Optimizing File Availability in a Secure Serverless Distributed File System

John R. Douceur and Roger P. Wattenhofer
Microsoft Research
{johndo, rogerwa}@microsoft.com

Abstract

Farsite is a secure, scalable, distributed file system that logically functions as a centralized file server but that is physically realized on a set of client desktop computers. Farsite provides security, reliability, and availability by storing replicas of each file on multiple machines. It continuously monitors machine availability and relocates replicas as necessary to maximize the effective availability of the system. We evaluate several replica-placement methods using large-scale simulation with machine availability data from over 50,000 desktop computers. We find that initially placing replicas in an availability-sensitive fashion yields pathological results, whereas very good results are obtained by random initial placement followed by incremental improvement using a scalable, distributed, fault-tolerant, and attack-resistant hill-climbing algorithm. The algorithm is resilient to severe restrictions on communication and replica placement, and it does not excessively co-locate replicas of different files on the same set of machines.

1. Introduction

We evaluate the utility, performance, and consequential effects of several candidate methods for replica placement in a distributed file system, for the purpose of optimizing file availability while maintaining system security. The context of our study is Farsite [6], a secure, highly scalable file system that logically functions as a centralized file server but that is physically distributed among a network of untrusted desktop workstations. In this potentially treacherous environment, wherein machines may be capriciously turned off or even maliciously subverted, Farsite provides a high degree of availability and security through randomized replication of both file content and directory infrastructure. This paper investigates scalable, distributed, fault-tolerant, and attack-resistant methods for assigning file replicas to machines to maximally exploit the availability diversity, failure independence, and threat isolation provided by different machines.

The need for replica placement decisions arises in three scenarios: where to place replicas when a new file is created, where to relocate a replica when it is evicted from a machine, and how to rearrange replicas as machine availabilities change over time. These scenarios yield two distinct problems: where to place a replica that is in need of a home (the *initial placement* problem) and how to improve a given arrangement of replicas in an incremental fashion (the *placement improvement* problem).

We find that it is disadvantageous to consider machine availability when determining initial replica placement. Simulations driven by large-scale measurement data show that placing replicas of individual files (or of indivisible sets of files) in an availability-sensitive fashion dramatically skews the distribution of free space among machines, completely consuming all available space on most machines and concentrating the free space on a small fraction of machines in a narrow range of measured availability levels, which severely impedes the system's ability to relocate replicas in the future.

For placement improvement, we consider a family of distributed hill-climbing algorithms that successively exchange the locations of two file replicas. These algorithms are randomized to thwart coercion by malicious machines. We find that the basis algorithm, in which files are selected completely randomly, is highly efficacious but very inefficient. We introduce two improvements that significantly increase the efficiency of the optimization process, with a simple augmentation to resist targeted attack by malicious machines.

Since an arbitrary initial placement followed by efficacious placement improvement yields a good final placement, we conclude that initial placement should be performed randomly, with a minor restriction to diminish security risk. Our placement improvement algorithm not only optimizes file availability but also improves the distribution of free space among machines, making it more evenly distributed than random placement.

Section 2 describes the Farsite system. Section 3 defines the placement problem and the requirements of a solution. Section 4 describes the simulated environment for our evaluation. Section 5 studies the initial placement problem, and section 6 studies the placement improvement problem. Section 7 considers the effects of two different restrictions on the replica placement system: one for reasons of scalability, the other for reasons of security. Section 8 examines the degree to which placement methods co-locate replicas of different files on the same set of machines. Section 9 discusses related work, and Section 10 summarizes our results and concludes.

2. Farsite system architecture

Farsite [6] is a scalable, serverless file system that exploits the underutilized [11] storage and communication resources distributed among the networked desktop computers of a large organization, such as a university or corporation. It provides high levels of availability, reliability, and security without reliance on centralized administration or physically protected infrastructure. Instead, every client desktop machine that stores files in the file system also serves both as a repository for replicas of encrypted file content and as a member of a group of machines that manage a region of the file-system namespace. Since Farsite's constituent machines function primarily as client computers for their local users and only secondarily as distributed storage and directory hosts, the system must provide remote file services without interfering with users' local tasks and without requiring users to modify their behavior.

Like any file system, Farsite has two classes of objects to maintain: directories and files. In the aggregate, directories consume little storage compared to files; however, they must be comprehensible and revisable directly by the system. In contrast, files consume a large amount of storage space, but they can be (and for security reasons, should be) completely opaque to the system. Farsite thus employs different techniques for maintaining these two classes of objects: Byzantine-fault-tolerant groups for directories, and encrypted replication for files.

Figure 1 illustrates a portion of a Farsite system from the perspective of a single *client*; aspects of the system that are not relevant from this perspective are grayed out. This figure shows a single client machine, a set of machines that collectively comprise a *directory host*, and a set of machines each of which functions as a *file host*. (In practice, every machine actually performs all of these roles.)

A directory host is a group of machines that interact using a Byzantine-fault-tolerant protocol [9]. All (non-faulty, non-compromised) machines in the group perform the same sequence of operations, and the protocol preserves the integrity of the group as long as fewer than one third of the machines misbehave in any arbitrary or malicious manner. Each directory host manages a set of directories in the file-system namespace, providing a hierarchical name-based catalog of storage-machine locations and other metadata for files and subdirectories. For operations that are not fully determined by client requests, directory hosts make decisions via cryptographically secure distributed random number generation [5] to prevent any single member from coercing the selection of another directory host or file host and thereby compromising the security of the system.

Each file in a Farsite directory is encrypted, replicated, and stored on multiple file hosts. Encryption provides data privacy, and a cryptographic file hash [32] stored in the directory host provides data integrity. Replication provides data persistence even if some file hosts die, suffer head crashes, or maliciously destroy their stored files. Replication also provides file availability even if some of the file hosts are unavailable when a file is requested. Since different machines are available for different fractions of time, Farsite continuously monitors machine availability, and it relocates replicas as necessary to maximize the availability of files to clients, subject to security and reliability constraints.

Files that a client has recently accessed are cached locally, providing guaranteed and immediate availability. Files not recently accessed must be retrieved from one of the remote file hosts that store replicas of the file, as indicated by the dotted lines between the client and file hosts in Figure 1. In the latter case, the client contacts the directory host to determine which machines contain replicas of the file.

