Measurement and Analysis of Large-Scale Network File System Workloads

Andrew W. Leung* Shankar Pasupathy† Garth Goodson† Ethan L. Miller*

*University of California, Santa Cruz
{aleung, elm}@cs.ucsc.edu

†NetApp Inc.
{shankarp, goodson}@netapp.com

Abstract

In this paper we present the analysis of two large-scale network file system workloads. We measured CIFS traffic for two enterprise-class file servers deployed in the NetApp data center for a three month period. One file server was used by marketing, sales, and finance departments and the other by the engineering department. Together these systems represent over 22 TB of storage used by over 1500 employees, making this the first ever large-scale study of the CIFS protocol.

We analyzed how our network file system workloads compared to those of previous file system trace studies and took an in-depth look at access, usage, and sharing patterns. We found that our workloads were quite different from those previously studied; for example, our analysis found increased read-write file access patterns, decreased read-write ratios, more random file access, and longer file lifetimes. In addition, we found a number of interesting properties regarding file sharing, file re-use, and the access patterns of file types and users, showing that modern file system workload has changed in the past 5–10 years. This change in workload characteristics has implications on the future design of network file systems, which we describe in the paper.

1 Introduction

Network file systems are playing an increasingly important role in today's data storage. The motivation to centralize data behind network file systems has been driven by the desire to lower management costs, the need to reliably access growing amounts of data from multiple locations, and is made possible by improvements in processing and network power. The design of these systems is usually guided by an understanding of file system workloads and user behavior [12, 19, 25], which is often obtained by measuring and analyzing file system traces.

While a number of trace-based file system studies have been conducted in the past [3, 5, 21, 24, 29], there are several factors indicating that further study is necessary to aid future network file system designs. First, no major study has been conducted of CIFS (Common Internet File System) [13], the network file transfer protocol used by Windows. Second, the last major trace study [5] analyzed traces from 2001, over half a decade ago; significant changes in the architecture and use of network storage since then have resulted in changes in workload. Third, no published study has ever analyzed large-scale enterprise file system workloads, focusing instead on research-type workloads, such as those seen in the university settings often available to systems researchers.

In this paper, we examine two real-world, large-scale enterprise network file system workloads, collected over three months from two network file servers deployed in NetApp's data center. One server hosts data for the marketing, sales, and finance departments, and the other hosts data for engineering departments. Combined, these systems contain over 22 TB of actively used storage and are used by over 1500 employees. The analysis of our trace data focused on: (1) changes in file access patterns and lifetimes since previous studies, (2) properties of file I/O and file sharing, and (3) the relationship between file type and client access patterns.

Our analysis found important changes in several aspects of file system workloads. For example, we found that read-write file access patterns, which are highly random, are much more common relative to read-only and write-only access patterns as compared to past studies. We also found our workloads to be more write-oriented than those previously studied, with only about twice as many bytes read as written. Both of these findings challenge traditionally-held assumptions about access patterns and sequentiality. A summary of our key observations can be found in Table 1.

In all, our contributions include:

1) The first published study of CIFS workloads.
2) A comparison with past file system studies.
3) A new study of file access, I/O and sharing patterns.

Compared to Previous Studies
1. Both of our workloads are more write-oriented. Read to write byte ratios have signifcantly decreased.
2. Read-write access patterns have increased 30-fold relative to read-only and write-only access patterns.
3. Most bytes are transferred in longer sequential runs. These runs are an order of magnitude larger.
4. Most bytes transferred are from larger files. File sizes are up to an order of magnitude larger.
5. Files live an order of magnitude longer. Fewer than 50% are deleted within a day of creation.
New Observations
6. Files are rarely re-opened. Over 66% are re-opened once and 95% fewer than five times.
7. Files re-opens are temporally related. Over 60% of re-opens occur within a minute of the first.
8. A small fraction of clients account for a large fraction of file activity. Fewer than 1% of clients account for 50% of file requests.
9. Files are infrequently shared by more than one client. Over 76% of files are never opened by more than one client.
10. File sharing is rarely concurrent and sharing is usually read-only. Only 5% of files opened by multiple clients are concurrent and 90% of sharing is read-only.
11. Most file types do not have a common access pattern.

Table 1: *Summary of observations.* *A summary of important file system trace observations from our trace analysis. Note that we define clients to be unique IP addresses, as described in Section 4.1.*

4) An analysis of file type and user session patterns in network file systems.
5) A discussion of the significant design implications derived from our study.

2 Background

In this section, we discuss previous studies, summarized in Table 2, and outline factors we believe motivate the need for new file system analysis. In addition, we provide a brief background of the CIFS protocol.

2.1 Past Studies

Early file system studies, such as those of the BSD [21], VAX/VMS [22], and Sprite [3] file systems, revealed a number of important observations and trends that guided file system design for over two decades. In particular, they observed a significant tendency towards large, sequential read access, limited read-write access, bursty I/O patterns, and very short file lifetimes. Other studies uncovered additional information such as file size distributions [18, 26] and workload self-similarity [10]. A more recent study of the Windows NT file system [29] supported a number of the past observations and trends. In 2000, Roselli, *et al.* compared four file system workloads [24]; they noted that block lifetimes had increased since past studies and explored the effect on caching strategies. The most recent study, in 2001, analyzed NFS traffic to network file servers [5], identifying a number of peculiarities with NFS tracing and arguing that pathnames can aid file system layout.

In addition to trace-based studies, which analyze file system workloads, several snapshot-based studies have analyzed file system contents [2, 4, 7, 8]. These studies showed how file attributes, such as size and type, are distributed throughout the name-space, and also how file system contents change over time.

2.2 The Need for a New Study

Although there have been a number of previous file system trace studies, several factors that indicate that a new study may aid ongoing network file system design.

Time since last study. There have been significant changes in computing power, network bandwidth, and network file system usage since the last major study in 2001 [5]. A new study will help understand how these changes impact network file system workloads.

Few network file system studies. Only a few studies have explored network file system workloads [3, 5, 22], despite their differences from local file systems. Local file systems workloads include the access patterns of many system files, which are generally read-only and sequential, and are focused on the client's point of view. While such studies are useful for understanding client workload, it is critical for network file systems to focus on the workload seen at the server, which often excludes system files or accesses that hit the client cache.

No CIFS protocol studies. The only major network file system study in the past decade analyzed NFSv2 and v3 workloads [5]. Though NFS is common on UNIX systems, most Windows systems use CIFS. Given the widespread Windows client population and differences between CIFS and NFS (*e. g.*, CIFS is a stateful protocol, in contrast to NFSv2 and v3), analysis beyond NFS can add more insight into network file system workloads.

Limited file system workloads. University [3, 5, 10, 21, 24] and personal computer [2, 4, 32] workloads have been the focus of a number of past studies. While useful, these workloads may differ from the workloads of network file systems deployed in other environments.

Study	Date of Traces	FS/Protocol	Network FS	Trace Approach	Workload
Ousterhout, *et al*. [21]	1985	BSD		Dynamic	Engineering
Ramakrishnan, *et al*. [22]	1988-89	VAX/VMS	x	Dynamic	Engineering, HPC, Corporate
Baker, *et al*. [3]	1991	Sprite	x	Dynamic	Engineering
Gribble, *et al*. [10]	1991-97	Sprite, NFS, VxFS	x	Both	Engineering, Backup
Douceur and Bolosky [4]	1998	FAT, FAT32, NTFS		Snapshots	Engineering
Vogels [29]	1998	FAT, NTFS		Both	Engineering, HPC
Zhou and Smith [32]	1999	VFAT		Dynamic	PC
Roselli *et al*. [24]	1997-00	VxFS, NTFS		Dynamic	Engineering, Server
Ellard, *et al*. [5]	2001	NFS	x	Dynamic	Engineering, Email
Agrawal, *et al*. [2]	2000-2004	FAT, FAT32, NTFS		Snapshots	Engineering
Leung, *et al*.	2007	CIFS	x	Dynamic	Corporate,Engineering

Table 2: ***Summary of major file system studies over the past two decades.*** *For each study, we show the date of trace data, the file system or protocol studied, whether it involved network file systems, the trace methodology, and the workloads studied. Dynamic trace studies involve traces of live requests. Snapshot trace studies involve snapshots of file system contents.*

2.3 The CIFS Protocol

The CIFS protocol, which is based on the Server Message Block (SMB) protocol that defines most of the file transfer operations used by CIFS, differs in a number of respects from oft-studied NFSv2 and v3. Most importantly, CIFS is stateful: CIFS user and file operations act on stateful session IDs and file handles, making analysis of access patterns simpler and more accurate than in NFSv2 and v3, which require heuristics to infer the start and end of an access [5]. Although CIFS may differ from other protocols, we believe our observations are *not* tied exclusively to CIFS because access patterns, file sharing, and other workload characteristics observed by the file server are influenced by the file system users, their applications, and the behavior of the file system client, none of which are closely tied to the transfer protocol.

