Experiments of Large File Caching and Comparisons of Caching Algorithms

Brad Whitehead[1], Chung-Horng Lung[2], Amogelang Tapela, Gopinath Sivarajah
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
{[1]bwhitehe, [2]chlung}@sce.carleton.ca

Abstract

File sizes have grown tremendously over the past years for music/video applications and the trend is still growing. As a result, large ISPs are facing increasing demand for bandwidth from the growth of file sizes. A main contribution to this bandwidth demand problem is inefficient use of bandwidth due to many ISP customers downloading the same large files multiple times. This paper first reports real experiments conducted on Carleton's Internet backbone by using the large file caching technique. Various cache replacement algorithms are then simulated and compared using traces of large file transfers. The results reveal that least recently used (LRU) performs better than others.

1. Introduction

The Internet today is inefficient in distributing large files which become very common. This leads to long wait times to download popular files. Whitehead [28] studied the wait times for a popular download site after a major release of a 210MB file. The file has been downloaded 35,000 times in 18 days and the wait time could reach an hour or even over 100 minutes after an hour of its release.

An efficient caching solution for large files could mitigate wait times and speed up downloads. Further, efficient caching of large files reduces ISPs' bandwidth usage. The concept of caching has been widely used for many years in computing systems and Web [20,23,27] and networks [7,11]. However, there is no explicit large file caching (LFC) solutions or experimental results reported in the literature. The currently available caching techniques are more suitable for caching Web pages and small files. There are many technical challenges in dealing with large file transfers [19]. A comparison of Web caching and LFC is presented in section 2. The first objective of this paper is to study the effect of large-file caching on general Internet traffic.

Another trend of the network inefficiency problem is the fact that the size of a "large" download is continually increasing. To save money, ISPs can extend Web caching to LFC. Very few companies are expanding these tools for larger files. Large file transfers are not perceived to be a major factor in bandwidth usage for an ISP. However, in our study, we will show that large file transfers account for about 20% of the Internet traffic on Carleton's Internet backbone based on real data traces and the amount is still growing.

The second objective is to evaluate various caching algorithms. Many caching algorithms have been used in computing systems and Web caching. Some of them are applicable to LFC; some need modifications. We have applied and adapted some caching algorithms and compared their performance based on the data traces collected from real experiments.

2. Background and Related Work

Web caching has been used since the early 90s. Web caching allows ISPs to save their bandwidth by storing frequently accessed files locally [20,23,27]. Currently available caching systems focus on Web pages almost exclusively. Downloading large files, on the other hand, is a much more challenging problem [19]. Table 1 summarizes the comparison between LFC and existing Web caching products. In short, LFC has to deal with any file type and various protocols. In addition, large files typically are stored in multiple locations, LFC techniques need to be able to identify the same files even if they could be downloaded from different locations.

Table 1. Large File Caching & Web Caching

Large File Caching	Web Caching
Any file type, large Web pages	Small files and Web pages
1MB-10GB	0-10MB
Any protocol; HTTP, FTP, FastTrack, Gnutella, BitTorrent, CoBlitz, etc.	HTTP only
Files, in chunks or in whole, mostly or always reside on multiple sites	Files reside on a single site or multiple sites

Content distribution networks (CDNs) are closely related to our technique. Many approaches to peer-to-peer (P2P) content distribution techniques have been reported in the literature or used in practice, such as BitTorrent [8], CHORD [5,14], CoBlitz [19], Coral [9], Fast-Replica [4], Gnutella [10], Kazaa [15], Shark [1],

978-0-7695-3192-2/08 $25.00 © 2008 IEEE
DOI 10.1109/NCA.2008.44

and etc.. Some focus on content search and sharing using keywords, e.g., [5,14,16,29]. Some techniques focus on reducing the download time of popular files.

