Alex - Caching NFS fileserver for FTP files

Vincent Cate

Thomas Gross

vac or trg @cs.cmu.edu

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

The Alex system provides uniform access to public ftp files on the internet (i.e., files that can be retrieved via annonymous ftp) as well as caching of remote files. Client systems mount an Alex server like any other NFS server, so access to a cached file is as fast as access to a file in a local file system. To provide high-speed access to a file, the Alex system cannot check on every access if the local copy is still up to date. So instead of guaranteeing complete consistency between its local cache and the remote files, Alex provides *bounded weak consistency*: Alex guarantees that a file is not out of date by more than 1/10 of its lifetime. That is, a day-old file may at most 2.4 hours out of date, whereas a 1 year old file may be about 5.5 weeks out of date. Bounded weak consistency implies the risk that a client is given a stale copy of a file, but the actual frequency of handing out stale copies depends on many factors including the rate of updates to remote files as well as the usage patterns for file references. To address the needs of users who want to be sure to get the latest copy, Alex provides a user command to forcefully update the local copy.

The Alex system has been distributed freely, and there are now about 200 servers with approximately 4000 clients who mail back usage statistics. These data indicate that

- Public files in ftp repositories are largely archival and change only infrequently;
- The caching of remote files works well; there is significant reuse of files as well as of directory information (which is cached as well);
- Stale files are extremely rare, among over 1 million files served, less than 0.8% of them were stale. The Alex system allows the user to force the update of a directory (e.g., to be sure that the latest version of the software is installed), but such update requests are rare (only 0.18% of all file acceses).

Although the Alex system guarantees only weak consistency (a file may be out of date by 1/10th of its lifetime), the actual freshness of the cache is much better, in part through clever exploitation of local directory information.

This research was sponsored in part by The Defense Advanced Research Projects Agency, Information Science and Technology Office, under the title “Research on Parallel Computing”, ARPA Order No. 7330, issued by DARPA/CMO under Contract MDA972-90-C-0035 and in part by the National Science Foundation under grant number ECD-8907068. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

1. Introduction

The internet with its many hosts provides a great opportunity for the sharing of information. Many sites make a wealth of information available in files that can be accessed by any internet user. These files are available for annonymous ftp, and we refer to those files as *public files*.

A browser like Mosaic simplifies navigating through the internet's files, provides a sophisticated user interface, links related documents, and integrates access to non-textual information. However, if two users at the same site visit the same moasic page, then this information is transferred twice from the remote storage site.

Alex is an experimental internet wide file system that employs caching to reduce the latency of accessing a remote file. If a file is already in the local cache, then access to this file takes as much time as other access to a local file server. Caching files also cuts down on the number of transfers, since a file supplied by the local server does not have to travel another time over the internet.

One of the problems with local caching of remote internet files is that these files can change at their home site, so the local copy is no longer up to date. We call such a copy a *stale* file. Given the size of the internet, it is impractical for the Alex server to explicitly check if a file is stale whenever it is accessed. Not only can it take considerable time to perform such a check, if the remote host is down or unreachable, it is impossible to perform the check. Therefore, Alex introduces a more relaxed consistency model. Instead of keeping the local cache consistent with the remote file systems at any time, Alex uses *bounded weak consistency*: A file in the local cache is potentially out of date only for some amount of time, and this amount is a function of the file's recorded age. That is, a file is potentially out of date in the last 1/10 period of its recorded age. E.g., a file that is about a day old can only be out of date by only 2 hours. Notice that the age of the file determines the size of its weak consistency window, not when it was brought into the cache. Figure 1 depicts graphically events in the life of a file: it is created remotedly at time T0, at T1 is is brought into the local cache of an Alex server. Any subsequent request (e.g, T2) to this server in the gray zone W1 is satisfied by serving the file brought in at T1, the lenght of W1 is 1/10 of its age (T1- T0). A later request (e.g., at T3) results in checking if the local copy is still up to date and starts a new window of weakness (W2).

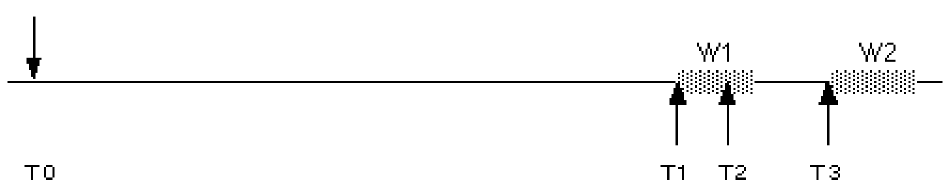

Figure 1 Caching and consistency checks for a file

Figure 1 depicts a simple case, the actual implementation includes a number of optimizations (the server may know from other activity if a file is stale or up to date) and also deals with the problems of internet unreachability. This is discussed in more detail in Section S2. relaxed consistency

The principal advantage of Alex over user-maintained private copies of remote files is that Alex offers a consistency guarantee. If a user makes a private copy of a file in a local directory (e.g., by tranferring the file with FTP), then the user is reponsible for keeping track of updates of the remote file; with the growing use of internet files, this task can soon become impossible. Alex keeps track for the user and guarantees that a file is not out of date by more than a preset amount of time.

Alex provides an NFS interface, so remote files are integrated into the (conventional) file system name space as provided by the local operating system, e.g., Unix. That is, the file "README.mss" in directory "/pub/reports/1991" on machine "reports.cs.cmu.edu" is accessed as /alex/edu/cmu/cs/reports/1991/README.ps.

Alex's files names are global, the same file name refers to this file from all locations in the internet (that have mounted an Alex server). Notice that Alex hides significant aspects of the heterogeneity of the hosts on the internet; cs.cmu.edu could be a VMS, Unix, or NT machine.

2. Related work

There are a number of other efforts to provide access to internet files. We can group those into classes:

Improvements over ftp: An example is "fetch", which provides a select-and-click user interface, as well as file conversion for transfers (e.g., from Macintosh Finder format to MacBinary). However, users must manage their local copies (i.e., keep track of updates to the remote file). Plan9 provides a uniform name space and e.g. allows symbolic links across machine boundaries; on file access, the operating system performs the ftp transfer on behalf of the user, but there is no caching. While potentially useful in a local area network, no results have been reported on how this approach scales to the internet domain. Archie and its descendants allow searches over the domain of public files and initiate ftp transfers of matching files but do not assist the user to maintain consistency between local and remote copies.

