Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs

William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer
Microsoft Research, Redmond, WA 98052
{bolosky, johndo, theimer}@microsoft.com; ely@cs.washington.edu

ABSTRACT

We consider an architecture for a serverless distributed file system that does not assume mutual trust among the client computers. The system provides security, availability, and reliability by distributing multiple encrypted replicas of each file among the client machines. To assess the feasibility of deploying this system on an existing desktop infrastructure, we measure and analyze a large set of client machines in a commercial environment. In particular, we measure and report results on disk usage and content; file activity; and machine uptimes, lifetimes, and loads. We conclude that the measured desktop infrastructure would passably support our proposed system, providing availability on the order of one unfilled file request per user per thousand days.

Keywords

Serverless distributed file system architecture, personal computer usage data, availability, reliability, security, trust, workload characterization, analytical modeling, feasibility analysis.

1. INTRODUCTION

We present an architecture for a serverless distributed file system and assess the feasibility of deploying this system on an existing desktop infrastructure. The distinguishing feature of our proposed system is that it does not assume the client computers to be carefully administered or mutually trusting. Instead, the system provides high availability (file access) and high reliability (file persistence) by making multiple replicas of each file and distributing them among the client machines. To determine how well such a system might work, we gathered usage data from a large number of client machines at Microsoft Corporation, and we use this data to analyze the feasibility of our architecture.

Our proposed serverless distributed file system is intended to provide a global name space for files, location-transparent access to both private files and shared public files, and improved reliability relative to a desktop workstation.

These goals can be achieved with a centralized, server-based file system, but this has several disadvantages: Servers tend to be expensive because they need special hardware, such as high-performance I/O (to support multiple clients simultaneously) and RAID disks (for reliability) [21]. They rely on system administrators, in whom the users must place their faith [28], both to authorize access and to perform reliability functions, such as regular backups. Finally, they are vulnerable to geographically localized faults, such as a failed router or a broken network link.

Serverless distributed file systems have been developed before (§ 6), but these have all assumed that client machines are mutually trusting. This assumption greatly eases the system design, but we believe it is unrealistic, especially in a large-scale system. If we can build a system that works without requiring trust, then the system can function with no central administration: The owner of each machine determines who can store files on it, and owners establish contracts with other machine owners to collaboratively share their resources. With no central administrator, any user can add resources to the system. One can even add userless client machines to function as dedicated servers.

The lack of trust between clients pervades our entire design. We need to use cryptographic techniques to ensure data privacy and integrity. We also need to create and securely distribute multiple replicas of each file throughout the system, both to prevent a malicious user from easily destroying all copies of any given file and because we cannot expect users to significantly alter their behavior with regard to keeping their machines on and available.

The need for multiple file replicas significantly increases the storage demand. However, since disk space is inexpensive and becoming more so all the time [25], we consider this an acceptable trade-off. Also, previous research [7] has shown a significant amount of free storage space on client machines. This inspired us to investigate how well our proposed system would work if deployed on an existing client-computing infrastructure, without adding any new resources to support our system. This question is the primary concern of the present paper, which we answer by measuring file-system space, machine availability, and machine load on client machines in a commercial environment.

The remainder of this paper describes our system architecture, explains our measurement methodology, presents our results, and analyzes these results in terms of our proposed architecture.

2. SYSTEM ARCHITECTURE

The principal construct of our proposed system is the *global file store*, which is a logically monolithic file system indexed by a hierarchical directory tree. Although logically monolithic, the global file store is physically distributed among the disks of the client machines participating in the distributed file system.

Each disk of a participating machine is partitioned into three regions: a scratch area, a global storage area, and a local cache. The scratch area holds ephemeral data, such as virtual-memory paging files and temporary files. The global storage area houses a portion of the global file store, which is accessible by other machines in the distributed system. The local cache is of variable size, caching all files accessed within a certain period of time (the *cache retention period*), like some in-memory file-system buffer caches [19, 22, 26] and unlike traditional fixed-size caches.

To provide high availability and reliability in a not-fully-trusted environment, our proposed system makes multiple replicas of

SIGMETRICS 2000 6/00 Santa Clara, California, USA

each file and distributes them among the global storage areas of the client machines. We define the *replication factor* as the number of distributed replicas of each file.

Since file availability and reliability are drastically affected by the file replication factor, it is important to maintain a minimum number of replicas for each file. To ensure sufficient disk space for these replicas, our proposed system enforces quotas [16] on the space available to the users of the file system. We propose that each user's space allotment depend on how much space the user has contributed to the global store.

Part of the local cache and global storage area is reserved for a highly available, reliable, global, distributed directory service, much like that provided by xFS [1]. Each machine can use this service to locate replicas of requested files. This directory must be maintained in a strongly consistent fashion, to ensure that clients see the latest version of each file.

2.1 Efficiency Considerations

Our proposed system can increase the usable space in the global file store through two techniques. First, it can compress files before storing them and decompress them on the fly when they are read [26]. Second, it can coalesce distinct files that happen to have identical contents [5]; for example, many users may store copies of a common application program in the global file store, and these can be coalesced into a single instance, prior to replicating this instance among multiple machines. For clarity, we refer to logically distinct files with identical content as *duplicates*, and we refer to the file copies our system generates as *replicas*.

We plan for our system to employ a lazy-update strategy, meaning that the system waits for a short time after a file is written before updating the file's replicas. Since a large fraction of written files are deleted or overwritten within a few seconds [29], lazy update can significantly reduce file-update traffic. Lazy update also allows a file to be written without requiring all (or even a quorum) of the replicas to be immediately available. The disadvantage of this approach is that the content of newly written files will briefly reside on only one machine, so loss of that machine will result in loss of the update. The directory service must keep track of which replicas contain up-to-date data, so users will not accidentally access out-of-date versions of files.

Since file replicas are distributed among multiple machines, our system can select which machine should be accessed to service a client request [30]. There are two primary considerations in this decision: First, the selected machine should be topologically close to the machine that is requesting the file, to minimize both transmission delay and generated network traffic. Second, the selected machine should be lightly loaded, to minimize both read delay and performance impact on the sending machine. The impact on the sending machine can also be reduced by performing non-cached reads and writes, to prevent buffer cache pollution.

If remote reads of very popular files are targeted at a small number of machines, those machines could become overloaded. Our system can avoid creating these hotspots by allowing machines to copy files from other machines' on-disk local caches. As a side benefit, this improves the availability of popular files, since the effective number of replicas is substantially increased.

2.2 Replica Management

In our proposed system, the selection of machines for storing file replicas is driven by the availability of those machines. The system measures machine uptimes and distributes replicas so as to maximize the minimum file availability. (§ 2.2.1 describes such a replica-placement algorithm.)

In selecting locations for replicas of a given file, the system could select a set of machines whose uptimes are negatively correlated, thus reducing the likelihood that all machines containing a replica will be down at the same time. However, our measurements (described in § 3.2.2 and reported in § 4.2.3) suggest that this would provide little marginal benefit.