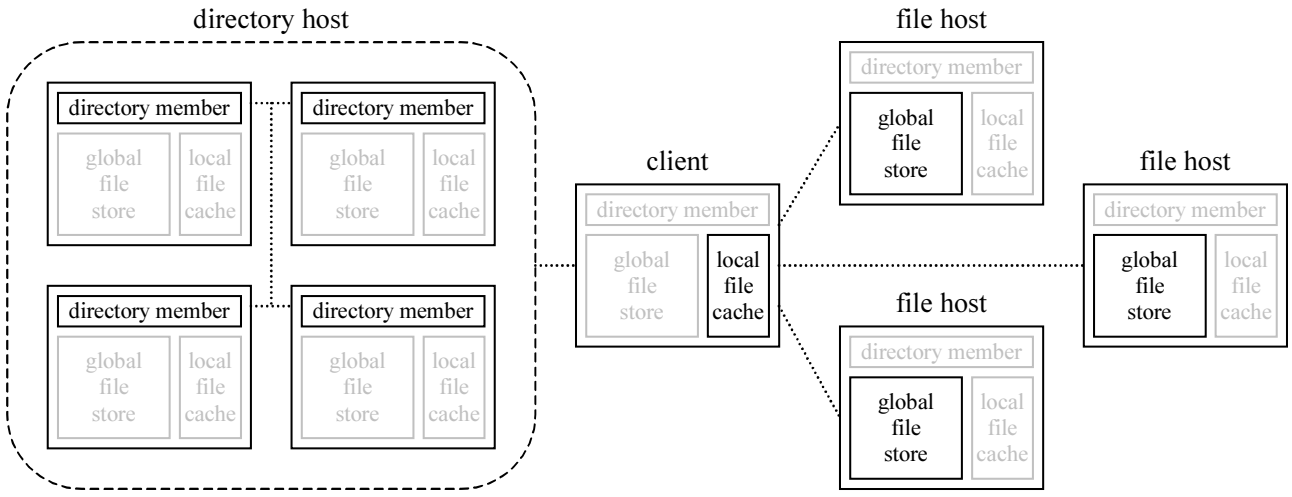


Figure 1. Portion of Farsite system architecture from one client's perspective

3. Problem statement, solution requirements

The problem we address is how to produce an assignment of file replicas to machines that maximizes security, reliability, and availability over all files. Although the security of a machine is hard to assess in absolute terms, the probity of two machines with the same owner is highly correlated [31], so no more than one replica of each file should be placed on machines owned by a single user. Since it is difficult to assess the remaining lifetime of a particular disk or machine with any accuracy [30], all machines are considered to have equal reliability, equitable allocation of which requires that each file have the same number of replicas as every other file. In contrast to the previous two qualities, availability varies widely between machines [6], so this is the principal criterion for determining the specific placement of replicas.

The *fractional downtime* of a machine is the mean fraction of time that the machine is unavailable. For pedagogical purposes, we will momentarily assume that the times at which different machines are unavailable are not significantly correlated with each other, an assumption which has some empirical justification [6]. Therefore, since a file is unavailable only if all of its replicas are unavailable, the fractional downtime of a file is equal to the product of the fractional downtimes of the machines that store replicas of that file. For convenience, we express machine and file availability values as the negative decimal logarithm of fractional downtime. Thus, the availability of a file is equal to the sum of the availabilities of the machines that store the file's replicas. The common unit for availability is the "nine"; for example, a machine with a fractional downtime of 0.01 has $-\log_{10}(0.01) = 2$ nines of availability, intuitively corresponding to its *fractional uptime* of $1 - 0.01 = 0.99$.

Since the client's local cache exploits and exhausts any temporal locality in file accesses, cache misses have relatively little temporal locality. Therefore, our file-placement objective is to maximize the *effective system availability* (ESA), defined as the negative decimal logarithm of the mean file downtime, μ. Given N files each with availability a_i, effective system availability can be calculated as:

$$\text{ESA} = -\log_{10}\mu = -\log_{10}\frac{1}{N}\sum_{i=0}^{N-1}10^{-a_i} \qquad (1)$$

This value is dominated by low-availability files and is maximized when all files have the same availability.

To be suitable for Farsite, a file-placement algorithm must be distributed, iterative, and randomized.

- distributed – Decisions must be made by individual machines or small groups of machines with no central coordination. Requirements for communication or storage must not grow with the size of the system.
- iterative – The algorithm must be able to improve an existing placement incrementally, rather than requiring a complete re-allocation of storage resources when conditions change.
- randomized – For security reasons, the placement algorithm must allow randomness to drive the selection of which other machines to engage in the placement process.

4. Simulated environment

The environment that we simulate is an approximation of a real-world commercial environment measured for an early study of Farsite feasibility [6]. We simulate file placement on a set of $M = 51{,}662$ machines for which we have availability data given by a 35-day series of hourly ping snapshots. The cumulative distribution of machine availabilities is shown in Figure 2a. It is approximately uniform in the range of 0 to 3 nines, which we exploit for the analytic model in Section 7.

Figures 2b and 2c show how the measured machine availabilities change over time. For both of these figures, we compute each machine's availability over a 15-day window. Figure 2b shows that mean machine availability remains fairly constant as this window slides from the beginning to the end of the sample period. Figure 2c shows the cumulative distribution of changes to each machine's availability from the first 15 days of the period

