A New Architecture for HTTP Proxies Using Workstation Caches

Author Names:

Prabhakar T V (tvprabs@cedt.iisc.ernet.in)
R. Venkatesha Prasad (vprasad@cedt.iisc.ernet.in)
Kartik M (kartik_m@msn.com)
Kiran K N (kiran@virusinternals.com)

Corresponding Author: R Venkatesha Prasad

Address: Centre For Electronics Design Technology,
Indian Institute of Science,
Bangalore 560 012, India

Phone +91-80-3092732
Fax: +91-80-3600808
A New Architecture for HTTP Proxies Using Workstation Caches

Prabhakar T V*, R. Venkatesha Prasad*, Kartik M and Kiran K N

*tvprabs@cedt.iisc.ernet.in, *vprasad@cedt.iisc.ernet.in, kartik_m@msn.com, kiran@virusinternals.com

Abstract

The Web connectivity to the Internet in LANs is usually through an application level Gateway such as a proxy. In LANs fetching time for a Web-Page is mainly dependent on the performance of the Proxy and the size of cache at the proxy, apart from the link speed to the Internet. A new architecture for Web Caching in LAN environments is proposed in this paper reduces the fetching time and scalable as well. In specific, Java Servlet based Web “Proxy Agent” is implemented with LAN workstation browser caches so that the workstations’ caches would become a part of the setup. The agent itself is an ICP [1, 2] enabled cache-less proxy. The proposed architecture consists of a number of Caching servers running on workstations managed by this central agent, which results in increased effective cache in the LAN. This ICP enabled “Proxy Agent” maintains information regarding web content in the Caching servers. On request to the agent via Caching servers, a hashing search mechanism is implemented on central database held by the ”Proxy Agent” and then, requested URL is fetched from the peer workstation cache thus reducing the fetching time. Communication between Caching servers and our “Proxy Agent” is done on UDP. This communication enables a number of operations on the database including, ‘Cache Hit’, ‘Cache Miss’ and ‘Cache Replace’. The results are discussed in detail that show a reduction in the fetching time. The implementation of the new architecture is already functional.

1. Introduction

Users on a local area network (LAN) usually access the Internet (the web) through a proxy server or a web cache, which stores copies of popular web objects. In this paper we describe a method for using workstation browser caches with HTTP [3] proxies. For the clients on the network, the proxy server represents the Internet and to the Internet, the same proxy server acts on behalf of all the clients on the
network. It intercepts all requests to the Internet for web pages, etc and tries to fulfill them locally through the cache. In the case of a cache miss, the page is fetched from the Internet. The proxy server stores a local copy of all such pages and for subsequent requests for the same page it is served from its local storage rather than fetching it again from the Internet.

It is not only on local networks that proxy servers are installed. They may be implemented across the Internet to make Internet access faster by effectively reducing the Round Trip Time (RTT) between requesting clients and origin servers. ISP’s may implement proxy servers locally to provide faster access to their clients and also conserve bandwidth.

In this work, the Proxy Agent’s cache is not implemented; instead, a distributed caching is implemented by including LAN workstations’ browser caches. Hitherto, the workstations’ cache were used only by the user on the respective workstations for the session. It is proposed in this work that, these caches are brought into the purview of the existing Proxies with the new functionality of building the HTTP connection between workstations. It also, regards these caches as its own cache when communicating with ICP enabled parent and siblings. This modified Proxy is named as **Proxy Agent**. (Agent or PA)

In this scenario, a **URL Database** maintained by the PA facilitates HTTP request processing between peer workstations. In the event of a “Cache Hit”, a requesting client contacts its peer to complete the processing. Prior to peer contact, the requesting client dispatches a UDP [9] packet with the URL as payload. In the event of a “Cache Miss”, the agent functions as a normal HTTP proxy and the new page fetched will be cached in the respective workstation cache.
In § 2, we describe the proposed architecture for previously described above scenario. In § 3, we propose an implementation of the hashing mechanism used for access, insert, delete and update operations within the “URL Database”. In § 4, we describe the power we exploited by using Servlets for implementation purposes. In § 5, the experimental setup for implementation of the proposed architecture is explained. Section 6 contains in depth discussion of the implementation. In § 7, we demonstrate the user request flow by way of Time Sequence Charts. In § 8 Results of our measurements and increase in response times when user connects to typical Internet Links such as 64 Kbit/s and 128 Kbit/s is discussed in detail and finally in § 9 we conclude our work.