3 Tracing Methodology

We collected CIFS network traces from two large-scale, enterprise-class file servers deployed in the NetApp corporate headquarters. One is a mid-range file server with 3 TB of total storage, with almost 3 TB used, deployed in the corporate data center that hosts data used by over 1000 marketing, sales, and finance employees. The other is a high-end file server with over 28 TB of total storage, with 19 TB used, deployed in the engineering data center. It is used by over 500 engineering employees. Throughout the rest of this paper, we refer to these workloads as *corporate* and *engineering*, respectively.

All NetApp storage servers support multiple protocols including CIFS, NFS, iSCSI, and Fibre Channel. We traced *only* CIFS on each file server. For the corporate server, CIFS was the primary protocol used, while the engineering server saw a mix of CIFS and NFS protocols. These servers were accessed by desktops and laptops running primarily Windows for the corporate environment and a mix of Windows and Linux for the engineering environment. A small number of clients also ran Mac OS X and FreeBSD. Both servers could be accessed through a Gigabit Ethernet LAN, a wireless LAN, or via a remote VPN.

Our traces were collected from both the corporate and engineering file servers between August 10th and December 14th, 2007. For each server, we mirrored a port on the network switch to which it was connected and attached it to a Linux workstation running `tcpdump` [28]. Since CIFS often utilizes NetBIOS [1], the workstation recorded all file server traffic on the NetBIOS name, datagram, and session service ports as well as traffic on the CIFS port. The trace data was periodically copied to a separate file server. Approximately 750 GB and 1.5 TB of uncompressed `tcpdump` traces were collected from the corporate and engineering servers, respectively. All traces were post-processed with `tshark 0.99.6` [30], a network packet analyzer. All filenames, usernames, and IP addressed were anonymized.

Our analysis of CIFS presented us with a number of challenges. CIFS is a stream-based protocol, so CIFS headers do not always align with TCP packet boundaries. Instead, CIFS relies on NetBIOS to define the length of the CIFS command and data. This became a problem during peak traffic periods when `tcpdump` dropped a few packets, occasionally causing a NetBIOS session header to be lost. Without the session header, `tshark` was unable to locate the beginning of the next CIFS packet within the TCP stream, though it was able to recover when it found a new session header aligned with the start of a TCP packet. To recover CIFS requests that fell within this region, we wrote a program to parse the `tcpdump` data and extract complete CIFS packets based on a signature of the CIFS header while ignoring any NetBIOS session information.

Another issue we encountered was the inability to correlate CIFS sessions to usernames. CIFS is a session-based protocol in which a user begins a session by connecting to the file server via an authenticated login pro-

cess. However, authentication in our environment almost always uses Kerberos [20]. Thus, regardless of the actual user, user authentication credentials are cryptographically changed with each login. As a result, we were unable to match a particular user across multiple sessions. Instead we relied on the client's IP address to correlate users to sessions. While less accurate, it provides a reasonable estimate of users since most users access the servers via the LAN with the same IP address.

4 Trace Analysis

This section presents the results of our analysis of our corporate and engineering CIFS workloads. We first describe the terminology used throughout our analysis. Our analysis begins with a comparison of our workloads and then a comparison to past studies. We then analyze workload activity, with a focus on I/O and sharing distributions. Finally, we examine properties of file type and user session access patterns. We italicize our key observations following the section in which they are discussed.

4.1 Terminology

Our study relies on several frequently used terms to describe our observations. Thus, we begin by defining the following terms:

I/O A single CIFS read or write command.

Sequential I/O An I/O that immediately follows the previous I/O to a file within an open/close pair (i.e., its offset equals the sum of the previous I/O's offset and length). The first I/O to an open file is always considered sequential.

Random I/O An I/O that is not sequential.

Sequential Run A series of sequential I/Os. An opened file may have multiple sequential runs.

Sequentiality Metric The fraction of bytes transferred sequentially. This metric was derived from a similar metric described by Ellard, *et al.* [5].

Open Request An open request for a file that has at least one subsequent I/O and for which a close request was observed. Some CIFS metadata operations cause files to be opened without ever being read or written. These open requests are artifacts of the CIFS client implementation, rather than the workload, and are thus excluded.

Client A unique IP address. Since Kerberos authentication prevents us from correlating usernames to users, we instead rely on IP address to identify unique clients.

	Corporate	Engineering
Clients	5261	2654
Days	65	97
Data read (GB)	364.3	723.4
Data written (GB)	177.7	364.4
R:W I/O ratio	3.2	2.3
R:W byte ratio	2.1	2.0
Total operations	228 million	352 million
Operation name	%	%
Session create	0.4	0.3
Open	12.0	11.9
Close	4.6	5.8
Read	16.2	15.1
Write	5.1	6.5
Flush	0.1	0.04
Lock	1.2	0.6
Delete	0.03	0.006
File stat	36.7	42.5
Set attribute	1.8	1.2
Directory read	10.3	11.8
Rename	0.04	0.02
Pipe transactions	1.4	0.2

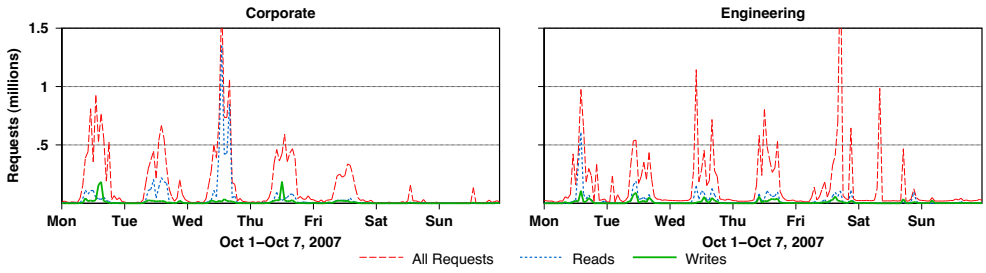
Table 3: ***Summary of trace statistics.*** *File system operations broken down by workload. All operations map to a single CIFS command except for file stat (composed of* `query_path_info` *and* `query_file_info`*) and directory read (composed of* `find_first2` *and* `find_next2`*). Pipe transactions map to remote IPC operations.*

4.2 Workload Comparison

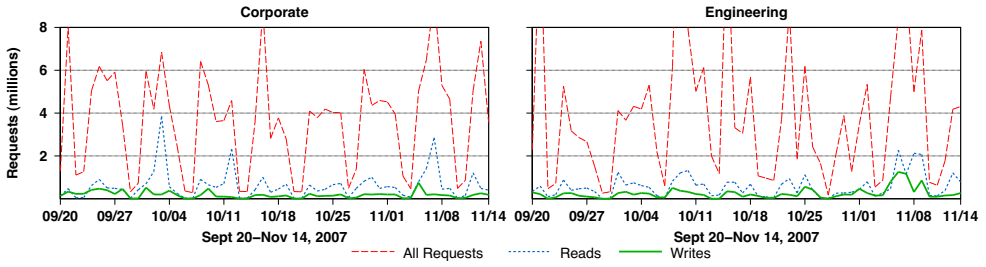
Table 3 shows a summary comparison of overall characteristics for both corporate and engineering workloads. For each workload we provide some general statistics along with the frequency of each CIFS request. Table 3 shows that engineering has a greater number of requests, due to a longer tracing period, though, interestingly, both workloads have similar request percentages. For both, about 21% of requests are file I/O and about 50% are metadata operations. There are also a number of CIFS-specific requests. Our I/O percentages differ from NFS workloads, in which 30–80% of all operations were I/O [5, 27]. This difference can likely attributed to both differences in workload and protocol.

