Large-Scale Simulation of Replica Placement Algorithms for a Serverless Distributed File System

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

Abstract

Farsite is a scalable, distributed file system that logically functions as a centralized file server but that is physically implemented on a set of client desktop computers. Farsite provides high degrees of reliability and availability by storing replicas of files on multiple machines. Replicas are placed to maximize the effective system availability, using a distributed, iterative, randomized placement algorithm. We perform a large-scale simulation of three candidate algorithms using machine availability data collected from over 50,000 desktop computers. We find that algorithmic efficiency and placement efficacy run counter to each other. We fit analytic functions to the improvement rates and provide explanations for the fitted curves. We explore the algorithms' properties through study of their dynamic behavior. We visualize algorithmic placements and compare them to theoretical worst cases. We quantify the degree of machine failure correlation and develop a formula to approximate its effect.

1. Introduction

This paper analyzes algorithms for automated placement of file replicas in the Farsite [6] system, using simulations based on large-scale measurement data. In the Farsite distributed file system, multiple replicas of files are stored on multiple machines, so that files can be accessed even when some of the machines are unavailable (either turned off or inaccessible). The purpose of the placement algorithm is to determine an assignment of file replicas to machines that maximally exploits the different levels of availability provided by different machines.

The number of replicas of each file, *R*, is fixed by the system. Given measurements of machine availabilities [6] and the efficacy of our algorithms, we find that to attain overall system availability in the desirable range of 4 to 5 nines requires 3 or 4 replicas of each file, so we study the behavior of our algorithms given these two values for *R*.

We require algorithms that can improve an existing placement, so we concentrate on hill-climbing algorithms that successively exchange the machine locations of two file replicas. We investigate the properties of distributed versions of three such algorithms: (1) RAND-RAND, which swaps replica locations between any pair of files, (2) MIN-RAND, which swaps replica locations between a minimum-availability file and any other file, and (3) MIN-MAX, which swaps replica locations between a minimum-availability file and a maximum availability file.

We find that in terms of algorithmic efficiency, the MIN-MAX algorithm performs best, MIN-RAND second, and RAND-RAND the worst. In terms of placement efficacy, RAND-RAND is the best, MIN-RAND is again second, and MIN-MAX is the worst. The algorithms thus present a trade-off between these two desirable qualities.

Section 2 overviews the Farsite system. Sections 3 and 4 describe the algorithms and the environment of our simulation. Sections 5 – 8 detail the algorithms' transient improvement, dynamic behavior, final placement patterns, and influence by machine failure correlation. Sections 9 and 10 wrap up with related work and conclusions.

2. Background

Farsite [6] is a secure, highly scalable, serverless, distributed file system that logically functions as a centralized file server without requiring any physical centralization whatsoever. The system's computation, communication, and storage are distributed among the client computers participating in the system. Farsite runs on a networked collection of desktop computers in a large corporation or university without interfering with users' local tasks and without requiring users to modify their behavior. As such, it needs to provide a high degree of security and fault tolerance without the physical protection and continuous support enjoyed by centralized servers.

Since people turn off their desktop machines whenever they wish, without regard for other users who may wish to remotely access the machine's resources, Farsite employs a high degree of replication in its storage of file and directory data. Since desktop machines are not physically secured, Farsite must be resilient to arbitrary malicious behavior on a subset of the machines that form the system infrastructure. It resists such attacks using two techniques: a Byzantine-fault-tolerant protocol and cryptographically secure distributed random number generation.

Directories are implemented by groups of machines that interact using a Byzantine-fault-tolerant protocol [9], which guarantees correctness if fewer than one third of the machines misbehave in any manner. A group of machines collectively managing a directory is called a *directory host*. Each directory host implements multiple directories, since there are ~10^4 directories on a typical machine [11].

If any single member of a directory host can force the selection of another host for an arbitrary operation, then a single malicious machine can compromise system security. Farsite resists such attacks via cryptographically secure distributed random number generation [5] when determining values for non-deterministic operations.

Files are stored on *file hosts*, which are undistinguished machines in the system. Every machine functions as a file host, as a component of one or more directory hosts, and as a local client. Farsite provides four properties for the files that it stores in file hosts: privacy, integrity, reliability, and availability. Data privacy is afforded by encryption, and data integrity by one-way hash functions and digital signatures [30]. Reliability (data persistence) is provided by making multiple replicas of each file and storing the replicas on different machines. The topic of the present paper is file availability, in the sense of a user's being able to access a file at the time it is requested.

Like reliability, file availability is provided by storing multiple file replicas on different machines. However, whereas the probability of permanent data-loss failure (such as disk head crashes) is assumed to be identical for all machines, the probability of transitory unavailability (such as a machine's being powered off temporarily) has been shown to be heterogeneous by a five-week series of hourly measurements of more than 50,000 desktop machines at Microsoft [6]. This study also concluded that the times at which different machines are unavailable appear predominantly uncorrelated with each other.