2 Proposed Architecture

The architecture as shown in Fig 1 consists of a high-speed LAN, which contains a number of caching servers. The agent functions as a central controlling agent. The initial interaction is always though the agent. The agent may transfer control to the “caching servers”. The siblings and the parent are a part of the existing cache hierarchy as explained in RFC 2187.

![Proposed architecture for usage of workstation browser caches.](image-url)
Proxy Agent performs the following functions

A. Making use of Workstation Browser Caches:

The agent is aware of the contents of each workstation browser caches, by a URL database maintained within the PA. If requests for web objects from a requesting client are contained in the cache of another, the PA redirects the requesting client to its peer to complete the processing.

B. Maintaining a URL Database

A URL Database is maintained by the PA to facilitate HTTP request processing within peer workstations and other ICP peers. The database contains the URLs of all pages in the client caches along with the corresponding hostname. Operations on the database include ‘search’ for URL, ‘delete’ for deleting URL entries and ‘insert’ for new URL entries.

C. ICP Queries:

ICP is a message format used for communicating between caches. The agent exchanges ICP queries and replies with siblings and parent to determine the most appropriate location from which to retrieve an object as in conventional proxies.

D. Conventional Proxying:

In the event of a ‘cache miss’ the PA automatically enables conventional Proxying, to fetch the requested page.

E. Denying access to Restricted sites:

Initially, all requests need to pass through an Access Control List (ACL) database. If a match is found, access to the site is denied.

3 Hashing Mechanisms

Hashing mechanism permits a record to be retrieved from a table directly without any comparisons. A record is positioned in the table at a location reserved for a particular value of the record key. Simply using the key as an index to the table retrieves the record. A function that transforms a key into a table index is called the hash function. If ‘h’ is the hash function and ‘key’
is a key, \( h(key) \) is called the hash-key and is the index at which a record with the key ‘key’ should be placed. If two distinct keys hashes to the same index, such a situation is called a hash collision or a hash clash. Mechanisms to deal with a hash clash include rehashing, chaining, open-addressing etc. [4]

We propose and implement a mechanism for hashing within our PA to perform various operations on the URL Database. These operations include search for a URL, inserting an URL and delete an URL. To convert the URL’s from character form to numeric form to form a hash-key, the mapping table 3.1 is used. For other special characters, the ASCII equivalent of that character is used.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>C</th>
<th>d</th>
<th>e</th>
<th>F</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>j</th>
<th>k</th>
<th>l</th>
<th>m</th>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>107</td>
<td>108</td>
<td>109</td>
<td>110</td>
<td>111</td>
<td>112</td>
<td>113</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
</tr>
</tbody>
</table>

Table 3.1: The mapping table used to generate the hash key

We used the above mapping table due to ease of implementation. All hashing are done only on the host name of the URL, which is stripped of the ‘www’ tag, and the domain name. For example, If the URL is http://www.virusinternals.com/mail.html, then hashing is done only on the string “virusinternals”.

We illustrate the computation of the hash-key for a URL with the help of an example. Suppose a hash key for the URL http://www.hotmail.com/main.html is to be found, computation takes place as follows:

1. The entire URL is converted into lower case.
2. The URL is stripped of it’s www tag, domain name and URI and the string “hotmail” is considered
3. The mapping table of fig is used to map the string “hotmail” to string “1041111610997105108”

4. This numeric string is split into numeric substring of length 5 (Except the last substring whose length is <=5)

5. i.e. “10411”,”,11161”,,”09971”,”05108”

6. All the numeric substrings are converted to “Number” form.

7. Then all of them are added and “MOD” with 99997¹ which results in the hash-key value of 36651.

Hash clashes are resolved by ‘overflow bucketing’, also known as ‘separate chaining’. In case of collision, we append the record, followed by a separation character ‘|’ after the End-of-line (EOL). The records in the database are homogeneous and contain two fields namely the URL and the hostname of the machine, which has that URL as shown in Tables 3.1 and 3.2. The structure of URL database is as shown in Table 3.3.

