Modeling Replica Placement in a Distributed File System: Narrowing the Gap between Analysis and Simulation

John R. Douceur and Roger P. Wattenhofer

Microsoft Research, Redmond WA 98052, USA
{johndo,rogerwa}@microsoft.com
http://research.microsoft.com **

Abstract. We examine the replica placement aspect of a distributed peer-to-peer file system that replicates and stores files on ordinary desktop computers. It has been shown that some desktop machines are available for a greater fraction of time than others, and it is crucial not to place all replicas of any file on machines with low availability. In this paper we study the efficacy of three hill-climbing algorithms for file replica placement. Based on large-scale measurements, we assume that the distribution of machine availabilities be uniform. Among other results we show that the `MinMax` algorithm is competitive, and that for growing replication factor the `MinMax` and `MinRand` algorithms have the same asymptotic worst-case efficacy.

1 Introduction

Farsite [10] is a distributed peer-to-peer file system that replicates and stores files on ordinary desktop computers rather than on dedicated storage servers. Multiple replicas are created so that a user can access a file if at least one of the machines holding a replica of that file is accessible. It has been shown [3] that some desktop machines are available for a greater fraction of time than others, and it is crucial not to place all replicas of any file on machines with low availability, or the availability of that file will suffer.

In earlier work [9], we evaluated the efficacy and efficiency of three hill-climbing algorithms for file replica placement, using competitive analysis and simulation. The scenario under consideration was a static problem in which the availability of each machine was fixed, and each replica stably remained on the machine to which the placement algorithm assigned it. Our study found that algorithmic efficiency and efficacy ran counter to each other: The algorithm with the highest rate of improvement yielded a final placement with the poorest quality relative to an optimal placement.

** Due to lack of space we omit most of the proofs in this extended abstract. The complete paper is available as Microsoft Research technical report MSR-TR-2001-62.

In actual practice, the replica placement problem is not static. The availability of each machine (defined loosely as the fraction of time it is accessible) varies over time as user behavior changes. In addition, file replicas may be evicted from machines by other processes in the system. The replica placement algorithm does not produce a static final placement that thereafter persists; rather, it continuously operates to correct for dynamic changes in the system. Viewed from this dynamic perspective, extensive Monte Carlo simulation shows that the `MinMax` algorithm consistently out-performs the other two algorithms, even though it was proven [9] to be non-competitive. Hence, our theoretic worst-case competitive analysis opposes use of the algorithm that appears best in practice.

We thus face an apparent dilemma: Either we fail to exploit an algorithm that is demonstrably efficient, or we risk the possibility that our system will encounter a distribution of machine availabilities that renders our algorithm useless. In the present paper, we make stronger assumptions about the algorithm's input, based on large-scale measurement of machine availability [3]. Given these assumptions, which – we stress – are a close approximation of the behavior of actual machines, we show that the `MinMax` algorithm is competitive for the levels of replication we intend to use in actual deployment. Obtaining these new results requires completely different analysis methods from those used for our earlier general-distribution results, which relied on highly unusual availability distributions utterly dissimilar to those found in real systems.

Furthermore, our earlier studies evaluated competitiveness in terms of the least available file, which is a straightforward quantity to analyze. However, from a systems perspective, a better metric is the effective availability of the overall storage system, which is readily computable in simulation. In the present paper, we show that all worst-case results for minimum file availability are also worst-case results for effective system availability, further legitimizing the relevance of our theoretic analyses.

In our opinion, the significance of this work lies in the fusion of four elements: an important problem from an emerging area of systems research, simulation results that demonstrate the practical performance of a suite of algorithms, large-scale measurements of deployed systems that provide a tractable analytic model, and rigorous theoretic analysis to provide confidence in the algorithm selected for use in the actual system. We consider this an exemplary synergy of systems, simulation, measurement, and theory.

The remainder of the paper is organized as follows. 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. In Section 4 we further motivate this paper, followed by a summary of results in Section 5. Section 6 presents a simplified model, which is used in Section 7 to analyze the efficacy of the algorithms. Section 8 compares the two measures of efficacy. In Section 9 we conclude the paper by presenting related work.