However, downloading large files is a qualitatively different problem for CDNs, as [19] reported based on Akamai's experience. Some techniques break a large file into pieces and exchange those pieces among clients instead of always downloading from the origin server [4,8,19]. Other techniques, e.g., FatNemo [2], SplitStream [3], ESM [6], Bullet [17], Astrolabe [21] mainly deal with load balancing and link utilization, but they require clients to simultaneously transmit the content.

BitTorrent is wildly used to support file downloads and it scales well. CoBlitz has been explicitly proposed to support large files and it outperforms BitTorrent [19]. One of CoBlitz's design goals is to trade bandwidth for disk seek times, because bandwidth price is continually dropping and disk seek times are high and hence the solutions may not be scalable [19]. However, high bandwidth along does not guarantee high performance, because every element of the delivery chain (intermediate nodes and links) can affect the overall performance. Also, if the cache hit ratio is high, using caching could have better performance than that of high bandwidth [22].

The cache in our approach resides on the local ISP side where many clients using the same ISP may try to access the same large files. The chance that many clients share similar interests is high in some environment, such as universities. Moreover, our approach could be used together with other large file distribution services such as BitTorrent and CoBlitz. In other words, large files can be downloaded using those techniques from other peers or servers if those files are not in the local cache.

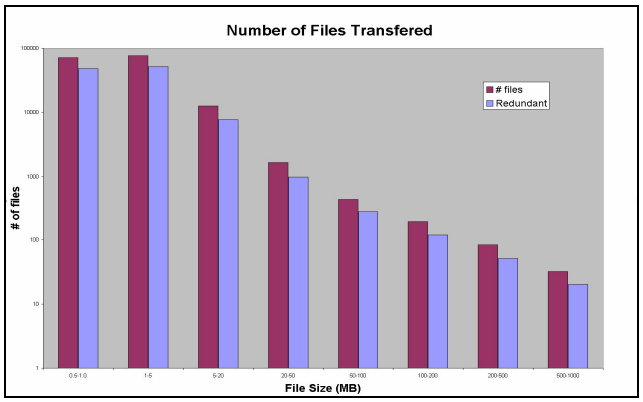
3. Experiments and Empirical Results

The main functionalities required to support LFC are cache lookup, redirection of connection, and identifying large files and tracking of all file transfers on the backbone server for every TCP connection, which presents high challenge to the design and implementation. Tracking the TCP connections on the backbone server requires that information be stored for every computer on the network and for each connection for that computer.

Filestats is a network-sniffing program that records a log of all file transfers over a network. *Filestats* was used to validate the need for LFC in two ways. First, large-file network traffic must make up a large enough percentage of the total network traffic that it was significant; and second, that enough of the files transferred were the same (redundant) that a caching solution was viable.

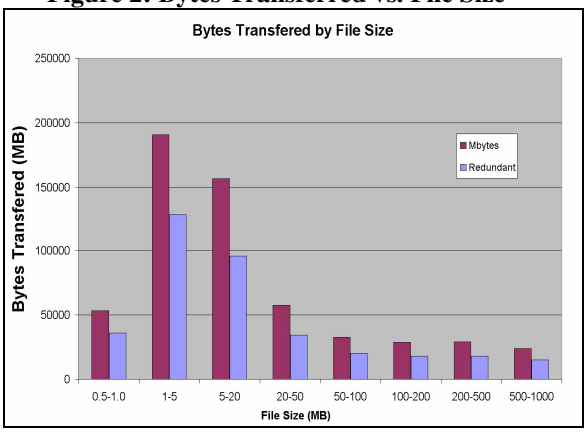
All network traffic of the university going through the backbone server was processed, but only file transfers over 500KB in size were logged. The logs from Jan. 5 to Jan. 23, 2004 were processed to form Figures 1 and 2**.** Figures 1 and 2 show the correlation between file size and the network impact. Figure 1 shows the expected decrease in the number of files transferred as the file size increases. Figure 2 shows that large files, over 1MB in size, account for a large percentage of network traffic.