Total data transferred in the two traces combined was just over 1.6 TB of data, which is less than 10% of the file servers' active storage of over 22 TB of data. Since the data transfer summaries in Table 3 include files that were transferred multiple times, our observations show that somewhat more than 90% of the active storage on the file servers was untouched over the three month trace period.

Read/write byte ratios have decreased significantly compared to past studies [3, 5, 24]. We found only a 2:1 ratio, in contrast to past studies that found ratios of 4:1 or higher, indicating workloads are becoming less read-centric. We believe that a key reason for the decrease in the read-write ratio is that client caches absorb a significant percentage of read requests. It is also interesting



(a) Requests over a week.



(b) Requests over 9 weeks.

Figure 1: ***Request distribution over time.*** *The frequency of all requests, read requests, and write requests are plotted over time. Figure 1(a) shows how the request distribution changes for a single week in October 2007. Here request totals are grouped in one hour intervals. The peak one hour request total for corporate is 1.7 million and 2.1 million for engineering. Figure 1(b) shows the request distribution for a nine week period between September and November 2007. Here request totals are grouped into one day intervals. The peak one day intervals are 9.4 million for corporate and 19.1 million for engineering.*

that the corporate and engineering request mix are similar, perhaps because of similar work being performed on the respective clients (*e. g.*, Windows office workloads) or because client caching and I/O scheduling obfuscate the application and end-user behavior. **Observation 1** *Both of our workloads are more write-heavy than workloads studied previously.*

Figures 1(a) and 1(b) show the distribution of total CIFS requests and I/O requests for each workload over a one week period and a nine week period, respectively. Figure 1(a) groups request counts into hourly intervals and Figure 1(b) uses daily intervals. Figure 1(a) shows, unsurprisingly, that both workloads have strongly diurnal cycles and that there are very evident peak and idle periods throughout a day. The cyclic idle periods show there is opportunity for background processes, such as log-cleaning and disk scrubbing to run without interfering with user requests.

Interestingly, there is a significant amount of variance between individual days in the number and ratio of both requests and I/O. In days where the number of total requests are increased, the number of read and write requests are not necessarily increased. This is also the case between weeks in Figure 1(b). The variation between total requests and I/O requests implies any that single day or week is likely an insufficient profile of the overall workload, so it is probably inaccurate to extrapolate trace observations from short time periods to longer periods, as was also noted in past studies [5, 10, 29]. It is interesting to note that the overall request mix presented in Table 3 is different from the mix present in any single day or week, suggesting that the overall request mix might be different if a different time period were traced and is influenced more by workload than by behavior of the file system client.

4.3 Comparison to Past Studies

In this subsection, we compare our CIFS network file system workloads with those of past studies, including those in NFS [5], Sprite [3], VxFS [24] and Windows NT [29] studies. In particular, we analyze how file access patterns and file lifetimes have changed. For comparison purposes, we use tables and figures consistent with those of past studies.

4.3.1 File Access Patterns

Table 4 provides a summary comparison of file access patterns, showing access patterns in terms of both I/O requests and bytes transferred. Access patterns are categorized by whether a file was accessed read-only, write-only, or read-write. Sequential access is divided into two categories, *entire* accesses, which transfer the entire file, and *partial* accesses, which do not.

File System Type	Network							Local		
Workload	Corporate		Engineering		CAMPUS	EECS	Sprite	Ins	Res	NT
Access Pattern	I/Os	Bytes	I/Os	Bytes	Bytes	Bytes	Bytes	Bytes	Bytes	Bytes
Read-Only (% total)	39.0	52.1	50.6	55.3	53.1	16.6	83.5	98.7	91.0	59.0
Entire file sequential	13.5	10.5	35.2	27.4	47.7	53.9	72.5	86.3	53.0	68.0
Partial sequential	58.4	69.2	45.0	55.0	29.3	36.8	25.4	5.9	23.2	20.0
Random	28.1	20.3	19.8	17.6	23.0	9.3	2.1	7.8	23.8	12.0
Write-Only (% total)	15.1	25.2	17.3	23.6	43.8	82.3	15.4	1.1	2.9	26.0
Entire file sequential	21.2	36.2	15.6	35.2	37.2	19.6	67.0	84.7	81.0	78.0
Partial sequential	57.6	55.1	63.4	61.0	52.3	76.2	28.9	9.3	16.5	7.0
Random	21.2	8.7	21.0	3.8	10.5	4.1	4.0	6.0	2.5	15.0
Read-Write (% total)	45.9	22.7	32.1	21.1	3.1	1.1	1.1	0.2	6.1	15.0
Entire file sequential	7.4	0.1	0.4	0.1	1.4	4.4	0.1	0.1	0.0	22.0
Partial sequential	48.1	78.3	27.5	50.0	0.9	1.8	0.0	0.2	0.3	3.0
Random	44.5	21.6	72.1	49.9	97.8	93.9	99.9	99.6	99.7	74.0

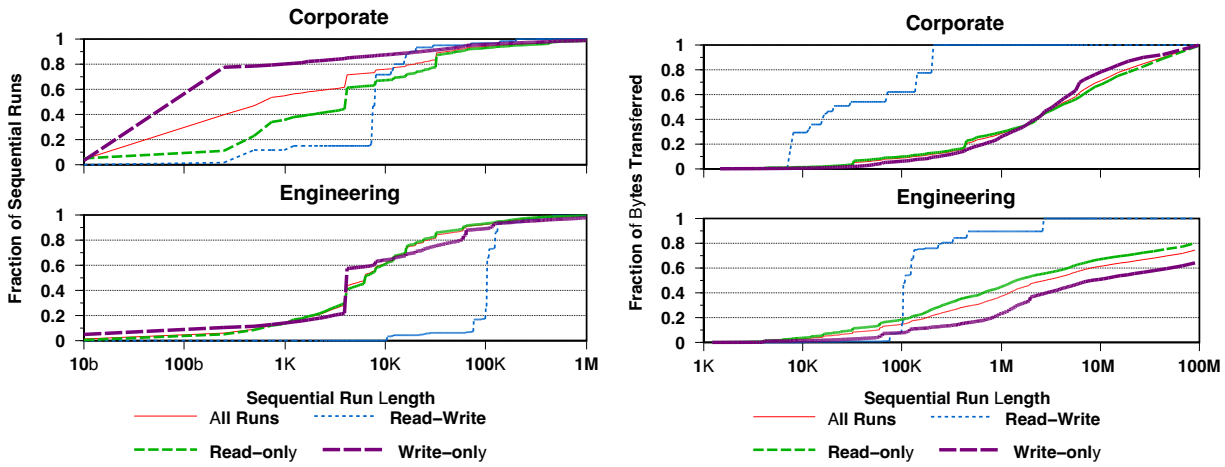
Table 4: ***Comparison of file access patterns.*** *File access patterns for our corporate and engineering workloads are compared with those of previous studies. CAMPUS and EECS [5] are university NFS mail server and home directory workloads, respectively. Sprite [3], Ins and Res [24] are university computer lab workloads. NT [29] is a combination of development and scientific workloads.*

Table 4 shows a remarkable increase in the percentage of read-write I/O and bytes transferred. Most previous studies observed less than 7% of total bytes transferred to files accessed read-write. However, we find that both our corporate and engineering workloads have over 20% of bytes transferred in read-write accesses. Furthermore, 45.9% of all corporate I/Os and 32.1% of all engineering I/Os are in read-write accesses. This shows a diversion from the read-only oriented access patterns of past workloads. When looking closer at read-write access patterns we find that sequentiality has also changed; 78.3% and 50.0% of bytes are transferred sequentially as compared to roughly 1% of bytes in past studies. However, read-write patterns are still very random relative to read-only and write-only patterns. These changes may suggest that network file systems store a higher fraction of mutable data, such as actively changing documents, which make use of the centralized and shared environment and a smaller fraction of system files, which tend to have more sequential read accesses. These changes may also suggest that the sequential read-oriented patterns for which some file systems are designed [16] are less prevalent in network file systems, and write-optimized file systems [11, 25] may be better suited. **Observation 2** *Read-write access patterns are much more frequent compared to past studies.*