<table>
<thead>
<tr>
<th>url</th>
<th><a href="http://www.virusinternals.com">http://www.virusinternals.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>edpc303.cedt.iisc.ernet.in</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.1 – Record Structure</th>
<th>Table 3.2 – Example of Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>hostname</td>
</tr>
<tr>
<td><a href="http://url1.com">http://url1.com</a></td>
<td>edpc1.cedt.iisc.ernet.in</td>
</tr>
<tr>
<td><a href="http://www.virusinternals.com">http://www.virusinternals.com</a></td>
<td>edpc1.cedt.iisc.ernet.in</td>
</tr>
<tr>
<td><a href="http://url1.com/index.html">http://url1.com/index.html</a></td>
<td>edpc11.cedt.iisc.ernet.in</td>
</tr>
</tbody>
</table>

Table 3.3 – Structure of URL Database

4 Java Servlets

We used Servlets⁵ [6,7] to implement our PA, thus showing that Servlets are no-more limited to only database applications, can also be used to implement proxy servers. We demonstrate that it is still possible to build application servers with Servlets. A Servlet’s initialization code is executed only the first time the Web server loads it. The power of java Servlets lies in the fact
that it runs inside a Java Virtual Machine (JVM) on the server. So, it is safe and portable. Servlets are used in lieu of server side CGI scripts and expand the functionality of a server. Unlike CGI, which use multiple processes to handle separate programs, separate threads within the web server process handle all Servlets. This means that servers are efficient and scalable. Because Servlets run within the web server, they can closely interact with the server to do things that are not possible with CGI scripts. We believe that Java Servlets offer the best possible platform for web application development. Further, Servlet invocation is highly efficient. Once a Servlet is loaded, it generally remains in the server’s memory as a single object instance. Thereafter, the server invokes the Servlet to handle a request using a simple, lightweight method invocation. VQServer [5] is used as web server. The modules, ‘ICP Servlet’ & ‘Proxy Servlets’ using Servlets and the modules ‘URL Locator’ & ‘ICP Replier’ (see § 6.1) using JDK [10] are implemented. After the Servlet is loaded, calling a service method does handles new requests. This is a more efficient technique than loading a completely new executable with every request.

5 Experimental Setup

Our experimental setup consists of at least 200 computers running caching servers within our LAN. This was to verify the scalability of our architecture. However, measurements were performed using six machines whose configuration is as shown in Table 5.1 and Table 5.2. Furthermore, we show the inter-working of our Proxy Agent with popular HTTP Gateways such as SQUID[8]. ICP protocol enables such a communication between neighbor caches. The actual setup is shown in Fig 2. In Fig 2, the systems shown in Table 5.2 are connected using an 10/100 Ethernet switch.

---

1 It is found out that ‘mod’ with a prime number minimizes collisions and spreads the records somewhat uniformly
2 http://www.sun.com/products/servlet
Table 5.1 – Software Configuration of setup

<table>
<thead>
<tr>
<th>Machine Name</th>
<th>Operating System</th>
<th>Supporting Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-blr</td>
<td>REDHAT LINUX 6.1</td>
<td>Squid</td>
</tr>
<tr>
<td>Kavya</td>
<td>SLACKWARE LINUX 7.1</td>
<td>Squid</td>
</tr>
<tr>
<td>Sadhana</td>
<td>RED HAT LINUX 7.1</td>
<td>Squid</td>
</tr>
<tr>
<td>Edpc1</td>
<td>WINDOWS 95</td>
<td>VQServer + web server</td>
</tr>
<tr>
<td>Edpc303</td>
<td>RED HAT LINUX 6.0</td>
<td>Cache Server + Browser</td>
</tr>
<tr>
<td>Edpc105</td>
<td>WINDOWS 95</td>
<td>Cache Server + Browser</td>
</tr>
</tbody>
</table>

Table 5.2 – Hardware Configuration of setup

<table>
<thead>
<tr>
<th>Machine Name</th>
<th>Designated Role</th>
<th>RAM (MB)</th>
<th>Processor (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-blr</td>
<td>PARENT</td>
<td>128</td>
<td>PIII 860</td>
</tr>
<tr>
<td>Kavya</td>
<td>SIBLING#1</td>
<td>128</td>
<td>PIII 650</td>
</tr>
<tr>
<td>Sadhana</td>
<td>SIBLING#2</td>
<td>128</td>
<td>PIII 620</td>
</tr>
<tr>
<td>Edpc1</td>
<td>AGENT</td>
<td>32</td>
<td>PENTIUM 100</td>
</tr>
<tr>
<td>Edpc303</td>
<td>CACHING SERVER#1</td>
<td>32</td>
<td>PENTIUM 133</td>
</tr>
<tr>
<td>Edpc105</td>
<td>CACHING SERVER#2</td>
<td>32</td>
<td>PENTIUM 133</td>
</tr>
</tbody>
</table>

Fig 2 – Experimental setup
6. Implementation

Fig 3 shows the various modules developed within our architecture. ICP peers include both parent and siblings as shown earlier in figure 1. ICP requests and responses are forwarded to URL locator. While Cache ‘Hits’ need to fetch Web content from connected workstations, the PA itself handles Cache ‘misses’. The functioning of each block is clearly mentioned below.

