Replication Strategies for Highly Available Peer-to-peer Storage Systems

Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker
Department of Computer Science and Engineering
University of California, San Diego

Abstract

Failure is inevitable: disks fail, hosts crash, networks partition, applications stop. Consequently, the principal challenge in designing highly-available systems is to tolerate each failure as it occurs and recover from its effects. For large systems, or systems with unreliable components, such failures can cease to be exceptional events, but instead may become the common case. Perhaps no design point is more challenging in this respect than that faced by heterogeneous peer-to-peer systems. Such systems are typically composed of very large numbers of hosts, of which only a minority may be available at any one time. In this environment, failure is not only common, but pervasive. This paper analyzes the challenges and limitations in building a highly-available storage system in such a peer-to-peer environment. In particular, we explore the design requirements on failure tolerance and failure recovery in environments with limited host availability. Our contributions are threefold: First, we provide an analytic model for reasoning about the efficiency of replication and erasure encoding as temporary storage redundancy mechanisms. Second, we extend this framework to model the availability of groups of files or file systems. Finally, we incorporate the costs of maintaining a given level of availability in the long term by recovering from persistent storage failures. We show that even in environments with pervasive failure it is possible to offer a storage service with a high degree of availability at a moderate cost in storage overhead.

1 Introduction

In the past few years, peer-to-peer systems have become an extremely popular platform for large-scale content sharing. Unlike traditional server-based storage systems, which centralize the management of data in a few highly reliable servers, peer-to-peer storage systems distribute the burden of data storage and communications among thousands of individual client workstations. The wide-spread attraction of this model arises from the promise that idle resources may be efficiently harvested to provide scalable storage services. Unfortunately, while significant progress has been made towards providing scalable data access in peer-to-peer systems [16, 17, 19, 22], the challenges in providing acceptable levels of availability are poorly understood by comparison.

At the heart of this challenge is the ad hoc manner in which peer-to-peer systems are constructed. In contrast to traditional distributed systems, peer-to-peer systems are composed of components with extremely heterogeneous availabilities – individually administered host PC's may be turned on and off, join and leave the system, have intermittent connectivity, and are constructed from low-cost low-reliability components. For example, one recent study of a popular peer-to-peer file sharing system found that the majority of peers had application-level availability rates of under 20 percent [18]. In such an environment, failure is no longer an exceptional event, but is a pervasive condition. At any point in time the majority of hosts in the system are unavailable and those hosts that are available may soon stop servicing requests. While most peer-to-peer systems employ some form of data redundancy to cope with failure, these solutions are ad hoc and generally poorly matched to the underlying host failure distribution or the level of availability desired by users. Consequently, it remains unclear what availability guarantees can be made using existing systems, or conversely how to best achieve a desired level of availability using the mechanisms available.

In this paper, we present an analytic framework for evaluating the minimal storage costs required to provide a given level of availability as a function of the underlying host failure distribution. We consider several data redundancy mechanisms including conven-

tional replication strategies (whole file and per-block) as well as file-level erasure coding with variable redundancy. Each of these approaches has its advantages and disadvantages and we recognize that workload and system implementation issues may favor one approach over the other in particular scenarios. For example, whole-file replication allows simple and low-overhead implementations of naming and lookup, while block-level replication allows increased performance through parallel downloads at the cost of additional storage to provide a given level of availability. The models we present in this paper allow a system designer to evaluate such tradeoffs with full knowledge of the costs required for each mechanism to achieve a desired level of availability.

We also consider availability at two different time scales, reflecting the clear cut division between short-term transient failures dominated by periodic user behavior and long-term persistent failures that reflect permanent data loss. For example, in peer-to-peer client populations many users switch off their computers at night, making them unavailable for a period of hours every day. This behavior is independent of the user deciding to stop the use of the system altogether, or of the machine experiencing a hardware failure. We categorize the former behavior as one that determines the *short-term availability* of files in the system, while the latter is more likely to affect a file's *long-term availability*. A system designer addresses short-term availability by ensuring that there is sufficient data redundancy to tolerate transient failures during a relatively short period of time – such as a 24 hour period – over which the host availability distribution is known. By contrast, long-term persistent failures, modeled by a decay distribution, require the system to explicitly recover by adding additional redundancy into the system to match the storage services that are lost. Finally, we show how to combine these two approaches to guarantee a specified level of availability over an extended period.

The remainder of this paper is structured as follows: Section 2 provides background and describes the related work that led to our analysis. In Section 3 we describe our general analytic approach and the simulation framework we use to validate it. Sections 4 and 5 develop and validate analytic models for evaluating the short-term fault tolerance of conventional replication and erasure coded replication strategies respectively. We present the analysis of failure recovery for providing long-term availability in Section 6 and Section 7 discusses the practical implementation issues not covered by our formal model. Finally, we summarize our results in Section 8.

2 Related Work

All peer-to-peer systems must employ some form of storage redundancy to provide acceptable service to their users. In systems such as Napster and Gnutella [8], data replication occurs implicitly as each file downloaded by a user is replicated at the user's workstation. However, since these systems do not explicitly manage replication, the availability of an object is fundamentally linked to its popularity and rarely accessed objects cannot be reliably accessed. Moreover, since this approach does not mask failures, users must repeatedly access different replicas until they find one on an available host. By contrast, next-generation peer-to-peer storage systems, such as the Cooperative File System (CFS) [4], address both of these limitations through an explicit replication strategy that hides failures from the user and is workload independent.

Systems using replication must also choose the granularity at which data is replicated: whole-file vs block-level. Whole-file replication, as used in Gnutella, is simple to implement and has a low state cost – it must only maintain state proportional to the number of replicas. However, the cost of replicating entire files in one operation can be cumbersome in both space and time, particularly for systems that support applications with large objects (e.g., audio, video, software distribution).

