Pachyderm: The Web Proxy That Never Forgets

Junfeng Yang and Xin Li
{junfengy, lx}@cs.wisc.edu

CS736 Advanced Operating System
Fall 2000
Professor: Remzi Arpaci-Dusseau

December 17, 2000

Abstract

Web pages are changing everyday in the world with great amount of information updated and perhaps disappearing. At the same time, disks are becoming ever cheaper and larger. We argue that it is desirable and feasible to record a history of web pages using the specially designed proxy.

This paper describes the design, implementation and performance of the Pachyderm system, a web proxy that automatically retains all the versions of web pages retrieved from the web servers to form a history on disk, and make them accessible to users. The pages are versioned by the file names on disk. Storage in Pachyderm is managed using user-specified storing policies. Access of history is implemented with glimpse as the searching tool, and the Pachyderm is built on the base of Junkbuster, a simple filtering proxy.

1 Introduction

The web pages in our world change everyday and everywhere. Huge amount of information updates on the webs. The pages we saw a few days ago may disappear forever. At the same time, disks are becoming ever cheaper and larger. Since disk space is not a big problem anymore, we may find a way to store all the web page history on disks and then make that access history available to users. We believe that to remove the deletion and overwriting is probably a trend.

The Elephant file system [Elephant99] is a store-history approach on files. In Elephant, old versions of files are automatically retained and storage is managed by the file system. Users specify retention policies for individual files, groups of files, or directories. The same approach can be applied to the proxy server.

Net caches we used today can store the web pages previously visited by some users, but it doesn't provide a real history [Ircache]. The purpose of the web caches is to save precious bandwidth or reduce response time. Although it is possible that there are old versions of pages in the proxy cache, they will be overwritten once the newer page is retrieved from the web server and therefore cannot retain history.

Because of the huge amount of pages updated every day in the world, it's not very possible to find a way to store all those page histories. However, we can do this in a limited range. Since the proxy fetches the web pages on behalf of the client, it is possible to store all the versions of web pages ever visited by any client, and form a relatively complete history. Besides, we believe that the amount of storage dedicated to pages of old versions is growing slowly compared to the total amount of pages stored in the proxy cache.

This paper describes the design, implementation, and performance of the Pachyderm, a web proxy that never forgets. In Pachyderm, any pages requested by any client are automatically retained on the proxy server's disk to form a history of web pages. The users can specify storing policies for individual URLs. The goal of Pachyderm is to provide a convenient way for the users to access any pages they have visited before. Thus the updating action by the web servers will not eliminate the old but valuable information.

The remainder of this paper is divided into the following sections: section 2 introduces two other systems related closely to the Pachyderm; section 3 describes the design and implementation of the Pachyderm; section 4 is the measurement and evaluation of the performance of our proxy; section 6 describes some of our ideas about the future work; section 7 is the conclusion of our work; the acknowledgements are in section 8.

2 Related systems

The Pachyderm is developed on top of two other systems. The first is a web proxy, Junkbuster [Junkbuster]. The reasons we used it is that it's written in C, and that it's only a simple filter with no caching. So we can add our own cache to it conveniently.

The second system is Glimpse [Glimpse94][Glimpse]. Glimpse (which stands for Global IMPlicit Search) is a popular UNIX indexing and query system that allows users to search through a large set of files quickly. The pachyderm uses glimpse to build the index as well as search the index.

3 Design and implementation

Our design of Pachyderm can be outlined by Figure 3.1. The local disk on Pachyderm not only performs a common proxy cache, but also retains all the page versions to form a history. For every URL, the newest version is used as a common cached page, and only the newest version pages will be accessed for the normal web browsing. However, whenever a new page is retrieved from the web server, it doesn't overwrite the old one, but is stored as a new version. When the users request to access the history, the proxy searches all the page versions to find the required ones and send them back. So the Pachyderm doesn't bring any extra disk operation when performing a normal proxy cache. Actually it may require less disk operations because the old files needn't be deleted in the Pachyderm.