Figure 1: No. of Files Transferred vs. File Size



By comparing Figure 1 and Figure 2, several conclusions can be drawn that can improve the efficiency of LFC. The number of requests (files transferred) is an important factor determining caching system performance. Figure 1 shows that files between 0.5-1MB account for almost the greatest number of transfers at 71226 transfers, behind 1-5MB at 76328. However Figure 2 shows that these 0.5-1MB transfers only make up 55GB of traffic, far lower than the 180GB for 1-5MB transfers. Therefore, for our study, we focus on files over 1MB in size. This can also reduce the amount of connections that need to be monitored. The specific file size is actually configurable depending on different needs or environments.

Figure 2: Bytes Transferred vs. File Size



4. Large File Caching Algorithms

This section presents a comparison of various LFC algorithms with respect to hit ratio, byte hit ratio, bandwidth saved, and cache size. The comparison was

conducted based on simulation of data traces collected through experiments. The total number of requests from these files (at least 1MB) was 130,866, and the total size for the files was 380.81GBytes. There were 62,530 unique requests and their total size was 205.86GBytes. The files in the trace file were collected between January 05 and January 14, 2004 inclusive.

Each algorithm is further explained as follows.

LRU –Least Recently Used. LRU removes the file that was least recently accessed. In this method, time is used to determine the file to be removed. More than one file can be replaced if the incoming file size is greater than the size of the replaced least recently object.

LRU Size – Least Recently Used with Size. This method determines the file to be removed by looking at the least recently used and the size of the file. The file to be removed must have size which is greater or equally to the size of the incoming file to be cached. The main aim of the algorithm is to replace one file if the cache is full.

LRU Threshold – LRU with Threshold. In this algorithm a value known as a threshold and the time for the last access to the file are used to choose a file to replace from the cache. The file that has the longest recent-accessed time and its size is less than the threshold value is evicted from the cache if a new file is to be cached cannot fit in the current cache. Files that are larger than the threshold value are not replaced from the cache. The idea is to keep larger files in cache to save bandwidth.

LFU –Least Frequently Used. This algorithm removes the file that has been accessed the least. Frequency is the key parameter that is used to determine the object to be replaced.

LFU Size –Least Frequently Used with Size. This algorithm replaces the file that has been accessed the least. Frequency is the key parameter that is used to determine the file to be replaced. The file to be removed must have size which is greater or equally to the size of the incoming file to be cached. Similar to LRU Size, the main aim of the algorithm is to replace just one file if the cache is full.

LFU Threshold – LFU with Threshold

This algorithm removes the file that has been accessed the least and its size is less than or equal to the threshold value. The idea is similar to LRU Threshold.

Greedy Dual Size Frequency [13]. The key parameters in this method are file size, number of references (frequency) to the file and the cost. The cost of downloading the file from the remote server is equal to the average bandwidth usage of the file. Each file that is cached is assigned a key calculated using the above parameters. If the cache is full, the file that has the least key will be evicted. The key is calculated using the formula shown below:

$$Key(f) = Lvalue + freq(f) * bandwidth(f) / size(f) \ldots(A)$$

The value of *Lvalue* tarts from 0 and is updated every time a file is evicted. The new value for *Lvalue* will be the key value of the removed file. If a request is a miss, the file will be fetched from the remote server. If the free space in the cache is smaller than the size of the incoming file, one or more files that has the minimum key will be removed. The file will then be cached and:

- the frequency (f) count is set to 1.
- key is recalculated from formula (A)
- cache used = cache used + size(f)

If a request is a hit:

- file frequency is increased by 1;
- key is recalculated from formula (A)
- cache size does not change

Static Cache. The static cache algorithm is a simple algorithm that caches all the popular requests that were made on the current day. The cache is deleted and reset at the end of each day.

