Competitive Hill-Climbing Strategies for Replica Placement in a Distributed File System

John Douceur and Roger Wattenhofer
{johndo,rogerwa}@microsoft.com

June 2001

Technical Report
MSR-TR-2001-60

The Farsite distributed file system stores multiple replicas of files on multiple machines, to provide file access even when some machines are unavailable. Farsite assigns file replicas to machines so as to maximally exploit the different degrees of availability of different machines, given an allowable replication factor R. We use competitive analysis and simulation to study the performance of three candidate hill-climbing replica placement strategies, `MinMax`, `MinRand`, and `RandRand`, each of which successively exchanges the locations of two file replicas. We show that the `MinRand` and `RandRand` strategies are perfectly competitive for $R = 2$ and 2/3-competitive for $R = 3$. For general R, `MinRand` is at least 1/2-competitive and `RandRand` is at least 10/17-competitive. The `MinMax` strategy is not competitive. Simulation results show better performance than the theoretic worst-case bounds.

To appear in the Proceedings of the 15th International Symposium on Distributed Computing (DISC), Lisbon, Portugal, October 2001.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1 Introduction

This paper analyzes algorithms for automated placement of file replicas in the Farsite [3] system, using both theory and simulation. In the Farsite distributed file system, multiple replicas of files are stored on multiple machines, so that files can be accessed even if some of the machines are down or inaccessible. The purpose of the placement algorithm is to determine an assignment of file replicas to machines that maximally exploits the availability provided by machines.

The file placement algorithm is given a fixed value, R, for the number of replicas of each file. For systems reasons, we are most interested in a value of $R = 3$ [9]. However, to ensure that our results are not excessively sensitive to the file replication factor, we also provide tight bounds for $R = 2$ and lower bounds for all R (tight at different values of R).

Our theoretic investigations cover an arbitrary distribution of machine availabilities and show worst-case behavior for a slightly abstracted model of the problem. For these studies, we assume an adversary that can establish – and continuously change – the availability characteristics for all machines, and we assess the ability of our algorithms to maximize the minimum file availability relative to the optimally achievable minimum file availability, for any given assignment of machine availability values. We do not attempt to classify the computational complexity of the problem, because it is not a classic input-output algorithm.

Our simulations are driven by actual measurements of machine availability [9] and show average-case behavior for a specific set of real-world measurements. For these studies, we consider not only the minimum file availability but also the distribution of file availability values. In all cases, we use a logarithmic measure for machine and file availability values, in part because of its standard usage [15] and computational convenience, but also because a linear measure understates the differences between results, since for all algorithms the minimum-availability file is available for a fraction of time that is very close to unity.

The next section describes the Farsite system and provides some motivation for why file replica placement is an important problem. Section 3 describes the algorithms, followed by a summary of results in Section 4. Section 5 presents a simplified theoretic model of the Farsite system environment, which is used in Section 6 to analyze the performance of the algorithms. Section 7 describes the environment for our simulations, the results of which are detailed in Section 8. Related work is discussed in Section 9.

2 Background

Farsite [3] 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 all of the client computers that participate in the system. Farsite runs on a networked collection of ordinary desktop computers in a large corporation or university, without interfering with users' local tasks, and without requiring users to modify their behavior in any way. As such, it needs to provide a high degree of security and fault tolerance without benefit of the physical protection and continuous support enjoyed by centralized server machines.

There are four properties that Farsite provides for the files that it stores: privacy, integrity, reliability, and availability. Data privacy is afforded by symmetric-key and public-key encryption, and data integrity is afforded by one-way hash functions and digital signatures. Reliability, in the sense of 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 replicas of each file on different machines. However, whereas the probability of permanent data-loss failure is assumed to be identical for all machines, the probability of transitory unavailability (such as a machine's being powered off temporarily) is demonstrably not identical for all machines. A five-week series of hourly measurements of more than 50,000 desktop machines at Microsoft [9] has shown that (1) machine availabilities vary dramatically from machine to machine, (2) the measured availability of each machine is reasonably consistent from week to week, and (3) the times at which different machines are unavailable are not significantly correlated with each other.

A file is not available if all the machines that store the replicas of the file are temporarily down. Given uncorrelated machine downtimes, the fraction of time that a file is unavailable is equal to the product of the fractional downtimes of the machines that store replicas of that file. We express availability as the negative logarithm of fractional downtime, and then the availability of a file is equal to the sum of the availabilities of the machines that store the file's replicas. The goal of a file placement algorithm is to produce an assignment of file replicas to machines that maximizes the minimum file availability over all files without exceeding the available space on any machine.

Measurements of over 10,000 file systems on desktop computers at Microsoft [3] indicate that machines experience permanent data-loss failures (e.g. disk head crashes) in a temporally uncorrelated fashion. We do not allow the algorithm to vary the number of replicas on a per-file basis, because this would introduce variance into the distribution of file reliability. The measurements show that a value of $R = 3$ is achievable in a real-world setting [9].