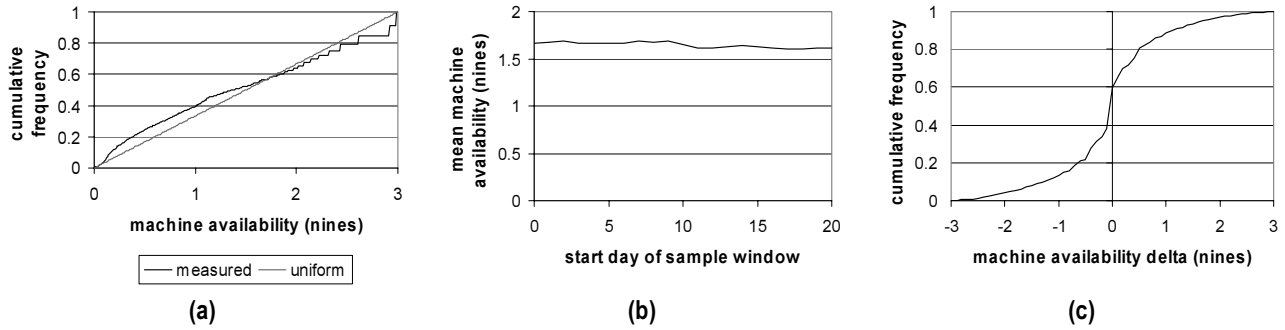


Figure 2. (a) Machine availability distribution, (b) Mean availability vs. time, (c) Availability change in 20 days

to the last 15 days of the period. 75 % of machines have availabilities that change by less than one nine over these 20 days, but 7 % of machine availabilities change by more than two nines, so although there is significant consistency in machine availabilities over time, there is sufficient change to warrant a continual reassessment of availability and concomitant rearrangement of files.

We simulate the placement of 2,583,100 files, which is several orders of magnitude smaller than it would be in a real system of 51,662 machines, but we cannot significantly increase it without exceeding the memory limit of the 512-MB computer we use for simulation. We have run our simulations with smaller counts of files per machine, and the algorithms do not appear to be sensitive to this value.

To allow for replicas to be rearranged, it is necessary to maintain some amount of free space on machines. In a real system, this "free space" does not actually have to be unused: It could hold additional file replicas or local cache entries, but it must be readily reclaimable when needed. For our study, we set the mean value of this excess capacity to 10 % of each machine's storage space.

Our algorithms do not perform well when placing files larger than the mean free space per machine. The smallest hard disk currently available from Seagate [33] has a capacity of 9.2 GB, and fewer than one in two million files [11] is larger than 10 % of this size, so Farsite can prioritize the placement of such extremely large files without significantly affecting other files. For our simulations, file sizes are governed by a binary lognormal distribution with $\mu^{(2)} = 12.2$ and $\sigma^{(2)} = 3.43$ [11], limited to the mean machine free space.

The number of replicas of each file, R, is determined by the fraction of storage capacity that holds unique file content. Measurements of over 10,000 file systems of commercial desktop computers in 1999 [11] indicate that a replication factor of $R = 3$ is readily achievable [6]. More recent measurements (unpublished) indicate that the fraction of free disk space is steadily increasing, such that a replication factor of $R = 4$ is now likely achievable. The present paper investigates replica placement for both of these replication factors.

A recent study [12] has shown that although different machines' unavailability times are mostly uncorrelated, there is sufficient correlation to noticeably weaken the effective system availability for replication factors beyond $R = 3$. This study empirically found a second-order polynomial relationship between the actual ESA and the ESA as calculated by summing replica availabilities:

$$\mathrm{ESA}_{\mathrm{actual}} = -0.078\ \mathrm{ESA}_{\mathrm{sum}}{}^{2} + 1.5\ \mathrm{ESA}_{\mathrm{sum}} - 0.84 \qquad (2)$$

This model is roughly valid for ESA values in the range of 2 to 6 nines. Values of ESA reported in the present paper are calculated using this model.

5. Initial placement

The problem of initial placement is to select machine locations for homeless replicas. Since the primary goal of replica placement is to optimize file availability, it seems reasonable to base this selection on the measured availability of machines. However, given an efficacious method for placement improvement, it is not necessary to account for availability in initial placement, and the results in this section show that it is actually disadvantageous.

We perform the following experiment for each candidate initial placement method: Beginning with an arbitrary placement, we randomly select a machine, evict a replica from it, have the replica's directory host place the replica on another machine, and iteratively repeat this process until the ESA has converged to an apparent asymptote. For brevity, this section shows results only for $R = 3$, but those for $R = 4$ are substantially similar.

Figure 3a shows the count of replicas, the mean file size, and the fraction of free space on each machine versus

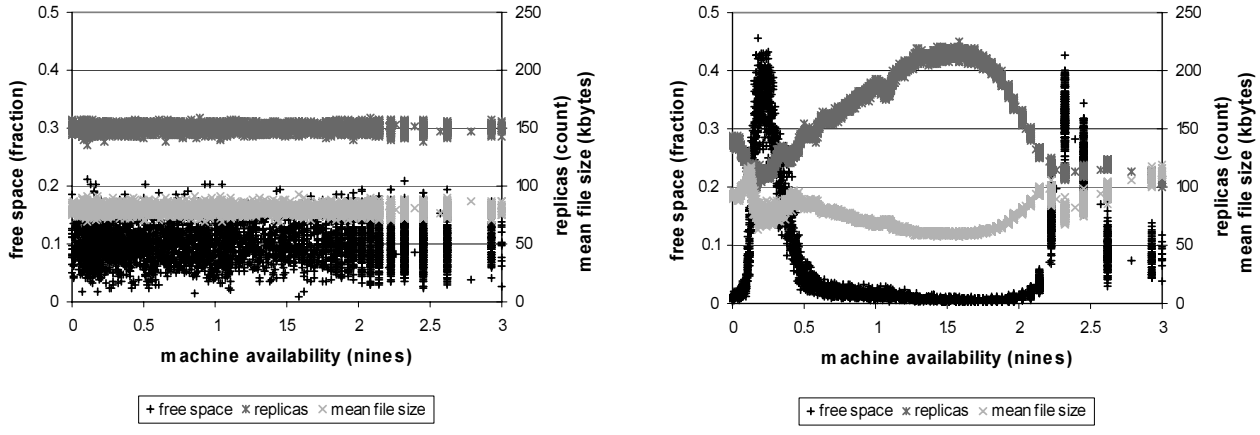


Figure 3. Initial placements for $R = 3$: (a) random, (b) availability-constrained