We state availability as the negative decimal logarithm of the fraction of time a machine or file is inaccessible, yielding a unit of "nines." For example, a machine with a fractional uptime of 0.99 has $-\log_{10}(1 - 0.99) = 2$ nines of availability. Given uncorrelated machine downtimes, the fraction of time a file is unavailable equals the product of the fractional downtimes of the machines that store the file's replicas. Therefore, expressed logarithmically, the availability of a file equals the sum of the availabilities of the machines that store the file's replicas.

Farsite monitors machine availability and places file replicas to maximize the availability of files to users. Files that a client has recently accessed are stored in a cache on the client machine; files not recently accessed must be retrieved from a remote file host. Since we make no assumptions about the likelihood of accesses to files not recently accessed, we set the file-placement objective to be maximizing the success probability of accessing a random file at a random time. We express this objective as the negative logarithm of the access failure probability, which we call the *effective system availability* (ESA), measured in units of nines. Given N files each with availability a_i, ESA can be calculated as:

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

This value is dominated by low-availability files. For a given value of mean file availability, ESA is maximized by minimizing the file availability variance.

3. Algorithms

To be suitable for a secure, serverless, distributed file system, a placement algorithm must have three properties:

- distributed – Decisions must be made by small machine groups without central coordination. Communication and storage must not grow with the system size.
- iterative – The algorithm must improve an existing placement incrementally without requiring a complete re-allocation of storage when conditions change.
- randomized – Security requires that randomness drive the selection of machines that determine a placement.

We thus investigate a family of randomized, swap-based, hill-climbing algorithms. At a high level, a directory host selects a file, randomly selects another directory host (possibly itself) which also selects a file, and determines whether it can bring the availability values of the two files closer together by swapping machine locations of one replica from each file. If so, it performs the swap. Swaps can only be made if there is sufficient free space on each machine to accept the replicas that are being relocated.

We investigate three algorithms: (1) RAND-RAND, in which each directory host randomly selects a file, (2) MIN-RAND, in which one host selects its minimum-availability file and the other selects a random file, and (3) MIN-MAX, in which one host selects its minimum-availability file and the other selects its maximum-availability file. RAND-RAND is the most general strategy, so it represents a baseline for comparison with the other algorithms. MIN-RAND focuses on low-availability files, since they have the greatest impact on ESA. MIN-MAX exploits the fact that high-availability files afford the most opportunity for improving low-availability files.

4. Simulated environment

Our simulated environment is an approximation of a real environment measured for an initial study of Farsite feasibility [6]. We simulate file placement on a set of 51,662 machines for which we have availability data given by a 5-week set of hourly ping snapshots. The cumulative distribution of machine availabilities, shown in Figure 1, is approximately uniform in the range of 0 to 3.0 nines.

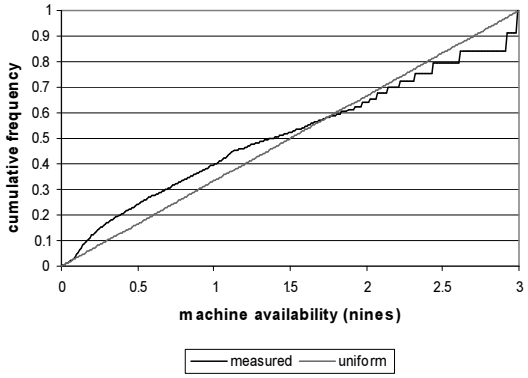

Figure 1. Machine availability distribution

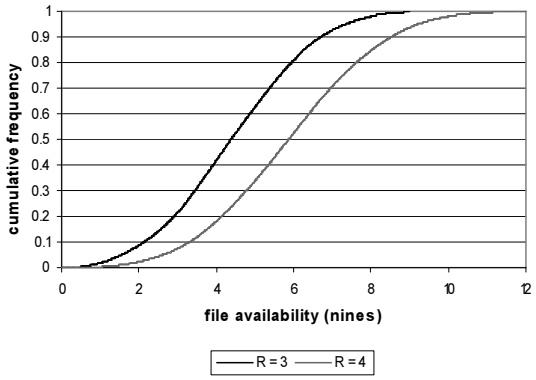

Figure 2. File availability distribution with random placement

File sizes are governed by a binary lognormal distribution with $\mu^{(2)} = 12.2$ and $\sigma^{(2)} = 3.43$ [11]. We simulate the placement of 2,583,100 files, averaging 50 files per machine. This value is far smaller than it would be in a real system, but we cannot significantly increase it without exceeding the memory limit of the 512-MB computer we use for simulation. In simulations with smaller counts of files per machine, the algorithms do not appear to be sensitive to this value. We maintain excess storage capacity in the system, without which it would not be possible to swap file replicas of different sizes. The mean value of this excess capacity is 10 % of each machine's storage space, and we limit file sizes to less than this mean value per machine.

At each step, a pair of files is selected randomly. To account for the distributed nature of this selection in a real system, we set a *selection range* for minimum- and maximum-availability files to 2 %, to be consistent with a mean value of 50 files per machine. In other words, the "minimum-availability" file is drawn from the set of files with the lowest 2 % of availabilities, and the "maximum-availability" file is drawn from the set of files with the highest 2 % of availabilities.