3 Algorithms

To be suitable for a distributed file system, a replica placement algorithm must satisfy two essential requirements: It must be incremental and distributable. Because the system environment is constantly changing, the algorithm must be able to improve an existing placement iteratively, rather than requiring a complete re-allocation of storage resources when a file is created or deleted, when a machine arrives or departs, or when a machine's availability changes. Because the file system is distributed, the algorithm must scale with the size of the system and must operate by making small changes of strictly local scope. To satisfy these requirements, we concentrate on hill-climbing algorithms, in particular those that perform an ordered succession of swap operations, in which the machine locations of two file replicas are exchanged.

Specifically, we investigate the properties of three algorithms: (1) `MinMax`, in which the only allowed replica-location swaps are between the file with the minimum availability and the file with the maximum availability, (2) `MinRand`, in which swaps are allowed only between the file with the minimum availability and any other file, and (3) `RandRand`, in which swaps are allowed between any pair of files. In general, file replicas are swapped between machines only if the swap reduces the absolute difference between the availabilities of the files and only if there is sufficient free space on each machine to accept the replicas that are being relocated. If there is more than one successful swap for two given files, our algorithm chooses one with minimum absolute difference between the file availabilities after the swap. However, for our theoretical worst-case analysis, this does not matter.

The intuition behind these algorithms is as follows: `RandRand` is the most general swap-based strategy, in that it allows swaps between any pair of files, so it represents a baseline against which to compare and contrast the other algorithms. We are most concerned with improving the minimum file availability, and since a replica exchange only affects the two files whose replica locations are swapped, it makes sense for one of these files to be the one with minimum availability, hence `MinRand`. The motivation behind `MinMax` is that the maximum availability file seems likely to afford the most opportunity for improving the minimum availability file without excessively decreasing its own availability.

In actual practice, the file placement algorithm executes in a distributed fashion, wherein the files are partitioned into disjoint sets, and each set is managed by an autonomous group of a few machines. At each iterative step, one of the groups contacts another group (possibly itself), each of the two groups selects one of the files it manages, and the groups jointly determine whether to exchange machine locations of one of the replicas of each file. Therefore, the `MinMax` and `MinRand` algorithms are not guaranteed to select files with globally extremal availability values. For our theoretic analyses, we concentrate on the more restrictive case in which only extremal files are selected. For our simulation studies, we model this extremal discrepancy by selecting from a range of files with the highest or lowest availability rank.

4 Summary of Results

In this paper we perform a worst-case analysis and a simulation to determine the efficacy of three hill-climbing algorithms, where the efficacy of an algorithm is specified by the availability of a file with minimum availability. We denote the efficacy of an algorithm by its competitive ratio $\rho = m/m^*$, where m is the efficacy of the hill-climbing algorithm, and m^* is the efficacy of an optimal algorithm. We show – for both theory and simulation – that the `MinRand` algorithm performs (almost) as well as the `RandRand` algorithm. The `MinMax` algorithm performs poorly throughout. Here is a detailed summary of our results:

Algorithm	`MinMax`	`MinRand`	`RandRand`
Worst-case $R = 3$	$\rho = 0$ (Thm. 6.5)	$\rho = 2/3$ (Thm. 6.3)	$\rho = 2/3$ (Thm. 6.3)
Simulated $R = 3$	$\rho \approx 0.74$ (Fig. 2)	$\rho \approx 0.93$ (Fig. 2)	$\rho \approx 0.91$ (Fig. 2)
Worst-case $R = 2$	$\rho = 0$ (Thm. 6.5)	$\rho = 1$ (Thm. 6.4)	$\rho = 1$ (Thm. 6.4)
Lower bounds any R	$\rho = 0$ (Thm. 6.5)	$\rho > 1/2$ (Thm. 6.8)	$\rho \geq 10/17$ (Thm. 6.10)

For $R = 3$ and machine availabilities as measured in [9] an optimal algorithm will assign replicas to machines such that the least available file has an successful access probability of .99997. A competitive ratio of $\rho = 2/3$ means that our algorithm achieves a successful access probability of .999 for the least available file.

5 Theoretic Model

We are given a set of N unit-size files, each of which has R replicas. We are also given a set of $M = N \cdot R$ machines, each of which has the capacity to store a single file replica. Machines have *availabilities* $a_i \geq 0$, $i = 1, \ldots, M$, given as negative logarithms of machines' downtimes.

Let the R replicas of file f be stored on machines with availabilities $a_1^f, \ldots, a_R^f$. To avoid notational clutter, we overload a variable to name a file and to give the availability value of the file. Thus, the *availability* of file f is $f = a_1^f + \cdots + a_R^f$.

Let m be a file with minimum availability when the algorithm has exhausted all possible improvements. Let m^* be a file with minimum availability given an optimal placement for the same values of N, R, and a_i ($i = 1, \ldots, M$). We compute the ratio $\rho = \min m/m^*$ over all allowable a_i as $N \to \infty$. We say that the algorithm is ρ-competitive.

In the practical algorithms, the particle "`Rand`" stands for a random choice, i.e. the `MinRand` algorithm tries to exchange machines between minimum-availability file m and a *randomly* chosen file f. In the theoretical analysis however, "`Rand`" is treated as "`Any`", i.e. the `MinRand` algorithm tries to exchange machines between the minimum-availability file m and *any* other file f. The algorithm stops ("freezes") only after all legal pairs of files have been tested. We use the particle "`Rand`" rather than "`Any`" to have a consistent terminology to the simulation part of this work.