Internew browsers and navigators: Examples are Gopher or Mosaic. These programs provide sophisticated interfaces to a network of servers. But if a user wants to work on a file or data set with local programs, then it is necessary to explicitly "save" the file and assume responsibility for keeping track of updates. The files operated on by gopher or mosaic cannot be operated on by another, local program without an explicit local copy. Some of the servers may provide mirroring of other servers to reduce network latencies. However consistency depends on how often the systems synchronize (often once per night at a fixed time), and there is no way for a user to enforce updates on the server. And while these tools may cache information during a single session, no inter-user caching or inter-session caching takes place.

Network file systems: A network files system like AFS provides some services that are not provided by Alex, e.g. the ability to authenticate a user or group of users to perform file updates. But AFS requires modification to the kernel of the operating system, and consequently, both parties (i.e. the user's local system and the remote file storage site) have to run the AFS system. Alex requires no kernel modification (it is mounted as an NFS server) and requires no cooperation from remote sites beyond supporting anonymous ftp.

A good way to think about Alex is as a platform or infrastructure for caching internet files. E.g., mosaic could use an alex server to transfer files. The Alex server provides a complete file system interface, and remote Alex files can be used like a user's local files in any operation (other than write or delete). The interesting research aspect of Alex is that it attempts to provide caching of files in a lage domain, for the class of public files, and it demonstrates the practicality of a relaxed consistency model for internet wide file access. In this paper, we report on the lessons learned from this experiment, and the outline of the paper is then as follows: In Section S2, we state the basic assumptions of the Alex design. Clearly, Alex is not intended for a database with frequent updates. Then, in Section S2, we review those aspects of the Alex implementation that are crucial for this paper. Section S4 contains data gathered on many Alex servers; these data support our view that Alex works extremely well for its domain. These data also provide an interesting view of the world of public files in the internet. Finally, Section S5 contains our conclusions.

3. Alex assumptions

One observation that motivated the design of Alex was that public ftp files in the internet, i.e. those files that can be retrieved via anonymous ftp, are often archival. A directory may contain e.g. technical reports, position papers, or public-domain software, and updates happen infrequently. Furthermore, if a user by accident retrieved a file that was "a little" out of date, then there would be no serious harm. Of course, this observation of infrequent updates does not hold true for all public files. A number of systems make periodic information available, like e.g. weather maps, which are updated once every hour. But even then, a user may accept that a file is out of date by a few minutes. So there is a tradeoff between user-managed consistency (as good as the user cares) and automatic, bounded weak consistency that can be provided by the operating (file) system. The assumption is that some types of files change on a regular basis while most change infrequently or not at all. A further assumption is that the age-based bounded consistency model provided by Alex is sufficient for the types of files found in FTP sites.

The second major assumption is that there is sufficient reuse of information in public ftp repositories. If there is only a single user at each site that reads a public file just once, then caching is not effective, and the bounded consistency service has no impact.

Section S4 discussed to what extend these assumptions hold in practice for the domain of public ftp files.

4. Alex implementation

The core of the Alex files system is the Alex server. In a typical installation, this server has significant local file space to cache many files as well as directory information. Requests for files by clients are satisfied out of the local cache, or the files are retrieved from the remote site. The ftp program is used as a transport layer protocol, all interaction with the remote file system is via ftp commands (get, dir, etc.). The Alex server employs the NFS protocol for client operations; to a client, the Alex file system is yet another NFS server that is mounted. Details of the Alex implementation have been described earlier\cite{michigan}; we restate here only those details that are relevant for the evaluation of Alex.

There are three implications of the Alex implementation that pertain to this paper:

The first benefit provided by Alex is the local caching done by the server. A subsequent reference to a file can be satisfied locally if the file is still in the cache. However, caching does not only work for files, but also for directory information, ftp connections (which are kept open to avoid the startup overhead), status information about a host (like its time zone, etc.). We discuss these aspects of Alex in detail in Section S4.

Second, Alex provides a consistency guarantee: a file will never be out of date by more than a bounded amount of time. This bound is a function of the file's age. As seen in Figure F1, whenever the servers learns that its current copy is still up to date (or a fresh copy is brought in), this starts another window during which requests can be satisfied blindly, without checking over the internet. So the Alex server attempts to keep track of its files as efficiently as possible. Explicit requests to a file outside its window (e.g., at T3) trigger an inquiry. As a result, the Alex server obtains meta data about all files in the file's directory. If any of the local copies of files in this directory are stale, this fact is recorded, and subsequent accesses result in a refetch of the file. We discuss this in more detail in Section S4.

Third, since Alex is a file system, applications access Alex files the same way they access other (local and networked) files. Consequently, an application that issues a "read" request can start working on the first dataset as soon as the first block of data is provided by the file system. And the operating system may be able to exploit the fact that a process is waiting for a response from the file system and schedule the process when the requested response is delivered. If the FTP program is used explicitly to fetch a file over the network, then the application has to wait until the complete file has arrived. This benefit translates into a real savings for the user, who would otherwise have to wait for the completion of these file transfers.

There are also other benefits, e.g. the seamless integration of the networked files into the user's file space. There is no need to relink programs (this would be necessary if operations on remote files were provided by some separate library, as proposed by Prospero), nor is it necessary to change the programs (this would be necessary if we introduced new functions for remote acces, as e.g. done in WWFS). So all the user's tools continue to work as expected. And the Alex system hides the heterogeneity of the systems connected via the internet. Alex is able to to deal with internet hosts running Unix, VMS, NT (and until dropped for lack of interest), CMS. Users use files the same way independant of the type of the remote FTP site. However, it is hard to evaluate empirically these benefits, and therefore, we concentrate in this paper on those aspects of Alex for which we can obtain solid measurement data.

5. Alex evaluation

Our evaluation of Alex attempts to throw some light on the novel aspects of Alex. Our primary focus is evaluation of the effectiveness of the caching model including its concept of bounded weak consistency.

To gather meaningful data, we have instrumented the Alex server to keep track of a number of different events (to be elaborated on later in detail). If a new server is installed (and a user does not disable this feature), the server reports back to CMU at a regular interval. The fact that data are collected is announced openly. Reporting statistics back is purely optional and can be disabled by the installer, but our suspicion is that most installations do not disable reporting. Since Alex is used to access public files on the internet, security may be less issue for most users than in a file system that may contain protected data.

This data collection provides a rich foundation for the evaluation of an internet wide file system. Each server reports back every 24 hours using “mail”. This communication protocol is not reliable, and some information is invariably lost. No attempt is made to interpolate or reconstruct missing data; all summaries included in this paper are derived from actual data.