2 Farsite

Farsite [10] is a distributed peer-to-peer file system that runs on a networked collection of desktop computers in a large organization, such as a university or corporation. It provides a logically centralized storage repository for the files of all users in the organization. However, rather than storing these files on dedicated server machines, Farsite replicates them and distributes them among all of the client computers sitting on users' desktops. As compared to centralized storage, this architecture yields great savings in hardware capital, physical plant, system administration, and operational maintenance, and it eliminates a single point of failure and single target of attack. The disadvantage of this approach is that user's desktop machines lack the physical security and continuous support enjoyed by managed servers, so the system must be designed to resist the threats to reliability and security that are inherent in a large-scale, distributed, untrusted infrastructure.

For files stored in Farsite, the following properties are maintained: privacy, integrity, persistence, and availability. Data privacy and integrity are ensured by encryption and digital signatures. File persistence is provided by generating R replicas of each file and storing the replicas on different machines. The data will persist as long as one of the replicas resides on a machine that does not suffer a destructive failure, such as a disk head crash. Since it is difficult to estimate the remaining lifetime of a particular disk with any accuracy [16], the degree of data persistence is considered to be determined entirely by the replication factor R and not by any measurable aspect of the particular machines selected for storing the replicas.

In this paper we focus on file availability, meaning the likelihood that the file can be accessed by a user at the time it is requested, which is determined by the likelihood that at least one replica of that file can be accessed at the requested time. The *fractional downtime* of a machine is the mean fraction of time that the machine is unavailable, because it has crashed, has been turned off, has been disconnected from the network, etc. A five-week series of hourly measurements of over 50,000 desktop machines at Microsoft [3] has shown that the times at which different machines are unavailable are not significantly correlated with each other, so 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 simplicity, we express machine and file availability values as the negative logarithm of fractional downtime, so the availability of a file equals the sum of the availabilities of the R machines that store replicas of the file.

The goal of a file placement algorithm is to produce an assignment of file replicas to machines that maximizes an appropriate objective function. We consider two objective functions in this paper: (1) the *minimum file availability* over all files and (2) the *effective system availability* (ESA), defined as the negative logarithm of the expected fractional downtime of a file chosen uniformly at random. When we evaluate the efficacy of a file placement algorithm, we are gauging its ability to maximize one of these objective functions. For our theoretic analyses in both our earlier work [9] and the present paper, we focus on the metric

of minimum file availability, because it is more readily tractable. Our simulation results, such as those described in Section 4, relate to ESA because it is more meaningful from a systems perspective. One of our current findings (Section 8) is that all of our theoretic worst-case results for minimum file availability are also theoretic worst-case results for effective system availability.

Measurements of over 10,000 file systems on desktop computers at Microsoft [8] indicate that a replication factor of $R = 3$ is achievable in a real-world setting [3]. Thus, we have a special interest in the case $R = 3$.

3 Algorithms

Files in Farsite are partitioned into disjoint sets, each of which is managed by a small, autonomous group of machines. This imposes the requirement that a file placement algorithm must be capable of operating in a distributed fashion with no central coordination. Farsite is also a highly dynamic system in which files are created and deleted frequently and in which machine availabilities continuously change. This imposes the requirement that a file placement algorithm must be able to incrementally improve an existing placement, rather than require a complete re-allocation of storage resources. These and other considerations [10] have led us to a family of iterative, swap-based algorithms: One group of machines contacts another group (possibly itself), each of which selects a file from the set it manages; the groups then decide whether to exchange the machine locations of one replica from each file. The groups select files according to one of the following algorithms:

- `RandRand` swaps a replica between two randomly chosen files,
- `MinRand` swaps a replica between a minimum-availability file and any other file, and
- `MinMax` swaps a replica between a minimum-availability file and a maximum-availability file.

(We use the particle "`Rand`" rather than "`Any`" because this reflects the way files are selected in the system, even though all that matters for our theoretic analysis is the absence of a selection restriction.) The groups swap replicas only if doing so reduces the absolute difference between the availabilities of the two files, which we call a successful swap. If a pair of files has more than one successful swap, the algorithm chooses one with minimum absolute difference between the files' availabilities after the swap (although this does not affect theoretical efficacy). Because the algorithms operate in a distributed fashion, their selection restrictions are weakened, i.e., the `MinMax` and `MinRand` algorithms might select files whose availability values are not globally minimum or maximum. For our theoretic analysis, we concentrate on the more restrictive case in which only extremal files are selected.