Block-level replication, as used by eDonkey [7] and CFS [4], divides each file object into an ordered sequence of fixed-size blocks. Because individual parts of an object may be named independently, a block-level system may download different parts of an object simultaneously from different peers and reduce overall download time. Also, because the unit of replication is small and fixed, the cost to replicate an individual block can be small and can be distributed among many peers. Finally, block-level representation allows large files to be spread across many peers even if the whole file is larger than any single peer is able to store. However, an important drawback of block-level replication is that if enough replicas fail such that any single block cannot be found, then the entire file object is unavailable. For example, measurements of the CFS system using six block-level replicas show that when 50 percent of replicas fail the probability of a block being

unavailable is less than two percent [4]. However, if an object consists of 8 blocks then the expected availability for the *entire object* will be less than 15 percent.

Several systems, including Intermemory [3], Swarmcast [20], Oceanstore [11], and FreeHaven [5], have recognized the need to address this issue and use erasure codes (EC) to achieve higher availability than whole-file and block replication using reduced storage. Erasure codes, such as Reed-Solomon [15] and Tornado [2] codes, provide the property that a set of b original blocks can be reconstructed from any m coded blocks taken from a set of cb (where m is typically close to b, and c is typically a small constant). For large files, erasure codes are a more efficient method for providing file availability guarantees compared to whole-file replication.

In all of these systems the amount of redundancy, either the number of replicas or the *stretch factor c*, is determined in a static ad hoc fashion and is not closely coupled with the required level of file availability or the shape of the underlying host availability distribution. This could lead to overestimating the needed redundancy, wasting storage needlessly. Or worse, the required redundancy might be underestimated, providing a lower level of availability than necessary.

The closest analog to our work is that of Weatherspoon and Kubiatowicz [21] who compare the availability provided by erasure coding and whole file replication under particular failure assumptions. The most critical differences between this work and our own revolve around the failure model. In particular, Weatherspoon and Kubiatowicz focus on disk failure as the dominant factor in data availability and consequently miss the distinction between short and long time scales that is critical to deployed peer-to-peer systems. Consequently, their model is likely to dramatically overestimate true file availability in this environment.

3 System Model

In this section, we describe the common aspects of our system model that we use to evaluate the conventional and erasure coded replication strategies in the rest of the paper. In particular, we describe our replica placement policy, our probabilistic models for characterizing host availability and host storage requirements, and our simulation methodology for validating our analytic methods.

3.1 Replica placement policy

Our goal is to guarantee a certain level of availability for all files in a peer-to-peer system. For any replication strategy, the optimal solution is a combinatorial function of per-host availability and the number of replicas per file that produces a mapping between individual replicas and individual hosts. This approach has several drawbacks that make it impractical. First, it imposes a challenging system requirement that the availability of each host be accurately tracked and estimated. This requirement is methodologically difficult and presents significant communication overheads as the system scales to large numbers of hosts. Second, even given perfect per-host availability information the solution to this combinatorial optimization problem is NP-hard.

To make our analysis both practical and tractable, we assume that host failures are independent and identically distributed (iid), allowing replicas to be assigned to hosts randomly. Implementing this random placement policy is straightforward in practice, scales well as the number of hosts increase, and also fits well with the way current peer-to-peer systems like Chord and Pastry operate. In practice the iid assumption is reasonable with the exception of diurnal correlation between hosts (hosts tend be turned on and off according to daily work patterns that are highly correlated in individual time zones). We discuss the impact of this deviation and how to accommodate it in practice in Section 7.

Table 1 summarizes the variables used in the analysis. Using y random replica placements and a known host availability distribution $H(x)$, we can estimate the probability P that there are sufficient replicas available at the time the file is accessed as follows. If we randomly pick h hosts from an n-host system, then the probability of exactly y of the hosts being up follows a binomial distribution with the mean of the original host availability distribution $H(x)$. This implies that any host chosen at random from the pool of hosts has an expected availability of μ_H.

To represent distributing replicas of a file across all hosts, we can pick h hosts at random and calculate the probability that exactly y hosts out of the h are available. This is equivalent to tossing a coin with bias μ_H h times, and calculating the probability of obtaining exactly y heads. It is known that the number y follows a binomial distribution with mean μ_H and variance $\mu_H(1-\mu_H)$ [6]. So, if Y is the random variable for the number of hosts available, then the following

Variable	Definition
n	No. of hosts in the system
f	No. of files in the system
c	Replica factor for conventional replication, stretch factor for erasure coding
b	No. of blocks in a file with erasure coding
$A_j, 1 \leq j \leq f$	Required availability of file j
$H(x)$	Host availability distribution
μ_H	Mean of host avail. distribution
${\sigma_H}^2$	Variance of host avail. distribution
Y	Random var. for number of hosts available out of h randomly picked hosts
$W = Y/cb$	Random var. for fraction of number of hosts out of cb randomly picked hosts that are available
Z	Random var. for the storage per host

Table 1: Summary of variables used in the probabilistic analysis.

formula estimates the number of available hosts:

$$P(Y = y) = \binom{h}{y} {\mu_H}^y (1 - \mu_H)^{(h-y)} \qquad (1)$$

We use this result in later sections to derive formulas for file availability for both conventional and erasure coded replication.

3.2 Storage requirements

The overall system storage requirements SS for a particular replication strategy can be directly calculated using the degree of replication c and number of files in the system f: $SS = cf$.