If two or more files have minimum availability, or if two or more files have maximum availability, we allow an adversary to choose which of the files can be swapped.

The number of possible machine exchanges between two given files grows exponentially with the number of replicas R. In this paper we do not study this problem. We are predominantly interested in systems where R is small; for large R we will give an recipe linear in R that finds machines to be exchanged (see Lemma 6.6).

Note that the set of legal pairs of files for the `MinRand` algorithm is a subset of the set of legal pairs of files for the `RandRand` algorithm. That is, if the `MinRand` algorithm freezes, it is possible that the `RandRand` algorithm would still find a successful exchange. A freeze of the `RandRand` algorithm however also implies that the `MinRand` algorithm would not find a successful exchange. Similarly, the singleton set of legal pairs for `MinMax` is a subset of the legal pairs for `MinRand`. Thus, the efficacy of the `RandRand` (`MinRand`) algorithm is at least as high as the efficacy of the `MinRand` (`MinMax`) algorithm. Formally,

Lemma 5.1 $\rho_{\texttt{MinMax}} \leq \rho_{\texttt{MinRand}} \leq \rho_{\texttt{RandRand}}$.

With Lemma 5.1 we are in the position to find the competitive ratio of different algorithms by simply giving a worst-case example for the stronger algorithm (e.g. `RandRand`), and a qualitative argument on the weaker algorithm (e.g. `MinRand`). When discussing the competitive ratio of an algorithm with a specific replication factor, we append the replication factor R to the algorithm name, i.e. the competitive ratio of the `MinRand` algorithm for replication factor 3 is $\rho_{\texttt{MinRand3}}$.

If possible we simplify the arguments by linearly scaling the machine availabilities such that $m = 1$ throughout this paper. Note that this does not change the competitive ratio ρ.

6 Competitive Analysis

We start with $R = 3$, the case we are most interested in.

Lemma 6.1 $\rho_{\texttt{MinRand}3} \geq 2/3$.

Proof: The intuition of the proof is as follows: We define a non-decreasing function g, the argument of g is a non-negative real (a machine availability), and g returns a real. We define $G := \sum_{a \in A} g(a)$, where the set A is the availabilities of all machines.

In the first part of the proof we show that the `MinRand` algorithm freezes with $G < N$. In the second part of the proof we consider an optimal assignment of machines to files. If all files in the optimal assigment have availability strictly greater than $3/2$, we show that $G \geq N$. Since $N \leq G < N$ is a contradiction, the minimum file of the optimal assignment has availability $m^* \leq 3/2$. With $m = 1$ the proof will follow.

Here are the details: Let $m = a_1 + a_2 + a_3$ be the file with minimum availability, with $a_1 \geq a_2 \geq a_3$. Of particular interest is a_3, the minimum-availability machine of minimum file m. Let $g(a)$ be a function applied on the availability a of a machine, with

$$g(a) = \begin{cases} 1 & \text{if } a > 1 - a_3 \\ 1/2 & \text{if } 1/2 < a \leq 1 - a_3 \\ 1/4 & \text{if } a_3 < a \leq 1/2 \\ 0 & \text{if } a \leq a_3 \end{cases}$$

The function $g(f)$ applied to file f is simply $g(f) = g(b_1) + g(b_2) + g(b_3)$, if file $f = b_1 + b_2 + b_3$.

Let $f = b_1 + b_2 + b_3$ be another file with availability $f > 1$, and with $b_1 \geq b_2 \geq b_3$. We assume that there is no successful machine exchange between the files f and m. We denote the file f (m) after an exchange with f' (m'). Note that $m' > 1$ and $f' > 1$ would contradict the assumption that the `MinRand` algorithm freezed, since $f \geq m = 1$.

We distinguish several cases.

Case 1: Let $b_1 > 1 - a_3$. If $b_2 > a_3$ we can exchange the machines b_2 and a_3, such that $f' \geq b_1 + a_3 > (1-a_3)+a_3 = 1$, and $m' = m+b_2-a_3 > 1$. Therefore $b_2 \leq a_3$. Thus $g(f) = g(b_1)+g(b_2)+g(b_3) = 1+0+0 = 1$.

In all other cases we therefore have $b_1 \leq 1 - a_3$.

Case 2: If $b_3 \leq a_3$, then $g(f) \leq 1/2 + 1/2 + 0 = 1$.

In all other cases we have $b_3 > a_3$.

Case 3: Let $b_2 > 1/2$. Since $b_3 > a_3$ we can exchange the machines b_3 and a_3, such that $f' \geq b_1 + b_2 > 1$, and $m' = m + b_3 - a_3 > 1$, which is a contradiction to the assumption that there was no successful exchange. Therefore we have $b_2 \leq 1/2$, and thus $g(f) \leq 1/2 + 1/4 + 1/4 = 1$.