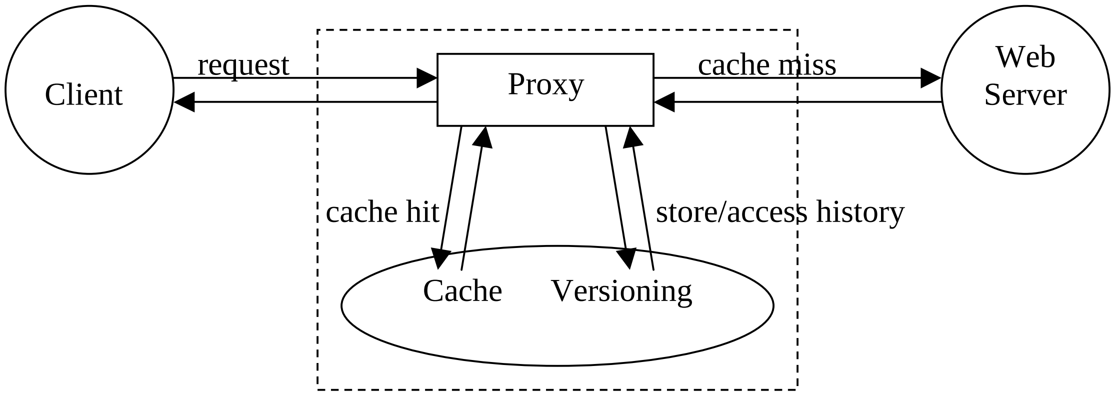

Figure 3.1

3.1 Store the history

This section describes how the history is stored on disk. When receiving a new page for a URL, the Pachyderm first projects the URL to a fixed length digest using MD5 algorism [MD5] and then uses the first two digits of the digest to decide the directory to store this file and takes the other digits combined with the last modified time of the page as the file name. In this way, the files can be uniformly stored on the disk.

3.1.1 A former design in the storage structures

In the beginning, we saved the pages in the same way as in the web. For example, for the URL http://www.cs.wisc.edu/directories/classes.html, we first make directories www.cs.wisc.edu/directories/, and then store the file classes.html under that path.

The benefit of this design is simpleness, but it cannot be applied to all URLs because some URLs contain characters that can't be used as the file name in UNIX system, such as ~, which is quite common in many URLs. Besides, this approach usually needs to make many directories before storing a new page, and to make a new directory is quite expensive in the UNIX system.

For these reasons, we turned to our current design. A two-level directory is used to store every file. The directory name of each level ranges from 0 to f. After projecting the URL to a 32-digit digest (because each byte ranges from 0 to 255, it is represented by two hexadecimal digits and thus the digest is 32 digits), the system takes the first 2 digits of the digest to decide the path to store the page. Take the same example as above,

URL: http://www.cs.wisc.edu/directories/classes.html

Digest generated by this URL: f5e83ee91463e14ad131062bdd7ebce7

Therefore this page will be stored under the path of f/5/. With this storing strategy, the files can be evenly distributed into the two-level directories, and any directory will not grow too large. A comparison of the time overhead to store a page in the initial design and the current design is given below.

	Average overhead
Store a page in the initial design	9.82 ms
Store a page in the current design	4.31 ms
MD5 computing in the current design	4.12 us

Table 3.1 Comparison of the time overhead to store a page in the initial design and the current design. The store operation is repeated 10,000 times for each design. Because computing MD5 digest is very fast, it is hard to break down the time overhead in the test for current design. The MD5 overhead is calculated by computing MD5 digests 10,000 times.

The comparison shows that the time overhead to store a new page in the current design is only half of the previous. The time to compute the MD5 digest is very small and can be neglected.