Static Algorithm

Gather all the requests for the day
For each requested file {
  Set value = #references / size of file
}
Sort these files from descending order
Populate the cache from the top of the sorted list

Largest Size. This algorithm determines the victim file by removing the largest file from the cache. Size is the key parameter in this method. The rationale here is that more disk space will be available by replacing the largest file.

Table 2 summarizes the implemented caching algorithms and the rationale in determining the file for replacement. Each algorithm is further explained as follows.

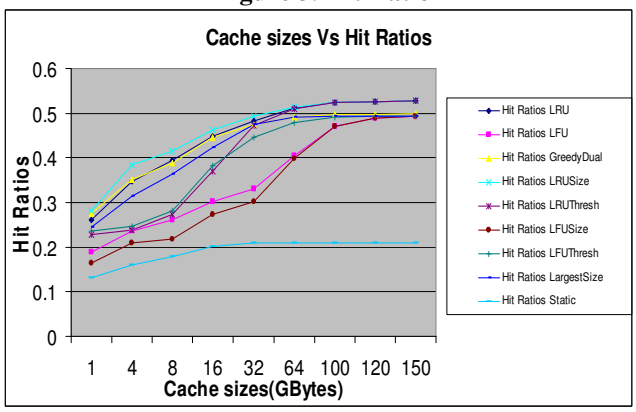
Table 2. Summary of the Implemented Algorithms

Algorithm	Replacement Policy
LRU	Least recently accessed first
LRU Size	LRU and file size in a specified range with respect to incoming file size
LRU Threshold	LRU and file size less than a certain threshold
LFU	Least frequently accessed first
LFU Size	LFU and file size in a specified range with respect to incoming object
LFU Threshold	LFU and file size less than a certain threshold
GreedyDual SizeFreq	Least value first according to Key(f) = L + (F*B)/S;
LargestSize	Largest file first
Static	No files are evicted, the cache is reset at the end of the day, based on yesterday's request value (frequency/# of files)

5. Results and Analysis

Hit Ratios. Figure 3 shows the hit ratios for each of the implemented algorithms.

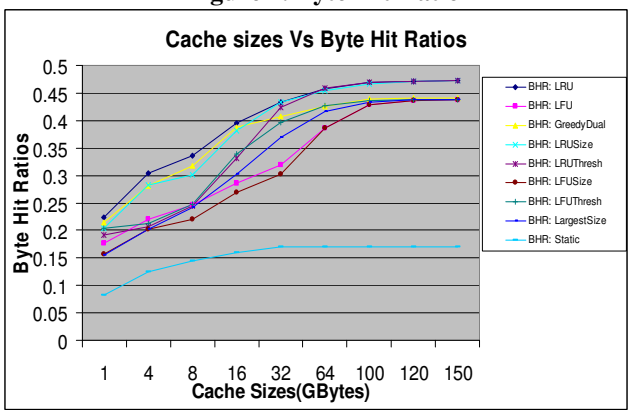
Figure 3. Hit Ratio



The LRU Size has higher hit ratio for all cache sizes. The hit ratios for the Static Cache were the lowest for all the cache sizes simulated. The hit ratio only deals with the count of files that were found in the cache. However, it is possible to have a high hit ratio and still save less bandwidth especially if most of the hit were smaller files.

Byte Hit Ratios. Figure 4 shows the byte hit ratios for each of the implemented algorithms. The byte hit ratios were calculated for each of the cache size that has been used during the simulation.

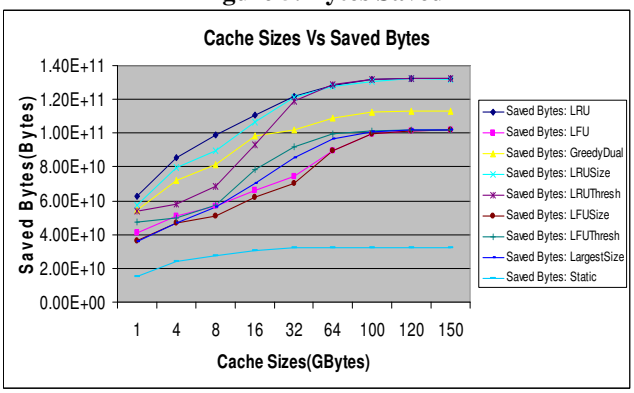
Figure 4. Byte Hit Ratio



The byte hit ratio gives more information about the bandwidth saved, because this parameter deals with the actual size of the file. High byte hit ratio means more bandwidth saved. The LRU algorithm has the highest byte hit ratios for all the cache sizes.