At the time of this writing, about 200+ Alex servers are in use all over the internet, with sites in 20+ countries, and report back to CMU. About 4000 machines have mounted an Alex server. There are more than 6000 current users. These servers have fetched over 1,000,000 files, with a cumulative total of over 80 GBytes. The meta data (directory information, size, creation time, etc.) for an even larger number of files is kept in its caches. Even a server with a medium size disk of a few 100 MBytes can easily know about 1,000,000 files. We draw upon this collection of data to support our evaluation of the Alex system. In addition, we have performed more detailed studies (including timing studies) using the 3 local CMU servers.

Table T1 provides some information about the major systems that we use to provide detailed information. Notice that some of these systems have significant disk caches.

Server	Cache Size	Active Users	Client Machines
Burgundy	600 MB	31	203
System2	1,400 MB	144	376

Table 1: Sample sytems

.There are two types of data collection methods used on a server, event counting (which reflects the server’s activity) and a scan of the cache (done once per day). The results of both methods for each day are mailed back to CMU.

Since Alex caches directory information as well, an Alex server contains information about many more files than it actually keeps in the cache. This source is used to collect general statistics about public files on the internet. At CMU we can do a more detailed scan of the information in the cache. The program to do this is derived from a program for walking Alex’s internal directory hierarchy and checking the integrity of the information. It was modified to collect various statistics about the files (e.g. age, creation time, size, etc.) and the directories in the cache..

The server keeps track of various events (e.g., cache hits, consistency checks, forced updates, etc.). To keep the overhead of data collection low, only the most important events are counted.

One of the assumptions of the Alex system is that the characteristics of public files on the internet are different from local files on most general purpose machines, and our data allow us to verify this assumption. The public files change much less often. Figure 2 depicts in a histogram of the age distribution of public files known to the Alex server on “burgundy” (i.e. the file appears in a directory for which the directory information is stored on “burgundy”). The percentage of older files is much larger than has been reported by numerous file system studies (see, e.g., \cite{asplos4}).

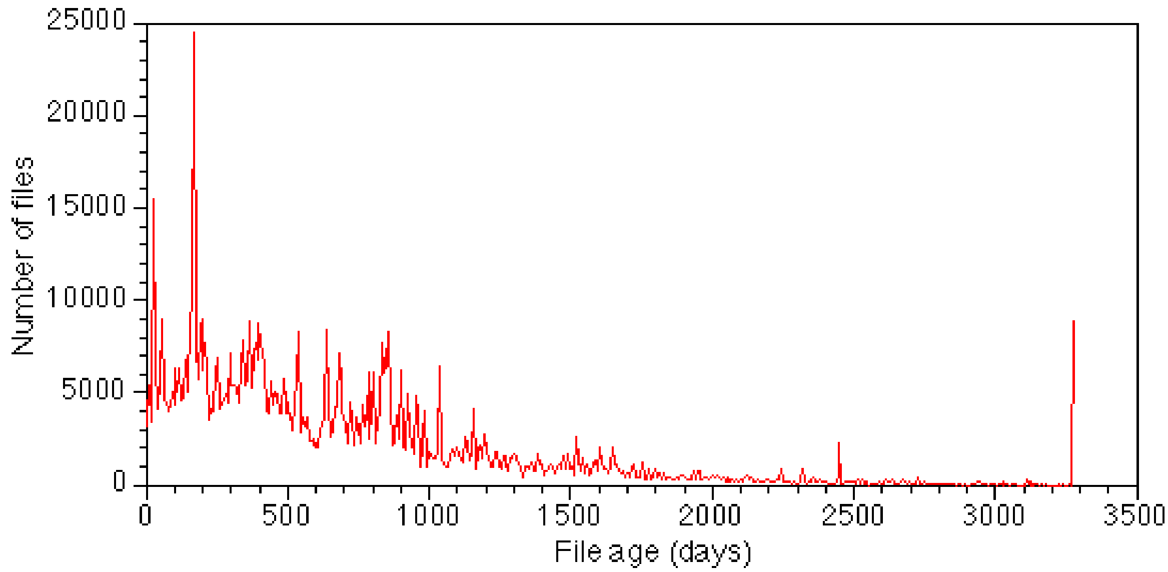

Figure 2 Age distributionof files known to server burgundy.

The preponderance of old files is no surprise since many FTP sites are really archives for papers or software. To fully understand Figure 2, one has to take into account that the internet is not a static, confined laboratory. The number of users on the internet is doubling approximately every 18 months \cite{?}, and we assume the new users were archiving files at the same rate as older users. When looking at the age distribution of these public files, it is clear that most of them are archival. If files were purely archival (i.e. never modified), we would expect 2 times as many current files as 1.5 year (547 days) old files, and 4 times as many current files as 3 year (1,095 day) old files. Looking at Figure 2, this relationship is a fairly accurate approximation of what we observed, which lends support to the claim that most FTP files are not modified on an ongoing basis, but archival in nature.

Figure 3 depicts the distribution of the total file space known to the server burgundy based on the size of the files. This figure shows what percentage of the total file space is taken occupied by files up a certain size. E.g., files with a size of 100KByte or less take up about 14% of the total space.

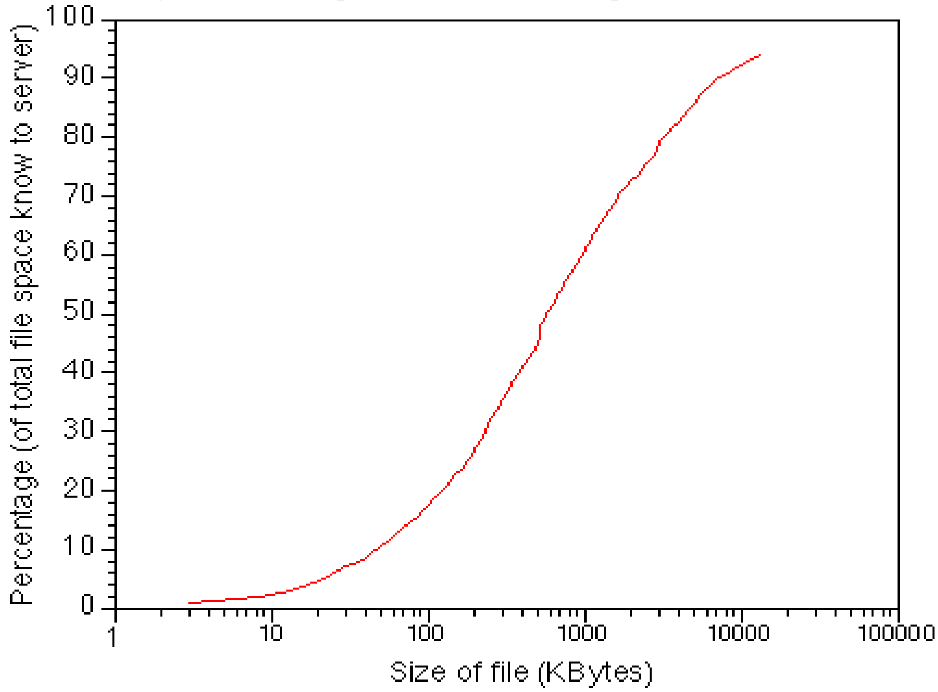

Figure 3 Distribution of total file space based on file size

Figure 4 shows the distribution of the same file space based on the number of files. E.g., 94% of all files are less than 193 KB in size. However, as shown in Figure 3, 50% of the total storage is in files that are over 570 KB. So while the large files are much smaller in number, they make up a significant part of the space occupied on FTP sites.