3.1.2 File versioning

One of the goals of the Pachyderm is to access the page versions quickly. For this purpose, we design a three-part file name for the files. The first part of the file name is the other 30 digits of the digest. The second part (URI number) is prepared for the collision. That is, if two URLs generate the same digest by the MD5 algorism, we must use this part to differentiate them. Although the chance of collision in the Pachyderm is only $1/2^{128}$, we put this part in the file name to prepare for the future. The last part in the file name is the last modified time of the page. This is the part we used to determine the versions. The different versions of a URL are only different in this part. We can still take the above example to explain the three-part file name. Given the page and the generated digest, the name of the file will be

$$\underbrace{\text{e83ee91463e14ad131062bdd7ebce7}}_{\text{(last 30 digits of the digest)}}\text{-}\overset{\text{(uri no.)}}{0}\text{-}\underbrace{\text{977031052}}_{\text{(last modified time)}}$$

With this three-part file name, the Pachyderm can easily get the version time by reading the file name.

3.1.3 Hash table

A hash table is built to speed the look-up of the history files. The hash function takes the 32-digit digest as the key, and stores the corresponding URL, policy, version number and a list of last modified times. The size of the information is small enough for the hash table to fit in the memory. Therefore this hash table reduces the number of disk operations when accessing the history.

The hash table is also a redundancy of the disk files. It can be regenerated easily by a rebuilder.

3.2 Access the history

In order to use glimpse to search the versioned files, the Pachyderm periodically make index with glimpseindex. To minimize the searching overhead, the medium-size index is used, which takes up 20% -- 30% of the total size of files being indexed.

A simple protocol is designed for the user to access history. If the domain name of the client request is “pachyderm”, then the path in the request indicates the searching command. For example, this URL

http://pachyderm/k/operating+system

tells the proxy to search the page history with keyword “operating” and “system”. The “k” in the path means keyword searching. Another searching command is “f”, which tells the proxy to directly return the file indicated. For instance, in the result page of the above URL (keyword searching command), there will be links as

http://pachyderm/f/<file name on the proxy disk>

which will retrieve the history file if clicked.

In general, the proxy parses the client request to get the searching command, then uses glimpse to make the searching, and returns the results as an html file.

3.3 Policies

As in the Elephant file system, the users can specify policies for the history storage. Why do we use policies? The main reason is that the users usually have different requirement, such as the privacy, or they don’t care about some pages at all. Certainly not all pages are useful to the users, so we are trying to retain only the useful ones.

We have identified the following five page retention policies for Pachyderm and have implemented these policies in our prototype.

- Keep None
- Keep One
- Keep All
- Keep Most
- Active Fetching

For Keep None, Keep One and Keep All, users can use regular expression [Regexp] to assign policies, such as *.edu, *.Microsoft.com/windows(98|99|2000). Keep Most is the default policy of Pachyderm. For Active Fetching, the actual URL must be specified.

3.3.1 Keep None: no caching

Keep None prevents the proxy from caching the web pages. Every time such web pages are requested, the proxy directly forward the request to the web server to get the page without searching cache, and will not cache the page when it’s returned from the web server.

This policy is mainly provided for users’ privacy. The users may not want to leave any copy of certain web pages outside the web server.

3.3.2 Keep One: no versioning

Keep One provides semantics identical to a standard proxy, retaining only the current version of a page. The newest page retrieved from the web server overwrites the older one immediately.

The pages that are applied this policy are usually the ones that users don’t care. The users don’t care about their updates and the proxy just performs the normal caching on these pages to save bandwidth.

3.3.3 Keep All: complete versioning

Keep All retains every version of every page. Every time a new page is received from the web server, it is stored as a new version, and the previous page will not be overwritten. This policy is often applied to the pages that contain information cared by the users and are also visited frequently by them.

3.3.4 Keep Most: default policy

When a page is retrieved from a URL with no user-specified policy, the default will be applied, which is Keep Most. The Pachyderm will automatically retain the pages that are considered meaningful by the proxy. For instance, some dynamically generated pages without a user-specified policy will be discarded, and the HTTP response without an OK status code will also be discarded [Http].