4 Motivation

If, beginning with a random assignment of replicas to machines, we run each algorithm until it *freezes* (meaning no more swaps can be found), we find that the three algorithms differ substantially in both the efficacy of their final placements and the efficiency with which they achieve those placements. Simulations show that the `MinMax` algorithm improves the availability of the minimum file more quickly than the other two algorithms. On the other hand, `MinMax` tends to freeze at a point with lower minimum file availability, since swaps are only considered between the minimum-availability file and the maximum-availability file.

In earlier work [9], we performed a worst-case analysis to determine each algorithm's competitive ratio $\rho = m/m^*$, where m is the availability of a minimum-availability file when the algorithm freezes, and m^* is the availability of a minimum-availability file given an optimal placement, for a worst-case availability distribution. The results were that `MinMax` (the most efficient algorithm) was not competitive ($\rho = 0$), whereas `MinRand` and `RandRand` were 2/3-competitive for $R = 3$.

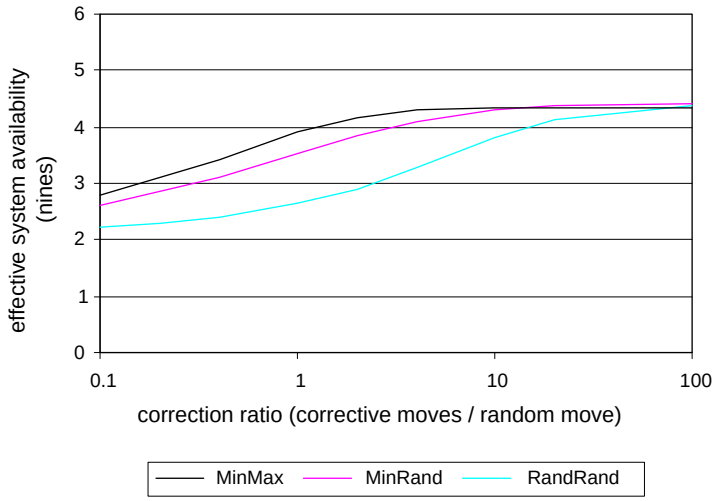

Fig. 1. Steady-state behavior of the algorithms

If we exercise each algorithm in a dynamic scenario that more closely matches the environment in which the Farsite system operates, the results are even more disconcerting. Figure 1 shows the result of a steady-state simulation in which two processes operate concurrently on the placement of replicas. One process (maliciously) moves random replicas to random machines, simulating the dynamic behavior of users and machines. The other process performs one of our

three hill-climbing algorithms, trying to repair the damage caused by the random moves. With the exception of unrealistically high correction ratios, `MinMax` performs significantly better than the other two algorithms.

We are in the unpleasant situation that a theoretical worst-case result ($\rho = 0$ for `MinMax`) opposes the use of an algorithm that works best for real-world data. In this paper, we begin to address this discrepancy by noting the distribution of availability values found in a large-scale study of desktop machines in a commercial environment [3], reproduced here as Figure 2. This figure shows that, when expressed logarithmically, machine availabilities follow a distribution that is nearly uniform. This finding, coupled with the observation that most of our worst cases [9] need rather unusual distributions of machine availabilities, suggests that we can improve the bounds of the worst-case analysis by making stronger assumptions about the input, namely that we have a uniform distribution of machine availabilities.