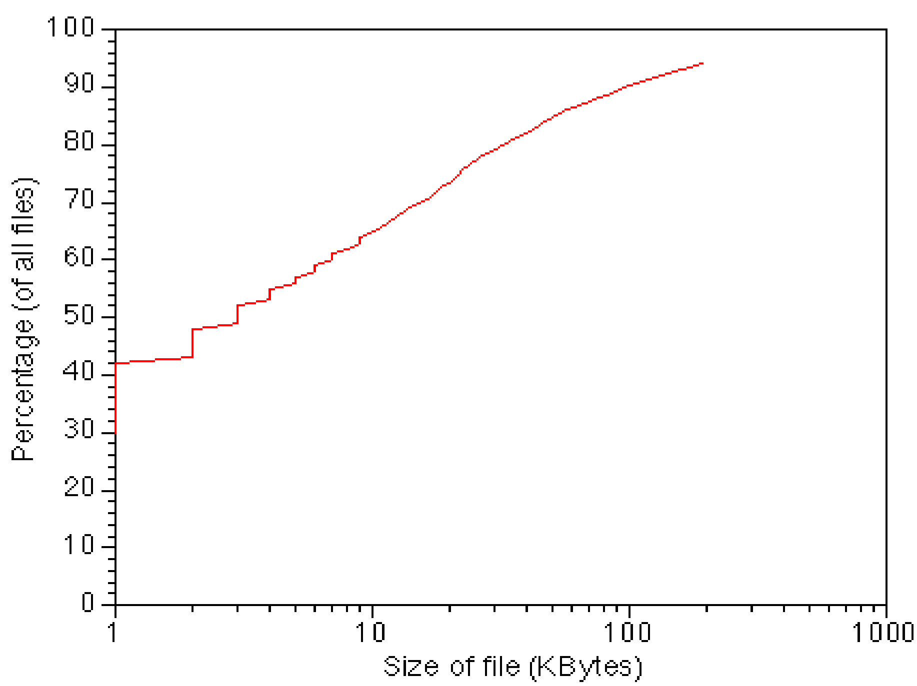

Figure 4 Distribution of total file space based on number of files

So far we have presented only the static data about FTPable files. Next we will address the effectiveness of Caching in Alex. Then we report on the effectiveness of the bounded weak consistency model.

There are three types of caching done in Alex: caching of FTP connections, files, and meta-data of filesCaching connections

As shown in Figure 5,setting up an FTP connection usually takes 5 to 10 seconds, but can take as long as 3 minutes. Once Alex has an FTP connection it keeps it open for up to 5 minutes after it is last used. This means that subsequent accesses to the same site do not need to have this connection setup overhead. Our data shows that on average Alex does 5.4 commands per connection. So the startup overhead is avoided for over 80% of the FTPs.

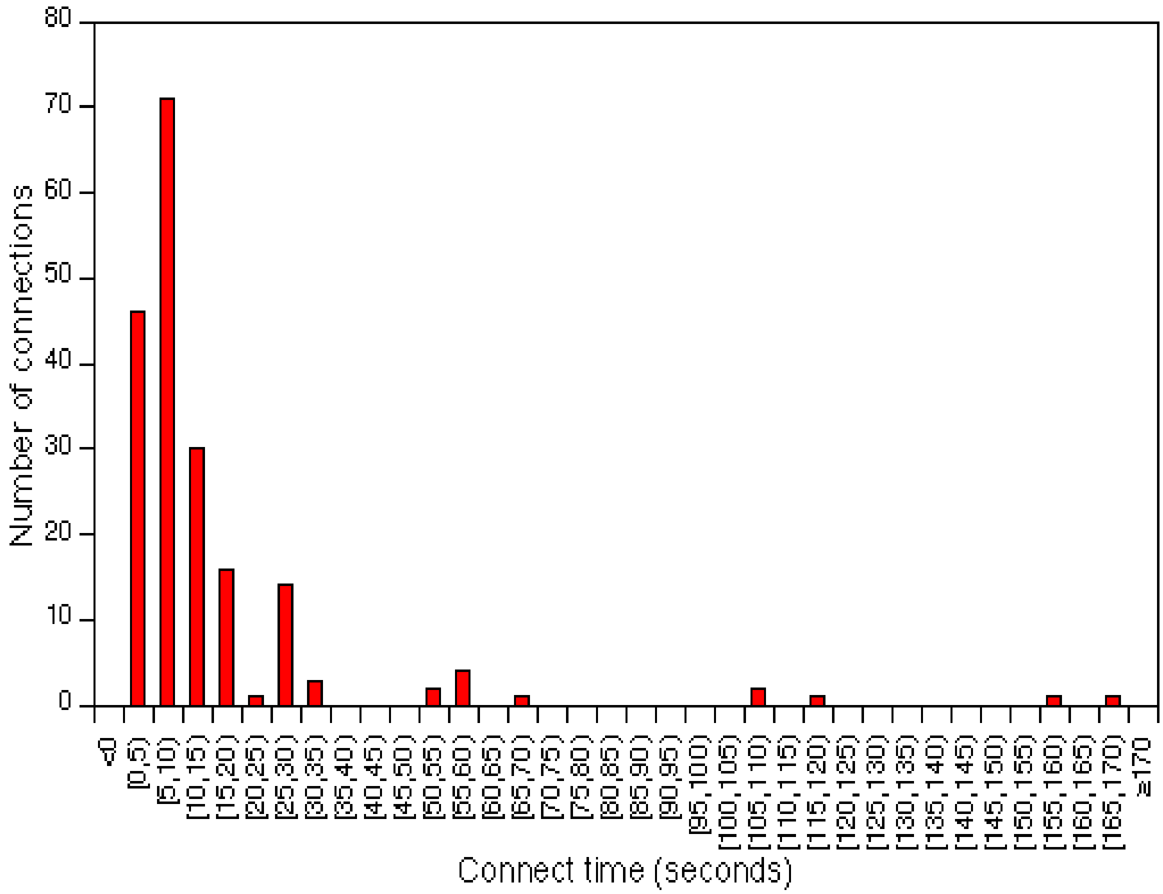

Figure5 FTP Connectlatency

5.1. Files

In this section we report on the cache hit ratio for files. The next figures show the hit rate some representative machines. (A file access counts as hit if the data can be supplied from the cache, even if the Alex server has to first verify that the file is not stale. We discuss bounded weak consistency in the next section).