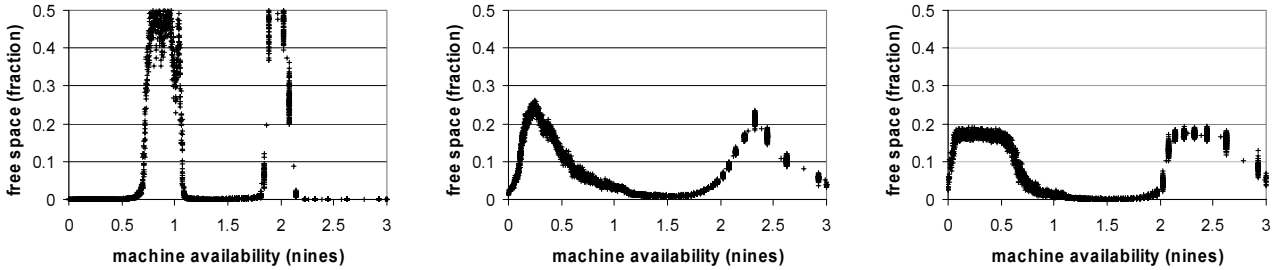


Figure 4. Modified availability-constrained placements for *R* = 3: (a) moderate replica, (b) space-biased, (c) 15% limited eviction

machine availability, when each evicted replica is placed on a randomly selected machine. (If the machine already contains another replica of the same file, or if there is insufficient free space to accommodate the replica, a different machine is selected until these conditions no longer obtain.) All three values in Figure 3a are uncorrelated to machine availability, which is desirable. The resulting ESA (not conveyed in the figure) is 2.2 nines, well below the mean file availability of 4.3 nines, but this is correctible by the placement improvement method described in the next section.

Figure 3b shows results when the evicted replica is disallowed from any machine that would cause the availability of the file to be further from the mean file availability than it was before its replica was evicted (thus never permitting the ESA to drop). Eventually, all machines with availabilities between 0.5 and 2.0 become full, and all free space is concentrated on machines with extremal availability values. This placement makes poor use of high-availability machines, weakens the dispersal of remote file requests among machines, and impedes future relocation of replicas as system conditions change.

We have attempted numerous modifications to the availability-constrained placement method to prevent this detrimental effect on free space distribution, none of which has been successful. One approach is to constrain (or bias) the selection of the evicted replica according to its relative availability; Figure 4a shows the result of evicting the replica whose availability lies between that of the other two replicas of the same file. Another approach is to specifically account for machine free space; Figure 4b results from disallowing a placement if the target machine has less free space than the machine from which the replica was evicted. We have even changed the rules of the experiment by not evicting a replica if the free space on the machine exceeds a threshold; Figure 4c sets this threshold to 15 %. From these and many other negative results, we conclude that Farsite should initially place each replica randomly, subject to a minor constraint for security we explore in Section 7: disallowing more than one replica of each file on machines with the same owner.

6. Placement improvement

The problem of placement improvement is to incrementally improve the ESA of a given arrangement of file replicas. The straightforward approach of progressively relocating replicas of individual files is equivalent to evicting files from machines and placing them so as to improve the ESA, and it thus leads to the same skewed free-space distribution described in Section 5. In this section, we consider an alternate method, in which each directory host selects a file, randomly selects another directory host (possibly itself) which also selects a file, and – if there is sufficient free space and if the availability values of the two files can be brought closer together – exchanges the machine locations of one replica from each file, which we call *swapping* the replicas. For $R > 3$, it could be beneficial to swap more than one replica at a time; however, this increases the security risk from a compromised directory host and has empirically shown to reduce the efficiency of the improvement process, so we do not consider it further.

Figure 5 shows the result when this method is applied to a random initial placement and allowed to bring the ESA to within 1 % of the mean file availability. The three illustrated quantities are satisfactorily uncorrelated to machine availability, unlike those shown in Figure 3b.

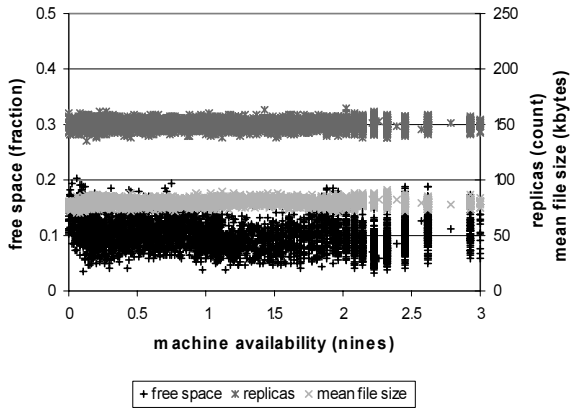


Figure 5. Swap-improved random placement for *R* = 3

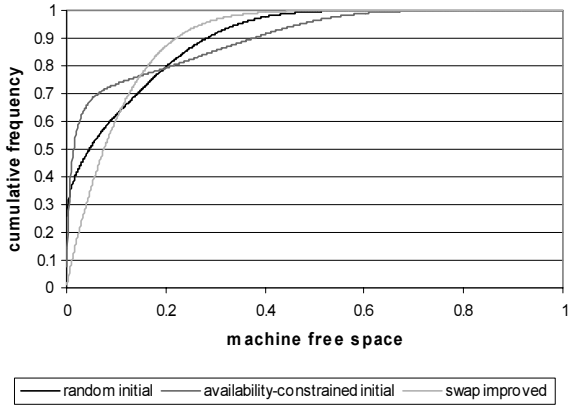


Figure 6. Machine free space distribution for $R = 3$

Furthermore, Figure 6 shows that machine free space is actually more evenly distributed than in the random initial placement: The median free space is 7.6 %, compared to 4.6 % for the random initial placement and 1.4 % for the availability-constrained initial placement. Figures 5 and 6 show results only for $R = 3$, but those for $R = 4$ are substantially similar.

We have not yet specified how each directory host selects a file to replica-swap with the other. The simplest approach is to have each host randomly select one of its files, so this establishes a basis method. Since ESA is dominated by low-availability files, we can focus the improvement on these by having one host select its minimum-availability file, which yields a second method. Furthermore, high-availability files should afford the most opportunity for improving low-availability files, so we can exploit this by having the other host select its maximum-availability file, thus providing a third method. We refer to these three methods as RAND-RAND, MIN-RAND, and MIN-MAX, respectively. For the latter two methods, it is important that the initiating host be the one to select its minimum-availability file, so it is effectively asking for help from – rather than offering help to – the contacted host, thereby making it more difficult for a compromised directory host to launch a targeted attack.

We evaluate the efficiency of these methods at performing two tasks: (1) correcting for random initial placement and (2) adjusting to changes in machine availability over time.