We begin each simulation run by placing the file replicas randomly on machines. Figure 2 shows the distribution of file availability with random replica placement. With 3 replicas, the mean file availability is 4.4 nines; placing the replicas randomly yields an ESA of 2.2 nines. With 4 replicas, the mean file availability is 5.9 nines; placing the replicas randomly yields an ESA of 2.9 nines. In both cases, the minimum file availability is near zero. Random placement thus makes poor use of the availability of the machines in the system.

5. Transient Analysis

Figures 3 and 4 show the evolution of effective system availability for replication factors of 3 and 4, respectively, beginning with a random placement and progressively applying one of the algorithms. For three replicas, MIN-MAX achieves a slightly lower final ESA than the others: For $R = 3$, $ESA_{RR} = ESA_{MR} = 4.4$ and $ESA_{MM} = 4.3$. For $R = 4$, ESA = 5.9 for all algorithms.

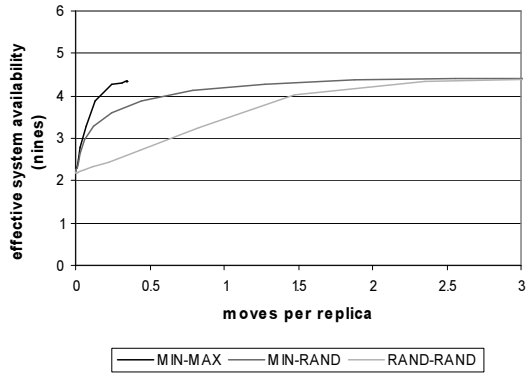

Figure 3. ESA vs. replica relocations (R = 3)

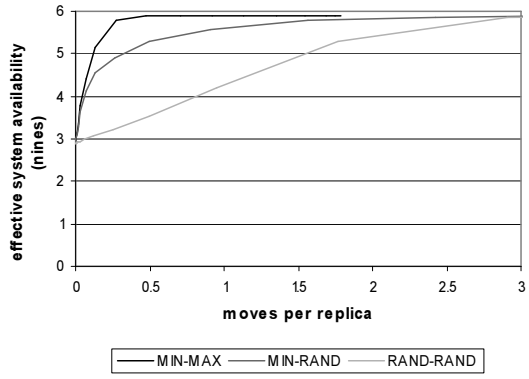

Figure 4. ESA vs. replica relocations (R = 4)

In both cases, the RAND-RAND algorithm makes the slowest improvement to ESA, with a progress half-life of 0.88 ($R = 3$) or 1.1 ($R = 4$) moves per replica. MIN-RAND is considerably faster, with a progress half-life of 0.12. MIN-MAX is the fastest, with a progress half-life of 0.06.

We have attempted to model the transient ESA curves in Figures 3 and 4 with exponential approximations. The MIN-MAX curves are well approximated (RMS error < 2 %) by an exponential with a time constant of 0.093 moves per replica, as shown in Figure 5. The MIN-RAND curves are well approximated by a 2-stage hyperexponential with a primary ($\alpha = 0.61$) time constant of 0.093 and a secondary ($1 - \alpha = 0.39$) time constant of 0.76, as shown in Figure 6.

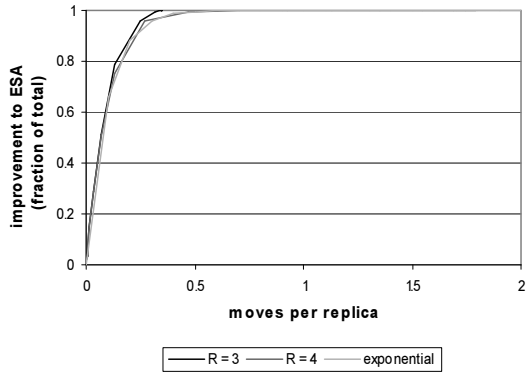

Figure 5. ESA improvement vs. relocations (MIN-MAX)

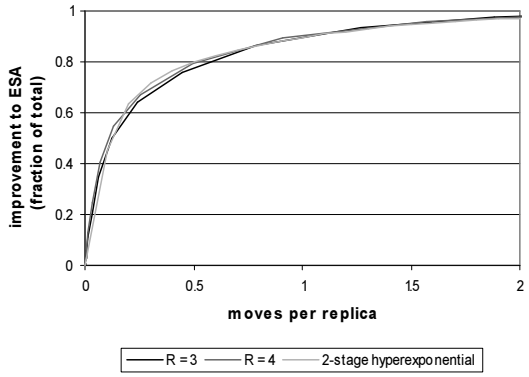

Figure 6. ESA improvement vs. relocations (MIN-RAND)

To explain the suitability of exponential fits to these algorithmic progressions, we offer the following conjectures: Aggressive hill climbing algorithms, such as MIN-MAX, tend to follow decay curves because they make the best available improvement at each step, and over time, the quality of remaining available improvements decreases as the good ones are used up. These decay curves tend to be exponential because each increment affects a fixed-size fraction of the system (i.e., one pair of files), so at each step, the realized improvement is proportional to the remaining potential improvement.