4.3.2 Sequentiality Analysis

We next compared the sequential access patterns found in our workloads with past studies. A sequential run is defined as a series of sequential I/Os to a file. Figure 2(a) shows the distribution of sequential run lengths. We see that sequential runs are short for both workloads, with almost all runs shorter than 100 KB, consistent with past studies. This suggests that file systems should continue to optimize for short sequential common-case accesses. However, Figure 2(b), which shows the distribution of bytes transferred during sequential runs, has a very different implication, indicating that many bytes are transferred in long sequential runs: between 50–80% of bytes are transferred in runs of 10 MB or less. In addition, the distribution of sequential runs for the engineering workload is long-tailed, with 8% of bytes transferred in runs longer than 400 MB. Interestingly, read-write sequential runs exhibit very different characteristics from read-only and write-only runs: most read-write bytes are transferred in much smaller runs. This implies that the interactive nature of read-write accesses is less prone to very large transfers, which tend to be mostly read-only or write-only. Overall, we found that most bytes are transferred in much larger runs—up to 1000 times longer—when compared to those observed in past studies, though most runs are short. Our results suggest file systems must continue to optimize for small sequential access, though they must be prepared to handle a small number of very large sequential accesses. This also correlates with the heavy-tailed distributed of file sizes, which we discuss later; for every large sequential run there must be at least one large file. **Observation 3** *Bytes are transferred in much longer sequential runs than in previous studies.*

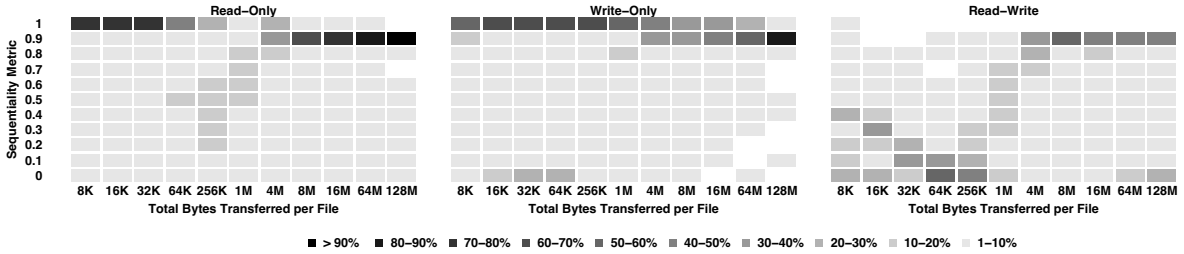
We now examine the relationship between request sizes and sequentiality. In Figure 3(a) and Figure 3(b) we plot the number of bytes transferred from a file against the sequentiality of the transfer. This is measured using the sequentiality metric: the fraction of bytes transferred sequentially, with values closer to one meaning higher sequentiality. Figures 3(a) and 3(b) show this information in a heat map in which darker regions indicate a higher fraction of transfers with that sequentiality met-



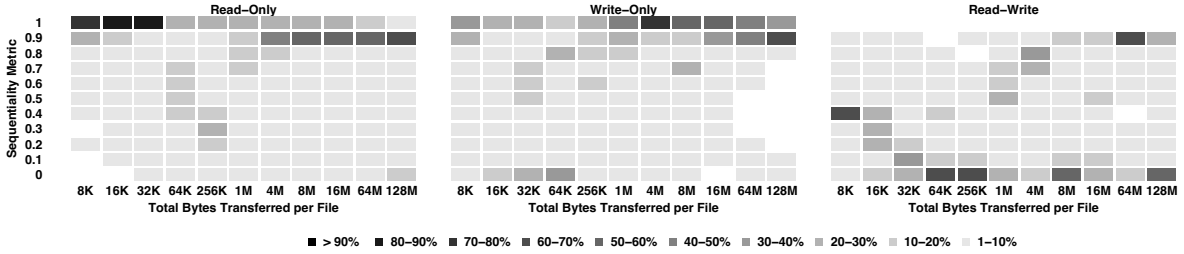
(a) Length of sequential runs.

(b) Bytes transferred in sequential runs.

Figure 2: ***Sequential run properties.*** *Sequential access patterns are analyzed for various sequential run lengths. Figure 2(a) shows the length of sequential runs, while Figure 2(b) shows how many bytes are transferred in sequential runs.*



(a) Corporate.

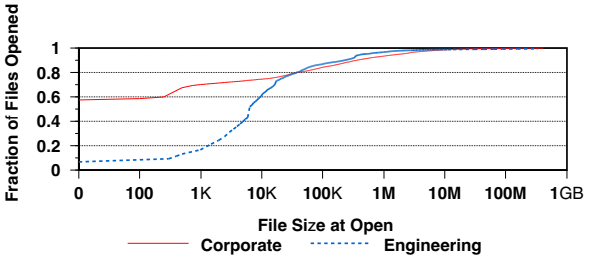


(b) Engineering.

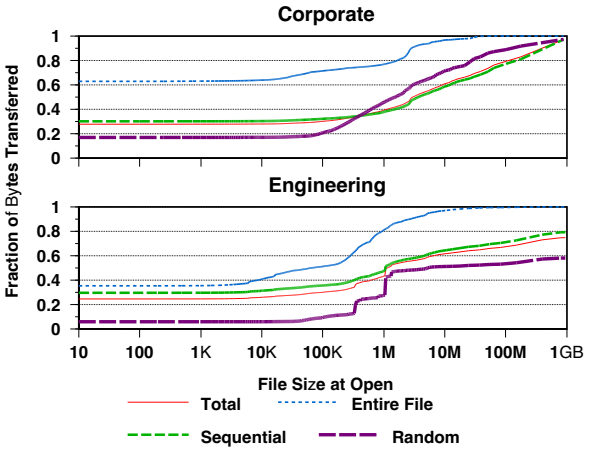
Figure 3: ***Sequentiality of data transfer****. The frequency of sequentiality metrics is plotted against different data transfer sizes. Darker regions indicate a higher fraction of total transfers. Lighter regions indicate a lower fraction. Transfer types are broken into read-only, write-only, and read-write transfers. Sequentiality metrics are grouped by tenths for clarity.*

ric and lighter regions indicate a lower fraction. Each region within the heat map represents a 10% range of the sequentiality metric. We see from Figures 3(a) and 3(b) that small transfers and large transfers are more sequential for read-only and write-only access, which is the case for both workloads. However, medium-sized transfers, between 64 KB and 4 MB, are more random. For large and small transfers, file systems may be able to anticipate high sequentiality for read-only and write-only access. Read-write accesses, on the other hand, are much more random for most transfer sizes. Even very large read-write transfers are not always very sequential, which follows from our previous observations in Figure 2(b), suggesting that file systems may have difficulty anticipating the sequentiality of read-write accesses.

Next, we analyze the relationship between file size and access pattern by examining the size of files at open time to determine the most frequently opened file sizes and the file sizes from which most bytes are transferred. It should be noted that since we only look at opened files, it is possible that this does not correlate to the file size distribution across the file system. Our results are shown



(a) Open requests by file size.



(b) Bytes transferred by file size.