For the former, we randomly evict and place replicas as we did in Section 5, but we randomly intersperse these random relocations with one or more steps of each iterative improvement method, iterating until the ESA has converged to an apparent asymptote. We vary the correction ratio, which is the mean number of corrective moves per random move, counting each swap as two corrective moves since it relocates replicas of two files. The results are shown in Figure 7. With sufficient effort, all three algorithms achieve an ESA of 4.3 nines ($R = 3$) or 5.3 nines ($R = 4$), beyond which our model for the effect of machine failure correlation is no longer valid. MIN-MAX is the most efficient method, keeping the ESA at 90 % of its maximal value with a unity correction ratio.

To assess the responsiveness of each method to the machine availability changes shown in Figures 2b and 2c, we set a window size of 15 days, compute the availability of each machine from our windowed measurement data, and run the method until the ESA reaches a target value. We then slide the window forward by a day, recompute the machine availabilities, and rerun the method, iterating through all days in our sample period. Figure 8 plots the daily replica-relocation rate required to sustain the full range of achievable target ESA values. MIN-MAX is again the most efficient method, sustaining the maximum attainable value of ESA by relocating fewer than 2 % of all file replicas per day. RAND-RAND requires an order-of-magnitude greater relocation rate, but MIN-RAND is almost as good as MIN-MAX for low ESA targets.

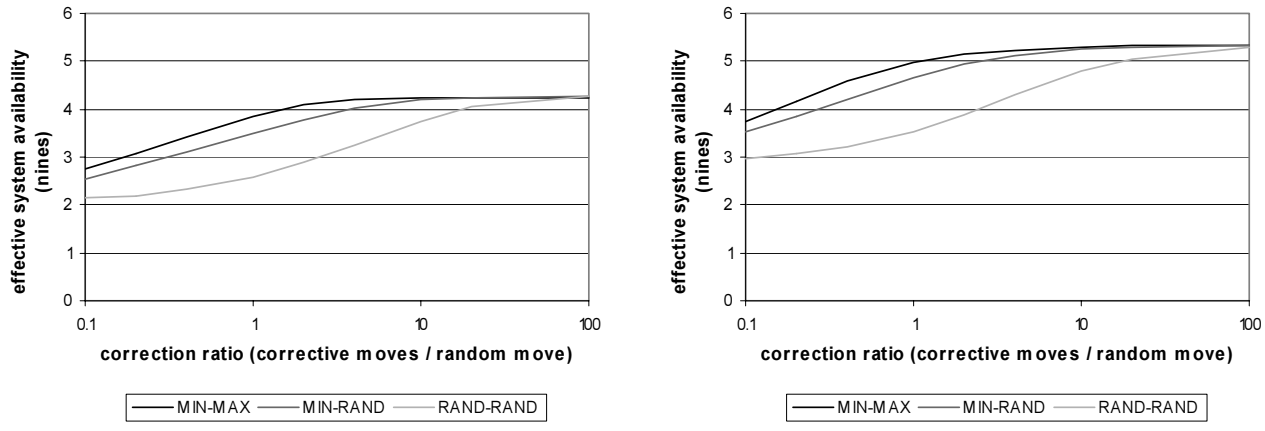


Figure 7. Correcting for random replica relocations: (a) $R = 3$, (b) $R = 4$

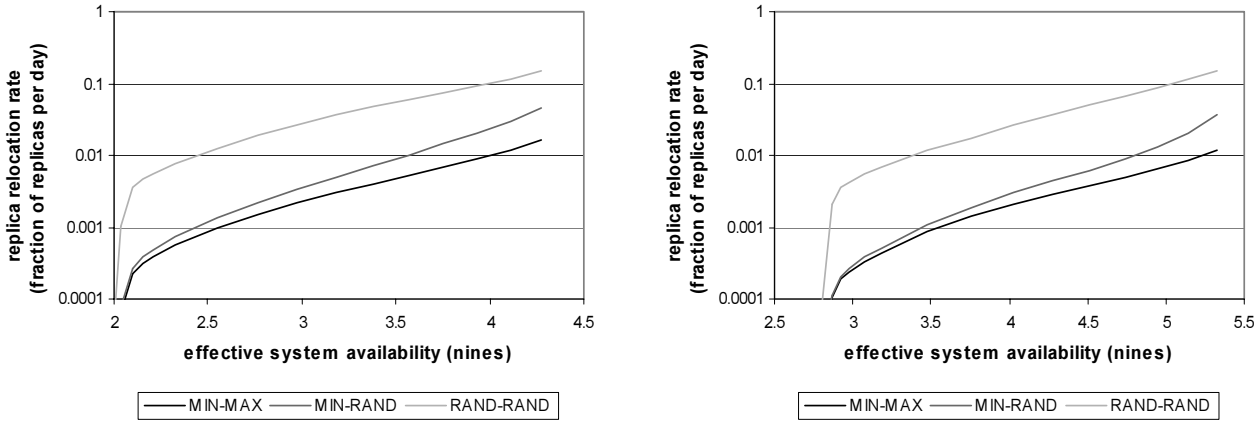


Figure 8. Responding to machine availability changes: (a) $R = 3$, (b) $R = 4$

7. Communication and placement restrictions

For scalability reasons, it is undesirable to require every machine to maintain contact with every other machine in the system. For security reasons, it is unacceptable to place more than one replica of each file on machines with the same owner. In this section, we analyze the effect of each of these restrictions on our placement methods.

Rather than exploring the combinatorially immense array of incomplete communication graphs, we consider only the most extreme case of limited interconnectivity, in which all machines are randomly partitioned into non-overlapping sets of fixed size, providing no indirect path via which to swap replicas with non-neighbors. Figure 9 shows achieved ESA versus number of connections per machine. With merely 30 connections per machine, random placement plus each improvement method retains 90% of the ESA attainable with full connectivity.