![Diagram showing interaction among various software modules]

6.1 Module Specifications

a. Caching Server Module

The Caching server intercepts the HTTP requests generated by clients. This is made possible since browsers are set to pass through the proxy.

Each intercepted request is repackaged into payload of a UDP datagram and routed to the URL locator module residing within the agent. The responses from the URL locator can either be a ‘hit’ in which case the hostname of peer workstation is served. In the event of a ‘miss’ the hostname of the PA is served. Furthermore, the Caching server generates an additional UDP message such as ‘request deletion message’ for informing the agent about its cache replacement.
While the Caching servers logical cache is the Browser cache, for implementation simplicity a physical cache mirroring this logical cache is enabled.

We also propose and implement a new method for replacing browser caches. Four parameters including size of document, latency, bandwidth and number of references are used to determine the cache replacement policy. A detailed discussion on cache replacement policy is not considered in this paper.

b. *Proxy Servlet Module*

This module performs conventional Proxying function. HTTP requests from Caching Server Modules are repackaged into ICP queries and routed to peer proxies. In the event of an ICP miss CACHE-BLR is contacted to fetch the page from the Internet.

c. *ICP Servlet Module*

This module accepts TCP [14,15] connections on one side and either interfaces with URL locator to serve pages from cache server or serves pages from hierarchical Cache servers or Origin server.

d. *ICP Replier Module*

This module is an interface between the peer-proxies and agent. On one side it accepts ICP queries from peer proxies and on the other side it interfaces with the URL locator through UDP messages to find if the page is in the “URL Database”.

e. *URL Locator Module*

This critical module of the PA listens on port 8990. It has access to the URL database. It should be noted that the URL database also resides within the PA. This module’s functionality encompasses operations including ‘search’ for URL, ‘delete’ for deleting URL entries and ‘insert’ for new URL entries. Search and delete operations are differentiated by setting a flag. The flag may be set either by a Caching server or by ICP replier module and ICP Servlet module. While search and delete operations by a flag may be performed by Caching server, only ‘search’ is possible by ICP Servlet and replier module.
7 Time Sequence Chart

Time Sequence chart shows the sequence of events that are handled by the protocols within our architecture. Fig 4 and 5 we trace the path of user HTTP requests to responses under scenarios such as workstation peer Cache Miss and Cache Hit respectively. The requests interact with caching servers, URL locator and Proxy Servlet modules.

Fig 4 – Dynamics of user requests in the event of Cache Miss

Fig 5 – Dynamics of user requests in the event of Cache Hit
Time sequence charts in Fig 6 and 7 show the transparency with which workstation browser caches are utilized by the PA for requesting ICP peers. It should be noted that ICP peers include both neighbors and parents. To facilitate ICP, requests are accepted by the ICP server and passed on to the URL locator. In the event of an ICP miss and the peer configuring PA as the only parent, the PA is now obliged to fetch the web content on behalf of its ICP peer. In the event of an ICP hit, the ICP Servlet module transparently obtains web content from the workstation Caching servers which is in turn served to the requesting host.

Fig 6 – Dynamics of peer requests in the event of ICP Miss

Fig 7 – Dynamics of peer requests in the event of ICP Hit
8 Results and Discussions

a. Part A - Measurements

Measurements were done by building an application in Java, which generates multiple simultaneous HTTP requests. It also records the average service time per request. A total of 50 sites participated in the experimental run. The sites included educational institutes, research laboratories, commercial companies and network service providers.

From the measurements, it is found that the processing time of the PA is on an average 50 ms per request at normal network conditions. The throughput increases as the number of requests increases.

In addition, the "URL Locator" takes an average processing time of about 50 ms to search for a URL entry in the "URL Database". In the case of a "cache-miss", an additional processing time of 60 ms is taken to insert the new URL entry. An average processing time of 200 ms is taken to delete the URL entry in the event of cache-replacement on the caching servers.