Moreover, using our random placement policy, we can bound the amount of data stored on a given host to be no more than x with probability P. To do this, note that we are assigning r data objects to n hosts in a random fashion. This is equivalent to the well-known *occupancy problem* which states that if r balls are randomly assigned to n bins, the number of balls in any of the bins is an independent random variable Z that follows a Poisson distribution with mean r/n and variance r/n [6]. So the probability that a certain host contains exactly x data objects is given by:

$$P(Z = x) = \frac{\left(\frac{r}{n}\right)^x e^{-r/n}}{x!} \qquad (2)$$

and the probability that a host contains at most x objects is:

$$P(Z \leq x) = \sum_{k=0}^{x} \frac{\left(\frac{r}{n}\right)^k e^{-r/n}}{k!} \qquad (3)$$

With these formulas for P, we can determine bounds using confidence intervals on the required storage per host required by a particular replication strategy.

3.3 Simulation Methodology

In order to verify our analyses, we use a Monte-Carlo simulation of a system of 1000 hosts. The number of files in the system is chosen to be 1000. Since we use a randomized replica placement strategy without any storage limits per host, increasing the number of files in the system does not change the behavior of the system in terms of availability of the files. File sizes follow a normal distribution with mean 700 MB and standard deviation 100 MB. These numbers were chosen to reflect the size of large multimedia files, such as movies, that are roughly around 700 MB in size. We chose a mean host availability of 0.5 based upon the measurements in [18], which found that the host availability distribution roughly follows a uniform random distribution between 0 and 1.

4 Conventional replication

In this section, we characterize file availability, system-level availability, and the storage requirements for the conventional replication strategy.

4.1 File Availability

With conventional replication, the goal of the system is to use the minimum number of replicas of a file, or *replication factor*, to provide a desired level of file availability. We consider both whole-file and block-level replication schemes.

Whole-file replication. Suppose we make c copies of the file and put them on different hosts. We need at least one of those c hosts to be available to recover the file. The file availability is therefore defined as the probability that one or more hosts are up. Using

Required availability	Replication factor (c)
0.800	3
0.900	4
0.950	5
0.990	7
0.995	8
0.999	10

Table 2: Number of replicas needed for said file availability for mean host availability 0.5.

Equation 1 from Section 3.1, the probability that one or more hosts are up is (1 - $P(Y = 0)$), or $P(Y \geq 1)$, where Y is the random variable for the number of available hosts. Since Y follows a binomial distribution,

$$A = P(Y \geq 1) = 1 - (1 - \mu_H)^c$$

$$c = \frac{log(1 - A)}{log(1 - \mu_H)} \quad (4)$$

This formula enables us to calculate the replication factor for a desired degree of file availability. Suppose we want the file availability to be 0.99. The number of replicas needed for this with random placement is given by substituting A by 0.99 and solving the above equation. Table 2 gives the value of c for a range of required file availabilities, assuming a mean host availability of 0.5. The replication factor should not be affected by variations in the host availability distribution, as long as the mean stays the same.

Figure 1 shows the replication factor for varying values of host availability and required file availability. The replication factor can get quite large for low values of mean host availability. For example, for mean host availability of 0.2, and required file availability of 0.99, the replication factor is 42. For large files, this can be a problem because the system would not scale well in terms of total storage required to replicate the file such that the system can meet the required availability.

Block-level replication. The advantage of block-level replication is that, for variable file sizes, storage burdens would be more evenly distributed between the hosts. Also, it has the advantage of the ability to perform parallel downloads and load balancing.

If we divide the file into b blocks, make c copies of each of them and place one block per copy per host, as in CFS and eDonkey, the availability of the file is given by:

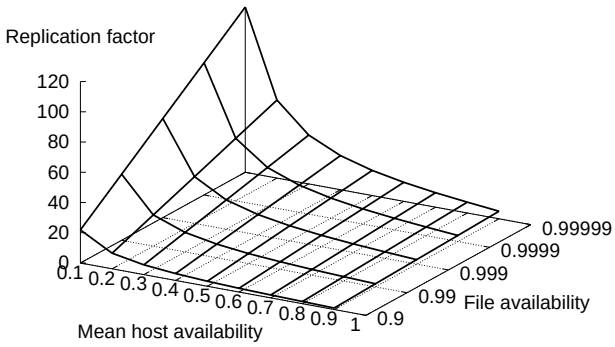

Figure 1: The number of replicas that we predict we need with conventional replication.

$$A = P(Y \geq 1) = (1 - (1 - \mu_H)^c)^b$$

To achieve the same level of availability in the block-level replication case as when doing whole-file replication, the number or replicas c would need to be larger. So the scalability of this scheme with respect to total storage required is even lower than that of the whole-file replication scheme. In other words, the availability of whole-file conventional replication is always better than block-level replication.

Validation. To validate our analytic model of file availability for conventional replication, we simulated file availabilities according to our random replica placement algorithm and compared them to the required file availability. In Figure 2, we plot the density of deviations of simulated per-file availability from required per-file availability for 1000 files. Most of the files either show equal or better actual availability than the required file availability. However, there are some files that are less available than the required value. This variation arises from the fact that the number of hosts chosen on which to replicate the files is not very large; for example, for 0.9 availability, we make only 4 copies of the file.

4.2 System-level availability

In addition to file availability, we are also interested in the system availability provided by conventional replication. System-level availability is a global measure of the availability of all of the files stored in the system, and reflects the availability of the system as a whole.