3.3.5 Active Fetching

Active Fetching automatically retrieves the page from a certain web server every period of time, and store it as a new version if the page is updated since the last fetch. The time interval to actively fetch is specified by the users. By assigning this policy, the user will not miss any version of the page even if the user don’t visit the URL for a long time, unless the page is modified more than once during a time interval.

4 Evaluation

We have three main performance concerns. First, how much disk space is needed to store the history. Second, how fast it is to access the history. The last one is how much additional overhead is introduced to do versioning.

4.1 Disk Space

To determine the amount of disk space to store the history, we did a 10-day trace (from Nov. 27 to Dec 06). We had 2 full time users and 4 part time users. Both of us used our proxy whenever we browsed the Web. 4 of our friends were persuaded to use our proxy. However, it is hard to let them use the Pachyderm when they browsed personal pages. So they were only occasional users. We run the Pachyderm on a tux machine (tux24). Because tux machines are shutdown every night, we wrote a shell script to start our Pachyderm automatically every day. The policy for this trace is configured as keep-all for every URL.

The trace stored over 4,000 files. Each file contained the client request line, the server response header and body. The information included in the trace is very complete.

Table 4.1 shows the number of files and the amount of disk space to store the history files. On average, the amount of disk space needed to store the history is less than 150 KB (2987KB/10 days/2 users) per user per day.

Figure 4.1 shows the distribution of URLs with different numbers (more than 1) of versions. About 3990 (97%) of URLs have only one version. Only 121(3%) have more than one. The distribution implies that most of the web pages are constant while only a small portion changes frequently.

Figure 4.2 shows a comparison of the amount of disk space occupied by different file types. An interesting result is that the amount of disk space to store the history is dominant by Text files. If compression is necessary to save disk space, Text files should be considered first.

	Number of files	**Size**
Latest version files	4,111 (90%)	34,234KB (92%)
History files	448 (10%)	2,987KB (8%)
Total	4,559	37,221KB

Table 4.1 Disk space occupied by the latest version files and the history files. Latest version files are the files that a common web cache will store. History files are the additional files that the Pachyderm proxy will store.

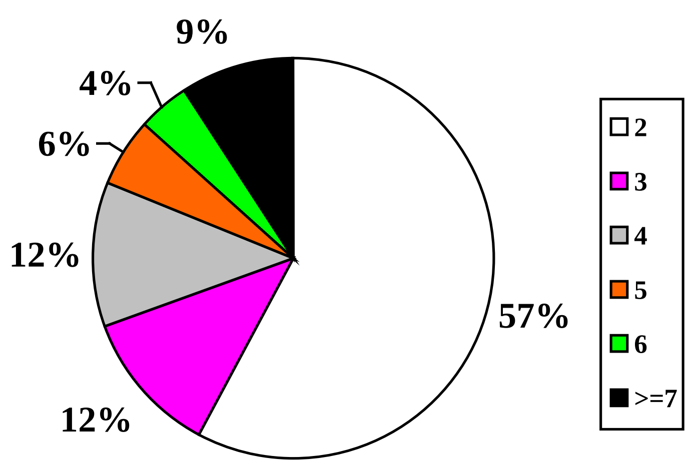

Figure 4.1 The distribution of URLs with different number (>=2) of versions. Total 121 URLs have more than one version.