Table 8.1 indicates response times both when a single user request and multiple (50) requests are allowed to initially pass through the PA. Measurements are taken again when user requests are allowed to reach the parent directly, bypassing the PA. In each of the cases, Servlet initialization being a one time initialization, was found to be 23 milliseconds. By further arithmetic [(242-23) – (213-23)] we observe that on a range 48 – 55 milliseconds are required by the PA to process requests. In other words, this is may be considered the path delay between Proxy Agent and the Parent. Furthermore, we have found that 700 milliseconds is the HTTP connection time. Therefore, in the event of a cache hit, the requesting client will incur same connection time delay between our architecture or by connecting to a parent.
Table 8.1 Average Response times when requests are via Proxy Agent and direct to parent

<table>
<thead>
<tr>
<th>Number of Requests</th>
<th>Response time through Proxy Agent (ms)</th>
<th>Response time without Proxy Agent, but direct to parent (ms)</th>
<th>Difference (ms)</th>
<th>Servlet Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>545</td>
<td>303</td>
<td>242</td>
<td>23</td>
</tr>
<tr>
<td>45</td>
<td>450</td>
<td>237</td>
<td>213</td>
<td>23</td>
</tr>
</tbody>
</table>

Part B – Discussion

The measurements presented in Part A shows that on an average there is going to be a delay of 71 milliseconds to 78 milliseconds including the Java Servelet activation delay. The delay for processing and transmission of the request over UDP is observed to be 65msec. Definitely the new proxy agent introduces additional delay for processing of the requests. A user is interested in getting the response faster as well as less fetching time of the requested Web-Page. Though the new architecture introduces a delay for the initial response compared to the existing architectures, this is insignificant, as it is in terms of a few tens of milliseconds. This slight increase in the response time is not perceivable by the user requesting the Web-Page. However, it is interesting to see the total time for fetching the requested pages significantly reduces. Fig. 8 shows two significant results of our measurements where we have considered typical 64kbps and 128kbps links to the Internet. The new architecture setup allows a scalable addition of Cache using the workstations’ cache in the Intranet so as to increase the percentage of Cache-Hits. The percentage of cache hits or cache miss depends on many factors including cache-replacement algorithms.

In the case of Cache-Miss, the page will be downloaded directly through the proxy server from the Origin server. It takes RTT time for fetching the requested page depending on the network speed. However, this is an extreme case. Whenever there are requests for new pages, because of the workstation cache in the Intranet, the request will be served locally, thus saving in time by avoiding page fetching from the Internet. Fig. 8 shows a definite reduction in fetching time. It also shows the gain with PA for a fraction of cache-hit vis-à-vis in the absence of PA. Further more, it shows the factor of gain in fetching the requested page for the proposed architecture with
respect to the existing architectures without the proposed PA. There is always an improvement in fetching time for more than 5% (refer Fig 8) of cache hit. The negative factor of gain is due the normalized difference between the fetching time with and without Proxy Agent. The throughput of the LAN is 4Mbits/sec and the average page size as 25KB. Higher the payload (web-page size) higher is the gain. Alternately, higher the link speed to the Internet, lower is the factor of gain in fetching the requested page. This architecture is scalable as many computers can act as a Web-Page cache inside Intranet.

![Fig 8 The factor of gain in fetching time for requested page in the proposed Architecture](image)

9 Conclusion

Java Servlet proxy agent has been implemented. Although Servlets are generally used for database applications, we demonstrate that it is still possible to build applications. End Workstation caches are effectively utilized and thus integrated with the Internet Caching mechanisms. Scalability of the architecture comes naturally. Also, no additional hardware investment is required to increase cache. Although Cache Hit will depend on the replacement algorithms, we prove there is an improvement in cache hits and thus a reduction in fetching time.
The drawback of this proposal is that the inherent weakness of Java is carried over. A better cache replacement algorithm can enhance system cache hits. Also, our sample of web sites needs to be increased for more accurate predictions. Overall, we think this architecture will improve the performance of Web Proxies.

10 Future work
We plan to introduce intelligence into such PA. For example, we could do prefetching based on bookmarks of connected users. While the proposed architecture can easily support FTP [13], we need to implement ‘FTP Servlet’ to support FTP.

11 References