So far we have shown that $g(f) \leq 1$ for each file. A simple case study reveals that the minimum file m itself has $g(m) \leq 3/4$. Since each machine is part of exactly one file we have $G = \sum_{f \in F} g(f)$, where F is the set of all files. Since $|F| = N$ we can conclude

$$G = \sum_{f \in F} g(f) \leq (N-1) \cdot 1 + 3/4 < N.$$

In the second part of the proof we will show that if a file f has a sufficiently high availability, then $g(f)$ will be at least 1. Specifially, we will show that for any file $f = b_1 + b_2 + b_3$ (with $b_1 \geq b_2 \geq b_3$) we have

$$f > 3/2 \Rightarrow g(f) \geq 1.$$

We distinguish two cases:

Case 1: If $b_1 > 1 - a_3$ or $b_2 > 1/2$ then $g(f) \geq 1$, because $g(b_1) \geq 1$ or $g(b_1) + g(b_2) \geq 1$.

Case 2: We have $b_1 \leq 1 - a_3$ and $b_2 \leq 1/2$. If $b_1 \leq 1/2$, then $f = b_1 + b_2 + b_3 \leq 3/2$ (not satisfying the precondition that $f > 3/2$). Thus $1/2 < b_1 \leq 1 - a_3$ and $b_2 \leq 1/2$. We have $3/2 < f = b_1 + b_2 + b_3 \leq (1 - a_3) + 1/2 + b_3 \Rightarrow b_3 > a_3$. Then $g(f) = 1/2 + 1/4 + 1/4 = 1$.

In the second part of the proof we have shown that files f with availability $f > 3/2$ necessarily have $g(f) \geq 1$.

The optimal algorithm assigns files to machines such that the file with minimum availability is m^*. Suppose, for the sake of contradiction, that an optimal algorithm manages to raise the availability of each file f such that $f \geq m^* > 3/2$. With the second part of the proof we know that in this case $g(f) \geq 1$ for all N files. Since the function g of a file is defined as a sum of the function g of the machines of the file, we know that $G \geq N$. With the conclusion of the first part of the proof we get $N \leq G < N$ which is a contradiction. Therefore $m^* \leq 3/2$, and $\rho = m/m^* \geq 2/3$. ■

Lemma 6.2 $\rho_{\texttt{RandRand}3} \leq 2/3$.

Proof: We give a constructive proof for a worst-case example with the three files m, f_1, and f_2: The `RandRand` algorithm freezes with $m = 1 + 0 + 0$ (the minimum-availability file), $f_1 = 1 + 1 + 0$, and $f_2 = 1/2 + 1/2 + 1/2$, that is, no machine exchange between any two files decreases the difference of the availabilities of the two files. We have nine machines with availabilities 3×1, $3 \times 1/2$, and 3×0. An optimal algorithm generates three files $1 + 1/2 + 0 = 3/2$, thus $m^* = 3/2$. Therefore $\rho_{\texttt{RandRand3}} = m/m^* \leq 2/3$. ■

Theorem 6.3 $\rho_{\texttt{MinRand}3} = \rho_{\texttt{RandRand}3} = 2/3$.

Proof: The Theorem follows directly with the Lemmas 6.1, 6.2, and Lemma 5.1. ■

For replication factor 2 the `MinRand` algorithm is optimal:

Theorem 6.4 $\rho_{\texttt{MinRand}2} = \rho_{\texttt{RandRand}2} = 1$.

Proof: The proof is a "light" version of the proof of Lemma 6.1, and the details are omitted in this extended abstract. The function g is defined as

$$g(a) = \begin{cases} 1 & \text{if } a > 1 - a_2 \\ 1/2 & \text{if } a_2 < a \leq 1 - a_2 \\ 0 & \text{if } a \leq a_2 \end{cases}$$

■

The `MinMax` algorithm performs poorly in general, as the following example shows.

Theorem 6.5 $\rho_{\texttt{MinMax}} = 0$.

Proof: We give a constructive proof for a worst-case example with (at least) three files: Let $m = 0 + 0 + 0 + \cdots + 0$ be the file with minimum availability (note that $m = 0$), and $f = 3 + 0 + 0 + \cdots + 0$ be the file with maximum availability, and all other files (at least one) have the machines $1 + 1 + 0 + \cdots + 0$. The `MinMax` algorithm freezes since there is no exchange between the files m and f. For $N \geq 3$ we have $2(N - 2) + 1 \geq N$ machines with availability at least 1, and it is possible to reassign the machines to files such that each file has at least one machine with availability at least 1, that is $m^* \geq 1$. Thus $\rho \leq 0/1 = 0$. ■

We want to be confident that our algorithms do not fail with larger replication factors. In the following we give bounds on the performance for arbitrary R. All our bounds are tight for some R: As seen above, the `MinMax` algorithm is non-competitive for any R. The `MinRand` algorithm is worst when $R \to \infty$, and the `RandRand` algorithm is worst when R is 7.