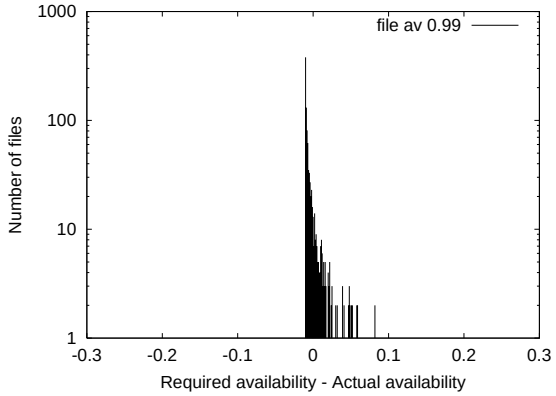

Figure 2: The density of deviations from the required availability for files, for conventional replication. The data are gathered from a simulation of 1000 files, with required availability per file = 0.99.

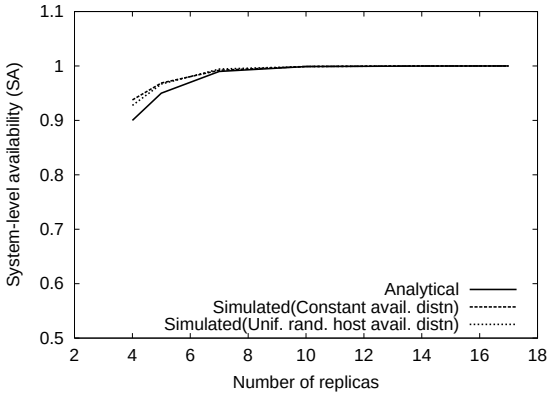

Figure 3: This graph shows how close the simulations of SA are to the required availability for erasure coding, which is shown by the bold line.

Formally, we define *system-level availability* SA as the average availability of all files in the system:

$$SA = \frac{1}{f}\sum_{j=1}^{f} A_j, 1 \leq i \leq f. \quad (5)$$

If all files use replication factors for a minimum availability A, the system-level availability will also evaluate to A:

$$SA = 1/f * (f * A).$$

Validation. Using the parameters from Section 3.3, we simulated how system-level availability adheres to the expected values obtained from the analysis. To see the effects of variance in the host availability distribution on system-level availability, we simulated both a constant host availability distribution with mean 0.5 and zero variance. Then we use a uniform random host availability distribution between 0 and 1, which has a mean and variance of 0.5 and 1/12, respectively. Figure 3 shows simulated values of SA for these distributions while varying the replication factor for the files. From the figure, we see that there is not much difference between the analytical results and the two simulations, indicating that variance in host availability distributions does not cause significant deviations from our analysis.

4.3 Storage requirements

To balance the storage requirements per host for conventional replication, we provide probabilistic bounds on the maximum storage required per host when using randomized placement and conventional replication. According to Equation 2 in Section 3.2, the number of files per host follows a Poisson distribution with mean cf/n. Because it is difficult to directly evaluate the Poisson distribution, we use the normal approximation to the Poisson distribution [6]. With the normal approximation, if we perform random placement of files on hosts then the number of files per host follows a normal distribution with mean $\mu_Z = cf/n$ and variance ${\sigma_Z}^2 = cf/n$. Table 3 shows the values of x for different values of $P(Z \leq x)$, which is the probability that the storage per host is less than or equal to x. These results are standard for any normal distribution. Using this table, we can calculate the value of x for given values of $P(Z \leq x)$. For example, with 0.9 probability, the storage per host will be less than or equal to $\mu_Z + 1.65\sigma_Z$. This can be calculated by substituting the values for c, f and n. With c=7, f=1000 and n=1000, this would evaluate to 11.

$P(Z \leq x)$	x
0.800	$\mu + 1.29\sigma$
0.900	$\mu + 1.65\sigma$
0.950	$\mu + 1.96\sigma$
0.990	$\mu + 2.58\sigma$
0.995	$\mu + 2.81\sigma$
0.999	$\mu + 3.30\sigma$

Table 3: Values of a random variable that follows a normal distribution for the given level of availability.

Validation. Figure 4 shows the simulated cumulative distribution of storage per host for this system, and

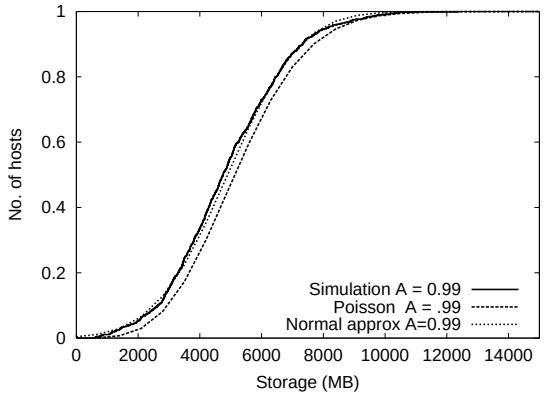

Figure 4: Cumulative distribution function of the storage per host for conventional replication.

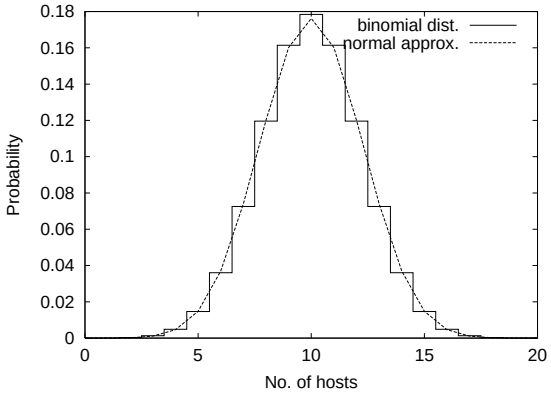

Figure 5: The graph shows the normal approximation to the distribution of the number of hosts available out of 20 randomly chosen hosts.

the Poisson and normal fits to it. Both distributions fit well with the simulated results. Using the storage per host analysis arguments, we can say that if we replicate files for an availability of 0.99, i.e., we make 7 copies of each file, then in a system with a thousand hosts and a thousand files, an individual host will store less than 7700MB of data 90 percent of the time.