Our system can improve the availability of sets of related files by storing them in the same locations. If a user needs a given set of files to accomplish a task, then if any of those files is inaccessible, the task cannot be completed; therefore, it is beneficial to store related files together, so that either all or none of the files are available. Our system could attempt to determine relatedness of files by observing file access patterns, but we propose a simpler approach for at least the initial implementation: Since files accessed together temporally tend to be grouped together spatially, replicas for all files in a given directory are stored on the same set of machines, and entire directories are cached together.

When a new machine joins the system, its files are replicated to the global storage areas on other machines; space for those replicas is made by relocating replicas of other files onto the new machine. Similarly, when a machine is decommissioned, the system creates and distributes additional replicas of the files stored on that machine.

Machines join the system by explicitly announcing their presence; however, machines can leave without notice, particularly if they leave due to permanent failure, such as a disk-head crash. If the system notices that a machine has been down for an extended period of time, it must assume that the machine has been decommissioned and accordingly generate new replicas; otherwise, the possibility of another failure can jeopardize the reliability of files with replicas on that machine.

2.2.1 Replica management – placement algorithm

Ideally, replicas should be assigned to machines so as to maximize the minimum availability of any file, while also maximizing the minimum reliability of any file. The latter goal merely requires minimizing the variance of the replica count, whereas the former is more involved. Measured logarithmically, the availability of a file equals the sum of the availability of all machines that store a replica of the file, assuming randomly correlated machine uptime.

The following heuristic algorithm yields a low availability variance: Set the provisional availability of all files to zero; then, iteratively select machines in order of decreasing availability; for each selected machine, assign the files with the lowest provisional availability to the selected machine, and update the provisional availability of those files; repeat until all machines are full.

Unfortunately, the above algorithm does not minimize reliability variance, so we modify it as follows: If an assignment of a file to a machine would reduce the remaining free space to below that necessary for any two files to differ in replica count by at most one, abort the above iteration and begin the following procedure: Iteratively select machines in order of increasing availability; for each selected machine, identify the file with the highest provisional availability from those with the lowest replica count, assign it to the selected machine, and update the provisional availability of that file; repeat until all machines are full.

2.3 Data Security

Distributing multiple replicas of a file protects not only against accidental failure but also against malicious attack. To destroy a file, an adversary must compromise all machines that hold replicas of that file. To prevent an adversary from coercing the system into placing all replicas of a given file on a small set of machines

under the adversary's control, the system must use secure methods for selecting machines to house file replicas.

File-update messages are digitally signed by the writer to the file. Before applying an update to a replica it stores, each machine verifies the digital signature on the update and compares the writer's identity to a list of authorized writers associated with the file. This prevents an adversary from forging and distributing file updates.

To prevent files from being read by unauthorized users, the contents of files are encrypted before they are replicated. However, encryption could interfere with the automatic detection and coalescing of duplicate files, since different users may encrypt identical plaintext files with different keys, which would normally produce different ciphertext files. We have developed a cryptographic technology, called *convergent encryption*, that allows the detection and coalescing of identical files even when these files are encrypted with separate keys. Rather than enciphering the contents of a user's files directly with the user's key, the contents of each file are one-way hashed, and the resulting hash value is used as a key for enciphering the file contents. The user's key is then used to encipher the hash value, and this enciphered value is attached to the file as meta-data. The user decrypts a file by first deciphering the hash value and then deciphering the file using the hash value as a key. With this approach, two files with identical plaintext will also have identical ciphertext, irrespective of the secret keys used to encrypt them.

3. METHODOLOGY

To analyze the feasibility of deploying our proposed system on an existing computing infrastructure, we collected usage data from a large number of desktop personal computers at Microsoft Corporation. We measured contents of file systems, availability of machines, and load on machines.

To judge the breadth of applicability of our results, we partitioned our measurements into six categories by the job function of each machine's owner: administration (clerical), business (marketing, accounting, legal), management, non-technical development (writers, artists), technical development (programmers, testers), and technical support. Roughly half of all machines belong to technical developers; the other half are distributed approximately equally among the remaining categories.

3.1 File System Measurement

We measured a set of file systems to determine the amount of disk space that is free and the amount of disk space that would be free if all duplicate files were eliminated. We also use the measured data to estimate the rate of cache misses, the size of local system caches, and the rate of file writes.

3.1.1 Disk usage measurement

We collected two data sets for our analysis of disk usage. In September 1998, we asked Microsoft employees to run a scanning program on their Windows and Windows NT computers that collected directory information (file names, sizes, and timestamps) from their file systems [7]. By this means, we obtained measurements of 10,568 file systems [8].

In August 1999, we remotely read the performance counters (free disk space, total disk space, and logon name of the primary user) of every Windows NT computer we could access. By this means, we obtained measurements of 8669 file systems.

3.1.2 Disk content measurement

In February 1999, we asked a random subset of the participants in the 1998 study to run a scanning program that computed and recorded hashes of all the files on their file systems. By this means, we obtained data from 550 file systems, which we used to determine the amount of duplicate file content.

3.1.3 File access rate estimation

We used the timestamps in the 1998 data set to estimate the rate at which files are read and written. We determined each file's last read time as the most recent timestamp (create, update, or access) on the file, and its last write time as the most recent of the create and update timestamps.

There are three aspects of file access rate that are relevant to our design: the miss rate of the local file cache, the size of the local file cache, and the amount of write traffic sent to file replicas on other machines. We consider only files that would be stored in the global file store, so we excluded inherently local files, including the system paging file and those files in temporary directories or the Internet browser cache, all of which account for 2% of the bytes in the data set.

Cache miss rate is a function of the cache retention period. We estimate miss rate as follows: For a cache retention period of n days, the files in directories last accessed n+1 days ago will exit the cache on the current day. Over the long term, the rate at which files exit the cache must match the rate at which they enter the cache, so the rate of file exit approximates the rate of file entry. Since a file enters the cache only in response to a miss, the cache entry rate equals the cache miss rate.

We estimate the cache size similarly. For a cache retention period of n days, the size of the local cache is equal to the sum of the file sizes in all directories accessed in the last n days.

Write traffic rate is a function of the lazy-update propagation lag. We estimate write traffic rate as follows: For a propagation lag of n hours, files last written between 2n hours ago and n hours ago will have been propagated during the past n hours.

3.2 Machine Availability Measurement

To determine file availability in our proposed system, we need to know machine uptimes, whether these uptimes are consistent over time, and whether the uptimes of different machines are correlated. When our proposed system sees a machine go down, it needs to predict how long the machine will stay down. Our reliability calculations require knowing machine lifetimes.

3.2.1 Machine uptime measurement

We measured machine availability by pinging 64,610 machines every hour for five weeks, from July 6 through August 9, 1999. To disregard any staleness of the name database as well as the attrition of machines during the sample period, we restricted our analysis to the 51,662 machines that responded to at least one ping during the week of August 10 through August 16, 1999.

3.2.2 Machine uptime correlation calculation

To determine whether the uptimes of different machines are correlated, we computed, for every pair of machines, a temporal correlation value by adding one for every ping snapshot that the machines were both up or both down, and subtracting one for every snapshot that one machine was up and the other was down. We normalized the result by dividing by the count of snapshots.