Figure 4: ***File size access patterns****. The distribution of open requests and bytes transferred are analyzed according to file size at open. Figure 4(a) shows the size of files most frequently opened. Figure 4(b) shows the size of files from which most bytes are transferred.*

in Figures 4(a) and 4(b). In Figure 4(a) we see that 57.5% of opens in the corporate workload are to newly-created files or truncated files with zero size. However, this is not the case in the engineering workload, where only 6.3% of opens are to zero-size files. Interestingly, both workloads find that most opened files are small; 75% of opened files are smaller than 20 KB. However, Figure 4(a) shows that most bytes are transferred from much larger files. In both workloads, we see that only about 60% of bytes are transferred from files smaller than 10 MB. The engineering distribution is also long-tailed with 12% of bytes being transferred from files larger than 5 GB. By comparison, almost all of the bytes transferred in previous studies came from files smaller than 10 MB. These observations suggest that larger files play a more significant role in network file system workloads than in those previously studied. This may be due to frequent small file requests hitting the local client cache. Thus, file systems should still optimize small file layout for frequent access and large file layout for large transfers. **Observation 4** *Bytes are transferred from much larger files than in previous studies.*

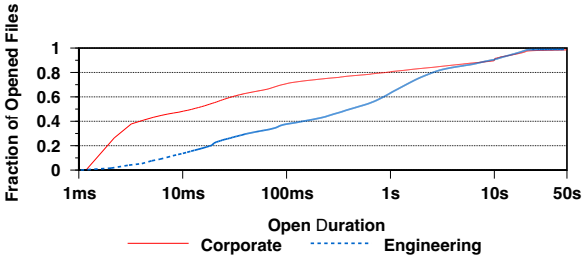


Figure 5: ***File open durations.*** *The duration of file opens is analyzed. Most files are opened very briefly, although engineering files are opened slightly longer than corporate files.*

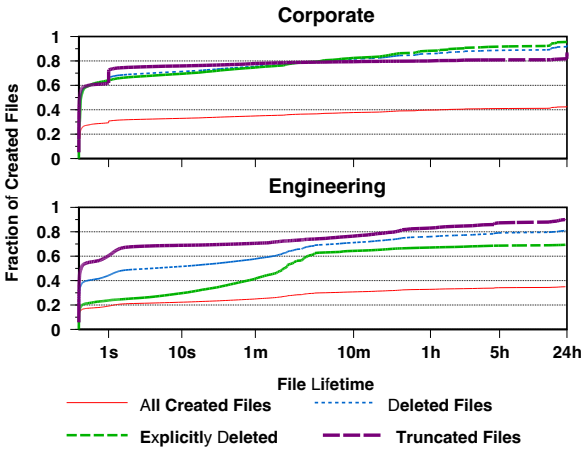


Figure 6: ***File lifetimes.*** *The distributions of lifetimes for all created and deleted files are shown. Files may be deleted through explicit delete request or truncation.*

Figure 5 shows the distribution of file open durations. We find that files are opened for shorter durations in the corporate workload than in the engineering workload. In the corporate workload, 71.1% of opens are shorter than 100 ms, but just 37.1% are similarly short in the engineering workload. However, for both workloads most open durations are less than 10 seconds, which is similar to observations in past studies. This is also consistent with our previous observations that small files, which likely have short open durations, are most frequently accessed.

4.3.3 File Lifetime

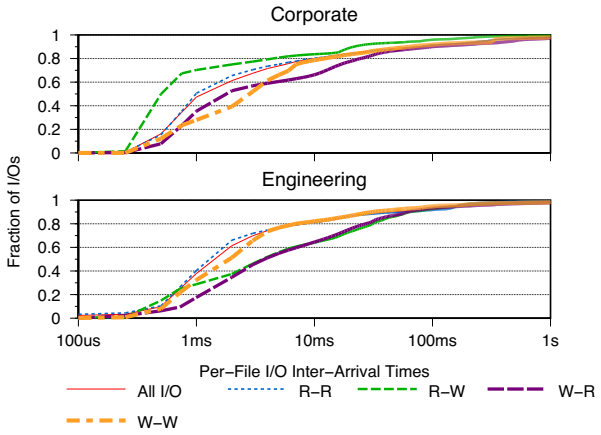
This subsection examines how file lifetimes have changed as compared to past studies. In CIFS, files can be either deleted through an explicit delete request, which frees the entire file and its name, or through truncation, which only frees the data. Figure 6 shows the distribution of file lifetimes, broken down by deletion method. We find that most created files live longer than 24 hours, with 57.0% and 64.9% of corporate and engineering files persisting for more than a day. Both dis-

tributions are long-tailed, meaning many files live well beyond 24 hours. However, files that *are* deleted usually live less than a day: only 18.7% and 6.9% of eventually-deleted files live more than 24 hours. Nonetheless, compared to past studies in which almost all deleted files live less than a minute, deleted files in our workloads tend to live much longer. This may be due to fewer temporary files being created over the network. However, we still find some files live very short lifetimes. In each workload, 56.4% and 38.6% of deleted files are deleted within 100 ms of creation, indicating that file systems should expect fewer files to be deleted and files that live beyond than a few hundred milliseconds to have long lifetimes. **Observation 5** *Files live an order of magnitude longer than in previous studies.*

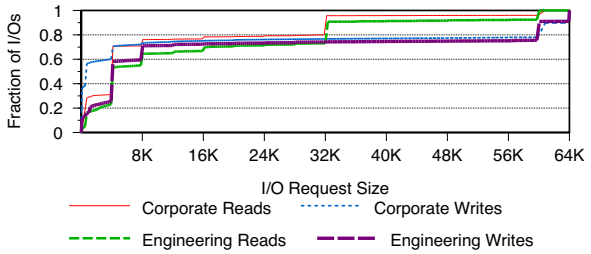
4.4 File I/O Properties

We now take a closer look at the properties of file I/O, where, as defined in Section 4.1, an I/O request is defined as any single read or write operation. We begin by looking at per-file, per-session I/O inter-arrival times, which include network round-trip latency. Intervals are categorized by the type of requests (read or write) that bracket the interval; the distribution of interval lengths is shown in Figure 7(a). We find that most inter-arrival times are between 100 μs and 100 ms. In fact, 96.4% and 97.7% of all I/Os have arrival times longer than 100μs and 91.6% and 92.4% are less than 100 ms for corporate and engineering, respectively. This tight window means that file systems may be able to make informed decisions about when to prefetch or flush cache data. Interestingly, there is little distinction between read-read or read-write and write-read or write-write inter-arrival times. Also, 67.5% and 69.9% of I/O requests have an inter-arrival time of less than 3 ms, which is shorter than some measured disk response times [23]. These observations may indicate cache hits at the server or possibly asynchronous I/O. It is also interesting that both workloads have similar inter-arrival time distributions even though the hardware they are running is of different classes, a mid-range model versus a high-end model.

Next, we examine the distribution of bytes transferred by a single I/O request. As Figure 7(b) shows, most requests transfer less than 8 KB, despite a 64 KB maximum request size in CIFS. This distribution may vary between CIFS and NFS since each buffers and schedules I/O differently. The distribution in Figure 7(b) increases for only a few I/O sizes, indicating that clients generally use a few specific request sizes. This I/O size information can be combined with I/O inter-arrival times from Figure 7(a) to calculate a distribution of I/Os per-second (IOPS) that may help file systems determine how much buffer space is required to support various I/O rates.



(a) Per-file I/O inter-arrival times.



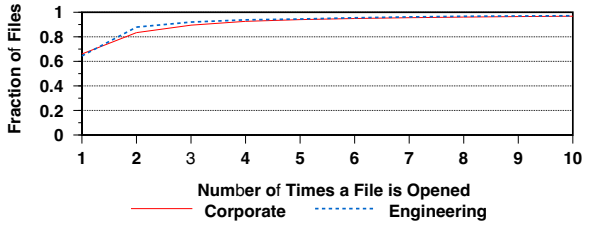
(b) I/O request sizes.