The aggressiveness of iterative improvement is substantially reduced by swapping the minimum-availability file with a random file (MIN-RAND) instead of with the maximum-availability file (MIN-MAX), thereby introducing wide variation into the quality of improvement steps. Consequently, rather than increasing the time constant of the exponential progression, MIN-RAND retains the same exponential time constant as MIN-MAX (0.093) and adds a second exponential stage with an order-of-magnitude greater time constant (0.76). The mixing coefficient, α, describes the combination of time constants τ_k. To express this instead as the mixing of improvement rates, we find a coefficient β by solving the following equation:

$$\frac{1}{\alpha\,\tau_1 + (1-\alpha)\,\tau_2} = \beta\frac{1}{\tau_1} + (1-\beta)\frac{1}{\tau_2} \qquad (2)$$

The result, $\beta = 0.16$, can be interpreted as a binary approximation of the continuum of step quality resulting from random selection of one of the files to swap: Each step has a 0.16 probability of being good (improving with a time constant of 0.093) and a 0.84 probability of being mediocre (improving with a time constant of 0.76).

The RAND-RAND progression does not appear to follow a hyperexponential at all. Instead, it appears nearly linear to around 1.5 moves per replica before it starts tapering off. Since the replica relocations that most affect the effective system availability are those of low-availability files, it is likely that many of the RAND-RAND swaps barely affect the ESA. As a consequence, the improvement per step is not heavily dependent upon the number of improvements made up to that point.

6. Dynamic behavior

The results in Section 5 showed significant speed differences among the algorithms. In the present section, we seek to understand the cause of these differences by examining the algorithms' dynamic behavior. Each swap changes the availabilities of the two files that exchange replica locations. For every change to a file's availability in the placement progression, we record the availability of the file before it is changed (the *source availability* A_S) and the availability of the file after it is changed (the *target availability* A_T). We then plot the target availability versus the source availability of every change, as a density plot on a two-dimensional grid (e.g., Figure 7).

We divide the density plot into quadrants in a non-standard way: The axes are rotated by 45° from the Cartesian axes, and they intersect at a point whose abscissa and ordinate both equal the mean file availability A_μ. We refer to the axis for which the source and target availabilities are identical as the *identity axis*; points along this axis indicate zero change to a file's availability. We refer to the other axis as the *complementary axis*; points along this axis indicate a change that maintains the magnitude but reverses the sign of the difference between a file's availability and the mean file availability.

The utility of a change can be evaluated by comparing the absolute difference between the mean file availability and the file's availability before the change to the absolute difference between the mean file availability and the file's availability after the change:

$$U = \left|A_S - A_\mu\right| - \left|A_T - A_\mu\right| \qquad (3)$$

The utility of a change is high if it succeeds in bringing the availability of the file significantly closer to the mean file availability. The utility is negative if it takes the availability of the file away from the mean file availability. A file can undergo a negative utility change when one of its replicas is swapped with a replica of another file that incurs a positive utility change of greater magnitude. Availability changes in the left and right quadrants have positive utility, and those in the upper and lower quadrants have negative utility.

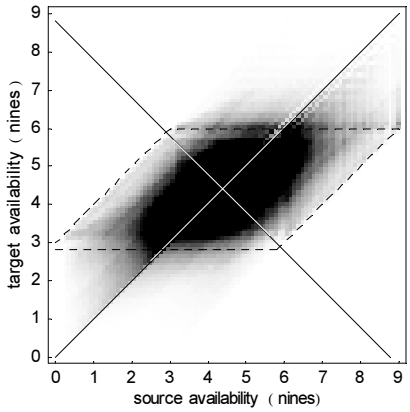

Figure 7. File availability change density (RAND-RAND, R = 3)

Figure 7 is a density plot of file availability changes during the execution of the RAND-RAND algorithm for $R = 3$. The plot for $R = 4$ (not shown) is substantially similar. The vast majority of points are confined within the dashed parallelogram. Exceeding the horizontal bounds $A_T = 2.8$ or $A_T = 6.0$ requires an extremely low or high target availability, which requires an even more extreme source availability, either of the file itself (in the left and right quadrants) or of the file with which it is swapped (in the upper and lower quadrants). Such extreme files are rare.

No points lie outside the diagonal bounds $A_T = A_S + 3.0$ and $A_T = A_S - 3.0$. It is not possible to change a file's availability by more than 3 nines in a single swap, because the replica availability range is only 3 nines. Since files are selected at random, few swaps are between extreme files, so most availability changes are within two nines.

Figure 8 is a density plot of file availability changes during the execution of the MIN-RAND algorithm for $R = 3$. The plot for $R = 4$ (not shown) is substantially similar. The dashed parallelogram is the same as that in Figure 7, but the dark region is nearer to the diagonal bounds, indicating a wider range of availability changes, since the mean availability distance between a replica of the minimum file and a replica of a random file is greater than that between replicas of two random files.