For each day, the Alex server collects the number of files accessed and the number of files FTPed during that day. These data are then depicted in three different formats: In Figure 6, the X axis represents time. The raw data for a given day are averaged to compute a hit ratio for this day, and the hit rate for a day's acesses is depicted on the Y axis. The size of the circle corresponds to the number of files accessed during that period.

Figure 7 shows the same data for a different system (code named system2, since revealing the names of these NSF servers would allow anyone to mount them). .

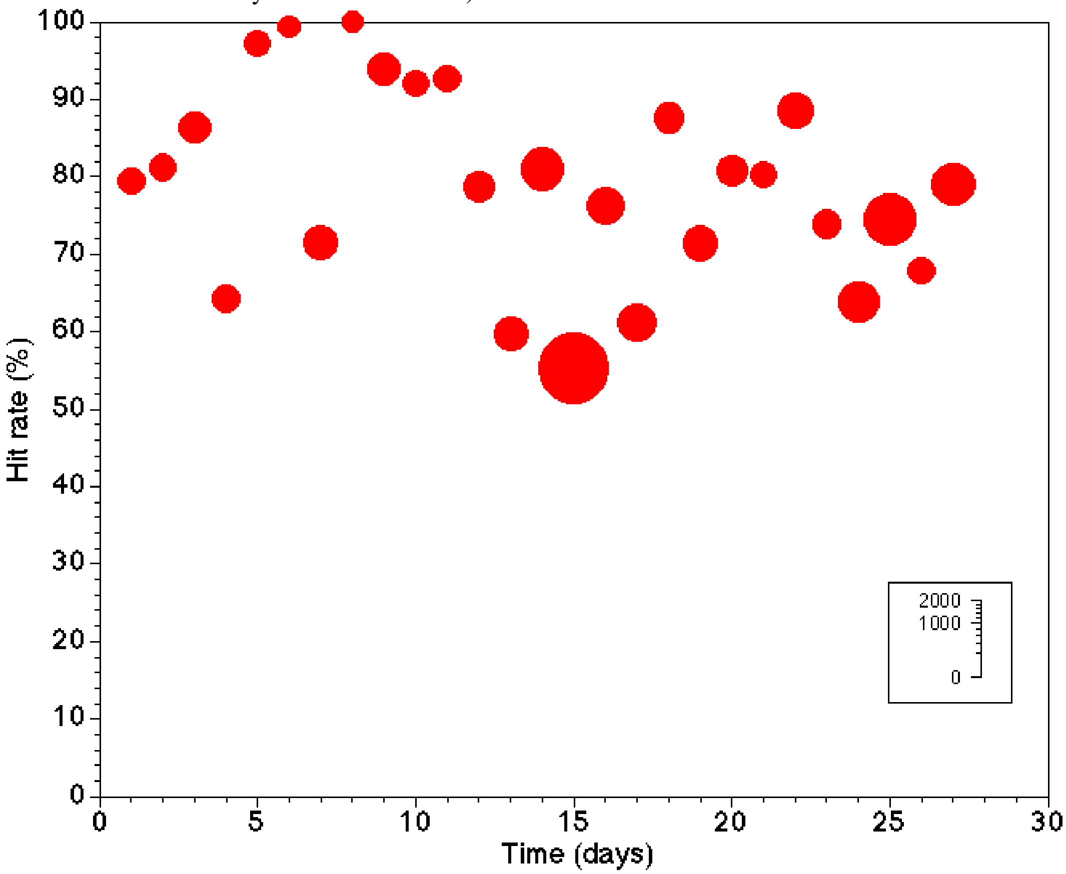

Figure 6 File hit rate (size of circle corresponds to number of files) for system 1

.