3.2.3 Machine downtime prediction calculation

We can use the time t_p a machine has been off to predict the amount of additional time t_f it will remain off, using the formula:

$$t_f = \int_{t_p}^{\infty} f(x)\,dx \Big/ \int_{t_p}^{\infty} \frac{f(x)}{x}\,dx \; - \; t_p \qquad (1)$$

In this equation, $f(x)$ is the distribution of downtime as a function of the enclosing interval, which we determine from our data.

3.2.4 Machine lifetime measurement

To analyze machine attrition rates, we pinged 64,610 machines several times per day for 100 days, from July 6 through October 13, 1999. We scanned backwards through the data to determine the last time each machine responded to a ping. If machines have deterministic lifetimes, then the rate of attrition is constant, and the count of remaining machines decays linearly. The expected machine lifetime (meaning the lifetime of the machine name, not the physical hardware) is the time until this count reaches zero.

3.3 Machine Load Measurement

We measured CPU and disk load by running a program on six data-collection computers that iteratively selected a random machine and remotely read the performance counters (CPU load and disk load) of that machine. Over 18 days in October 1999, we collected 178,801 measurements from 3908 machines.

CPU load was measured as the fraction of cycles expended in processes other than the idle process. Disk load was measured as the number of disk operations performed per second.

3.4 Measurement Errors and Biases

Our data-collection techniques are vulnerable to many sources of experimental error and measurement bias. In general, we are not able to quantify the effects of these errors on our results, but we can at least enumerate them, so their presence will not be ignored.

Since each machine's owner determined whether to run the file-system scanning program (§ 3.1), these results could be affected by self-selection bias.

The file timestamps we collected (§ 3.1.1) and analyzed (§ 3.1.3) can be reset by user-level software; therefore, they may be unreliable. In addition, it took several minutes to perform the scan of each file system, so this limits the granularity with which we can assess the elapsed time since file accesses.

Our use of static timestamps to determine dynamic access rates prevents us from observing burstiness in file accesses. We therefore confine ourselves to mean-value analysis.

The file access times derived from the file timestamps are biased towards integral multiples of one week, for the following reason: Individual users ran the scanning program at times of their own choosing, so they were disproportionately likely to do so on a weekday. Since files are more likely to be accessed on a weekday, the time since last access reflects this congruence.

Our remote reads of performance counters were restricted to a subset of the Windows NT/2000 machines on our corporate network, since performance counters are not present in Windows 95/98, a machine's owner can restrict access to them in Windows NT, and they are restricted by default in Windows 2000. Windows 2000 is most likely to be running on newer machines, which are likely to have larger-than-average disks. We used remote performance counters not only for file-system and machine-load measurements (§ 3.1 & 3.3) but also to determine each machine's primary user (and thereby job category) for machine-lifetime measurements (§ 3.2).

Each hourly ping snapshot (§ 3.2.1) took between 19 and 28 minutes to perform. This fuzziness slightly weakens our analysis of the correlation among different machines' uptimes.

For the machine-lifetime measurements (§ 3.2.3), we did not determine the machines' users until August 27, so we have no job-specific data for the first half of the observation period.

The machine-load measurements (§ 3.3) overestimate the CPU and disk loads, because the monitoring itself increases these loads: The operating system dynamically loads the performance counter libraries when interrogated, and it unloads them after a period of non-use that is smaller than our mean time between same-machine samples. Therefore, sampled machines commonly loaded some amount of monitoring code before responding to our query, thus increasing their workload. The exact amount of code loaded depends on the particulars of each machine and on whether any other application or system component is already using some part of the monitoring subsystem.

4. RESULTS

4.1 File System Results

From the file-system data we collected as described in § 3.1, we determined the amount of disk space that is free and the amount of disk space that could be freed by eliminating duplicate files. We also constructed an analytical model to show the relationship between content duplication and population size. From the recorded file timestamps, we estimate the rate at which directories of files are accessed and the rate at which files are written.

4.1.1 Free disk space

The self-selected September 1998 data show 53% of overall disk space in use. The remotely read August 1999 data show 50% of overall disk space in use. Since these two data sets incur different types of bias, it is reassuring that their results are quite similar.

4.1.2 Duplicate file content

From the February 1999 measurement of disk content, we calculate the space savings from removing duplicate files from the population of file systems, as illustrated in Figure 1.

The open circle on the graph shows that removing duplicates from the whole population of 550 file systems reclaims 47% of used file space. The small dots show the space reclamation for random subsets of the total population; we selected ten subsets in each power-of-two size from 1 to 512 file systems, with the selection probability weighted by file-system size.

The diamonds, squares, and triangles in Figure 1 show the effect of removing duplicate files within each subset of file systems corresponding to one of our six job categories. In general, the job-specific subsets contain a larger fraction of duplicated content than do comparably sized random populations. This indicates a greater commonality of content among members of a specific job category than among arbitrary sets of file systems.

Most of the space savings comes from eliminating duplicates of files with small duplication factors. Figure 2 shows the space reclaimed by eliminating duplicates of only those files with a

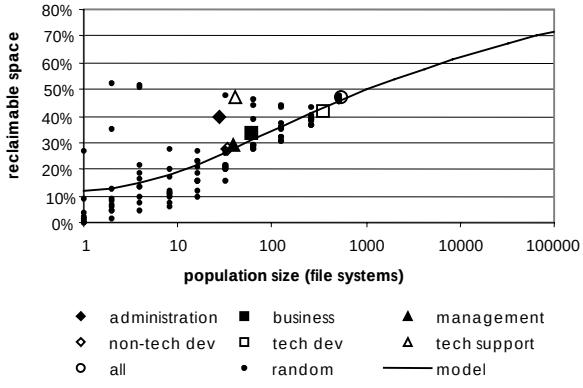

Figure 1: Reclaimable space versus population size

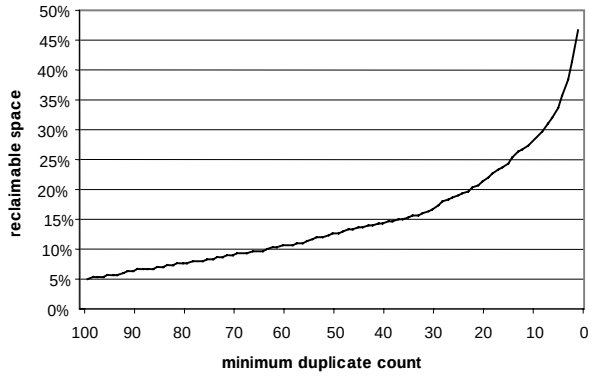

Figure 2: Reclaimable space versus minimum duplication

minimum number of duplicates, based on the entire population of 550 file systems. Only about 5% of file space is reclaimed by eliminating duplicates of files with at least 100 duplicates, less than 30% with a minimum of 10, and 47% when considering files with at least one duplicate.