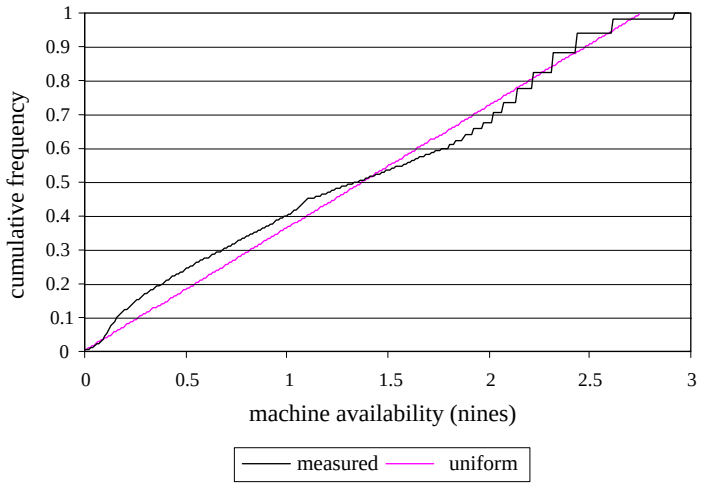

Fig. 2. Distribution of machine availabilities

5 Results

In this paper we will take for granted that the distribution of machine availabilities be uniform. With this assumption we show that the `MinMax` algorithm is competitive. More surprisingly, when the replication factor R grows the `MinRand` and `MinMax` algorithms have the same asymptotic worst-case efficacy. This is counterintuitive when looking at our earlier results [9]. We study the case $R = 3$

with special care, since the real Farsite system is expected to be deployed with $R = 3$. We also give detailed results for $R = 2$ since they are considerably different from $R > 2$. Here is a detailed summary of our results:

Algorithm	general R	$R = 2$	$R = 3$
`MinMax`	$\rho = 1 - \Theta(1/R)$ (1)	$\rho = 0$ (5)	$\rho = 1/2$ (4)
`MinRand`	$\rho = 1 - \Theta(1/R)$ (1)	$\rho = 1$ [9] & (2)	$\rho = 22/27$ (2)
`RandRand`	$\rho = 1 - \Theta(1/R^2)$ (3)	$\rho = 1$ [9] & (2)	$\rho = 8/9$ (3)

6 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. Throughout this paper we assume that machines have (uniformly distributed) *availabilities* $0\gamma, 1\gamma, 2\gamma, \ldots, (M-1)\gamma$, for an arbitrary constant γ.

Let the R replicas of file f be stored on machines with availabilities $a_1, \ldots, a_R$. 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 $f = a_1 + \cdots + a_R$.

As in our earlier study [9], we examine the point at which the algorithms freeze. 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 and R. We compute the ratio $\rho = \min m/m^*$ as $N \to \infty$. We say that the algorithm is ρ-competitive. Note that the scale γ of the machine availabilities does not affect ρ; throughout this paper we therefore assume $\gamma = 1$.

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 will be considered for a potential swap.

7 Analysis

We start this section with a number of undemanding observations which will help us simplify the analysis.

`MinMax` searches for swap candidates in a subset of `MinRand`, and similarly `MinRand` $\subseteq$ `RandRand`, thus

Lemma 1. $\rho_{\texttt{MinMax}} \leq \rho_{\texttt{MinRand}} \leq \rho_{\texttt{RandRand}}$.

In this paper we study a restricted case (we assume uniform availability) of the general problem that was investigated in [9]. We have immediately:

Lemma 2. *For the same algorithm, we have* $\rho_{general} \leq \rho_{uniform}$.

The next Lemma shows a simple observation: There always is an optimal assignment. This simplifies calculating the competitive ratio ρ.

Lemma 3. *For any $R > 1$ and uniform distribution there is an optimal assignment, where all the files have the same availability $R(M-1)/2$.*

Lemma 4. $\rho_{\texttt{MinRand}_R} \leq 1 - c/R$, *where c is a positive constant. If $R = 3$ then $c = 5/9$.*

Theorem 1. $\rho_{\texttt{MinRand}_R} = \rho_{\texttt{RandRand}_R} = 1 - \Theta(1/R)$.

Theorem 2. $\rho_{\texttt{MinRand}_3} = 22/27$.

Proof. (We include this proof in the extended abstract as a representative for the proof techniques in this paper.) With Lemma 4 we have that $\rho_{\texttt{MinRand}_3} \leq 22/27$. For proving the Theorem it is therefore sufficient to show that $\rho_{\texttt{MinRand}_3} \geq 22/27$.

The intuition of the proof: We partition the set of machines into five regions. With a detailed case study we show which combinations of regions do not allow successful swaps with the minimum file. Then we classify the valid combinations of regions, and give a combinatorial proof about their quantity which ultimately leads to a lower bound for the availability of the minimum file. For simplicity we omit insignificant constants throughout this proof (i.e. we write M instead of $M-1$).