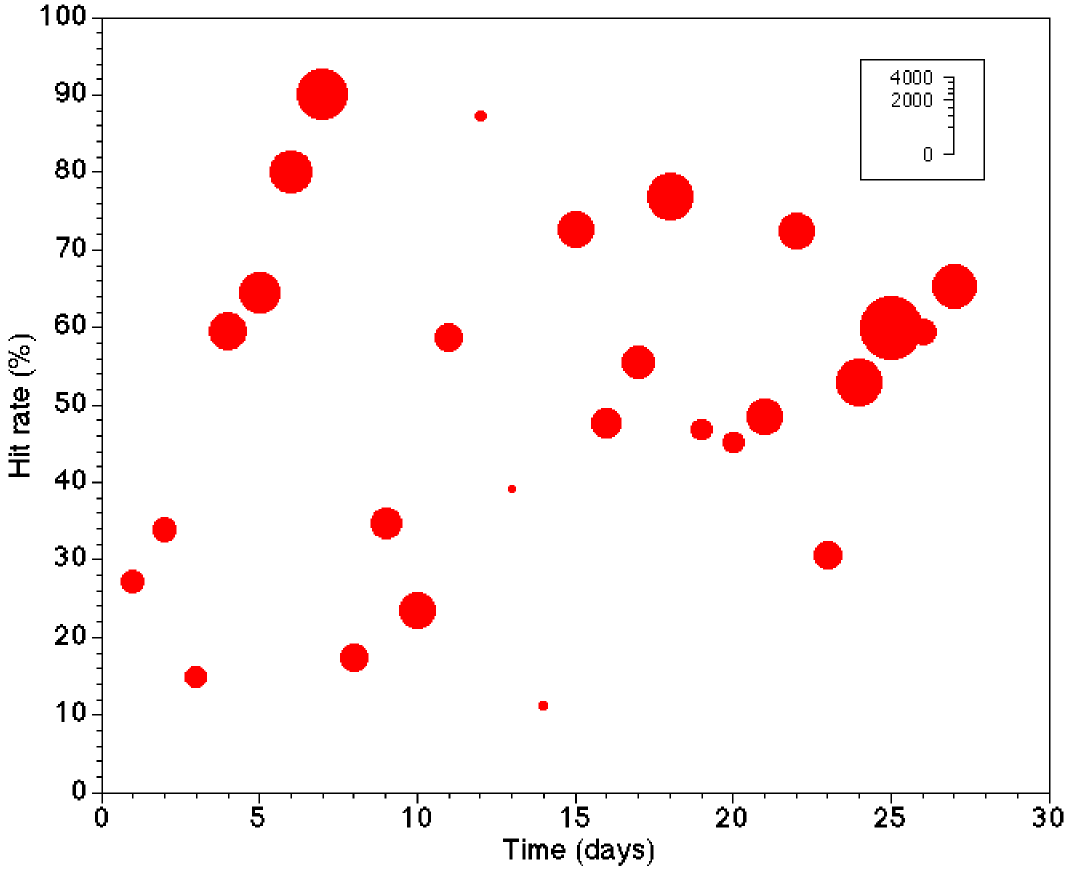

Figure 7 File hit rate (size of circle corresponds to number of files) for system 2

Figure 8 provides another look at the cache hit rate, using the sizes of the files accessed as basis. For each day, this figure shows what percentage of the bytes for Alex files is found in the cache. (It is not possible to use the total number of bytes accessed on this system as a baseline, since we do not instrument the local operating system. There are many reasons for the difference in the hit rate. The Alex server favors small files when faced with a shortage of disk space in its cache and removes larger files first.

Table 2 summaries the file hit rate of Alex servers, broken down for groups with different cache sizes. Obviously, a large cache increases the possibility that a file is still in the cache on a subsequent access, and the data demonstrate the impact of cache size on the hit rate. These data clearly indicate that there is sufficient reuse of information in the Alex servers. There is one very active server in Australia with very high hit rates. Since this skews the results noticably and is not typical of the other results, the second and third lines in the table have this server removed. The final line is just for those running the most recent code

.

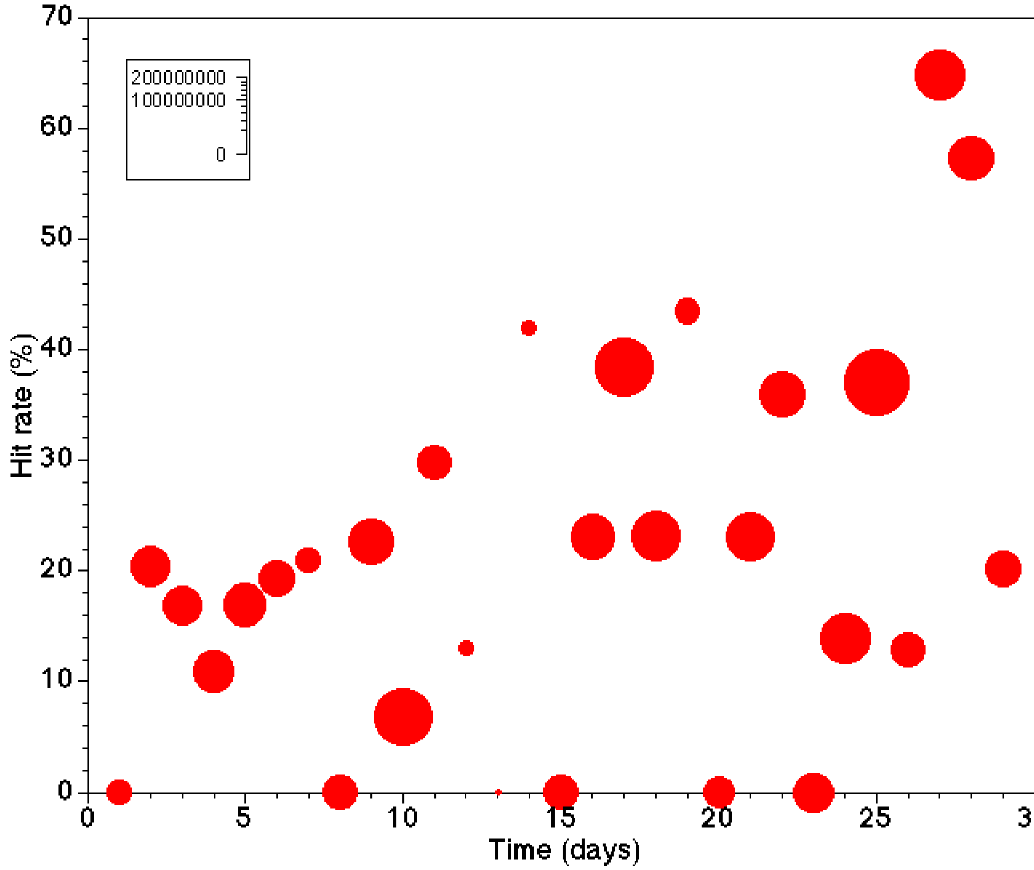

Figure 8 Byte hit rate for system 2

	All	200+MB	500+ MB	1 GB+
All systems	58	79	81	88
All but AU	44	64	61	60
Recent code	48	53	56	62

Table 2: Summary hit rates for files (% of all Alex accesses)

5.2. Meta information

The Alex server maintains information about any directory relevant to accessing the files in its cache. This means the directory a file is located in as well as its parent directories. If a user asks for a listing ("ls" in Unix), this listing can often be provided from the information in the cache. Meta information about files and directories is subject to the same bounded weak consistency model.

Figure 9 shows how often meta information is found in the cache. This figure shows the data for the same system as Figure 7. (To allow an easy comparion of the hit rates for meta data, files, and data bytes, the data form

Figures 7 and 8 are repeated in a more conventional format (hit rate alone without reference to volume). Again, the level of activity differs significantly from day to day.