4.1.2.1 Duplicate file content – analytical model

We have developed a model to describe how recoverable space relates to the population size. The solid line in Figure 3 shows the measured distribution of content popularity from the February 1999 data. The x-axis indicates the likelihood that any given file system will contain an item of file content (one file, where the distribution is weighted by file size), and the y-axis indicates the cumulative fraction of all content. For example, 90% of all file content has popularity of less than 0.1, so it will be found on no more than 1 out of 10 file systems.

Since the distribution of content popularity is a distribution of likelihood, we approximate it with a beta distribution [10], B(x), (α = 0.12, β = 4.2), shown by the dotted line in Figure 3. We use a beta distribution rather than the more commonly considered Zipf distribution [6] for two reasons: First, the Zipf distribution is discrete and applies to a specific count of objects, whereas we want a continuous distribution that applies to an indeterminate count of objects. Second, we are not hypothesizing a distribution for content popularity; we know the actual distribution (as shown by the solid line in Figure 3), and we merely want an analytical function to approximate it, to facilitate our analysis.

When adding a new file system to an existing population of k file systems, the amount of new content (content not already present in the existing population) added by the new file system is:

$$N(k) = \int_0^1 \mathrm{B}(x)\,(1-x)^k\,dx = \frac{\Gamma(\beta+k)\,\Gamma(\alpha+\beta)}{\Gamma(\beta)\,\Gamma(\alpha+\beta+k)} \qquad (2)$$

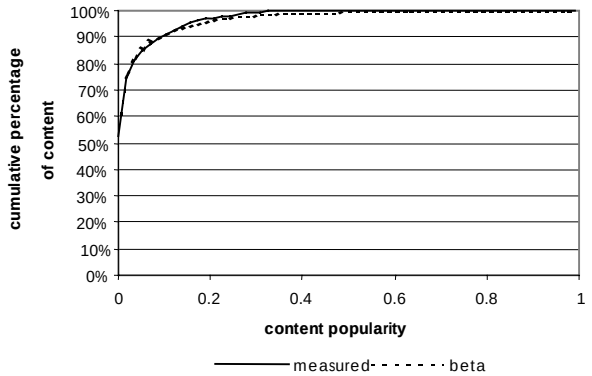

Figure 3: Popularity of file content

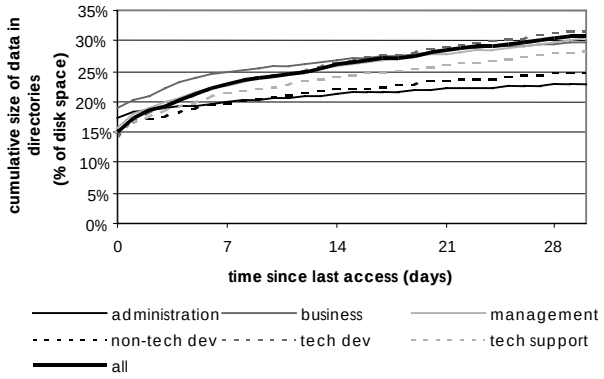

Figure 4: Cumulative directory-data size vs. last access time

Therefore, the fraction of used space that can be recovered by eliminating duplicate files in a population of n file systems is:

$$R(n) = 1 - c\sum_{k=0}^{n-1} N(k) = 1 - \frac{c\,\Gamma(\alpha+\beta)}{(\alpha-1)\,\Gamma(\alpha+\beta-1)} + \frac{c\,\Gamma(\alpha+\beta)\,\Gamma(\beta+n)}{(\alpha-1)\,\Gamma(\beta)\,\Gamma(\alpha+\beta+n-1)} \qquad (3)$$

The constant c accounts for files that are duplicated within a single file system. From our data, we calculated that the mean fraction of unique data within a single file system is c = 0.88.

The solid line in Figure 1 plots $R(n)$ versus population size n. Note the sigmoid shape of the graph and the fairly good correspondence to the results for random subsets of the actual population. The values predicted by the model pass Wilcoxon signed-rank tests [10] for each power-of-two-size group of 10 random subsets, at a 0.01 level of significance.

4.1.3 File access activity

From the September 1998 measurements, Figure 4 shows the cumulative amount of data in accessed directories, as a fraction of total disk space, versus the elapsed time since that data was last accessed. Most of the separation among job categories is due to variation in the fraction of disk space in use. For example, administration systems generally have a smaller fraction of disk space in use than other job categories have, so although the fraction of used disk space they access in a given time is comparable to that of other categories, this represents a smaller fraction of total disk space, which is what is plotted in Figure 4.

Figure 5 shows the count of directories that were last accessed on each day prior to the day each measurement was taken. Figure 6 shows the volume of data in the accessed directories on each day prior to the measurement date. Overall, 35 directories totaling 50 megabytes of data were touched per file system during the 24 hours preceding the measurement snapshot. There is noticeable

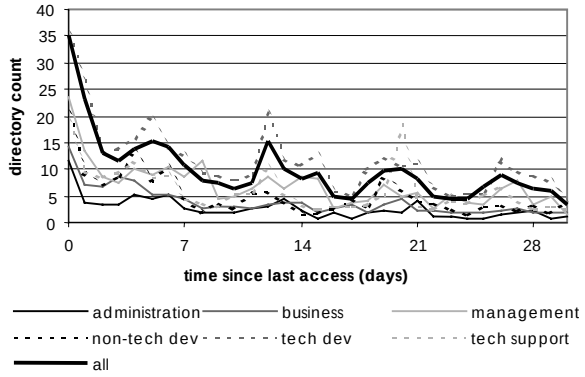

Figure 5: Directory count vs. last access time

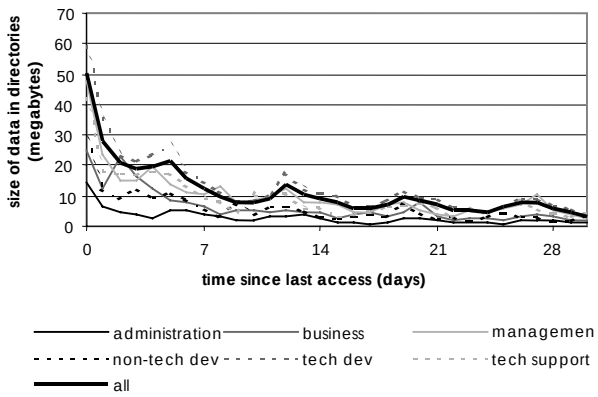

Figure 6: Directory-data size vs. last access time

variation among job categories, with administrators touching only 12 directories totaling 14 megabytes of data per file system. As described in § 3.4, our measurement technique biases the access times towards integral multiples of one week, as is shown by the weekly periodicity in Figures 5 and 6.

Figure 7 indicates the total size of all files written within each hourly period preceding the measurement snapshot. For files written within one hour of the snapshot, the overall rate was about 32 MB / hour / file system. The various job types ranged from 65 MB / hour / file system for business to 24 MB / hour / file system for technical developers. Merely 7 MB / hour / file system is written between one and two hours before the snapshot, which implies that most written data is overwritten fairly soon, consistent with several other studies [4, 20, 29]. As described in § 3.4, the granularity of the measurement data prevents assessing write rates for periods substantially less than one hour.