Lemma 6.6 *Let $m = a_1 + \ldots + a_R$ be a file with availability $m = 1$, and $a_1 \geq \ldots \geq a_R \geq 0$. Let $f = b_1 + \ldots + b_R > 1$ be another file, with $1 \geq b_1 \geq \ldots \geq b_R \geq 0$. Let $\bar{f}$ be f, but all the machines with availability less than a_R are replaced with machines with availability a_R, that is, $\bar{f} = \max(b_1, a_R) + \max(b_2, a_R) + \ldots + \max(b_R, a_R)$. If $\bar{f} > 2$ we can successfully exchange machines between m and f.*

Proof: Let l be the highest index such that $b_l > a_R$, that is, either $b_{l+1} \leq a_R$ or $l = R$. First we exchange the machines b_l and a_R, that is $m' = m + b_l - a_R > 1$ and $f' = f + a_R - b_l$. If $f' > 1$ we are done, because we found a way to exchange machines and both availabilities are strictly greater than one. In the remainder of the proof we need to consider the case where $f' \leq 1$ only.

As long as $m' > 1$ *and* $f' \leq 1$ we repeatedly exchange the machines a_{i+1} and b_{R-i}, for $i = 0, 1, \ldots$. We denote m' (f') after the ith exchange with m^i (f^i). More formally:

$$m^i = m' + \sum_{j=0}^{i} b_{R-j} - \sum_{j=0}^{i} a_{j+1} \text{ and } f^i = f' + \sum_{j=0}^{i} a_{j+1} - \sum_{j=0}^{i} b_{R-j}.$$

In the remainder of the proof we show that these repeated exchanges will eventually be successful, that is, there is a k (with $k \leq R - l - 1$) such that after k exchanges we have $m^k > m$ and $f^k > m$.

First, we show that the process of repeated exchanges *terminates*. In other words, there is a $k \leq R - l - 1$ such that either $m^k \leq 1$ *or* $f^k > 1$.

In particular, if $i = R - l - 1$, then

$$\begin{aligned} f^i &= f + a_R - b_l + \sum_{j=0}^{i} a_{i+1} - \sum_{j=0}^{i} b_{R-j} = \sum_{j=0}^{l-1} b_j + a_R + \sum_{j=0}^{R-l-1} a_{j+1} \\ &\geq \sum_{j=0}^{l-1} \max(b_j, a_R) + a_R + \sum_{j=l+1}^{R} \max(a_R, b_j) \text{ (using } b_l > a_R \leq a_j) \\ &= \bar{f} + a_R - b_l > 2 + a_R - b_l \geq 1. \text{ (using } b_l \leq 1) \end{aligned}$$

We have $f^i > 1$, and therefore the process of repeated exchanges terminates.

Since the process terminates after k exchanges, we have either $m^k \leq 1$ *or* $f^k > 1$.

We distinguish three cases.

Case 1: Let $m^k > 1$ and $f^k > 1$. We have found successful machine exchanges since $m^k > m$ ($m = 1$) and $f^k > m$. We are done.

Case 2: Let $m^k \leq 1$ and $f^k \leq 1$. This contradicts the assumptions that $m = 1$ and $f > 1$ because $2 \geq m' + f' = m + f > 2$.

The only remaining case is the most difficult.

Case 3: Let $m^k \leq 1$ and $f^k > 1$. Note that before the kth exchange it was decided to do another exchange, that is, $m^{k-1} > 1$ and $f^{k-1} \leq 1$.

The precondition of this Lemma is $\bar{f} = b_1 + \ldots + b_{l-1} + b_l + (R - l)a_R > 2$. For readibility we split $\bar{f}$ at "b_l" into two parts: $\bar{f} = \bar{f}_1 + \bar{f}_2$.

Then

$$\begin{aligned}
\bar{f}_1 &= b_1 + \ldots + b_{l-1} = f - b_l - \sum_{i=l+1}^{R} b_i = f' - a_R - \sum_{i=l+1}^{R} b_i \\
&= f' - a_R - \sum_{i=0}^{k-1} b_{R-i} - \sum_{i=l+1}^{R-k} b_i \le f' - \sum_{i=0}^{k-1} b_{R-i} - a_R \\
&= f^{k-1} - \sum_{i=0}^{k-1} a_{i+1} - a_R \le 1 - \sum_{i=0}^{k-1} a_{i+1} - a_R
\end{aligned}$$

Also we have

$$\begin{aligned}
\bar{f}_2 &= b_l + (R-l)a_R \le m + b_l - \sum_{i=1}^{l} a_i = m' + a_R - \sum_{i=1}^{l} a_i \\
&= m^k + \sum_{i=0}^{k} a_{i+1} - \sum_{i=0}^{k} b_{R-i} + a_R - \sum_{i=1}^{l} a_i \le 1 + a_R + \sum_{i=0}^{k} a_{i+1} - \sum_{i=1}^{l} a_i
\end{aligned}$$

Together we get

$$\begin{aligned}
\bar{f} &= \bar{f}_1 + \bar{f}_2 \le 1 - \sum_{i=0}^{k-1} a_{i+1} - a_R + 1 + a_R + \sum_{i=0}^{k} a_{i+1} - \sum_{i=1}^{l} a_i \\
&\le 2 + a_k - a_1 \le 2.
\end{aligned}$$

This contradicts with the precondition $\bar{f} > 2$.