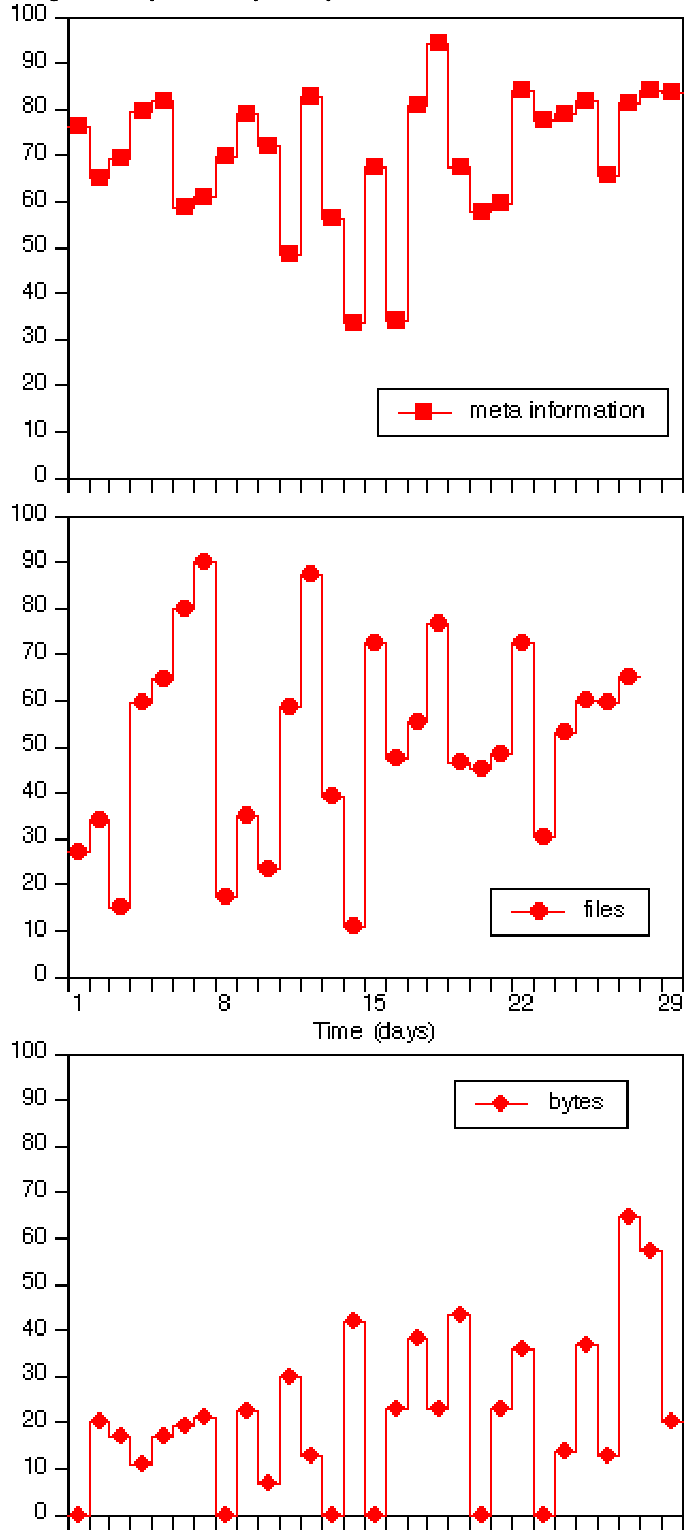

Figure 9 Hit rates for meta information, files, and data bytes (system 2)

Alex also caches a number of other aspects. For each system, it caches the time zone, host operating system, network status, etc. Caching the network status (e.g., unreachable, rejects annonymous ftp) is important since otherwise NFS retries would cause long delays. Alex also maintains a table of known hosts (important to translate file names into host and directory names). These aspects are discussed in a forthcoming thesis.

5.3. Bounded weak consistency

Central to Alex is the idea of a relaxed consistency model. Alex does not always check back with an FTP site to see if the local copy is current before giving it to a user. Thus Alex sometimes gives users old files. The question of how often this happens, and how old the files are is addressed in this section. We will also look at how often users explicitly ask for a directory to be updated. This indicates how much confidence they have in the consistency Alex provides.

Finding out with reasonable effort that a stale file was handed out is not easy but can be done in some cases. For each file, the Alex server keeps track of its date of last use. When Alex updates a directory, it checks to see if any of the cached files have been modified. When a file has been modified on the FTP site and an old is in the cache, Alex removes the file from the cache. However, before Alex removes the file it checks to see if the recorded date of last use is after the time the master file was modified. If the cached copy has been used since the master file was updated, then the Alex server logs a consistency failure. Also, in this case the server can also record how far out of date the file was when it was given out.

Figure 10 illustrates this sequence of events. To get a stale file, a user must read the file twice within a time period less than 1/10th the time since the file was last modified, and between those two uses of the file, the remote file has to be modified.

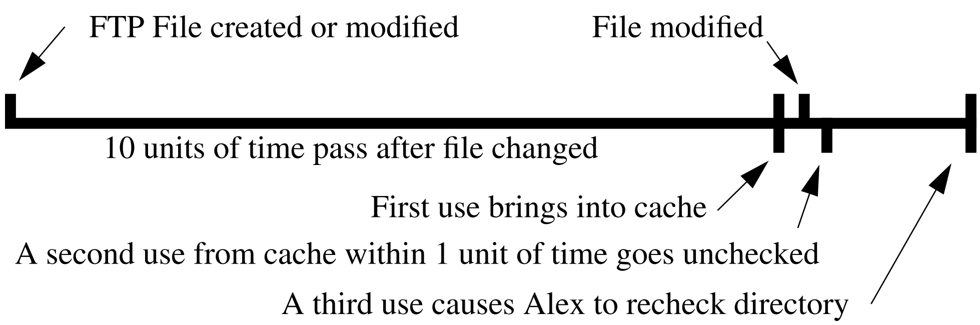

Figure 10 Using and detecting a stale file

To detect such a consistency failure, the local copy must still be in cache at the time the updated directory information is obtained. However, a file might be removed from the cache before the server updates the directory, so the above test alone does not detect all of the times that a stale file is given out.

To get better statistics on how often stale files are given out, the CMU servers check a file before removing it from the cache. If the file has been read since it was first brought into the cache, and if the directory has not been checked since the file was read last, then the CMU servers force an update of that directory. This improves the accuracy of the statistics at the cost of extra work for Alex, the server, and the remote FTP server. This gives us good consistency data for the CMU machines and an idea of how complete the statistics are at other places.

Averaged over all servers, back stale files are given out about 0.1% of the time. The average stale file was 10 days out of date. Using the more accurate method on the CMU servers, we measured that stale files are given out for 0.8% of the accesses. Although this number is higher than the number collected in the field, it is still fairly small.

Our data indicate that on the average, not many directories are updated due to a forced update requests. I.e., even when a user asks for a forced update, most of the time, the cache contains no stale data.