4.2 Machine Availability Results

From the machine-availability data we collected as described in § 3.2, we analyzed the uptime distribution of machines on our network, the consistency of those uptimes over time, and the correlation of the uptimes of different machines. We also estimated the length of time that a down machine will remain down. Lastly, we calculated the expected lifetime of machines.

4.2.1 Machine uptimes

From the July–August 1999 ping measurements, Figure 8 shows a time plot of machine availability. The count varies by about 2500 machines over a one-day period and about 5000 machines over a one-week period. We do not know what caused the negative

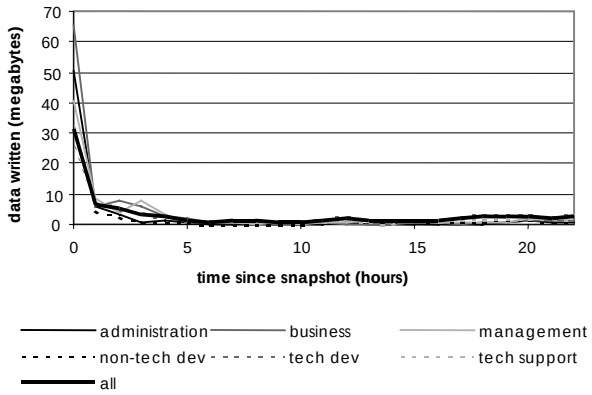

Figure 7: Write rates

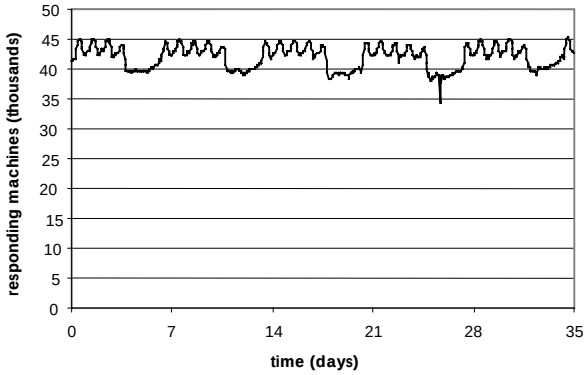

Figure 8: Count of available machines versus time

spike on July 31 at 15:00; this sample does not contain a large, contiguous series of non-responses, so it does not obviously appear to be a network problem.

The solid line in Figure 9 plots the fraction of time each machine is up, where the machines are sorted by their uptimes. Half of all machines are up over 95% of the time. The dotted line in Figure 9 shows machine availability measured in nines, calculated as the negative decimal logarithm of the fraction of downtime. It is interesting to note that this curve is approximately a straight line.

4.2.2 Machine uptime consistency

Figure 10 is a density plot, with logarithmically scaled shading, that shows the number of hours per week each machine is up during the second through fifth weeks of our sample period versus the number of hours it is up during the first week. Since most machines are up more than 95% of the time, this figure is heavily weighted toward the upper right corner.

There are three noticeable groups of machines in this figure: First are the machines that are turned off nightly, which are seen as a loose cluster in the region of 45 hours per week; the uptimes of these machines are fairly consistent from week to week. Second are the machines that are on for the first week but turned off for the first weekend, which form a vertical band around 105 hours per week; the vertical spread indicates that these are somewhat but not strongly likely to be turned off on subsequent weekends. Third are the machines left on for nearly all of the first week, which form a dense stripe along the upper part of the right edge; the very dark point at the upper right corner indicates that these tend to be on for most of the remaining time, but the tail down the right edge shows that this tendency is not absolute.

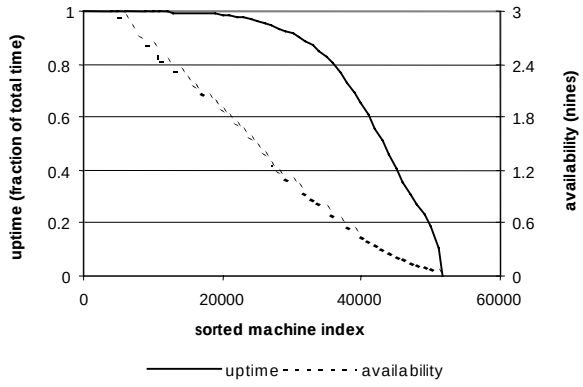

Figure 9: Uptime and availability vs. sorted machine index

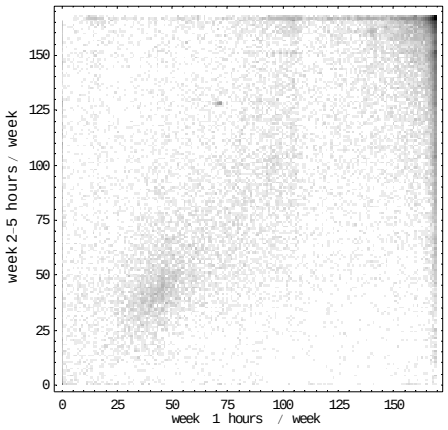

Figure 10: Week 2–5 uptimes vs. week 1 uptime (log scale)

4.2.3 Machine uptime correlation

We computed a temporal correlation value ρ for each pair of machines using the method described in § 3.2.2. The dark curve in Figure 11 shows the cumulative distribution of this correlation value for all pairs of machines. 73% of all pairs of machines exhibit positive uptime correlation, and 20% of all pairs exhibit very strong positive correlation ($\rho > 0.95$).

Figure 11 also plots what this cumulative correlation curve would be if uptimes for all machines were randomly correlated and if they were perfectly correlated, in each case using the measured distribution of machine uptimes from Figure 9. The actual curve is much closer to the random curve than to the perfect curve, indicating that the vast majority of the observed positive correlation is due to the fact that most machines are up most of the time, and the uptimes are not otherwise very correlated.

The degree of uptime correlation varies by job function. Administrator and business systems are more correlated than others; however, they are much closer to random than to perfect.

4.2.4 Machine downtime interval prediction

Figure 12 shows the distribution of machine downtime versus downtime interval length, including only downtime intervals that began and ended within our 5-week observation interval, accounting for 83% of all observed downtime. The distribution shows a daily periodicity, since changes in machine on-or-off state tend to happen during workdays. There are two large spikes near the left edge, indicating downtime due to nightly turnoffs and two-day (presumably weekend) turnoffs.