5 Erasure coded replication

The addition of erasure coding (EC) to block-level replication provides one key advantage. It dramatically improves file availability since the increased intra-object redundancy can tolerate the loss of many individual blocks without compromising the availability of the whole file. The *stretch factor* of an erasure code is the measure of the amount of redundancy added to the file. It is equal to the number of post-encoded blocks over the number of pre-encoded blocks. For example, for the same storage requirements, one can either duplicate b blocks of an object or code those blocks into $2b$ EC blocks. Here the stretch factor for the code is 2. If the blocks are distributed across $2b$ nodes, then, when using standard block replication, at least one of two hosts for each block must be available to reconstruct the object. When using EC block replication, however, any b of the $2b$ hosts storing EC replicas need be available to reconstruct the object.

5.1 File Availability

First, we perform a probabilistic analysis of file availability for erasure coded replication using the formulas from Section 3.1. The goal is to determine the stretch factor a file must have to achieve a desired file availability. Throughout the analysis we assume that we place one encoded block per file per host. In Section 7 we show how this assumption can be relaxed without affecting our results.

Our objective is to calculate the required stretch factor c for a file to obtain an availability greater than or equal to A, given the host availability distribution $H(x)$ and the value of b. We know from earlier discussions that, to recover the file, any b out of the cb blocks need to be available. Since we assume that we have one block per file per host, this means that at least any b out of the cb hosts need to be available. This corresponds to the sum of the probability that any b hosts are available and $cb-b$ hosts are down, the probability that any $b+1$ hosts are available and $cb-b-1$ of them are not, etc. Using equation 1 this can be written as

$$A = P(Y \geq b) = \sum_{j=b}^{cb} \binom{cb}{j} {\mu_H}^j (1-\mu_H)^{(cb-j)}. \quad (6)$$

This equation results from the fact that the number of hosts available follows a binomial distribution. The right-hand side of this equation does not have a nice closed-form expression. Hence, given a value for A, expanding it and solving for c is a non-trivial task. However, if cb is large enough, we can use the normal approximation to the binomial distribution [6] to solve for c, which is a direct result of the Central Limit Theorem. Using this, we can say that the random variable $W = Y/cb$ follows a normal distribution with mean $\mu_W = \mu_H$ and variance $\sigma_W = \mu_H(1-\mu_H)/cb$. W

Required Availability	x
0.800	$\mu - 1.29\sigma$
0.900	$\mu - 1.65\sigma$
0.950	$\mu - 1.96\sigma$
0.990	$\mu - 2.58\sigma$
0.995	$\mu - 2.81\sigma$
0.999	$\mu - 3.30\sigma$

Table 4: Values of a random variable that follows a normal distribution for the given level of availability.

is simply the fraction of hosts available out of the cb hosts that hold blocks of the file. Figure 5 shows how the normal distribution with these parameters fits the binomial distribution for $cb = 20$. The binomial distribution, which is a discrete distribution, is shown with the steps. It is well-approximated by the continuous normal distribution.

Using this approximation, we can rewrite Equation 6 as

$$A = P(Y \geq b) = P(W \geq 1/c). \quad (7)$$

Table 4 shows the values of x for which $P(W \geq x)$ evaluates to the required minimum availability. These values are fixed for the normal distribution. Equating these values for x with $1/c$ gives us the stretch factor required for achieving the corresponding level of file availability.

$$1/c = \mu_W - k\sigma_W \quad (8)$$

Given the values of b and μ_H, we can then calculate the stretch factor c.

Appendix A shows the derivation of c from the equation 8. We can choose k in Equation 8 depending on the required file availability from Table 4, and calculate the stretch factor required for the file to have that availability. For example, if we want the file to have three nines of availability, k should be 3.30. If μ_H = 0.5 and b = 100, c evaluates to 2.49.

The value of cb needs to be large enough for our result to hold given our use of the normal approximation to the binomial distribution. In practice, it has been found that the normal approximation to the binomial distribution works quite well for $cb \geq 16$ [6]. It is realistic to expect that blocking file systems such as CFS would break up a large file into more than 16 pieces to facilitate load balancing. So this analysis would hold for all such cases.

Required Availability	Replication factor (Conv. replication)	Stretch factor (EC)
0.800	3	2.13
0.900	4	2.19
0.950	5	2.25
0.990	7	2.36
0.995	8	2.40
0.999	10	2.49

Table 5: The stretch factor required for the corresponding file availability, $\mu_H = 0.5$, $b = 100$.

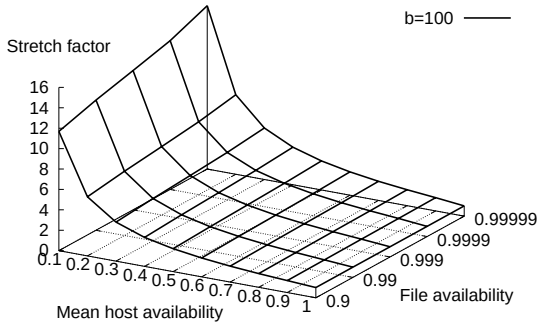

Figure 6: The graph shows the stretch factor required for varying levels of file availability, and for varying mean host availabilities, given that b is 100.