Only case 1 did not contradict with the assumptions and preconditions. The Lemma follows immediately.
■

Lemma 6.7 $\rho_{\texttt{MinRand}} > 1/2$.

Proof: Let $m = a_1 + \ldots + a_R = 1$ be a file with minimum availability, with $a_1 \ge \ldots \ge a_R \ge 0$. Let $f = b_1 + \ldots + b_R > 1$ be another file, with $b_1 \ge \ldots \ge b_R \ge 0$, and assume that there is no successful machine exchange.

We distinguish two types of files f:

Type A: Let $b_1 > 1$. If $b_2 > a_R$ we can exchange the machines b_2 and a_R such that $m' = m + b_2 - a_R > 1$, $f' \ge b_1 > 1$. Therefore $b_2 \le a_R$.

Type B: $b_1 \le 1$.

Of the N files, one is the minimum file m, x files are of type A, $N-1-x$ files are of type B. We have exactly x machines with availability strictly greater than 1, that is, an optimal assignment will end up with at least $N-x$ files that can only use machines with availability 1 or less. Since type A files have $b_2 \le a_R$ we can at most replace the machines of the type B files that have availability less than a_R with machines that have availability a_R. With Lemma 6.6 we know that such an improved file of type B can at most have availability 2. An optimal algorithm can redistribute the files such that the total sum of availabilities is at most $G = 2(N-x-1)+1$. An optimal redistribution to $N-x$ files gives the new minimum file an availability of at most $m^* \le G/(N-x)$, thus $m^* < 2$. Therefore $\rho = m/m^* > 1/2$. ■

Theorem 6.8 $\lim_{R\to\infty} \rho_{\texttt{MinRand}R} = 1/2$.

Proof: For simplicity let R be a power of 2, that is $R = 2^r$. We construct the following files:

- The minimum file $m = 1/R + \ldots + 1/R = 1$.
- $R/2$ files of type 0 with $f = 2 + 1/R + 1/R + \ldots + 1/R$.

- For $i = 1, \dots, r-1$: 2^{r-i-1} files of type i with $f = 2^i \times 2^{-i} + 0 + \dots + 0$

Note that there is no successful exchange between the minimum file m and any other file. We have used the following machines:

- $R/2$ machines with availability 2,
- For $i = 1, \dots, r-1$: $R/2$ machines with availability 2^{-i},
- $R/2 \cdot (R-1) + R$ machines with availability $1/R$
- The rest of the machines have availability 0.

With the same machines we can build

- $R/2$ files of type A with $f = 2 + \text{rest}$, that is $f \geq 2$, and
- $R/2$ files of type B with $f = 1/2 + 1/4 + 1/8 + \dots + 4/R + 2/R + 1/R + 1/R + \dots + 1/R$, that is $f = 1 + (R - r - 1)/R$.

Since $\lim_{R\to\infty} 1 + (R - r - 1)/R = 2$ we have $m^* \to 2^-$, and therefore $\rho = m/m^* \to 1/2^+$. With Lemma 6.7, $\rho \to 1/2^+$ is tight and the Theorem follows. ■

Lemma 6.9 $\rho_{\texttt{RandRand}} \geq 10/17$.

Proof: We omit this tedious proof in the extended abstract. Apart from some complications it is similar to the proof of Lemma 6.1. The function g is defined as

$$g(a) = \begin{cases} 1 & \text{if } a > 1 \\ 1/2 & \text{if } 2/3 < a \leq 1 \\ a/(2-a) & \text{if } 1/2 < a \leq 2/3 \\ a/(1+a) & \text{if } a \leq 1/2 \end{cases}$$

■

Theorem 6.10 $\rho_{\texttt{RandRand}} = 10/17$, *when we allow R to be a non-integer. If R must be an integer then* $\rho_{\texttt{RandRand}} \leq 3/5$.

Proof: First we are going to make a relaxation to our normal model by assuming that $R = 6 + \epsilon$, where $\epsilon > 0$. (Remark: A non-integer R is an ethereal construct without physical realization; we need ϵ to go towards 0 such that we can prove a tight bound.)

Let $m = 1+0+\dots$. Additionally we have a files of type $1+1+0+\dots$, b files of type $1/2+1/2+1/2+0+\dots$, and c files with $6+\epsilon$ machines with availability $1/(5+\epsilon)$. Note that $(6+\epsilon)/(5+\epsilon) = 1+1/(5+\epsilon)$. The `RandRand` algorithm does not find any successful exchange between any pair of files. If we do the accounting, we find $2a+1$ machines with availability 1, $3b$ machines with availability $1/2$, $c(6+\epsilon)$ machines with availability $1/(5+\epsilon)$, and $(5+\epsilon) + a(4+\epsilon) + b(3+\epsilon)$ machines with availability 0.