However, there is a failure mode that is not captured with this analysis. There is the possibility that a public file is modified several times, and the most recent modification is after the cached copy was read. Figure 11 shows this scenario. In this case, the Alex server detects a modification after the last use but has no knowledge of the prior modification. We know of no simple way to measure this consistency failure, and since the number of stale files is small, we do not include this case. Checking the status of all files places an unreasonable burden on the servers. Even the simpler test described above and implemented in the CMU servers was enough to bog down the server to the point that it made almost no progress. In the end, we had to set aside a sizeable portion of the cache to keep files that were logically removed but for which the server could not yet check if a stale copy was ever delivered to a client. But even then, performance was affected noticeably. Since a slow server that is shunned by users is not a good vehicle to gather usage information, we refrained from more extensive forms of monitoring.

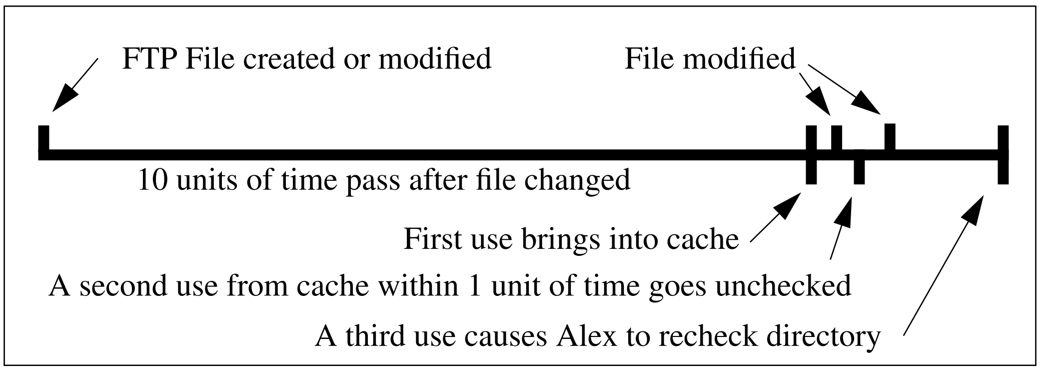

Figure 11 Pattern of usage that results in stale file that cannot be detected easily

The relaxed consistency model of Alex introduces stale files, but this is the only practical solution for an internet wide file system. If Alex were to use a *fixed* window, it would have to check remote FTP sites much more often. There are a number of files on the internet that change every hour. To provide reasonable consistency with a fixed window size, the window size would have to be small enough to work for these rapidly changing files. To do this a fixed size window would have to be something small like 5 minutes. In the data shown multiple NFS accesses within such a short period are counted as one. So with a 5 minute window Alex would have to check the remote site for every NFS access shown. By having an age-dependant window size we have reduced this checking by about a factor of 3.

An interesting issue is how to proceed in the case of an unreachable host at the time a consistency check should be done. An FTP site can be unreachable either because it is down or because of network problems. In this case it is not possible to check that the cached copy is current. In this case, Alex records this fact but lets the user access the cached files. Averaged over all servers, this does happen <in final paper> % of the time a consistency check is required in our model. Note, however, that consistency failures are still detected by the above tests.

6. Conclusions

There are a large number of files in the internet file repositories that match the Alex usage model, and these files are used. Alex provides an attractive method to integrate those files into a user's local system. To provide caching of files from the internet FTP servers, it is necessary to relax the consistency model. Just starting up an FTP connection takes a few seconds in the best and a few minutes in the worst case. If a caching file server had to check the consistency of each file on every access, the speed of the server would be so slow that it would be useless.

Local caching (of such files and other information) is sucessful. This is evidenced by a high hit rate, both for files and directories. The hit rate depends on many factors but even servers with a moderate sized disk cache realize a hit rate above 50%. The hit rate for data bytes is a little lower but still repectable.

Alex's consistency model of bounded weak consistency is well suited to an internet wide file system. The current guarantee (1/10 of the age maximum window of inconsistency) seldomly results in serving a stale file. It is impossible to get a completely acurate figure for the number of stale files served. For the CMU servers (which keep better track of out of date copies, at a higher cost), we record less than 1% stale files served to a client. Another indication for the success of the bounded weak consistency model is the large number of users and sites as well as the fact that users feel rarely the need to perform explicit sychronization. In return, the relaxed consistency model cuts down the work for the server by a factor of 3 and allows the operation of the server even if the original depository of a file is temporarily unreachable. Given our goal of caching files in the internet, relaxed consistency is price worth paying.

References

Vincent Cate, "Alex - a Global Filesystem", Proceedings of the 5th Usenix File System Workshop, May 1992, pp 1-11, /alex/edu/cmu/cs/sp/alex/doc/usenix.wofs92.ps.

A. Emtage, P. Deutsch, "archie - An Electronic Directory Service for the Internet", Usenix Conference Proceedings, pp. 93-110, San Francisco, CA, January 1992.

David J. Ewing, Richard S. Hall, Michael F. Schwartz, "A Measurement Study of Internet File Transfer Traffic", January 1992, Department of Computer Science, University of Colorado at Boulder, CU-CS-571-92, /alex/edu/colorado/cs/bruno/pub/cs/techreports/schwartz/RD.Papers/PostScript/FTP.Meas.ps.Z.

Brewster Kahle, "Wide Area Information Servers", Thinking Machines, April, 1991, /alex/com/think/quake/wais/wais-discussion/overview.text.

J. Howard, M. Kazar, S. Menees,D.Nichols, M. Satyanarayana, R. Sidebothamn, M. West, "Scale and Performance in a Distributed file System", Operating Systems Review, Vol 21, No 5, November 1987. See also /alex/org/central/grand/pub/doc/afs.

B. Clifford Neuman, "The Virtual System Model for Large Distributed Operating Systems", Department of Computer Science, University of Washington, TR-89-01-07, /alex/edu/washington/cs/pub/pfs/doc/UW-CS-89-01-07.PS.Z

Bill Nowicki, "NFS: Network File System Protocol Specification", Request for Comments: 1094, Sun Microsystems, March 1989, /alex/com/sri/nisc/phoebus/rfc/rfc1094.txt.

Francis Prusker, Edward Wobber, “The Siphon: Managing Distant Replicated Repositories”, Proceedings of the workshop on Management of Replicated Data, IEEE, Nov. 1990, pp. 44-47.

J. Postel, J. Reynolds, “File Transfer Protocol (FTP)”, Request for Comments: 959, /alex/com/sri/nisc/phoebus/rfc/rfc959.txt.

Herman Chung-Hwa Rao, “The Jade File System (Ph. D. Dissertation)”, Department of Computer Science, University of Arizona, TR 91-18. See also “Accessing Files in an Internet: The Jade File System”, /alex/edu/arizona/cs/llp/jade.ps