Here are the details: Let the minimum file be $m = a_1 + a_2 + a_3$ with $a_1 > a_2 > a_3$. Assume for the sake of contradiction that $m < 11/9 \cdot M$. We define the following sets of machines (see Figure 3): Machines in A have availability less than a_3, machines in B between a_3 and a_2, machines in C between a_2 and $(a_1+a_2)/2$, machines in D between $(a_1+a_2)/2$ and a_1, and machines in E more than a_1. With this partitioning the availability of the minimum file m translates into $m = 2|A| + |B| + M - |E|$, and with $m < 11/9 \cdot M$ we get

$$2|A| + |B| - |E| < 2/9 \cdot M = 2/3 \cdot N.$$

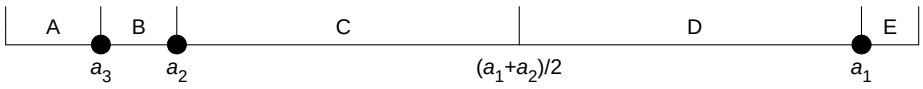

Fig. 3. Partition of the machines

Case 1: We consider all the files $f = b_1 + b_2 + b_3$ with $b_1 \in E$, and $b_2 > b_3$. If $b_2 + b_3 > a_2 + a_3$, then we swap the machines b_1 and a_1 and get $m' = b_1 + a_2 + a_3 > a_1 + a_2 + a_3 = m$ and $f' = a_1 + b_2 + b_3 > a_1 + a_2 + a_3 = m$. Thus $b_2 + b_3 \leq a_2 + a_3$, and therefore (with $b_2 > b_3$) $b_3 < (a_2 + a_3)/2$. Since each $b_1 \in E$ needs a b_3, we know that $|E| < (a_2 + a_3)/2$. Since $|A| + |B|/2 = (a_2 + a_3)/2$ and $2|A| + |B| - |E| < 2/9 \cdot M$ we get $(a_2 + a_3)/2 < 2/9 \cdot M$. On the other hand $|E| < (a_2 + a_3)/2$ shows that $(a_1 + a_2)/2 > 1/2 \cdot M$. Since $b_2 + b_3 \leq a_2 + a_3 <$

$4/9 \cdot M < 1/2 \cdot M < (a_1 + a_2)/2$, we know that $b_2 \in A \cup B \cup C$. If $b_2 > a_2$ then we have $b_3 < a_3$, in other words, if $b_2 \in C$ then $b_3 \in A$. If $b_2 \in A \cup B$ then $b_3 \in A \cup B$. Since we have used all machines in set E in this case, there are no machines in E in the following cases.

Case 2: We identify all remaining files $f = b_1 + b_2 + b_3$ with $b_1 \in C$, and $b_2 > b_3$. If $b_3 \in D$, then we swap machines b_1 and a_2, getting $m' = a_1 + b_1 + a_3 > a_1 + a_2 + a_3 = m$ and $f' = a_2 + b_2 + b_3 > a_2 + b_3 + b_3 \geq a_2 + 2(a_1 + a_2)/2 > m$. Therefore $b_3 \notin D$. Thus for each $b_1 \in C$ we have $b_2 \in A \cup B \cup C \cup D$ and $b_3 \in A \cup B \cup C$. We have used all machines in C; henceforth the sets C and E are taboo.

Case 3: We identify all the remaining files $f = b_1 + b_2 + b_3$ with $b_1 \in B$, and $b_2 > b_3$. If $b_3 \in D$, then we swap machines b_1 and a_3, getting $m' = a_1 + a_2 + b_1 > a_1 + a_2 + a_3 = m$ and $f' = a_3 + b_2 + b_3 > a_3 + b_3 + b_3 \geq a_3 + 2(a_1 + a_2)/2 = m$. Therefore $b_3 \notin D$. Thus for each $b_1 \in B$ we have $b_2 \in A \cup B \cup D$ and $b_3 \in A \cup B$. Henceforth the sets B, C, E are taboo.