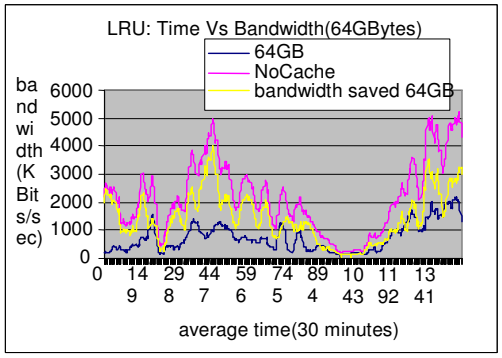
Bytes Saved. Figure 5 shows the bytes saved for different cache sizes used in the simulation. The LRU has the highest number of saved bytes for each cache size simulated. This bytes saved parameter and the byte hit ratio are related; both are good at determining how effective an algorithm is in saving bandwidth.

Figure 5. Bytes Saved



Bandwidth saved. Here we show the actual amount of bandwidth used without cache and the amount of bandwidth saved using cache. Only one case of using LRU with 64G cache is presented in Figure 6 as an demonstration. The first curve is the amount of bandwidth that will be used without cache (labeled *NoCache*). The second or bottom curve represents the bandwidth used with a cache in place (labeled *64GB*). Finally, the third middle curve represents the amount of bandwidth saved from having a cache. The difference between the *NoCache* curve and the bandwidth used with 64GB curve is the amount of bandwidth saved for that particular algorithm and corresponding cache size.

Figure 6. Bandwidth Saved on with 64 GB Cache



6. Conclusion

This paper presented a LFC technique and a comparison of various cache algorithms to support LFC. As the size of files will be continually increasing, caching large files at the local ISP can reduce download times and bandwidth usage, especially for popular files.

The comparison was based on data traces obtained from real experiments [28] in a university environment. The experiments revealed that many redundant large files had been downloaded from external servers. By caching those large files in the local ISP server, download times and bandwidth usage can be reduced. Generally speaking, LRU performs the best based on the data

collected in our experiments. GreedyDualSize-Freq has performed better than the LFU, because GreedyDualSize-Freq method optimizes the byte hit ratio and hit ratio with respect to LFU results.

LFU can be used if the user is only interested in the popular files that are used frequently. If the hit ratio is the main concern, then the LRUSize could be the appropriate algorithm to use. For maximum byte hit ratio and bandwidth saved LRU generates the highest value.

We are planning to repeat the experimental work as the nature of our network traffic has changed significantly since 2004. When the experiments were conduced, the university had 30Mbps of Internet bandwidth in January, 2004. For the 2006/2007 academic year the bandwidth was expanded to 60Mbps in September and by April it had been doubled again to 120Mbps [24]. It is expected that more bandwidth has been consumed for identical large files.

References

[1] S. Annapureddy, M. J. Freedman, and D. Mazires, "Shark: Scaling File Servers via Cooperative Caching", Proc. of *2nd USENIX/ACM Symp. on Networked Sys Design and Implementation*, May 2005.

[2] S. Birrer, D. Lu, F. E. Bustamante, Y. Qiao, and P. Dinda, "FatNemo: Building a resilient multi-source multicast fat-tree", *Proc. of 9th Int'l Workshop on Web Content Caching and Distribution*, October 2004.

[3] M. Castro, P. Drushcel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment", *Proc. of SOSP*, Oct 2003.