Table 5 shows the value of stretch factor for various file availabilities, given a mean host availability of 0.5 and b = 100. It shows that the difference in required stretch factor between two consecutive nines of availability is not much for reasonably high host availabilities ($\geq$ 0.5). For example, the stretch factor goes from 2.19 to 2.36 for a file availability increase from 0.9 to 0.99. In contrast, when using conventional replication the replication factor increases from 4 to 7. This shows that using erasure codes makes the system more scalable than conventional replication as the required availability gets higher and higher.

The graph in Figure 6 shows how the required stretch factor varies with different values of means of host availability μ_H and different values of required file availability.

Validation. To validate our analysis of erasure coding with randomized placement, we simulated the file availabilities resulting from our placement algorithm and compared them to the required file availability. Figure 7 shows the density of the deviations of the simulated availabilities from the required availabilities. As the figure shows, a few files have a lower avail-

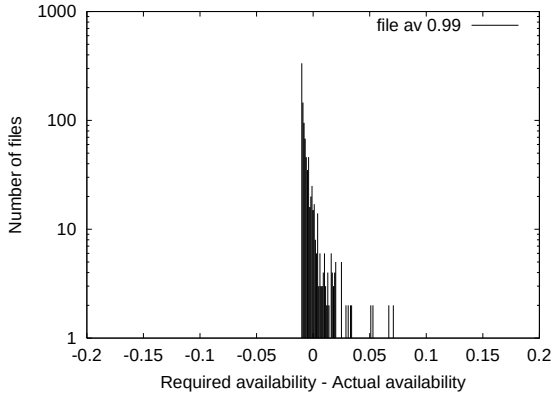

Figure 7: The density of deviations from the required availability for files for erasure coded replication. The data is gathered from a simulation of 1000 files with the required availability per file = 0.99.

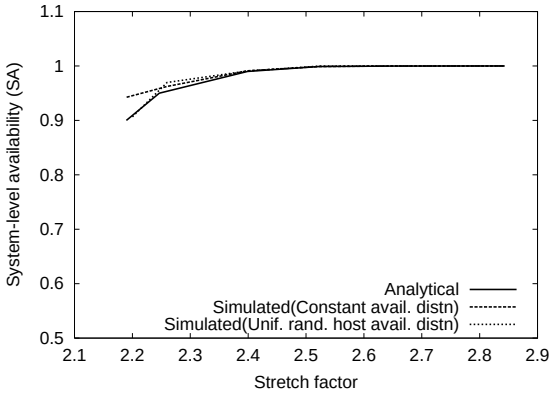

Figure 8: This graph shows how close the simulations of SA are to the required availability for conventional replication (shown by the bold line).

ability than the required availability, though most of them show equal or better availability than the required value. One reason for the lower availability is the fact that all our analyses were based on the assumption that only one block per file is placed on a host. However, when randomly placing blocks on hosts, some hosts might have more than one block of a file placed on them. This is a manifestation of the well-known birthday paradox [6]. In Section 7, we discuss how to compensate for this behavior in practice so that all the system will provide the required availability for all files.

5.2 System-level availability

System-level availability SA when using erasure coding is similar to the analysis for the conventional replication case (Section 4.2): if all files in the system have stretch factors providing a minimum availability A, then SA is equal to A.

Validation. To validate our analyses, we simulated system-level availability using two different distributions of host availability, constant and uniform random. Figure 8 shows the results of these simulations. As in the conventional replication case, the variations in the distribution of host availabilities do not affect the predicted system-level availability, thereby maintaining the expected value of SA through all our simulations.

5.3 Storage requirements

In terms of per-host storage, the analysis is again very similar to that in the conventional replication case and we will not repeat it here. For erasure coding, the total number of randomly placed blocks in the system is cfb. Hence the mean number of blocks per host μ_Z is cfb/n. Using Table 3 again, we can say that with c = 2.49, f=1000, b=1000 and n=1000, with 0.9 probability, hosts will store less than or equal to 276 blocks.

Validation. Figure 9 compares the results of simulating random block placement with our analytic model in terms of the storage used per host. It shows the cumulative distribution of storage used per host for 1000 files, with stretch factor 2.36, yielding a 0.99 file availability. As with the conventional replication case, we fit a normal distribution to this curve and use a constant block size of 7 MB for this fit. From this distribution, we get that an individual host will store less than 1747 MB with probability 0.9. Contrast this with the storage per host figure we got for the conventional replication case, which was 7700 MB. This shows that for the same level of availability, for our test case, the conventional replication strategy takes up roughly 4.5 times the storage required for the erasure coded case.

6 Long-term availability

As we have described in earlier sections, the differentiation of long-term and short-term availability is important. There is a constant inflow and outflow of users in a peer-to-peer system, that is not captured by a short-term availability model. Here, we introduce the concept of rate of decay of the system, which is the rate at which users “die” in the system, i.e. they leave never to come back. We have considered various different kinds of decay patterns. Due to the lack of relevant

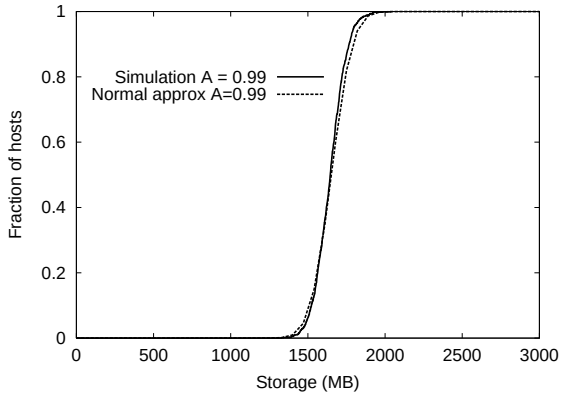

Figure 9: Cumulative distribution function of the storage per host for erasure coded replication.