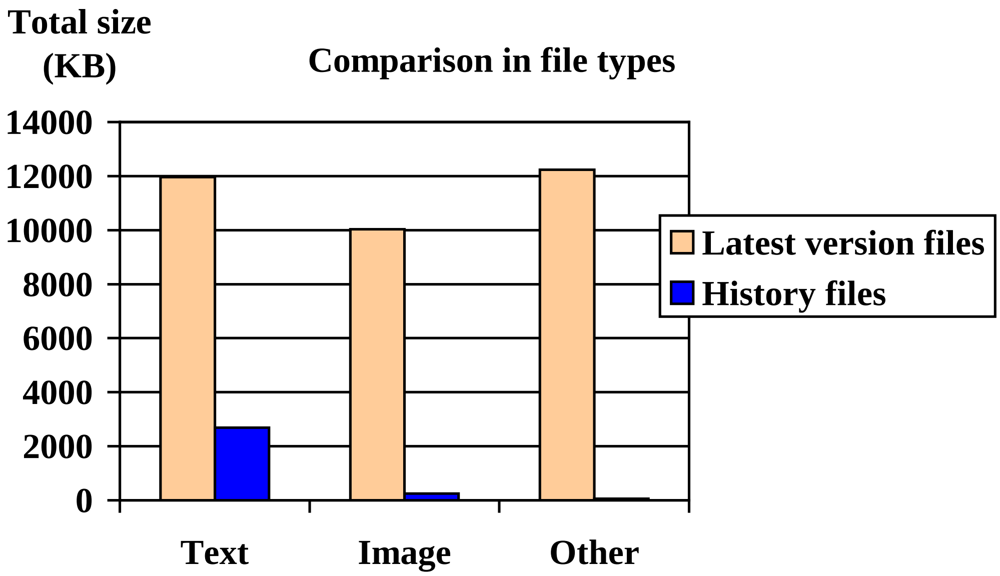

Figure 4.2 Disk space occupied by different file types. File types are determined by the content-type field in the response header.

As described, the trace may not be very representative. All users are graduate students, and they were working for their final. However, a long-term representative trace study can be done in the same way.

4.2 History Access Time

To evaluate the history access time, we wrote a benchmark program and conducted a test. The program will first get key words from the file .glimpse_index, then repeatedly pick one word randomly, generate a request and sent it to the Pachyderm. We hope that by picking randomly some irrelevant factors such as file system caching can be eliminated.

We did the benchmarking on 2 tux machines. A medium size index was used for the test. We generated 100 requests, and recorded the response time for each request. The total access time is 37.5s, and the average is 0.375s. It is not as good as what we expected. So we broke down the access time, and we found glimpse consumed almost 99% of the total access time. We think this is due to the extra overhead to run html-2-txt converter. When glimpseindexing, we run the converter to filter junk words in html files [Html], and every time we run glimpse to search the history, we run the converter again. So we optimized the Pachyderm by keeping a text copy for each html file when glimpseindexing, and did the benchmarking again. The performance improved dramatically. The total access time for 100 requests is 4.1s, the average is 0.041s, and glimpse consumes 3.7s(90%). Considering the LAN in tux lab is very fast (1K packet round trip time is only 0.3ms), it is reasonable that glimpse consumes that much. The additional disk space to store the text copies is 4.3M, which is 12% of the total to store the history. Table 4.2 shows the benchmarking results before and after the optimization.

	Total	**Glimpse consumes**	**Average**	**Additional disk space to store text copies**
Before Opt.	37.5 s	37.3 s	0.375 s	————
After Opt.	4.1 s	3.7 s	0.041 s	4,321KB

Table 4.2 Benchmarking results for history access time.

4.3 Time Overhead to Do Versioning

We think it is hard to evaluate the additional overhead to do versioning. This is because Junkbuster does no caching, so it is hard to test how much overhead introduced by comparing the Pachyderm with original Junkbuster. Another reason is that it is hard to specify a realistic workload, and we use multithread implementation. It is general difficult to measure performance when using threads.

Although it is hard to measure, we have still made an attempt at doing so. Our compromised solution is to compare our Pachyderm with Squid [Squid], a well-known mature proxy. We focus on the mean response time.

We used Web Polygraph [Poly] as the benchmark and did the test on 3 tux machines, one for Web Polygraph server, one for Web Polygraph client and one for the proxy tested. The tests lasted about 10 minutes.

Figure 4.3 shows the comparison results. We can see the overall mean response times are roughly the same. There is no obvious slowdown in the Pachyderm.