Figure 7: ***File I/O properties****. The burstiness and and size properties of I/O requests are shown. Figure 7(a) shows the I/O inter-arrival times. Figure 7(b) shows the sizes of read and write I/O.*

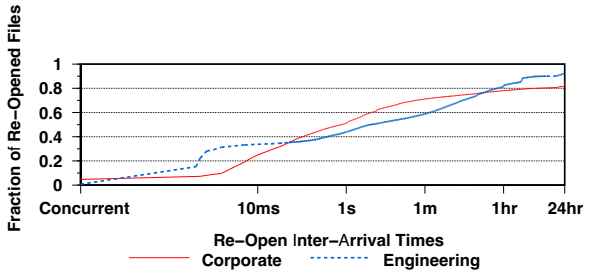
4.5 File Re-Opens

In this subsection, we explore how frequently files are re-opened, *i. e.*, opened more than once during the trace period. Figure 8(a), shows the distribution of the number of times a file is opened. For both workloads, we find that the majority of files, 65%, are only opened once during the entire trace period. The infrequent re-access of many files suggests there are opportunities for files to be archived or moved to lower-tier storage. Further, we find that about 94% of files are accessed fewer than five times. However, both of these distributions are long-tailed—some files are opened well over 100,000 times. These frequently re-opened files account for about 7% of total opens in both workloads. **Observation 6** *Most files are not re-opened once they are closed.*

We now look at inter-arrival times between re-opens of a file. Re-open inter-arrival time is defined as the duration between the last close of a file and the time it is re-opened. A re-open is considered *concurrent* if a re-open occurs while the file is still open (*i. e.*,, it has not yet been closed). The distribution of re-open inter-arrival times is shown in Figure 8(b). We see that few re-opens are concurrent, with only 4.7% of corporate re-opens and



(a) File open frequencies.



(b) Re-open inter-arrival times.

Figure 8: ***File open properties***. *The frequency of and duration between file re-opens is shown. Figure 8(a) shows how often files are opened more than once. Figure 8(b) shows the time between re-opens.*

0.7% of engineering re-opens occurring on an currently-open file. However, re-opens *are* temporally related to the previous close; 71.1% and 58.8% of re-opens occur less than one minute after the file is closed. Using this information, file systems may be able to decide when a file should be removed from the buffer cache or when it should be scheduled for migration to another storage tier. **Observation 7** *If a file is re-opened, it is temporally related to the previous close.*

4.6 Client Request Distribution

We next examine the distribution of file open and data requests amongst clients; recall from Section 4.1 that "client" refers to a unique IP address rather than an individual user. We use Lorenz curves [14]—cumulative distribution functions of probability distributions—rather than random variables to show the distribution of requests across clients. Our results, shown in Figure 9, find that a tiny fraction of clients are responsible for a significant fraction of open requests and bytes transferred. In corporate and engineering, 0.02% and 0.04% of clients make 11.9% and 22.5% of open requests and account for 10.8% and 24.6% of bytes transferred, respectively. Interestingly, 0.02% of corporate clients and 0.04% of engineering clients correspond to approximately 1 client for each workload. Additionally, we find that about 35 corporate clients and 5 engineering clients account for close to 50% of the opens in each workload. This suggests that the distribution of activity is highly skewed and

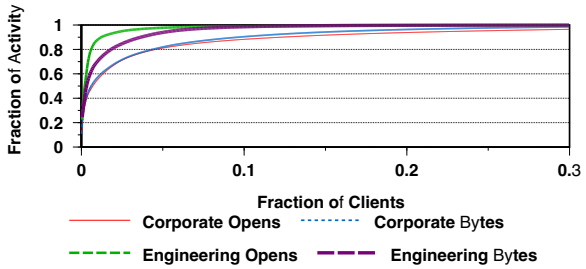
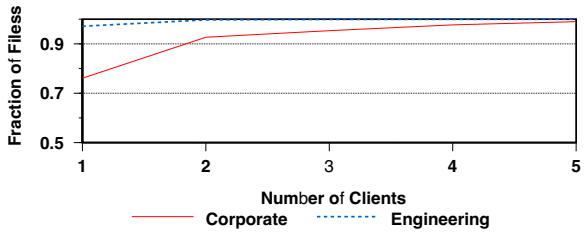
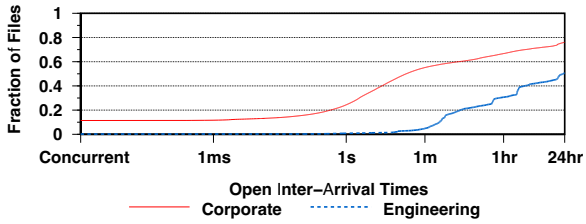


Figure 9: ***Client activity distribution*** *The fraction of clients responsible for certain activities is plotted.*



(a) File sharing frequencies.



(b) Sharing inter-arrival times.

Figure 10: ***File sharing properties***. *We analyze the frequency and temporal properties of file sharing. Figure 10(a) shows the distribution of files opened by multiple clients. Figure 10(b) show the duration between shared opens.*

that file systems may be able to take advantage of this information for doing informed allocation of resources or quality of service planning. **Observation 8** *A small fraction of clients account for a large fraction of file activity.*

4.7 File Sharing

This subsection looks at the extent of file sharing in our workloads. A file is shared when two or more clients open the same file *at some time* during the trace period; the sharing need not be concurrent. Since we can only distinguish IP addresses and not actual users, it is possible that two IP addresses may represent a single (human) user and vice versa. However, the drastic skew of our results indicates this likely has little impact on our observations. Also, we only consider opened files in our analysis; files which have only had their metadata accessed by multiple clients are not included in the these results.

Figure 10(a) shows the distribution of the frequency with which files are opened by multiple clients. We find that most files are only opened by a single client. In fact, 76.1% and 97.1% of files are only opened by one client in corporate and engineering, respectively. Also, 92.7% and 99.7% of files are ever opened by two or fewer clients. This suggests that the shared environment offered by network file systems is not often taken advantage of. Other methods of sharing files, such as email or web and wiki pages, may reduce the need for clients to share files via the file system. However, both distributions are long-tailed, and a few files are opened by many clients. In the corporate workload, four files are opened by over 2,000 clients and in the engineering workload, one file is opened by over 1,500 clients. This shows that, while not common, sharing files through the file system can be heavily used on occasion. **Observation 9** *Files are infrequently accessed by more than one client.*

In Figure 10(b) we examine inter-arrival times between different clients opening a file. We find that concurrent (simultaneous) file sharing is rare. Only 11.4% and 0.2% of shared opens from different clients were concurrent in corporate and engineering, respectively. When combined with the observation that most files are only opened by a single client, this suggests that synchronization for shared file access is not often required, indicating that file systems may benefit from looser locking semantics. However, when examining the duration between shared opens we find that sharing does have a temporal relationship in the corporate workload; 55.2% of shared opens occur within one minute of each other. However, this is not true for engineering, where only 4.9% of shared opens occur within one minute.

We now look at the manner (read-only, write-only, or read-write) with which shared files are accessed. Figure 11 shows the usage patterns for files opened by multiple clients. Gaps are present where no files were opened by that number of clients. We see that shared files are accessed read-only the majority of the time. These may be instances of reference documents or web pages that are rarely re-written. The number of read-only accesses slightly decreases as more clients access a file and a read-write pattern begins to emerge. This suggests that files accessed by many clients are more mutable. These may be business documents, source code, or web pages. Since synchronization is often only required for multiple concurrent writers, these results further argue for loose file system synchronization mechanisms. **Observation 10** *File sharing is rarely concurrent and mostly read-only.*

Finally, we analyze which clients account for the most opens to shared files. Equality measures how open requests are distributed amongst clients sharing a file. Equal file sharing implies all sharing clients open the shared file an equal number of times. To analyze equality, we use the Gini coefficient [9], which measures statistical dispersion, such as the inequality of income in economic analysis. We apply the equality concept to how frequently a shared file is opened by a client. Lower coefficients mean sharing clients open the file more equally (the same number of times), and higher coefficients mean a few clients account for the majority of opens. Figure 12 shows Gini coefficients for various numbers of shared clients. We see that, as more clients open a file, the level of equality decreases, meaning few clients begin to dominate the number of open requests. Gini coefficients are lower, less than 0.4, for files opened by fewer than 20 clients, meaning that when a few clients access a file, they each open the file an almost equal number of times. As more clients access the file, a small number of clients begin to account for most of the opens. This may indicate