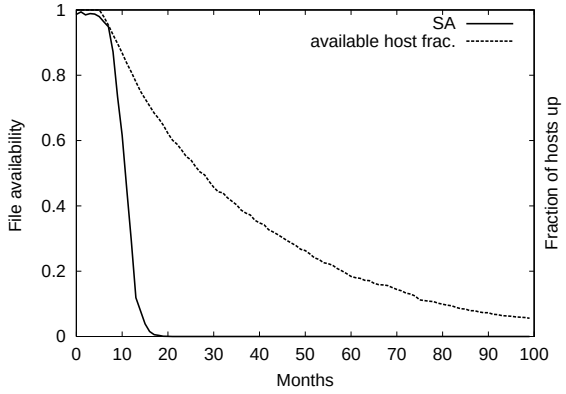

Figure 10: The decay in a peer-to-peer system affects the system availability drastically.

data on the rate of decay in a peer-to-peer system, we decided to use a pessimistic decay distribution, namely the exponential distribution. However, to mitigate the pessimistic approach to number of hosts decaying, we offset the exponential distribution by 3 months, which essentially means that in our simulation, no users disappear in less than 3 months.

In the best case, we can say that hosts in the system die when computers become too old to use. Let us make the reasonable assumption that the average lifetime of a PC is 3 years, or 36 months. Assuming that users in a peer-to-peer system decay follow an exponential distribution with rate 1/36, we performed simulations of system-level availability SA for the system using erasure coded replication for stretch factor 2.36 and file availability 0.99. We measure how it deteriorates over time. Figure 10 shows that the system-level availability SA drops drastically after the first 10 months, and is almost zero after 20 months. This indicates that periodic refreshes are needed in the system

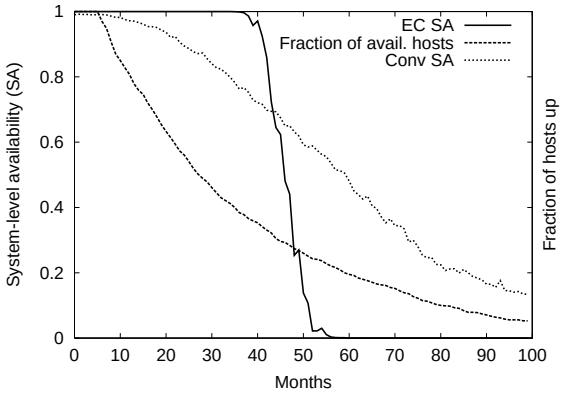

Figure 11: Decay in a peer-to-peer system affects both replication strategies, but in different ways, as shown in this graph.

to keep files in the system available over long periods of time. For example, if we refresh all files after every ten months, we will still get a very high level of system availability. Also, the time between refreshes can be lengthened while not compromising on availability if a larger stretch factor is used.

We also simulated both replication strategies with the same stretch factor/replication factor of 7, to see how their reaction to long-term decay compared. The results are shown in Figure 11. The erasure coded replication maintains a high SA until 40 months have passed before dropping off steeply, while the in the conventional replication case, the drop in SA is gradual, and after month 50, it is actually better than the erasure coded replication.

This can be explained in the following way. Recall that in the erasure coded case, the fraction of hosts available (W) out of cb hosts that hold blocks of the file follows a normal distribution with mean μ_H and variance $\mu_H(1-\mu_H)/cb$. The larger the value of cb, the lower the variance. In the case of conventional replication, the fraction of hosts available follows a much shallower distribution, since there are not that many hosts that contain replicas of the file. Figure 12 shows the two distributions.

In the erasure coded case, to recover the file b or more hosts must be available. As hosts die, b corresponds to an increasingly larger fraction of all hosts. Given the sharply concentrated probability distribution for erasure coding, even a small decrease in the number of hosts has a dramatic effect on file availability. By contrast, the less concentrated distribution of the conventional replication case is much less sensitive to decreases in the total number of hosts.

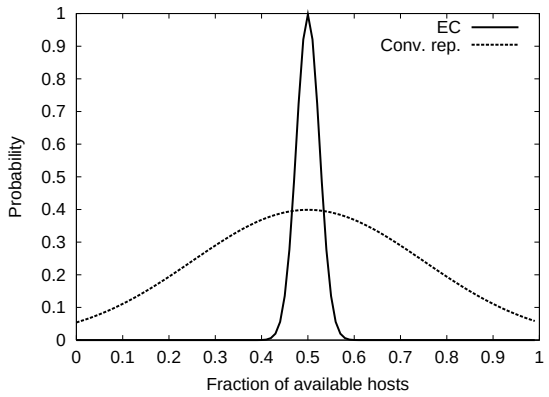

Figure 12: A probability distribution function of the fraction of hosts available. The steepness of the curve for erasure coded replication causes the drastic decrease in availability, while for conventional replication, the decay is gradual

If we require the system to be resilient to such decay, we need to periodically refresh it. Going back to Figure 11, for these particular simulation parameters, refreshing every 40 months approximately maintains roughly system-level availability of 0.9 for the erasure coded case. However, in the conventional case, though the overall drop in availability is more gradual, we would need to refresh every 20 months or so to maintain a system-level availability of 0.9.

7 Practical issues

The analyses and simulations up to this point have assumed an idealized system that ignores some practical issues that an actual implementation would need to address. In this section, we describe these implementation issues and how we compensate for idealized assumptions we made in the system model.

Variable file sizes: In our model for the use of erasure coded blocks, we have assumed that all files have the same number of blocks, b, and that at most one block of a file is stored on a host. This assumption holds if the system supports variable block sizes such that we can partition all files into the same number of blocks. However, it is often much more convenient to implement storage systems with a fixed block size.

