 1991, pp. 685-699.
[21] P. Barford and M. Crovella, Generating representative Web workloads for network and server performance evaluation, Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 151–160, 1998.
[22] A. Rousskov. Web Polygraph: Proxy performance benchmark. Online at http://www.web-polygraph.org/, 2004.
[23] J. Almeida and P. Cao, Wisconsin Proxy Benchmark 1.0. Available from http://www.cs.wisc.edu/~cao/wpb1.0.html, 1998.
[24] Y. Sato. DeleGate home page. Online at http://www.delegate.org/, 2004.
[25] J. Dilley and M. Arlitt, Improving Proxy Cache Performance-Analyzing Three Cache Replacement Policies, IEEE Internet Computing, Vol. 3, 1999, pp. 44-50.
[26] G. Barish and K. Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Communications Magazine Internet Technology Series, Vol. 2000, pp. 178-185.
[27] M. Busari and C. Williamson, Simulation Evaluation of Web Caching Hierarchies, Proc. IEEE MASCOTS 2001, 2001, pp. 379-388.