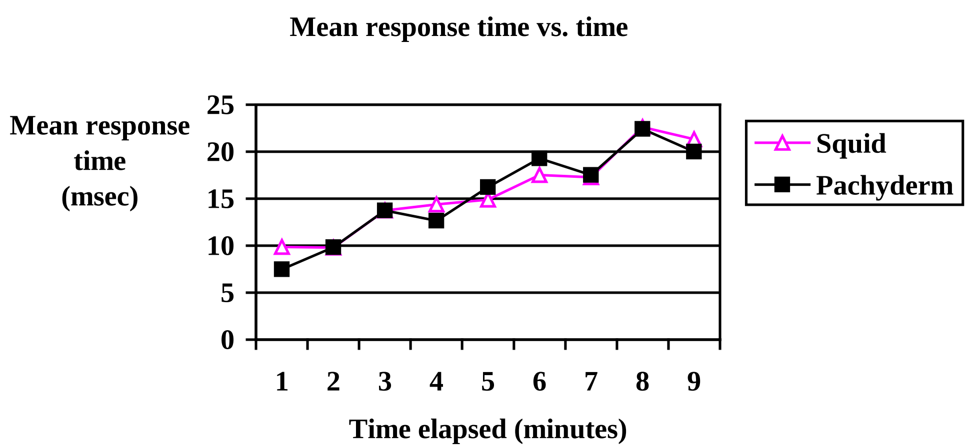

Figure 4.3 Comparison of mean response time with Squid. The benchmark program used is Web polygraph 2.5.4 with a simple workload configuration. Squid 2.3.4 is configured not to use Object Life Circle model and no cache replacement will occur by setting a very big limit for each cache directory. The overall mean response time is 15.71 ms for Squid and 15.45 ms for Pachyderm.

4.4 Summary

Our results show that the amount of disk space needed to store the history is less than 150KB per user per day. It is only a small percentage of modern disk drive capacity.

Although the trace may not be very representative, a long-term trace can be done in the same way. We believe that the same conclusion can be drawn.

The average history access time is only 41ms after the optimization. We can also use huge index to reduce the search time. Glimpse is a powerful tool, but it may be more powerful than needed for our Pachyderm. We think the access time can also be reduced by simplifying glimpse.

By comparing our Pachyderm with Squid, we can see that there is no noticeable slowdown in our implementation. Keep in mind that our Pachyderm does no more disk operations than any other web proxy, and it even does less because there are no deletions (unlinking) or truncating. We also use a hash table index to speed up searching and inserting, which can be fit in memory. The versioning overhead is small.

5 Related work

The Elephant file system is a store-history approach on files. In Elephant, old versions of files are automatically retained and storage is managed by the file system. Users specify retention policies for individual files, groups of files, or directories. The goal of Elephant is to allow users to retain important old versions of all of their files. There are many similarities between the Elephant and the Pachyderm. However, all the versioning operations are provide by the file system kernel. This might be over general. Only a few applications need such specified mechanisms. Besides, Elephant is not a mature file system compared with widely used FFS and LFS file systems. That's why we implement the Pachyderm with application level versioning, other than building it on top of the Elephant file system. In the Elephant file system a history version of a file can only be accessed with the proper time stamp. But it is hard for user to recall the creation time of a version. In our implementation, users can access the history by key word searching. We believe this is a more convenient way.

Another related technique is the net caches we use today. They provide a degree of web history. There are three locations where the page copies are stored: the cache of the web server, the net cache of the proxy server, and the net cache of the client machine. All of these three places can store the web pages previously visited by some users, but none of them provide a real history. The web server cache is used to reduce response time, and it only stores the newest version of pages on that server. The proxy cache's main objective is to save precious bandwidth. The proxy fetches web pages on behalf of the client. As it retrieves the web pages, the proxy also caches the pages that were requested. Because of the cache, frequently accessed pages will be retrieved from the cache of the proxy instead of fetching it again from the web site. So it is possible that there are old versions of pages in the proxy cache. But once a new version is retrieved, the old one will be overwritten. The client cache is similar as the proxy cache, but in the client machine and mainly for offline working.