An optimal algorithm assigns the machines so that it will get $1+a+b+c$ files of type $f = 1+1/2+1/(5+\epsilon)+0+\dots$ will have $1+a+b+c$ machines with availability $1, 1/2$, or $1/(5+\epsilon)$ each, and $(1+a+b+c)(3+\epsilon)$ machines with availability 0. This is possible if and only if the following equation system is solvable:

$$\begin{aligned} 2a+1 &= 1+a+b+c \text{ (for availibility 1)} \\ 3b &= 1+a+b+c \text{ (for availibility 1/2)} \\ c(6+\epsilon) &= 1+a+b+c \text{ (for availibility } 1/(5+\epsilon)) \\ (5+\epsilon)+a(4+\epsilon)+b(3+\epsilon) &= (1+a+b+c)(3+\epsilon) \text{ (for availibility 0)} \end{aligned}$$

We can solve this equation system with $a = 9/\epsilon + 1$, $b = 6/\epsilon + 1$, $c = 3/\epsilon$. Thus we have a worst case example that is tight with Lemma 6.9:

$$m^* = f = \lim_{\epsilon\to 0^+} 1 + 1/2 + 1/(5+\epsilon) = 17/10.$$

If R has to be integer we choose $\epsilon = 1$, and get $m^* = 1 + 1/2 + 1/6 = 5/3$. ■

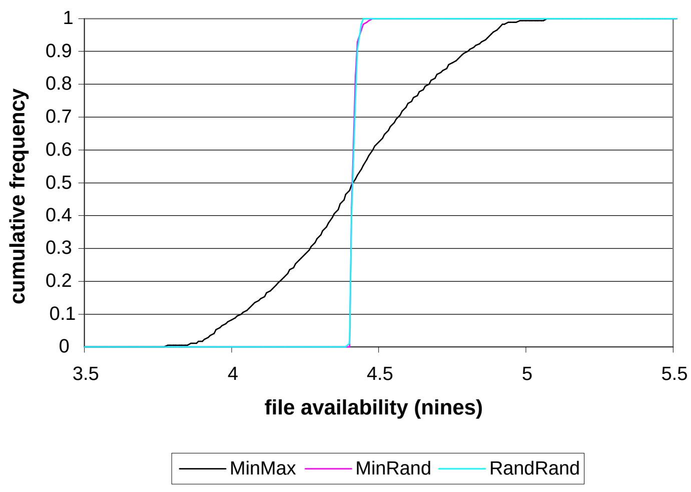

Figure 1: File availability distributions

7 Simulated environment

We are given a set of $M = 51,662$ machines with a measured distribution of availabilities in the range of 0.0 to 3.0 [9], calculated as the negative decimal logarithm of the machines' downtimes. (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 fraction uptime of $1 - 0.01 = 0.99$.) We are given a set of files whose sizes are governed by a binary lognormal distribution with $m(2) = 12.2$ and $s(2) = 3.43$ [7]. We simulate $N = 2,583,100$ files, averaging 50 files per machine, which runs at the memory limit of the 512-MB computer we use for simulation. 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. We fix the replication factor $R = 3$ [9].

At each step, a pair of files is selected randomly (uniform distribution). To model the fact that in a distributed environment the selection of files for swapping is done without global knowledge, we set a selection range for minimum and maximum availability files 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, uniformly at random.

8 Simulation results

We apply each of the algorithms to an initial random placement and run until the algorithm reaches a stable point. Fig. 1 shows file availability distributions for the three algorithms. The `MinMax` algorithm shows the widest variance, with an almost linear file availability distribution between 4 and 5 nines. The `RandRand` algorithm yields a much tighter distribution, and the `MinRand` distribution is almost exactly the same as that for `RandRand`, except for the upper tail.

To study how the minimum file availability varies, we apply each of the algorithms to 100 randomly selected subsets of 100 machines from the measured set of 51,662 machines. Fig. 2 shows a box plot [13] of the ratio of the minimum file availability to the mean file availability for the three algorithms. The "waist" in each box indicates the median value, the "shoulders" indicate the upper quartile, and the "hips" indicate

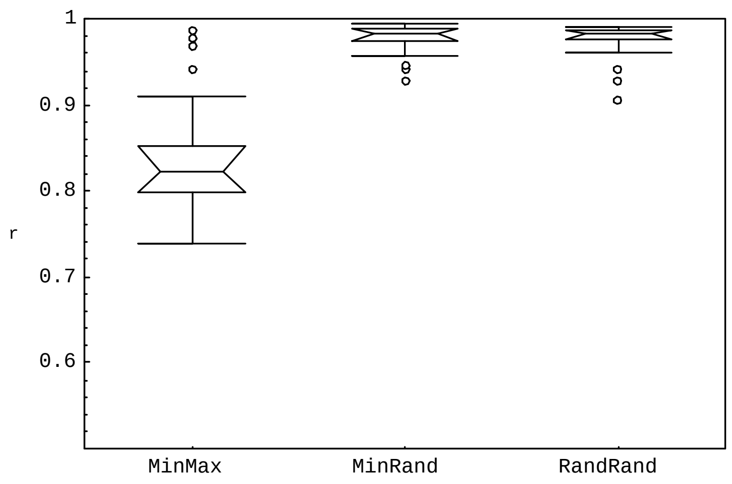

Figure 2: Minimum vs. Mean File Availability

the lower quartile. The vertical line from the top of the box extends to a horizontal bar indicating the maximum data value less than the upper cutoff, which is the upper quartile plus 3/2 the height of the box. Similarly, the line from the bottom of the box extends to a bar indicating the minimum data value greater than the lower cutoff, which is the lower quartile minus 3/2 the height of the box. Data outside the cutoffs is represented as points.