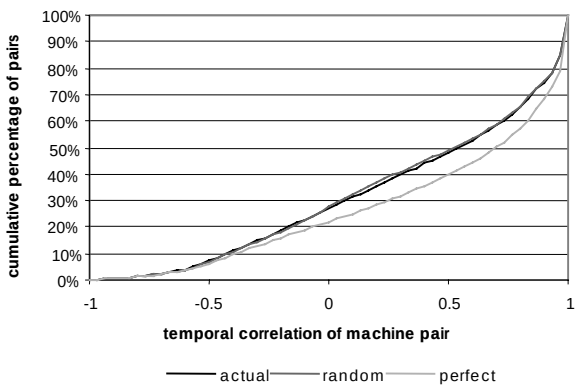

Figure 11: Machine uptime correlation

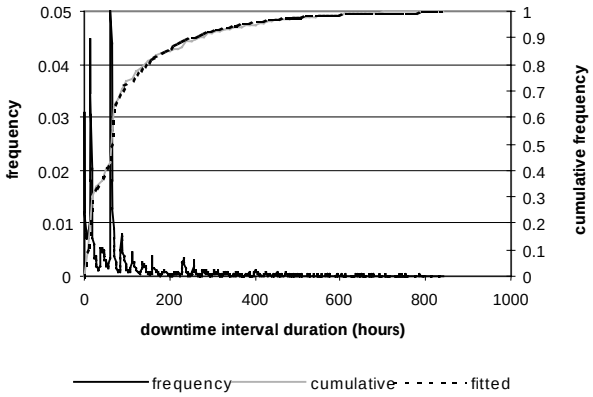

Figure 12: Duration of downtime intervals

We can use the time t_p a machine has been off to predict the amount of additional time t_f it will remain off, using Equation 1 defined in § 3.2.3. To estimate $f(x)$, we use a mix of three distributions: a normal distribution [11] ($\mu_1 = 14$, $\sigma_1 = 1.9$) for nightly 14-hour intervals, a normal distribution ($\mu_2 = 64$, $\sigma_2 = 2.1$) for weekend 64-hour intervals, and a gamma distribution [10] ($\alpha = 0.68$, $\beta = 180$) for the remainder. The mixing coefficients are 0.11, 0.18, and 0.71, respectively. This curve is shown by the dotted line in Figure 12.

Figure 13 shows the result of applying Equation 1 to the fitted distribution. After 72 hours of downtime have been observed, the expected remaining downtime increases monotonically with increasing observed downtime. Prior to this point, the curve is non-monotonic due to the high probability that the downtime is part of a nightly or weekend turnoff.

4.2.5 Machine lifetimes

From the July–October 1999 ping measurements, Figure 14 shows a plot of the count of remaining machines versus time. This curve is approximately a straight line, which is the expected result if machines have deterministic lifetimes (§ 3.2.4). The expected machine lifetime is the x-intercept of the line, which is 290 days.

As described in § 3.4, we have no job-specific data for the first half of the ping period. For the second half, fitting lines to the curves for different job categories yields x-intercepts that vary from 270 days (for business) to 380 (for technical development).

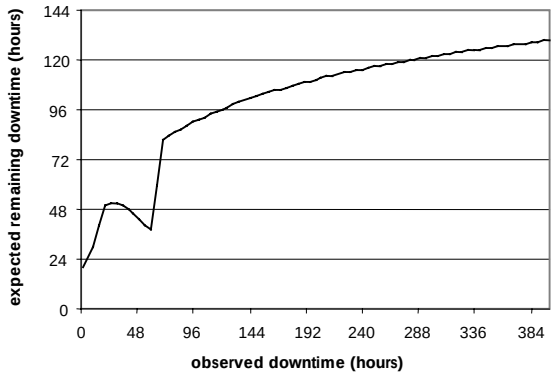

Figure 13: Predicted future downtime

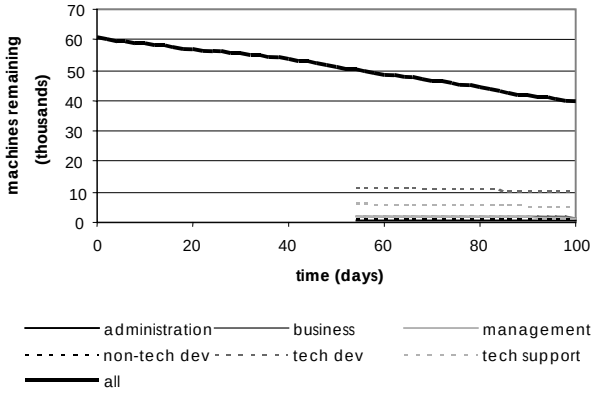

Figure 14: Machine attrition

4.3 Machine Load Results

From the machine-load data we collected as described in § 3.3, we analyzed the CPU load on machines, the disk loads on machines, the correlation of these two loads to each other, and – with the availability data described in § 3.2 – the correlation of CPU load to machine uptime.

4.3.1 CPU load

From the October 1999 load measurements, we examined CPU load versus time of week, as shown in Figure 15. Mean CPU load ranges from about 13% load during the night to about 18% during the day, and the median CPU load varies from about 1% to 2%. This figure also shows the fraction of sampled machines that are 100% loaded, which varies from about 7% to 13%, and which is weakly correlated to time of week, showing slightly higher values on nights and weekends.

4.3.2 Disk load

Figure 16 shows disk load versus time of week. Mean disk load ranges from about 9 operations per second to about 25, with one notable spike of 59 at midnight on Sunday evening. We conjecture that the spike corresponds to a common, periodic maintenance task, such as file-content indexing or backup. As with CPU load, the median disk load is much more uniform and varies from about 8 to 10 operations per second, which represents an essentially idle machine, since a typical modern machine has a peak performance of several hundred disk operations per second.

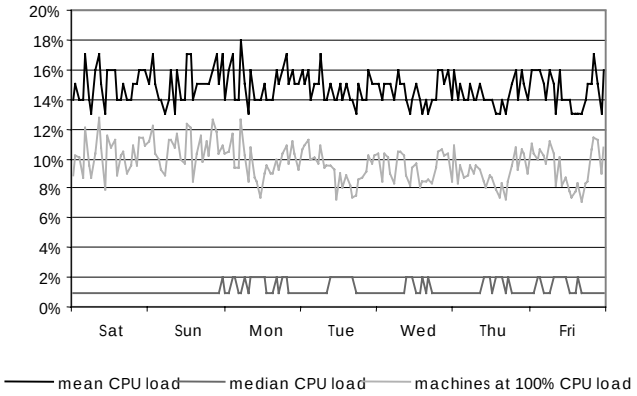

Figure 15: CPU load vs. time of week

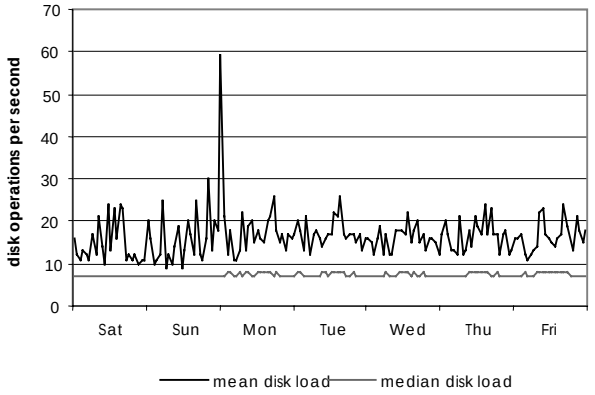

Figure 16: Disk load vs. time of week

Figures 17 and 18 are cumulative distributions of CPU load and disk load, respectively. The variation due to job category is relatively small. The CPU distribution is bimodal: 9% of samples reflect 100% load, and most other samples reflect near idleness.