Case 4: Finally we identify all the remaining files $f = b_1 + b_2 + b_3$ with $b_1 \in D$, and $b_2 > b_3$. If $b_3 \in D$, then we swap machines b_1 and a_2, getting $m' = a_1 + b_1 + a_3 > a_1 + a_2 + a_3 = m$ and $f' = a_2 + b_2 + b_3 > a_2 + b_3 + b_3 \geq a_2 + 2(a_1 + a_2)/2 > m$. Thus for each $b_1 \in D$ we have $b_2 \in A \cup D$ and $b_3 \in A$.

From above analysis we have seen that a file f can only consist of the these combinations of regions: (Case 1) $E + C + A$ or $E + (A \cup B) + (A \cup B)$ or (Case 2) $C + (A \cup B \cup C \cup D) + (A \cup B \cup C)$ or (Case 3) $B + (A \cup B \cup D) + (A \cup B)$ or (Case 4) $D + A + A$ or $D + D + A$. We define the two functions g_1, g_2:

$$g_1(f) = |C| - |D|$$
$$g_2(f) = 2|A| + |B| - |E|$$

Figure 4 shows all possible files f with respect to the functions $g_1(f)$ and $g_2(f)$.

Note that for all possible files f we have $g_2(f) \geq 0$. We put the files into three classes. Class X are the files f with $g_1(f) < 0$ (the black circles); class Y are the files with $g_1(f) = 0$ (the white circles); class Z are the files with $g_1(f) > 0$ (the grey circles). Note that for files $f \in X$ we have $g_1(f) \geq -2$ and $g_2(f) \geq 2$, and that for files $f \in Y$ we have $g_2(f) \geq 1$.

We have M machines, thus (ignoring the single mimimum file m) $|A| + |B| + |C| + |D| + |E| = M = 3N$. This translates into $|X| + |Y| + |Z| = N$ for the three classes X, Y, Z. The sets C and D were defined such that they exactly split the region of machines between the a_2 and a_1, hence $|C| = |D|$. Using $g_1(f) \geq -2$ for $f \in X$, and $g_1(f) \geq 1$ for $f \in Z$, the constraint $|C| = |D|$ translates into $2|X| \geq |Z|$. Both constraints together, we get $3|X| + |Y| \geq |X| + |Y| + |Z| = N$. We multiply with $2/3$: $2|X| + |Y| \geq 2|X| + 2/3 \cdot |Y| \geq 2/3 \cdot N$. We use this inequality to get:

$$2|A| + |B| - |E| = \sum_{f \in X} g_2(f) + \sum_{f \in Y} g_2(f) + \sum_{f \in Z} g_2(f) \geq 2|X| + |Y| \geq 2/3 \cdot N.$$

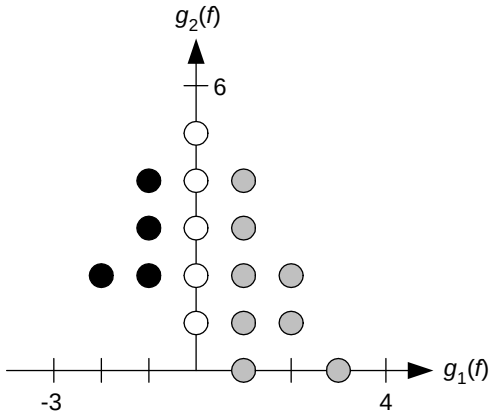

Fig. 4. Possible locations for a file f.

(The first equality is the definition of g_2; the middle inequality is because files $f \in X$ have $g_2(f) \geq 2$, files $f \in Y$ have $g_2(f) \geq 1$, and files $f \in Z$ have $g_2(f) \geq 0$.)

This contradicts our assumption that $2|A| + |B| - |E| < 2/3 \cdot N$ and therefore the assumption that $m < 11/9 \cdot M$. Thus $m \geq 11/9 \cdot M$. With Lemma 3 we know that $m^* = 3(M-1)/2$. Thus $\rho = m/m^* \geq 22/27$, as M goes to infinity.

Theorem 3. $\rho_{\texttt{RandRand}_R} = 1 - c/R^2$, *where c is a positive constant. If R is odd then $c = 1$.*

Theorem 4. $\rho_{\texttt{MinMax}_3} = 1/2$.