6 Future work

As previously stated, our 10-day trace may not be very representative. We believe that a long-term and more representative trace study of how people browsing web is essential to many web-related projects. Based on the representative trace results, people can understand web activity better and make tradeoffs. Although many traces have been done in past years, none of them is done in a history centric view. Their focuses are always network traffic, proxy throughput or mean response time. It is very necessary to do a long-term trace with a focus on how web pages change.

The web browsing history is not just web pages stored in disks. It should also include the user activities. For example, a user might follow an inner link in a web page to another web page days ago. These 2 web pages were stored to disks at that time. Now he wants to follow the link in the history version of that web page to the history version of the other web page again. To achieve this, all inner links in a history version of a web page should be replaced to disk links, which are links to the history files stored in disks.

Currently, the history access user interface is quite simple. Users have to type a search command to access the history. While this is acceptable for us, our intention is to develop a friendly interface that allows for accessing the history more conveniently.

Although our final implementation is to build the Pachyderm on a common file system, it is still interesting to build it on top of the Elephant file system. We tried to get an installation copy of the Elephant file system from the Elephant people. But at last we failed to do so.

In our initial design, we considered CVS to compress versioned text files. This consideration turned out to be unnecessary because the amount of disk space to store the history is quit small. But sometimes disk space is also a scarce resource. The disk quota for a CS graduate of UW Madison is only 300M. So if necessary, text files compression can be considered to save disk space.

In our current implementation, for a page without user specified policies, the Pachyderm will discard it if it is meaningless, or keep it if it is meaningful. For example, the Pachyderm will discard some dynamically generated pages without user specified policies. It will also discard pages without an OK HTTP response code. An intelligent Pachyderm can be developed, which can make intelligent policy decisions by its own understanding of the page contents, the relation of this page to other pages with user specified policies … This is a very broad area.

7 Conclusions

Nowadays, disks are become cheaper and larger, while web is changing very fast. It is desirable to develop a proxy that stores every page users have seen, and makes the history accessible to users.

Our implementation and evaluation of the Pachyderm, the proxy that never forgets, show that it is feasible to develop such a proxy. The amount of disk space to store the history is less than 150KB per user per day, which is only a small percentage of the disk drive capacity. The average access time is only 41 ms in our test. By comparing our proxy with Squid, we can see that there are no noticeable slowdowns in our implementation.

We think our Pachyderm is powerful, and we implement it in a neat way. We use a uniform file name to do versioning. A hash table index is used to speed up the searching and inserting. By using glimpse, users can access the history conveniently and uniformly by key word search. Users can also use regular expressions to assign different policies to different URLs in a configuration file.

8 Acknowledgements

We would like to thank Professor Remzi H. Arpaci-Dusseau and TA Jonathan Ledlie from University of Wisconsin for giving us much useful advice and helping us understand the problems. We would also like to thank our friends, Haijun Cao, Xidong Wang, Su Zhang and Li Yang for trying our Pachyderm voluntarily.

References

[Elephant99] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, and Jacob Ofir. Deciding When to Forget in the Elephant File System. In Proceedings of 17th ACM Symposium on Operating Systems Principles, December 1999.

[Ircache] http://www.ircache.net/

[Junkbuster] http://internet.junkbuster.com/

[Glimpse94] Udi Manber, Sun Wu. GLIMPSE: A Tool to Search Through Entire File Systems. In 1994 Winter USENIX Technical Conference.

[Glimpse] http://webglimpse.org/

[MD5] http://www.landfield.com/rfcs/rfc1321.html

[Regexp] http://sunland.gsfc.nasa.gov/info/regex/

[Http] http://www.w3.org/Protocols/

[Squid] http://www.squid-cache.org/

[Poly] http://polygraph.ircache.net/