4.3.3 Machine load correlation

CPU loads are somewhat correlated with disk loads. We computed a Spearman rank-correlation coefficient [10] of 0.27 for these two values across all samples.

The fraction of time that a machine is up is not correlated to the mean CPU load of the machine while it is up. We computed a Spearman rank-correlation coefficient of –0.0087 for these two values across all machines.

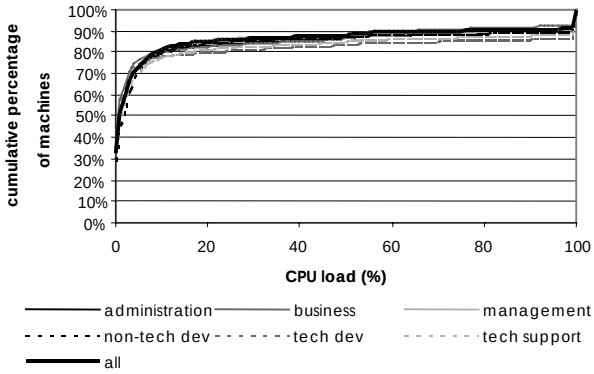

Figure 17: Cumulative distribution of CPU load

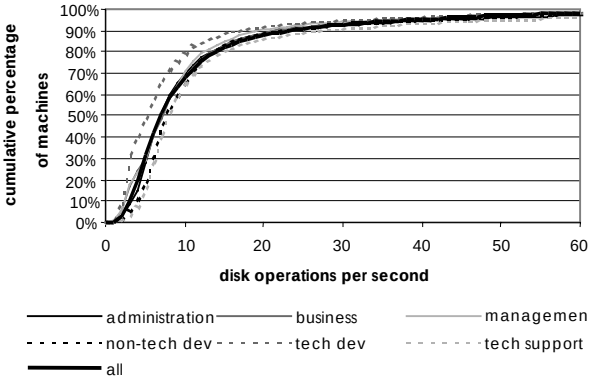

Figure 18: Cumulative distribution of disk load

5. FEASIBILITY ANALYSIS

In this section, we analyze the feasibility of deploying our proposed system on the desktop infrastructure measured in § 3. We determine how well our system would work if we added no new hardware resources to the environment. For this analysis, we use the September 1998 file-system data, but we conservatively assume only 47% space reclamation from duplicate elimination, which is the value obtained from the much smaller February 1999 data set. Since we have no data on file compressibility, we ignore this potential source of additional reclaimed space, although it may dramatically improve the feasibility results.

A critical design parameter is the cache retention period. Increasing the cache retention period increases the expected size of the cache, which decreases the space available for the global file store, thus decreasing the file replication factor. Figure 4, showing cumulative data size in accessed directories versus elapsed time since access, can be interpreted as cache size versus the cache retention period. The solid line in Figure 19 shows the file replication factor as a function of the cache retention period.

5.1 Availability Analysis

Figure 20 shows median file availability, as a function of replication factor, when files are placed as described in § 2.2.1. The y-axis is scaled in nines, which is the negative decimal logarithm of the fraction of time that a file is unavailable. Figure 5, showing directory count versus last access day, can be interpreted as the cache miss rate versus cache retention period, following the argument in § 3.1.3.

Given file availability a (nines) and cache miss rate m (directories per day), the probability of access failure for any given file system's files on any given day is:

$$p_F = 1 - \left(1 - 10^{-a}\right)^m \quad (4)$$

The dotted line in Figure 19 shows this probability expressed reciprocally as mean time between failures, versus the cache retention period. The fluctuation is due to measurement artifact in the access times, as discussed in § 3.4. The order of magnitude is one unfilled file request per file system per thousand days.

5.2 Reliability Analysis

If machines always notify the system before being permanently decommissioned, overall file reliability is governed by the disk failure rate [25]. In particular, the likelihood of permanently losing a file is exponentiated by the replication factor.

However, if machines do not always notify the system, reliability is substantially degraded. To determine a lower bound on reliability, we assume that machines always disappear without warning. The mean time between loss of one machine full of replicas is equal to the mean machine lifetime, l. A directory full of files will be permanently lost if all machines containing the other replicas of these files are decommissioned before the system recognizes the loss and creates new replicas. Given d directories per machine, r replicas, and lag τ between a machine's turnoff and the creation of replacement replicas, the mean time between losing a directory of files per machine is:

$$\mu_L = \frac{l}{d}\left(\frac{l}{\tau}\right)^{r-1} \quad (5)$$

As an example, for r = 3 (Figure 19), l = 290 days (§ 4.2.5), τ = 3 days (§ 4.2.4), and d = 1700 [7], the lower bound on the mean time between one directory loss per machine is μ_L = 1600 days.

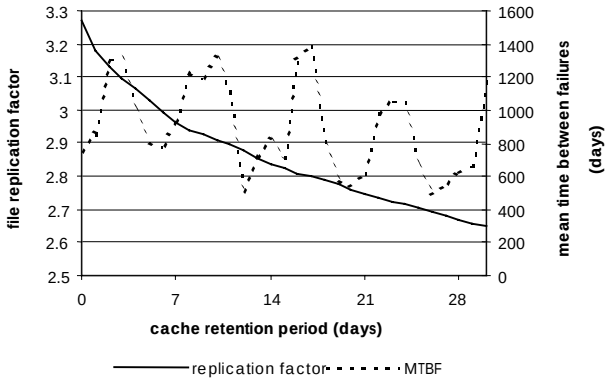

Figure 19: Replicas and MTBF vs. cache retention period

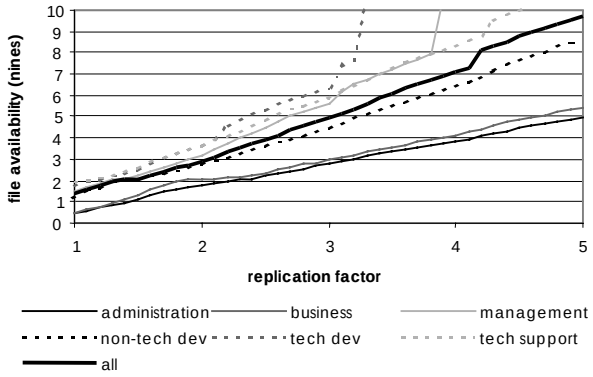

Figure 20: File availability vs. replication factor

5.3 Performance Analysis

Figure 6, showing data-access volume versus last access day, can be interpreted as the cache fill rate versus cache retention period, following the argument in § 3.1.3. For an example cache retention period of one week, the average network traffic from cache fills is a very modest 10 MB / day / file system.

Figure 7, showing data-write volume versus last write hour, can be interpreted as the update propagation rate versus lazy-update propagation lag. For an example propagation lag of one hour, the average network traffic from update propagation is 7 MB / hour / replica / file system, which is also quite modest.

Although these rates ignore burstiness in file activity (§ 3.4), they are so low that a typical modern desktop PC needs only a few seconds to service the entire traffic it would see in a typical day.