The worst-case ratio that our simulation encountered for the `MinMax` algorithm is 0.74, which is poor but significantly better than the value of 0 which our competitive analysis showed is possible. The worst-case ratio found for `MinRand` is 0.93, and the worst-case ratio found for `RandRand` is 0.91. These values are both better than the theoretic competitive ratio of 2/3 for the two algorithms. Note that the simulation ratios use mean file availability as the denominator, rather than the optimum minimum file availability, since the latter is not easily computable. Therefore, the ratios may be artificially lowered by the possibly incorrect assumption that the mean file availability is an achievable value for the availability of the minimum file.

9 Related Work

Other than Farsite, serverless distributed file systems include xFS [2] and Frangipani [18], both of which provide high availability and reliability through distributed RAID semantics, rather than through replication. Archival Intermemory [5] and OceanStore [16] both use erasure codes and widespread data distribution to avoid data loss. The Eternity Service [1] uses full replication to prevent loss even under organized attack, but does not address automated placement of data replicas. A number of peer-to-peer file sharing applications have been released recently: Napster [17] and Gnutella [10] provide services for finding files, but they do not explicitly replicate files nor determine the locations where files will be stored. Freenet [6] performs file migration to generate or relocate replicas near their points of usage.

To the best of our knowledge this is the first study of the availability of replicated files. We know of negative results of hill-climbing algorithms in other areas, such as clustering [11].

There is a common denominator of our work and the research area of approximation algorithms, especially in the domain of online approximation algorithms [14, 4] such as scheduling [12]. In online computing, an algorithm must decide how to act on incoming items without knowledge of the future. This is related to our work, in the sense that a distributed hill-climbing algorithm also makes decisions locally (without knowledge of the whole system), and where an adversary continously changes the parameters of the system (e.g. the availabilities of the machines) in order to damage a good assignment of replicas to machines.

Open Problems

There are a variety of questions we did not tackle in this paper. First, we focused on giving bounds for the efficacy of the three algorithms rather than efficiency. It is an interesting open problem how quickly the hill-climbing algorithms converge; both in a transient case (where we fix the availabilities of the machines and start with an arbitrary assignment of machines to files), and also in a steady-state case (where during an [infinite] execution of the algorithm an adversary with limited power can continuously change the availabilities of machines); see [8] for a simulation of the transient case and [9] for a simulation of the steady-state case.

It would also be interesting to drop some of the restrictions in this paper, in particular the simplification that each file has unit size.

Finally, it is an open problem whether there is another decentralized hill-climbing algorithm that has better efficacy and efficiency than the algorithms presented in this paper. For example, does it help if we considered exchanges between any group of three or four files? Or does it help to sometimes consider "downhill" exchanges too? In general, we would like to give lower bounds on the performance of *any* incremental and distributable algorithm. We feel that this area of research has a lot of challenging open problems, comparable with the depth and elegance of the area of online computation.

Acknowledgements

We would like to thank Peter Widmayer and the anonymous reviewers for their useful comments on the paper.

References

[1] Ross Anderson. The eternity service. *Proceedings of Pragocrypt*, 1996.

[2] Thomas E. Anderson, Michael Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.

[3] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems*, 2000. Also see http://research.microsoft.com/sn/farsite/.

[4] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] Yuan Chen, Jan Edler, Andrew Goldberg, Allan Gottlieb, Sumeet Sobti, and Peter Yianilos. A prototype implementation of archival intermemory. In *Proceedings of the Fourth ACM International Conference on Digital Libraries*, 1999.

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system, 2000.

[7] John R. Douceur and William Bolosky. A large-scale study of file-system contents. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems*, pages 59–70, New York, May 1–4 1999.

[8] John R. Douceur and Roger P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation on Computer and Telecommunication Systems*, 2001.

[9] John R. Douceur and Roger P. Wattenhofer. Optimizing file availability in a serverless distributed file system. In *Proceedings of the 20th Symposium on Reliable Distributed Systems*, 2001.

[10] Gnutella. See http://gnutelladev.wego.com.

[11] Nili Guttmann-Beck and Refael Hassin. Approximation algorithms for min-sum p-clustering. In *Discrete Applied Mathematics, vol. 89:1–3*. Elsevier, 1998.

[12] Leslie A. Hall. Approximation algorithms for scheduling. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1995.

[13] David M. Harrison. Mathematica experimental data analyst. *Wolfram Research, Champaign, IL*, 1996.

[14] Sandy Irani and Anna R. Karlin. Online computation. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1995.

[15] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[16] John Kubiatowicz, David Bindel, Patrick Eaton, Yan Chen, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Westley Weimer, Chris Wells, Hakim Weatherspoon, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35(11):190–201, November 2000.

[17] Napster. See http://www.napster.com.

[18] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the 16th Symposium on Operating Systems Principles*, volume 31,5 of *Operating Systems Review*, pages 224–237, New York, October 5–8 1997.