For the security restriction, we again consider the most extreme case: Machines are partitioned into R equal-sized sets, each of which has a single owner and whose members are selected randomly. We find that this partitioning has no significant effect on the result of random initial placement. For placement improvement, MIN-MAX and MIN-RAND show no significant change in efficiency; however, the efficiency of RAND-RAND is actually improved by this restriction. Although this result seems highly counterintuitive, it is plausible because the unrestricted case allows some swaps that make little improvement to ESA, whereas the security restriction disallows some of these swaps. The following analytical model predicts the result of our simulation (assuming uncorrelated machine downtimes):

Given a count of C files, a swap between files with availabilities v and w (expressed as negative decimal logarithms) that changes them to v' and w' will change the mean file downtime from μ to μ':

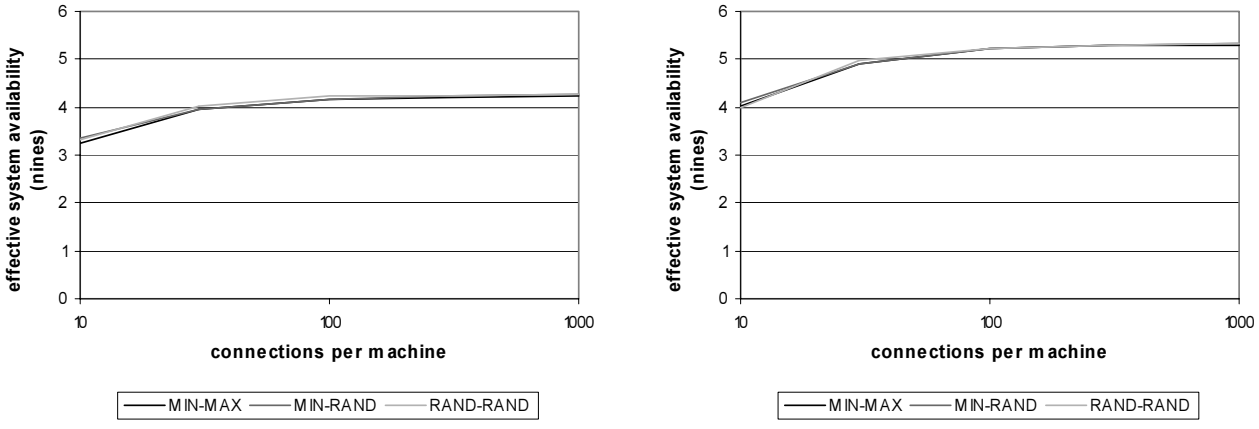


Figure 9. Effect of limited machine interconnectivity: (a) $R = 3$, (b) $R = 4$

$$\mu' = \mu + \left(10^{-v'} - 10^{-v} + 10^{-w'} - 10^{-w}\right)/C \quad (3)$$

For a large count of files, the change in effective system availability relative to the number of relocations per replica is as follows, since there are two relocations per swap and a total of *RC* replicas:

$$\begin{aligned}\lim_{C\to\infty} \frac{RC}{2}\left(\mathrm{ESA}' - \mathrm{ESA}\right) &= \lim_{C\to\infty} \frac{RC}{2}\left(\log_{10}\mu - \log_{10}\mu'\right) \\ &= -\frac{R}{2\mu\ln 10}\left(10^{-v'} - 10^{-v} + 10^{-w'} - 10^{-w}\right)\end{aligned} \quad (4)$$

Therefore, given two files, each with *R* replicas whose availabilities are respectively given by the *R*-element vectors **x** and **y**, the ESA improvement from the best possible swap is given by function *s*:

$$s(\mathbf{x},\mathbf{y}) = \max_{0\le i,j<R} \frac{R}{2\mu\ln 10}\left(10^{-\Sigma\mathbf{x}} - 10^{-\Sigma\mathbf{x}+x_i-y_j} + 10^{-\Sigma\mathbf{y}} - 10^{-\Sigma\mathbf{y}+y_j-x_i}\right) \quad (5)$$

Restricted swaps can only exchange replicas with matching indices (indicating the same set of machines):

$$s_r(\mathbf{x},\mathbf{y}) = \max_{0\le i<R} \frac{R}{2\mu\ln 10}\left(10^{-\Sigma\mathbf{x}} - 10^{-\Sigma\mathbf{x}+x_i-y_i} + 10^{-\Sigma\mathbf{y}} - 10^{-\Sigma\mathbf{y}+y_i-x_i}\right) \quad (6)$$

For machine availabilities uniformly distributed between 0 and $A = 3.0$ (as shown in Figure 2a) we calculate the expected change in ESA over all swaps that yield positive changes. The notation $\int d\mathbf{x}$ indicates a multiple integral over all elements of vector **x**:

$$\Delta = \int_0^A\int_0^A \max(s(\mathbf{x},\mathbf{y}),0)\,d\mathbf{y}\,d\mathbf{x} \Big/ \int_0^A\int_0^A (s(\mathbf{x},\mathbf{y}) > 0)\,d\mathbf{y}\,d\mathbf{x} \quad (7)$$

The expected change from restricted swaps, Δ_r, is calculated similarly. The ratio of these two values, $\rho = \Delta_r/\Delta$, shows the performance change from adding the restriction. For comparison, starting with a random initial placement of all replicas, we simulate the RAND-RAND algorithm with and without the security restriction and measure the improvement per swap. Table 1 compares the analytic results to the simulation results with 90 % confidence intervals. The model's predictions are well within reasonable error bounds.