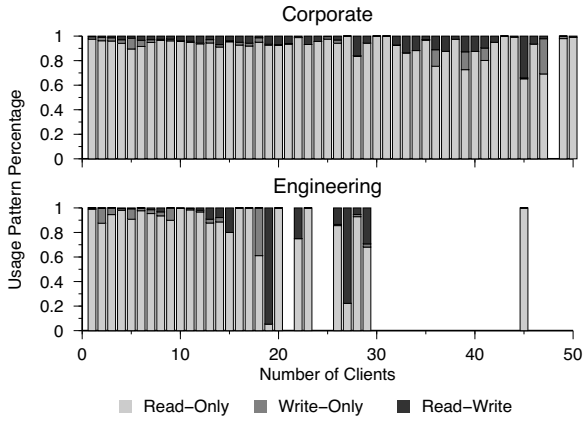


Figure 11: ***File sharing access patterns.*** *The fraction of read-only, write-only, and read-write accesses are shown for differing numbers of sharing clients. Gaps are seen where no files were shared with that number of clients.*

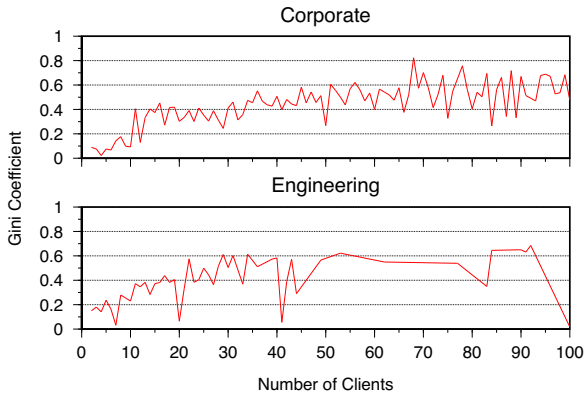


Figure 12: ***User file sharing equality****. The equality of sharing is shown for differing numbers of sharing clients. The Gini coefficient, which measures the level of equality, is near 0 when sharing clients have about the same number of opens to a file. It is near 1 when clients unevenly share opens to a file.*

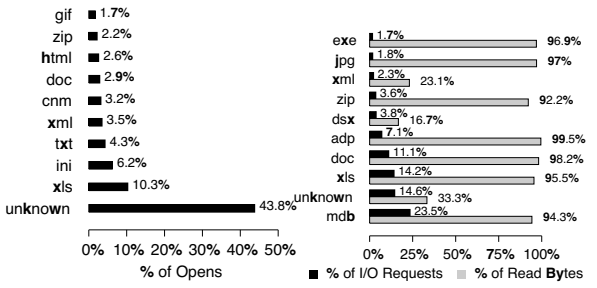
that as more clients share a file, it becomes less reasonable for all sharing clients to access the file evenly, and a few dominate clients begin to emerge.

4.8 File Type and User Session Patterns

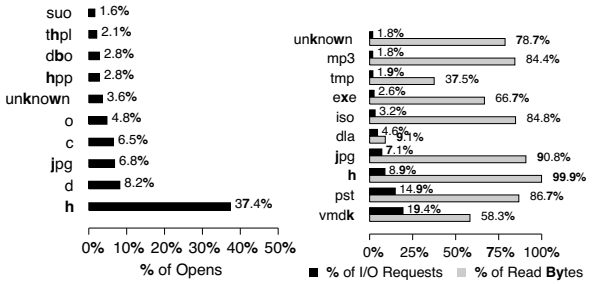
There have been a number of attempts to make layout, caching, and prefetching decisions based on how specific file types are accessed and the access patterns of certain users [6, 17]. In this subsection, we take a closer at how certain file types are accessed and the access patterns that occur between when a user begins a CIFS "user session" by logging on and when they log-off. Our emphasis is on whether file types or users have common access patterns that can be exploited by the file system. We begin our analysis by breaking down file type frequencies for both workloads. Figures 13(a) and 13(b) show the most frequently opened and most frequently read and written file types. For frequently read and written file types, we show the fraction of bytes read for that type. Files with no discernible file extension are labeled "unknown."

We find that the corporate workload has no file type, other than unknown types, that dominates open requests. However, 37.4% of all opens in the engineering workload are for C header files. Both workloads have a single file type that consumes close to 20% of all read and write I/O. Not surprisingly, these types correspond to generally large files, *e. g.*, `mdb` (Microsoft Access Database) files and `vmdk` (VMWare Virtual Disk) files. However, we find that most file types do not consume a significantly large fraction of open or I/O requests. This shows that file systems likely can be optimized for the small subset of frequently accessed file types. Interestingly, there appears to be little correlation between how frequently a file is opened and how frequently it is read or written. Only three corporate and two engineering file types appear as both frequently opened and frequently read or written; the `mdb` and `vmdk` types only constitute 0.5% and 0.08% of opens. Also, it appears file types that are frequently read or written are mostly read.

We now analyze the hypothesis that file systems can make use of file type and user access patterns to improve layout and prefetching [5, 6, 17]. We do so examining *access signatures*, a vector containing the number of bytes read, bytes written, and sequentiality metric of a file access. We start by defining an access signature for each open/close pair for each file type above, we then apply K-means clustering [15] to the access signatures of each file type. K-means groups access signatures with similar patterns into unique clusters with varying densities. Our results are shown in Figure 14. For clarity we have categorized access signatures by the access type: read-only, write-only, or read-write. We further group signatures by their sequentiality metric ranges: 1–0.81 is consid-



(a) Corporate.



(b) Engineering.

Figure 13: ***File type popularity.*** *The histograms on the right show which file types are opened most frequently. Those on the left show the file types most frequently read or written and the percentage of accessed bytes read for those types.*

ered highly sequential, 0.8–0.2 is considered mixed sequentiality, and 0.19–0 is considered highly random. Finally, access signatures are categorized by the number of bytes transferred; access signatures are considered small if they transfer no more than 100 KB and large otherwise. Darker regions indicate the file type has a higher fraction of access signatures with those properties shown on the y-axis, and lighter regions indicate fewer signatures with those characteristics.

Figure 14 shows that most file types have several distinct kinds of access patterns, rather than one as previously presumed. We also see that each type has multiple patterns that are more frequent than others, suggesting that file systems may not be able to properly predict file type access patterns using only a single pattern. Interestingly, small sequential read patterns occur frequently across most of the file types, implying that file systems should be optimized for this pattern, as is often already done. **Observation 11** *Most file types do not have a single pattern of access.*

Surprisingly, file types such as `vmdk` that consume a large fraction of total I/Os are frequently accessed with small sequential reads. In fact, 91% of all `vmdk` accesses are of this pattern, contradicting the intuition derived from Figure 13(b) that `vmdk` files have large accesses. However, a much smaller fraction of `vmdk` accesses transfer huge numbers of bytes in highly random read-write patterns. Several patterns read and write over

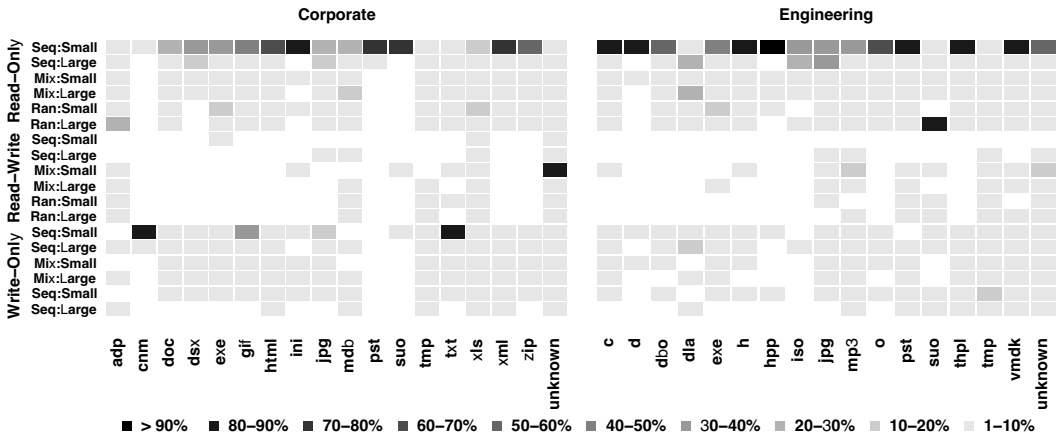


Figure 14: ***File type access patterns*** *The frequency of access patterns are plotted for various file types. Access patterns are categorized into 18 groups. Increasingly dark regions indicate higher fractions of accesses with that pattern.*

10 GB of data with a sequentiality metric less than 0.5, showing that frequent patterns may not be representative of the significant patterns in terms of bytes transferred or sequentiality. This argues that file systems should anticipate several patterns of access for any file type if layout or prefetching benefits are to be gained. Also, it is critical that they identify transitions between patterns. For example, a file system may, by default, prefetch data for `vmdk` files in small chunks: 100 KB or less. However, when over 100 KB of a `vmdk` file is accessed this signals the likely start of a very large transfer. In this case, the file system must properly adjust its prefetching.