[4] L. Cherkasova and J. Lee, "FastReplica: Efficient Large File Distribution within Content Delivery Networks", *Proc. of the 4th USITS*, March 2003.

[5] "Open Chord Specification", http://open-chord.sourceforge.net.

[6] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system Multicast", *IEEE J. on Selected Areas in Communication*, 2002.

[7] Cisco Systems, "Network Caching Technologies", http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/net_cach.htm.

[8] B. Cohen. Bittorrent, 2003, http://bitconjurer.org/BitTorrent.

[9] "The Coral Content Distribution Network", http://www.coralcdn.org/.

[10] "Gnutella Protocol Definition", http://rfcgnutella.sourceforge.net/.

[11] Internet Caching Resource Center, http://www.caching.com/caching101.htm

[12] Internet Systems Consortium Inc., "ISC Domain Survey: Number of Internet Hosts", 2004, http://www.isc.org/index.pl?/ops/ds/host-count history.php.

[13] S. Jin and A. Bestavros, "*Popularity-aware GreedyDual-Size Web Proxy Caching Algorithms*", *Proc. of ICDCS,* 2000.

[14] L. Karsten, K. Sven, "Open-CHORD: Distributed and Mobile Systems Group", http://www.lspi.wiai.unibamberg.de/dmsg/software/open_chord/.

[15] Kazza's Architecture, "How a Kazza's client finds a song", http://computer.howstuffworks.

[16] B. Kim, K. Kim, "Keyword Search in DHT-Based Peer-to-Peer Networks", *Proc. of 7th Int'l Conf. on Algorithms and Arch. for Parallel Processing,*, 2007, pp.338-347.

[17] D. Kosti´c, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh", *Proc. of 19th ACM SOSP*, 2003.

[18] T. Leighton, Akamai Technologies Inc., "The Challenges of Delivering Content on the Internet" Keynote Speech, *IASTED Conf. On Parallel and Distributed Systems*, 2002.

[19] K. Park and V. S. Pai, "Scale and Performance in the CoBlitz Large-File Distribution Service", *Proc. of the 3rd Symp. on Networked Sys Design and Implementation,* 2006.

[20] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*, Addison Wesley, 2002.

[21] R. van Renesse, K. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management and data mining", In *ACM Transactions on Computer Systems*, May 2003.

[22] S. van Rompaey, K. Spaey, and C. Blondia, "Bandwidth versus Storage Trade-off in a Content Distribution Network and a Single Server System", *Proc. of the 7th Int'l Conf. on Telecommunications,* 2003, pp. 315-320.

[23] W. Shi and Y. Mao, "Performance Evaluation of Peer-to-Peer Web Caching Systems", *Journal of Systems and Software*, May 2005, pp. 714-726.

[24] J. Stewart, *Personal communication* (email) with C.-H. Lung, May 29th 2007.

[25] D. C. Verma, "Overview of Content Distribution Networks" and "Site Design and Scalability Issues in Content Distribution Networks" in *Content Distribution Networks*, New York, NY: John Wiley & Sons, Inc., 2002, pp 1-61.

[26] D. C. Verma, "CDN Data Sharing Schemes" in *Content Distribution Networks*, John Wiley & Sons, Inc., 2002.

[27] "Web Caching: Making the Most of Your Internet Connection", http://www.web-cache.com/.

[28] B. Whitehead, *A Scalable Anycast Technology for Caching Content Distribution Networks*, Project Report, Dept of Sys and Comp Eng, Carleton Univ., Canada, 2004.

[29] K.-H. Yang and J.-M. Ho, "Proof: A Novel DHT-Based Peer-to-Peer Search Engine", *IEIEC Trans. on Communications*, E90-B4, 2007, pp. 817-825.

Acknowledgements:

We are grateful to John Stewart at Computer Computing Services, and Narendra Mehta and Dave Sword in the Systems and Computer Engineering Department for their trust, support, and encouragement.