Table 1. RAND-RAND performance changes from extreme security restrictions

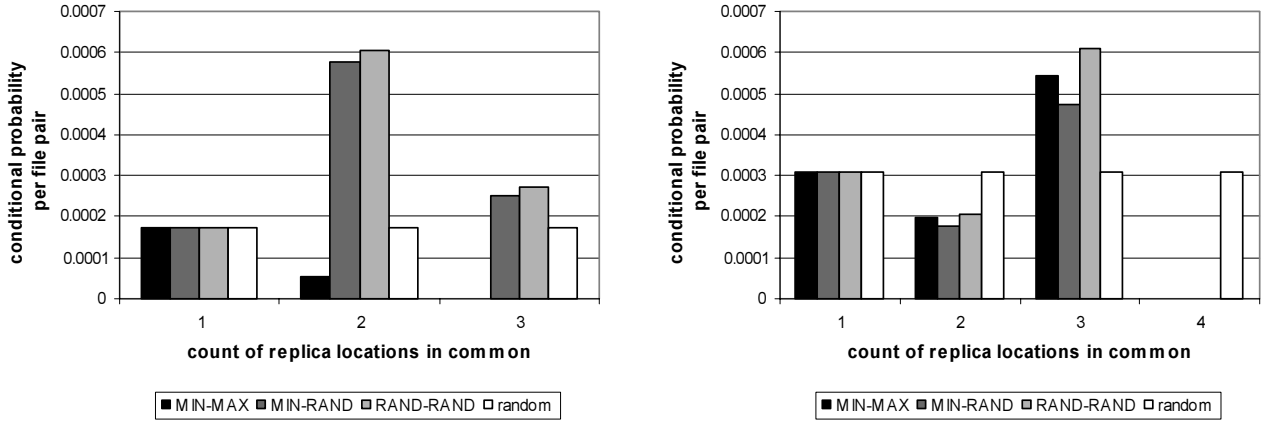
Replicas	Model	Simulation
$R = 3$	$\rho = 1.15$	$\rho = 1.12 \pm 0.09$
$R = 4$	$\rho = 1.14$	$\rho = 1.17 \pm 0.05$

8. Dispersion of file replicas

Since the replica placement system attempts to equalize the availability values of all files, it may seem that any two files with at least $R - 1$ replicas on the same machines are highly likely to have all *R* replicas on the same machines. If so, this would be a significant security weakness, since a malicious machine could adjust its own availability to coerce the placement of a replica of a particular file onto that machine. It would also cause file unavailability to occur in bursts during which multiple files (those on the same set of machines) would all become unavailable simultaneously. Thus, it is important to determine whether our algorithms do a good job of dispersing the replicas of each file relative to those of other files.

Beginning with a random initial placement, we run each algorithm until no further improvements are possible. Then, we examine the resulting layouts to calculate the conditional probability that at least *r* replicas of any two files will reside on the same set of machines if at least $r - 1$ of the two files' replicas reside on the same set of machines. For randomly placed replicas, this conditional probability equals R^2/M, for $1 \le r \le R$, given *R* replicas and *M* machines. Figure 10 shows the conditional probabilities calculated from the simulation runs.

Except for MIN-MAX at $R = 3$ (a special case with a sub-optimal local minimum [12]), there is a slightly elevated conditional probability for $R - 1$ replicas of two files to be co-located when $R - 2$ of the replicas already are, but the absolute probabilities are all very low. We conclude that our algorithms do an adequate job of replica dispersion.



(a) (b)

Figure 10. Dispersion of file replicas: (a) $R = 3$, (b) $R = 4$

9. Related work

In the Farsite environment, the dominant cause of machine unavailability is simply users' turning off their machines [6], rather than machine failures or network partitions. In systems built of dedicated components that are not capriciously turned off, understanding the causes of machine failure [22] can be used in detailed component-based reliability analysis [37] to evaluate the availability of a distributed system. Utterly different analysis strategies [4, 28] are appropriate for systems prone to network partitions, with results [29] that indicate a hard limit on the availability improvement due to replication, namely an improvement of $\log_{10}(1/2) \approx 0.3$ nines greater than the highest available replica. System availability is also limited by highly correlated failures such as site-wide power outages, but the appropriate techniques for dealing with such issues [19] are orthogonal to the concern of our present work, which is attaining a fairly high degree of availability when storing files on machines with comparatively low availabilities.

In Farsite, consistency is maintained by a distributed directory service using a Byzantine fault-tolerant protocol [9]. Updates to files are propagated lazily from the client machine that performs the modification to the machines that store replicas. Thus, we do not employ consistency protocols [1, 16, 20, 21] among machines storing replicas.

Some earlier work on file placement focused on access load balancing [7, 36], rather than availability. Others addressed availability [25] but not automated replica placement. A significant body of work concerns file migration [8, 15, 24, 34], relocating replicas to machines near points of high usage, whereas we explicitly ignore geographic issues, because in Farsite's target environment, all machines are interconnected by a low-latency network. McCue and Little [26] simulated a replica placement algorithm that yields significantly greater availability than random placement but which requires global coordination.

Other serverless distributed file systems include xFS [2] and Frangipani [35], which provide high availability and reliability through distributed RAID rather than full replication. Archival Intermemory [18] and OceanStore [23] both use erasure codes and widespread distribution to avoid data loss. The Eternity Service [3] uses replication in a very wide scale to prevent loss even under organized attack, but does not address automated placement of data replicas. Napster [27] and Gnutella [17] provide services for finding files but do not explicitly replicate files nor determine storage locations. Freenet [10] generates and relocates replicas near points of usage.

In addition to the current paper, our work on file replica placement includes competitive analysis [14], theoretic analysis using an analytic model of machine availability [13], and a simulation study of the detailed transient behavior of placement methods [12].

10. Summary and conclusions

Farsite is a secure, serverless, highly scalable, fully distributed file system that provides high degrees of file reliability and availability by replicating files and storing the replicas on multiple desktop machines. The system continuously monitors machine availability and relocates file replicas to maximize effective system availability without sacrificing system security, using a distributed hill-climbing algorithm that successively swaps the machine locations of two file replicas, constrained to avoid placing multiple replicas of a file on machines with the same owner.

Large-scale simulation using machine availability data from over 50,000 desktop computers shows that being sensitive to availability when placing replicas of individual files pathologically skews the distribution of free space on machines. By contrast, unbiased random placement does not negatively impact the machine free-space distribution, and swap-based hill climbing improves this distribution as it improves the effective system availability.

The hill-climbing algorithm is driven by a secure source of random numbers to diminish the security risk from malicious machines. The security need for randomness is in selecting groups of machines with which to perform replica swaps, not in selecting individual files for swapping. Therefore, it does not compromise the security of the system to improve the efficiency of hill climbing by selecting a pair of files that have maximally separated availability values. System security can be further improved by ensuring that the initiator of the swap is requesting to improve the availability of its file rather than offering to improve the availability of another file, thus obstructing an avenue of targeted attack.

Simulations based on real-world data show that three file replicas can yield over four nines of availability, and four replicas can yield over five nines of availability. Beyond this point, system availability is significantly diminished by correlated machine downtimes.

Farsite's replica placement methods are scalable, in that they continue to perform well even when the number of communication paths between machines is severely restricted. With a mere 30 connections per machine in a network of over 50,000 machines, the algorithms can achieve 90% of the results achievable with full connectivity.