Theorem 5. $\rho_{\texttt{MinMax}_R} = 1 - 2/R$, *for R even.*

8 Measures of Efficacy

We can show that any worst-case result for minimum file availability is also a worst-case result for effective system availability. We show that the effective system availability can be as low as the minimum file availability, and it cannot be lower.

Theorem 6. *Let b be the base for converting downtime d into availability a, that is $a = -\log_b d$. As $b \to \infty$, the effective system availability (ESA) equals the availability of the minimum file.*

Proof. Let $b = e^c$. Then $a = -\log_b d = -1/c \cdot \ln d$, where $\ln = \log_e$. If $b \to \infty$ then $c \to \infty$. Let m be the availability of the minimum file. Assume that there

are $X > 0$ files with availability m and $N - X$ files with availability f_i with $f_i > m$, for $i = 1, \ldots, N - X$. Then, applying the definition of ESA,

$$\begin{aligned}\lim_{c\to\infty} \text{ESA} &= \lim_{c\to\infty} -\frac{1}{c}\ln\left(\frac{1}{N}\left(Xb^{-m} + \sum_{i=1}^{N-X} b^{-f_i}\right)\right) \\ &= \lim_{c\to\infty} -\frac{1}{c}\ln\left(\frac{e^{-cm}}{N}\left(X + \sum_{i=1}^{N-X} e^{c(m-f_i)}\right)\right) \\ &= \lim_{c\to\infty} -\frac{1}{c}\ln\left(\frac{X}{N}e^{-cm}\right) = \lim_{c\to\infty}\left(m - \frac{1}{c}\ln\frac{X}{N}\right) = m.\end{aligned}$$

Similarly,

Theorem 7. *Let b be the positive base for converting uptime into availability. Then, ESA $\geq m$.*

9 Related Work

Other than Farsite, serverless distributed file systems include xFS [2] and Frangipani [17], both of which provide high availability and reliability through distributed RAID semantics, rather than through replication. Archival Intermemory [5] and OceanStore [14] 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 [15] and Gnutella [11] 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 [9] is the first study of the availability of replicated files, and also the first competitive analysis of the efficacy of a hill-climbing algorithm.

There is a common denominator of our work and the research area of approximation algorithms; especially in the domain of online approximation algorithms [13, 4] such as scheduling [12]. In online computing, an algorithm must decide how to act on incoming items without knowledge of the future. This seems to be related our work, in the sense that a distributed hill-climbing algorithm also makes decisions locally, without the knowledge of the whole system. Also, online algorithms research naturally focuses on giving bounds for the efficacy of an algorithm rather than for the efficiency.

Competitive analysis has been criticized as being too crude and unrealistic [4]. In this paper, we have narrowed the gap between theoretical worst-case analysis and real-world simulations, which has emerged because of unusual worst case, by making stronger and more realistic assumptions about the input. This is an approach that is well-known in the area of online algorithms; for an overview, see Chapter 5 in [4] for paging algorithms, and Section 2.3 in [7] for bin packing algorithms.

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.
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. Edward G. Coffman, M.R. Garey, and David S. Johnson. Appromxiation algorithms for bin packing: A survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1995.
8. John 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.
9. John Douceur and Roger Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proceedings of the 15th International Symposium on Distributed Computing*, 2001.
10. John Douceur and Roger Wattenhofer. Optimizing file availability in a serverless distributed file system. In *Proceedings of the 20th Symposium on Reliable Distributed Systems*, 2001. Also see http://research.microsoft.com/sn/farsite/.
11. Gnutella. See http://gnutelladev.wego.com.
12. Leslie A. Hall. Approximation algorithms for scheduling. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1995.
13. Sandy Irani and Anna R. Karlin. Online computation. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1995.
14. 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.
15. Napster. See http://www.napster.com.
16. Roger T. Reich and Doyle Albee. S.M.A.R.T. phase-II. *White paper WP-9803-001, Maxtor Corporation*, February 1998.
17. 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 (SOSP-97)*, volume 31,5 of *Operating Systems Review*, pages 224–237, New York, October 5–8 1997.