6. RELATED WORK

Most distributed file systems are implemented on centralized servers [23]. Performance is enhanced by local caching of files [13] on client machines. High availability is achieved with fault-tolerant hardware, replication across clustered servers [26, 30], and access to locally cached files when the server is inaccessible [14, 24]. High reliability is achieved through RAID [21] or replication [12].

Previous serverless distributed file systems include xFS [1] and Frangipani [27]. The xFS file system, part of the Berkeley NOW project [2], focused on providing support to distributed applications on workstations interconnected by a very high-performance network. Frangipani is a file system built on the Petal [15] distributed virtual disk, which is implemented in a decentralized fashion. Both systems provide high availability and reliability through distributed RAID semantics. Whereas xFS did not focus on administration issues, Petal provided support for

transparently adding, deleting, or reconfiguring servers. Both systems assume trusted machines and are intended primarily for use within a centrally administered machine cluster.

There have been many published studies of file systems and desktop workstations [3, 4, 9, 17, 18, 20], focusing mostly on Unix systems. Only two prior studies have examined PCs [7, 29].

7. SUMMARY AND CONCLUSIONS

We presented an architecture for a serverless distributed file system that does not assume mutual trust among the client computers. The system provides security, availability, and reliability by distributing multiple encrypted replicas of each file among the client machines.

To assess the feasibility of this system if deployed on an existing desktop infrastructure, we collected data from client machines at Microsoft Corporation.

We found that only half of all disk space is in use, and by eliminating duplicate files, this usage can be significantly reduced, depending on the population size. Half of all machines are up and accessible over 95% of the time, and machine uptimes are randomly correlated. Machines that are down for less than 72 hours have a high probability of coming back up soon. Machine lifetimes are deterministic, with an expected lifetime of around 300 days. Most machines are idle most of the time, and CPU loads are not correlated with the fraction of time a machine is up and are weakly correlated with disk loads.

Using our measurement data, we determined that if our proposed system were deployed on our measured desktop infrastructure, file availability is on the order of one unfilled file request per thousand days. The lower bound on reliability is around one directory loss per couple thousand days, but actual reliability should be much higher if machines generally notify the system before being permanently decommissioned.

Since availability and reliability are exponentially sensitive to the number of distributed replicas, small increases in disk space can profoundly increase availability and reliability. For example, adding an average of one dollar's worth of disk space per machine can nearly double the mean time between failures. Thus, in actual deployment of our proposed system, it would be wise to increase the available storage, either by increasing client-disk sizes or by adding userless client machines to the system. This is particularly true for small installations, which can reclaim little space through duplicate elimination. However, our measurements suggest that even deployed on an existing set of desktop computers in a large corporation, our proposed system would work fairly well.

REFERENCES

[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, R. Wang. Serverless Network File Systems. *15th SOSP*, p. 109-126, Dec 1995.

[2] T. Anderson, D. Culler, D. Patterson, the NOW team. A Case for NOW (Networks of Workstations). *IEEE Micro*. p. 54-64. Feb 1995.

[3] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, D. Patterson. The Interaction of Parallel and Sequential Workloads on a Network of Workstations. *1995 SIGMETRICS '95*, p. 267-278, May 1995.

[4] M. Baker, J. Hartmann, M. Kupfer, K. Shirriff, J. Ousterhout. Measurement of a Distributed File System. *13th SOSP*, p. 198-212, Oct 1991.

[5] W. J. Bolosky, S. Corbin, D. Goebel, J. R. Douceur. Single Instance Storage in Windows 2000. To appear in *4th Usenix Windows System Symposium*, Aug 2000.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. INFOCOM '99, p. 126-134, Mar 1999.

[7] J. R. Douceur, W. J. Bolosky. A Large-Scale Study of File-System Contents. *SIGMETRICS '99* 27(1), p. 59-70, May 1999.

[8] J. R. Douceur, W. J. Bolosky. Sanitized Data Set from "A Large-Scale Study of File-System Contents." CD-ROM. Microsoft Research, 1999.

[9] F. Douglis, J. Ousterhout. Transparent Process Migration; Design Alternatives and the Sprite Implementation. *Software - Practice and Experience* 21(8), p. 757-785, Aug 1991.

[10] J. E. Freund. *Mathematical Statistics*, Fifth Edition. Prentice Hall, 1992.

[11] K. F. Gauss. *Theoria combinationis observationum erroribus minimis obnoxiae*, 1823.

[12] D. K. Gifford. Weighted voting for replicated data. *7th SOSP*, p. 150-162, 1979.

[13] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, M. West. Scale and Performance in a Distributed File System. *TOCS* 6(1), p. 51-81, Feb 1988.

[14] J. Kistler, M. Satyanarayanan. Disconnected operation in the Coda File System. *TOCS* 10(1), p. 3-25, Feb 1992.

[15] E. Lee, C. Thekkath. Petal: Distributed virtual disks. *7th ASPLOS*, p. 84-92, Oct 1996.

[16] M. McKusick, W. Joy, S. Leffler, R. Fabry. A Fast File System for UNIX. *TOCS*, 2(3):181—197, Aug 1984.

[17] L. Mummert, M. Satyanarayanan. Long Term Distributed File Reference Tracing: Implementation and Experience. *Software - Practice and Experience* 26(6), p. 705-736, Nov 1994.

[18] M. M. Mutka, M. Livny. The Available Capacity of a Privately Owned Workstation Environment. *Performance Evaluation* 12(4), p. 269-284, Jul 1991.

[19] M. Nelson, B. Welch, J. Ousterhout. Caching in the Sprite Network Filesystem. *TOCS* 6(1), p. 134-154, Feb 1988.

[20] J. Ousterhout, H. Da Costa, D. Harrison, J. Junze, M. Kupfer, J. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. *10th SOSP*, p. 15-24, Dec 1985.

[21] D. Patterson, G. Gibson, R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *Internat. Conf. on Management of Data*, p. 109-116, Jun 1988.

[22] R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. Bolosky, J. Chew. Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures. *2nd ASPLOS*, p. 31-41, Oct 1987.

[23] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon. Design and Implementation of the Sun Network File System. *Summer USENIX Proceedings*, 1985.

[24] M. Satyanarayanan. Scalable, Secure and Highly Available Distributed File Access. *IEEE Computer* 23(5), May 1990.

[25] Seagate Technology, Inc. http://seagate.com

[26] D. Solomon. *Inside Windows NT Second Edition*. Microsoft Press, 1998.

[27] C. Thekkath, T. Mann, E. Lee. Frangipani: A Scalable Distributed File System. *16th SOSP*, p. 224-237, Dec 1997.

[28] S. Travaglia. BOFH. http://www.theregister.co.uk/cgi-bin/SearchArticles.pl?search=BOFH

[29] W. Vogels. File system usage in Windows NT 4.0. *17th SOSP*, p. 93-109, Dec 1999.

[30] B. Walker, G. Popek, R. English, C. Kline, G. Thiel. The LOCUS Distributed Operating System, *9th SOSP*, p. 49-70, Oct 1983.