Figure 8 differs from Figure 7 in several ways. The right half of the upper quadrant contains no points; these would indicate improvements to files with above-mean availabilities, but MIN-RAND only improves the minimum-availability file. Few points in the right quadrant lie near the identity axis beyond 6 nines; these would indicate small decreases to files with above-mean availability, but this requires swapping replicas with similar availability values, which are unlikely in a minimum-availability file. There are more points below $A_T = 2.8$ near the identity axis, indicating swaps between two low-availability files, which are more likely with MIN-RAND than RAND-RAND, since one of the swapped files always has minimum availability.

Figure 9 is a density plot of file availability changes during the execution of the MIN-MAX algorithm for $R = 3$. The dashed parallelogram is the same as that in Figure 7. The dark region meets the diagonal bounds, indicating the full range of possible availability changes, because the minimum- and maximum-availability files are likely to have replicas with extremal availability values.

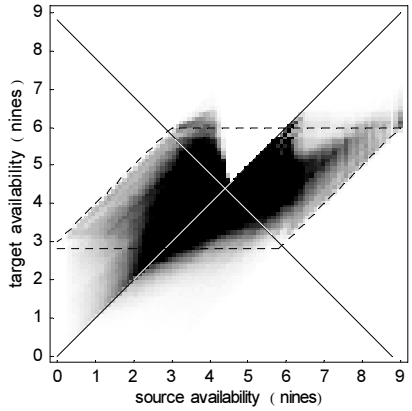

Figure 8. File availability change density (MIN-RAND, $R = 3$)

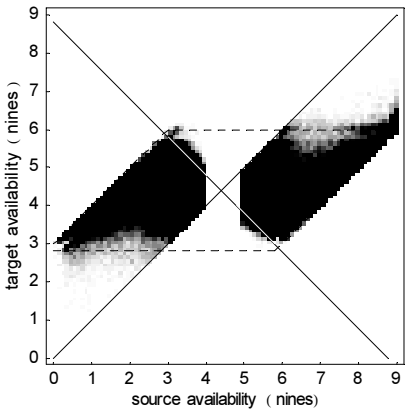

Figure 9. File availability change density (MIN-MAX, $R = 3$)

There are few points in the upper and lower quadrants, and all such points lie near the complementary axis. A replica swap between the minimum- and maximum-availability files can only increase the former and decrease the latter, and it is rare that this increase or decrease is sufficient to widen the absolute difference between either file's availability and the mean file availability

There are no points between source availabilities of 3.9 and 4.9 nines, since for $R = 3$, MIN-MAX freezes at a local minimum, where all files in the 2 % selection range have availabilities beyond these values.

Figure 10 is a density plot of file availability changes during the execution of the MIN-MAX algorithm for $R = 4$. Aside from several differences in scale, this plot is similar to that for $R = 3$, with the exception of points near the origin (5.9, 5.9) of the plot, since MIN-MAX does not freeze at a noticeably sub-optimal local minimum when $R = 4$.

In summary, MIN-MAX achieves its high rate of availability improvement by avoiding relocations that have negative utility. For the other two algorithms, one quarter of all relocations have negative utility, which retards the algorithms' improvements to effective system availability.

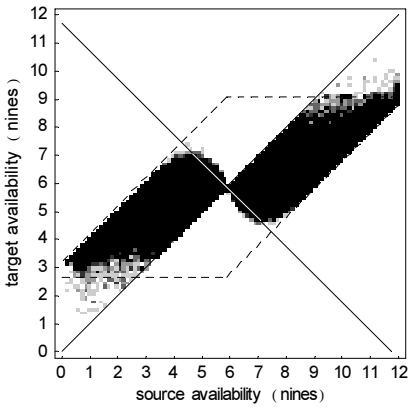

Figure 10. File availability change density (MIN-MAX, $R = 4$)

7. Placement details

The results in Section 5 showed minor differences among the algorithms with regard to the final ESA each could achieve. In the present section, we probe further by examining the distribution of file availabilities and of the replicas that compose each file. We also compare the placements achieved by each algorithm to worst-case placements found through theoretic analysis [13]. The metric used for determining the theoretic worst case is the availability of the minimum file, rather than the effective system availability. Our definition of the latter (Equation 1) assumes that file accesses are not correlated to file availability, but if low-availability files happen to be very popular, then the minimum file availability at least places a hard lower bound on the true system availability.

Theoretic placements use the uniform approximation of machine availability illustrated in Figure 1. We illustrate worst-case placements of 50 files, since this is the count of files per machine in our simulation.