Although security requires disallowing the placement of more than one replica of each file on machines with the same owner, this restriction does not have any detrimental effect on the effective system availability, even in the most restrictive case of R machine owners and R file replicas.

The placement algorithm does a good job of dispersing the replicas of each file, which prevents a malicious machine from coercing the location of a specific file.

References

[1] P. A. Alsberg and J. D. Day, "A Principle for Resilient Sharing of Distributed Resources", *2nd International Conference on Software Engineering*, IEEE, Oct 1976, pp. 562-570.

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

[3] R. J. Anderson, "The Eternity Service", *PRAGO-CRYPT '96*, CTU Publishing, Sep/Oct 1996, pp. 242-252.

[4] B. S. Bacarisse and S. Bek Baydere, "Reliability of Replicated Files in Partitioned Networks", *1st Workshop on Management of Replicated Data*, IEEE, 1990, pp. 98-101.

[5] J. Benaloh, "Dense Probabilistic Encryption", *Selected Areas in Cryptography '94*, May 1994, pp. 120-128.

[6] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs", *SIGMETRICS 2000*, ACM, Jun 2000, pp. 34-43.

[7] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, Distributed Data Placement Strategies for Storage Area Networks", *12th SPAA*, ACM, Jun 2000.

[8] G. Cabri, A. Corradi, and F. Zambonelli, "Experience of Adaptive Replication in Distributed File Systems", *22nd EUROMICRO*, IEEE, Sep 1996, pp. 459-466.

[9] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance", *3rd OSDI*, USENIX, Feb 1999, pp. 173-186.

[10] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System", *ICSI Workshop on Design Issues in Anonymity and Unobervability*, Jul 2000.

[11] J. R. Douceur and W. J. Bolosky, "A Large-Scale Study of File-System Contents", *SIGMETRICS '99*, ACM, May 1999, pp. 59-70.

[12] J. R. Douceur and R. P. Wattenhofer. "Large-Scale Simulation of a Replica Placement Algorithms for a Serverless Distributed File System." *9th MASCOTS*, IEEE, Aug 2001.

[13] J. R. Douceur and R. P. Wattenhofer, "Modeling Replica Placement in a Distributed File System: Narrowing the Gap between Competitive Analysis and Simulation", *ESA 2001*, Aug 2001.

[14] J. R. Douceur and R. P. Wattenhofer, "Competitive Hill-Climbing Strategies for Replica Placement in a Distributed File System", *15th DISC*, Oct 2001.

[15] B. Gavish and O. R. Liu Sheng, "Dynamic File Migration in Distributed Computer Systems", *CACM* 33 (2), ACM, Feb 1990, pp. 177-189.

[16] D. K. Gifford, "Weighted Voting for Replicated Data", *7th SOSP*, ACM, Dec 1979.

[17] Gnutella. http://gnutelladev.wego.com

[18] A. Goldberg and P. Yianilos, "Towards an Archival Intermemory", *International Forum on Research and Technology Advances in Digital Libraries*, IEEE, Apr 1998, pp. 147-156.

[19] R. Golding and E. Borowsky, "Fault-Tolerant Replication Management in Large-Scale Distributed Storage Systems", *18th SRDS*, IEEE, Oct 1999.

[20] R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page Jr., G. J. Popek, and D. Rothmeier, "Implementation of the Ficus Replicated File System", *1990 USENIX Conference*, Usenxi, Jun 1990, pp. 63-71.

[21] M. Herlihy, "A Quorum-Consensus Replication Method for Abstract Data Types", *TOCS* 4 (1), ACM, Feb 1986, pp. 32-53.

[22] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer "Failure Data Analysis of a LAN of Windows NT Based Computers", *18th SRDS*, IEEE, Oct 1999.

[23] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage", *9th ASPLOS*, ACM, Nov 2000.

[24] Ø. Kure, "Optimization of File Migration in Distributed Systems", *Technical Report UCB/CSD 88/413*, University of California at Berkeley, Apr 1988.

[25] K. Marzullo and F. Schmuck, "Supplying High Availability with a Standard Network File System", *8th ICDCS*, IEEE, Jun 1988, pp. 13-17.

[26] D. L. McCue and M. C. Little, "Computing Replica Placement in Distributed Systems", *2nd Workshop on Management of Replicated Data*, IEEE, Nov 1992, pp. 58-61.

[27] Napster. http://www.napster.com

[28] N. Natarajan and K. Kant, "Maintaining Availability of Replicated Data in Partition-Prone Networks", *1st Workshop on Management of Replicated Data*, IEEE, 1990, pp. 108-112.

[29] L. Raab, "Bounds on the Effects of Replication on Availabilty", *2nd Workshop on Management of Replicated Data*, IEEE, 1992, pp. 44-46.

[30] R. T. Reich and D. Albee. "S.M.A.R.T. Phase-II," White paper WP-9803-001, Maxtor Corporation, Feb 1998.

[31] J. D. Saltzer and M. D. Schroeder. "The Protection of Information in Computer Systems." *Proceedings of the IEEE* 63(9), pp. 1278–1308, Sep 1975.

[32] B. Schneier. *Applied Cryptography*, 2nd Edition. John Wiley & Sons, 1996.

[33] Seagate, Inc. Disc Products by Model Number. "Search Results: Disc Drives: All Capacities, All Interfaces, All Systems." http://www.seagate.com/cda/products/discsales/index, Apr 3, 2001.

[34] A. Siegel, K. Birman, and K. Marzullo, "Deceit: A Flexible Distributed File System", *Summer 1990 USENIX Conference*, USENIX, Jun 1990.

[35] C. Thekkath, T. Mann, and E. Lee, "Frangipani: A Scalable Distributed File System", *16th SOSP*, ACM, Dec 1997, pp. 224-237.

[36] J. Wolf, "The Placement Optimization Program: A Practical Solution to the Disk File Assignment Problem", *SIGMETRICS '89*, ACM, May 1989.

[37] S. M. Yacoub, B. Cukic, and H. H. Ammar, "A Component-Based Approach to Reliability Analysis of Distributed Systems", *18th SRDS*, IEEE, Oct 1999.