Our observation that many file types exhibit several access patterns of varying frequency and significance draws an interesting comparison to the results in Table 4. Table 4 shows significant read-write I/O and byte transfer activity. However, file types in Figure 14 rarely have read-write patterns. This implies, read-write file accesses are, in general, uncommon, however when they do occur, a large number of bytes are accessed.

Next, we apply the same K-means clustering approach to access signatures of access patterns that occur within a CIFS user session. Recall that CIFS users begin a connection to the file server by creating an authenticated user session and end by eventually logging off. We define signatures for all accesses performed while the user is logged on. However, we only consider sessions in which bytes are transferred. The CIFS client opens short, temporary sessions for various auxiliary functions, which we exclude from this study as they do not represent a normal user log-in. Like file types, user sessions have several common patterns and no single pattern can summarize all of a user's accesses. The majority of user sessions have read-write patterns with less than 30 MB read and 10 MB written with a sequentiality metric close to 0.5, while a few patterns have much more significant data transfers that read and write gigabytes of data.

5 Design Implications

In this section we explore some of the possible implications of our trace analysis on network file system designs. We found that read-write access patterns have significantly increased relative to previous studies (see Section 4.3.1). Though we observed higher sequentiality in read-write patterns than past studies, they are still highly random compared to read-only or write-only access patterns (see Section 4.3.1). In contrast, a number of past studies found that most I/Os and bytes are transferred in read-only sequential access patterns [3, 21, 29], which has impacted the designs of several file systems [16, 19]. Our observed shift towards read-write access patterns suggests file systems should look towards improving random access performance, perhaps through alternative media, such as flash. In addition, we observed that the ratio of data read to data written is decreasing compared to past studies [3, 5, 24] (see Section 4.2). This decrease is likely due to increasing effectiveness of client caches and fewer read-heavy system files on network storage. When coupled with increasing read-write access patterns, write-optimized file systems, such as LFS [25] and WAFL [11], or NVRAM write caching appear to be good designs for network file systems.

We also observed that files are infrequently re-opened (see Section 4.5) and are usually accessed by only one client (see Section 4.7). This suggests that caching strategies which exploit this, such as exclusive caching [31], may have practical benefits. Also, the limited reuse of files indicates that increasing the size of server data caches may add only marginal benefit; rather, file servers may find larger metadata caches more valuable because metadata requests made up roughly 50% of all operations in both workloads, as Section 4.2 details.

Our finding that most created files are not deleted (see Section 4.3.3) and few files are accessed more than once

(see Section 4.5) suggests that many files may be good candidates for migration to lower-tier storage or archives. This is further motivated by our observation that only 1.6 TB were transferred from 22 TB of in use storage over three months. While access to file metadata should be fast, this indicates much file data can be compressed, de-duplicated, or placed on low power storage, improving utilization and power consumption, without significantly impacting performance. In addition, our observation that file re-accesses are temporally correlated (see Section 4.5) means there are opportunities for intelligent migration scheduling decisions.

6 Conclusions

In this paper we presented an analysis of two large-scale CIFS network file system workloads gathered from enterprise-class file servers deployed in a corporate and in an engineering environment. We compared our workloads to previous file system studies to understand how file access patterns have changed and conducted a number of other experiments. We found that read-write file access patterns and random file access are far more common than previously thought, and that most file storage remains unused, even over a three month period. Our observations on sequentiality, file lifetime, file reuse and sharing, and request type distribution also differ from those in earlier studies. Based on these observations, we made several recommendations for improving network file server design to handle the workload of modern corporate and engineering environments.

Acknowledgments

This work was supported in part by the Department of Energy under award DE-FC02-06ER25768, the NSF under award CCF-0621463, and industrial sponsors of the Storage Systems Research Center at UC Santa Cruz, including Agami Systems, Data Domain, Hewlett Packard, LSI Logic, NetApp, Seagate, and Symantec. We would also like to thank our colleagues in the SSRC and NetApp's Advanced Technology Group, our shepherd Jason Flinn, and the anonymous reviewers for their insightful feedback, which greatly improved the quality of the paper.

References

[1] A. Aggarwal and K. Auerbach. Protocol standard for a netbios service on a tcp/udp transport. IETF Network Working Group RFC 1001, March 1987.

[2] N. Agrawal, *et al*. A five-year study of file-system metadata. In *Proc. of FAST '07*, Feb. 2007.

[3] M. G. Baker, *et al*. Measurements of a distributed file system. In *Proc. SOSP '91*, Oct. 1991.

[4] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *Proc. of SIGMETRICS '99*, 1999.

[5] D. Ellard, *et al*. Passive NFS tracing of email and research workloads. In *Proc. of FAST '03*. 2003.

[6] D. Ellard, *et al*. Attribute-based prediction of file properties. Technical Report TR-14-03, Harvard, 2004.

[7] K. Evans and G. H. Kuenning. A study of irregularities in file-size distributions. In *Proceedings of SPECTS '02*.

[8] T. J. Gibson and E. L. Miller. Long-term file activity patterns in a UNIX workstation environment. In *Proc. of the 15th IEEE Symposium on Mass Storage Systems*, pages 355–372, Mar. 1998.

[9] C. Gini. Measurement of inequality and incomes. *The Economic Journal*, 31:124–126, 1921.

[10] S. Gribble, *et al*. Self-similarity in file systems: Measurement and applications. In *Proc. of SIGMETRICS '98*.

[11] D. Hitz, J. Lau, and M. Malcom. File system design for an NFS file server appliance. In *Proc. of USENIX '94*.

[12] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM ToCS*, 10(1), 1992.

[13] P. J. Leach and D. C. Naik. A common internet file system (cifs/1.0) protocol. IETF Network Working Group RFC Draft, March 1997.

[14] M. O. Lorenz. Methods of measuring the concentration of wealth. *Publications of the American Statistical Associations*, 9:209–219, 1905.

[15] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297, 1967. University of California Press.

[16] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM ToCS*, 2(3), Aug. 1984.

[17] M. Mesnier, *et al*. File classification in self-* storage systems. In *Proc. of ICAC '04*.

[18] S. J. Mullender and A. S. Tanenbaum. Immediate files. *Software–Practice and Experience*, 14(4), April 1984.

[19] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite network file system. *ACM ToCS*, 6(1), 1988.

[20] B. C. Neumann, *et al*. Kerberos: An authentication service for open network systems. In *Proc. of USENIX '88*.

[21] J. K. Ousterhout, *et al*. A trace-driven analysis of the Unix 4.2 BSD file system. In *Proc. of SOSP '85*, 1985.

[22] K. K. Ramakrishnan, P. Biswas, and R. Karedla. Analysis of file i/o traces in commercial computing environments. In *Proc. of SIGMETRICS '92*, 1992.

[23] A. Riska and E. Riedel. Disk drive level workload characterization. In *Proc. of USENIX '06*, May 2006.

[24] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *Proc. of USENIX '00*.

[25] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM ToCS*, 10(1):26–52, Feb. 1992.

[26] M. Satyanarayanan. A study of file sizes and functional lifetimes. In *Proc. of SOSP '81*, Dec. 1981.

[27] Spec benchmarks. http://www.spec.org/benchmarks.html.

[28] Tcpdump/libpcap. http://www.tcpdump.org/.

[29] W. Vogels. File system usage in Windows NT 4.0. In *Proc. of SOSP '99*, Dec. 1999.

[30] Wireshark: Go deep. http://www.wireshark.org/.

[31] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proc. of USENIX '02*.

[32] M. Zhou and A. J. Smith. Analysis of personal computer workloads. In *Proc. of MASCOTS '99*, 1999.