For the replicas, the vertical axis is quantized in steps of size 0.1 nines to aggregate files with similar replica breakdowns. Files are sorted horizontally in increasing order of replica availability, from the most available replica (0) to the least available replica ($R - 1$). Densities are shown cumulatively, so the frequency of a particular configuration can be found by drawing vertical lines at each end of a region and taking the difference between the cumulative density values of the two points. For example, in Figure 11, there is a region (about 3 millimeters wide) between cumulative densities 0.45 and 0.50, which shows file availability of 4.4 and replica availabilities of 2.6, 1.6, and 0.1 (an inexact sum since the vertical axis is quantized for replicas but not for files), meaning 5 % (= 0.50 – 0.45) of all files have this particular replica availability makeup.

Figure 11 illustrates the placement achieved by RAND-RAND when $R = 3$. The placement for $R = 4$ (not shown) is substantially similar. When the algorithm freezes, all file availabilities are tightly distributed in the range 4.4 to 4.5 nines. Figure 12 shows a theoretic worst-case placement for RAND-RAND, in which the minimum file availability (at the left edge of the graph) is 4.0 nines. At this point, no replica exchange between any pair of files reduces the absolute availability difference between the files.

Figure 13 illustrates the placement achieved by MIN-RAND when $R = 3$. The placement for $R = 4$ (not shown) is substantially similar. When the algorithm freezes, the minimum file availability is 4.4 nines, and 98 % of files have availabilities no greater than 4.5 nines. The tails protruding upward from the main mass of file availabilities are a consequence of MIN-RAND's attention to low-availability files over high-availability files. Figure 14 shows a theoretic worst-case placement for MIN-RAND, in which the minimum file availability (at the right edge of the graph) is 3.7 nines. This is a far more complex placement than the worst-case placement for RAND-RAND.

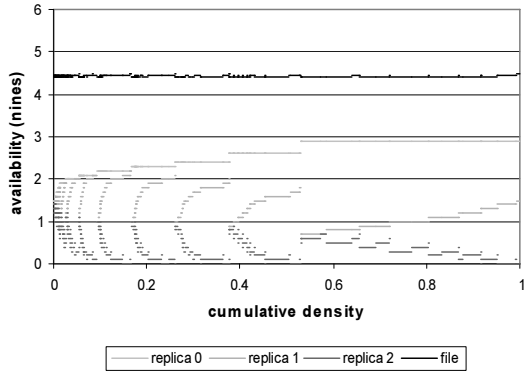

Figure 11. Simulated replica placement (RAND-RAND, $R = 3$)

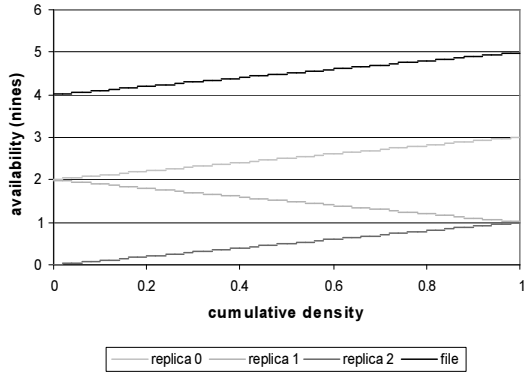

Figure 12. Worst-case replica placement (RAND-RAND, $R = 3$)

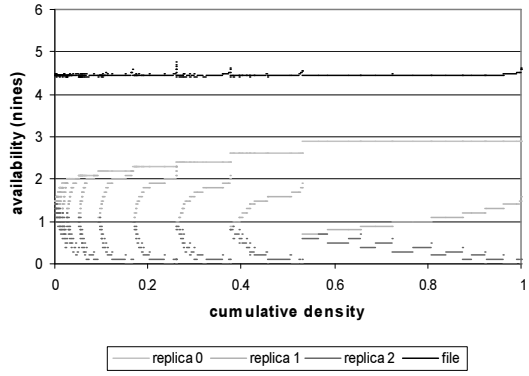

Figure 13. Simulated replica placement (MIN-RAND, $R = 3$)

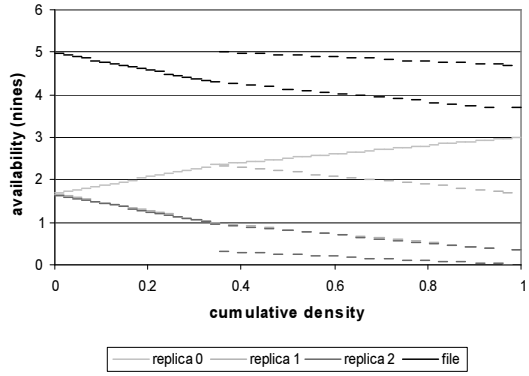

Figure 14. Worst-case replica placement (MIN-RAND, $R = 3$)

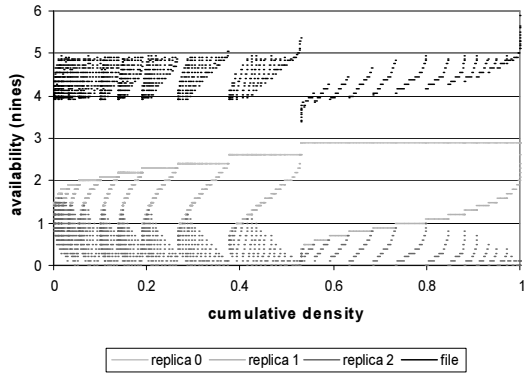

Figure 15. Simulated replica placement (MIN-MAX, *R* = 3)

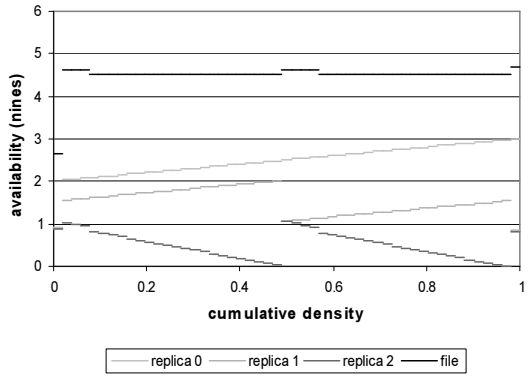

Figure 16. Worst-case replica placement (MIN-MAX, *R* = 3)

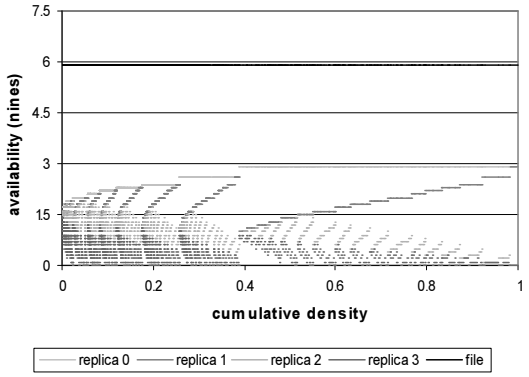

Figure 17. Simulated replica placement (MIN-MAX, *R* = 4)

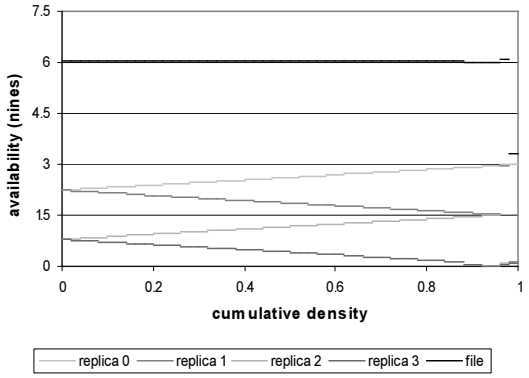

Figure 18. Worst-case replica placement (MIN-MAX, *R* = 4)

Figure 15 illustrates the file placement achieved by MIN-MAX when $R = 3$, which is dramatically different than the placements for the other algorithms. When the algorithm freezes, file availabilities are distributed throughout the wide range from 3.9 to 4.9 nines, but 2 % of files have availabilities above this range, and another 2 % are below. None of the files within the 2 % upper selection range can be swapped with any of the files in the 2 % lower selection range to decrease their absolute availability difference. The minimum file availability is 3.4 (and the maximum is 5.9). Figure 16 shows a theoretic worst-case placement for MIN-MAX when $R = 3$, in which the minimum file availability (at the left edge of the graph) is 2.7 nines.

Figure 17 illustrates the file placement achieved by MIN-MAX when $R = 4$, which appears more similar to the other algorithms than it does to MIN-MAX when $R = 3$. When the algorithm freezes, all file availabilities equal 5.9, which is extremely tight. Figure 18 shows a theoretic worst-case placement for MIN-MAX when $R = 4$, in which the minimum file availability (at the right edge of the graph) is 3.3.

In summary, MIN-MAX has a bad worst-case placement, and for $R = 3$, simulation yields a weak placement when judged by the metric of minimum file availability, which is important if the assumption of uniform access patterns turns out to be significantly incorrect. The worst case for MIN-RAND is significantly better, and that for RAND-RAND is better still. Both of these algorithms achieve good availability distributions under simulation.

8. Machine failure correlation

To assess our assumption that machine downtimes are sufficiently uncorrelated to calculate file availability as the sum of replica availabilities, we periodically interrupt the transient progressions of Figures 3 and 4 to calculate the actual downtime of each file, based on the vectors of ping responses of the machines holding replicas of the file. The negative decimal logarithm of the mean file downtime is the actual ESA of the placement. Figure 19 plots this actual ESA versus the ESA as estimated from summing replica availabilities. The values match within 3 % up to 5 nines but diverge by nearly 10 % for larger values.

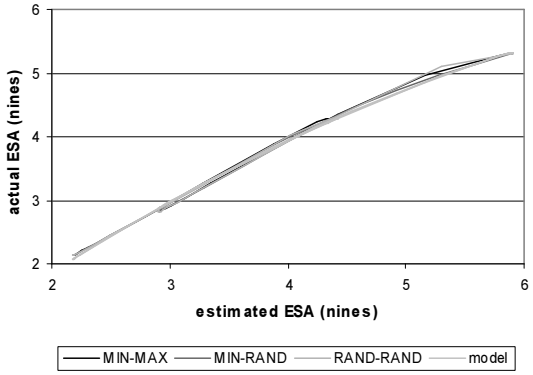

Figure 19. Actual calculated ESA vs. linearly estimated ESA

Figure 19 also shows a second-order polynomial model of the relationship between actual and estimated ESA, as determined by a least-squares fit. The fitted equation is:

$$\text{ESA}' = -0.9 \cdot \text{ESA}^2 + 1.5 \cdot \text{ESA} - 0.84 \quad (4)$$

These results suggest that $R = 4$ begins to exceed the limit of machine failure independence.

9. Related work

In the Farsite environment, the main cause of machine unavailability is users' turning off their machines [6], rather than machine failures or network partitions. In systems built of dedicated components, understanding the causes of machine failure [22] can be used in component-based reliability analysis [34] to evaluate a distributed system's availability. Different analysis strategies [4, 28] are appropriate for systems prone to network partitions, where results [29] indicate that replication cannot improve file availability by more than $\log_{10}(1/2) \approx 0.3$ nines. System availability is limited by highly correlated failures such as site-wide power outages, but techniques for dealing with such issues [19] are orthogonal to our work.

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, 33] rather than availability. Others addressed availability [25] but not automated replica placement. A significant body of work concerns file migration [8, 15, 24, 31], 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 [32], which provide high availability and reliability through distributed RAID rather than full replication. Archival Intermemory [18] and OceanStore [23] 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 [12], theoretic analysis using an analytic model of machine availability [13], and an exploration of systems issues [14].

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 monitors machine availability and places file replicas to maximize effective system availability, using a distributed hill-climbing algorithm that successively exchanges the machine locations of two file replicas. Large-scale simulation of three candidate placement algorithms using machine availability data from over 50,000 desktop computers yields the results summarized in Table 1. The theoretic results are for three file replicas, and simulation results are for the worse of three or four replicas.

When viewed from the perspective of algorithmic efficiency, MIN-MAX is the best of these algorithms: It improves the effective system availability with a progress half-life of 0.06 moves per replica, in contrast to the 0.12 of MIN-RAND or the 1.1 of RAND-RAND. MIN-MAX achieves this high rate of progress through efficient replica relocation: 99 % of all relocations have positive utility, and the mean relocation utility (Eq. 3) is 0.37. By contrast, only 77 % of MIN-RAND relocations and 72 % of RAND-RAND relocations have positive utility, and their mean relocation utility values are 0.16 and 0.13, respectively.

However, if judged by placement efficacy, RAND-RAND wins, and MIN-RAND is a close second. Simulations show that all three algorithms achieve good values of effective system availability, but when $R = 3$, MIN-MAX results in a minimum file availability that is only 3/4 of the mean, whereas RAND-RAND and MIN-RAND have minimum file availabilities very close to the mean. Competitive analysis (for $R = 3$) [12] shows that RAND-RAND and MIN-RAND are equivalently competitive in the general case, whereas MIN-MAX is not competitive. Theory also shows [13] that with a model of actual machine availability (Figure 1), RAND-RAND finds slightly better worst-case placements than MIN-RAND, which is significantly better than MIN-MAX.

Table 1. Summary of results

Metric	RAND-RAND	MIN-RAND	MIN-MAX	Reference
ESA progress half-life (moves per replica)	$\tau \approx 1.1$	$\tau \approx 0.12$	$\tau \approx 0.06$	§ 5
Mean utility of replica relocation	$U_\mu \approx 0.13$	$U_\mu \approx 0.16$	$U_\mu \approx 0.37$	§ 6
Percentage of relocations with positive utility	72 %	77 %	99 %	§ 6
Minimum file availability competitive ratio	$\rho \approx 0.99$	$\rho \approx 0.99$	$\rho \approx 0.77$	§ 7
theoretic (uniform machine availability)	$\rho = 8/9$	$\rho = 22/27$	$\rho = 1/2$	[13]
theoretic (general machine availability)	$\rho = 2/3$	$\rho = 2/3$	$\rho = 0$	[12]

The results of the studies presented in this paper show that MIN-RAND provides a reasonable trade-off between efficiency and efficacy, and if our system were constrained to use a single algorithm, this one appears to be the best choice. A perhaps better alternative is to use a combination of algorithms: using MIN-MAX unless and until it freezes at a noticeably sub-optimal local minimum and then using MIN-RAND for further refinement if necessary.

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, "Competitive Hill-Climbing Strategies for Replica Placement in a Distributed File System", Microsoft Research Technical Report *MSR-TR-2001-60*, Jun 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", (to appear) *ESA 2001*, Aug 2001.

[14] J. R. Douceur and R. P. Wattenhofer, "Optimizing File Availability in a Secure Serverless Distributed File System", (to appear) *20th SRDS*, IEEE, 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] B. Schneier. *Applied Cryptography*, 2nd Edition. John Wiley & Sons, 1996.

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

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

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

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