We can accommodate the constraint of fixed-sized blocks by placing multiple blocks per host. Since the availability of a file depends only on the number of hosts on which its blocks are placed, and not on the size of the blocks themselves, it does not matter whether we place one large block per host or multiple smaller fixed-size blocks per host as long as the distribution of data across the hosts is even.

More formally, the availability of a file F_i that has b pre-encoded blocks and cb encoded blocks, with one block placed per host, is equal to the availability of a file F_j that has bl pre-encoded blocks and cbl encoded blocks, with l blocks placed per host; c is the same stretch factor for both files. The number of hosts storing blocks of F_i is cb, and, for F_j, this number is cbl/l, or also cb. Note that the number of hosts responsible for the file is the same in both cases. Now, for F_i to be available at any given time, we need b blocks, and since there is only one block per host, the number of hosts that need to be up is b or more out of the cb hosts. For F_j to be available, we need bl blocks. Since each host stores l blocks, we need b or more hosts to be available. In both cases, the minimum fraction of hosts that need to be available is $b/cb = 1/c$ (see Equation 8 in Section 5.1) and they have the same availability.

For example, assume that F_i is a 700 MB file and F_j is a 7 GB file. If F_i is divided into a 100 blocks, each of size 7 MB, and put one block per host, and F_j is divided into 1000 blocks each of 7 MB as well, and we put those 10 blocks per host, both files will have the same availability.

Restricting replication: When using erasure coding, the system partitions files into blocks. If the system uses fixed-size blocks, then large files will result in the distribution of blocks to many hosts. Distributing blocks widely has a number of benefits, such as increasing file availability and providing additional opportunity for parallel downloads. However, the system may want to restrict the number of hosts on which it distributes blocks for a given file so that, e.g., it can limit the number of hosts that need to be contacted to download the file.

Restricting the maximum number of hosts on which to distribute blocks is entirely a system design decision, and the model easily accommodates the restriction. Analyses using the model assume that erasure-coded blocks are distributed to at least 16 hosts (see Section 5.1).

Time dependence: Our model for short-term file availability assumes that the host availability distribution is stationary. In practice, though, the host availability distribution varies throughout the day. Figure 13 shows the number of Gnutella sessions and hosts over a period of three days based upon measurements we made of host and application availability of

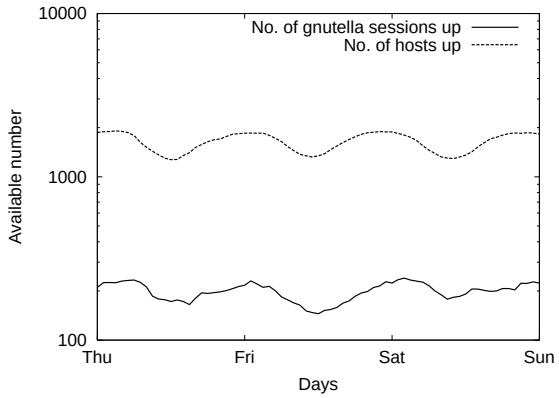

Figure 13: The two graphs show how many of the probed hosts were running gnutella, and how many were available, but not necessarily running gnutella over time. Both show a strong daily periodicity.

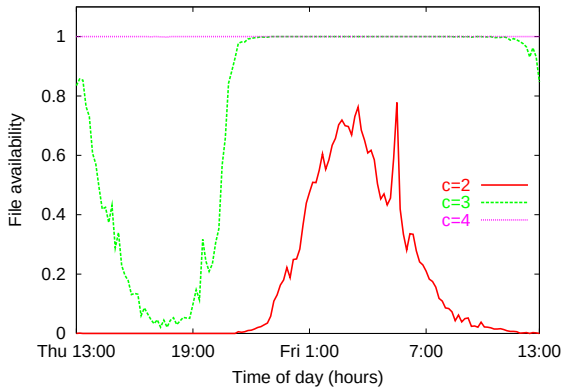

Figure 14: Increasing the stretch factor c to compensate for time-of-day effects in the host availability distribution.

the Gnutella system using techniques similar to those described in [18] in March, 2002. To account for these time-of-day effects, the system can compensate by using a conservative stretch factor c. Rather than choosing c to attain a desired degree of file availability according to the mean host availability across the entire day, we can choose c according to the host availability distribution of just those hosts available at night (when the fewest number of hosts are available). Although the larger stretch factor will result in a higher file availability than needed during the day, it will ensure that the system can always attain the minimum file availability required across the entire day.

Figure 14 shows the effect of adjusting the stretch factor to compensate for time-of-day effects. The figure shows the results of simulating file availability for three different stretch factors using the first day of a trace of Gnutella host availability.

Using a stretch factor of 3, for example, results in a file availability close to 1 for much of the day, but a larger stretch factor is needed to have the desired file availability for the entire day.

Block propagation: The fact that the number of hosts available changes throughout the day also impacts block placement. An implicit assumption in the model is that all hosts are available when the system randomly assigns blocks to hosts. However, the fundamental nature of the problem is that hosts are only available during a part of the day. As a result, at the time a file is introduced the system is likely to randomly choose to place a subset of its encoded blocks on hosts that are not available at that instant. In practice, the system can address this problem in two ways.

First, it can rely upon the conservative stretch factor used to compensate for time-of-day effects above, and propagate blocks onto hosts as they become available. When randomly choosing hosts on which to place blocks, the system will choose among all hosts in the system. For those hosts that are available, their blocks will be stored immediately. For those hosts that are unavailable, the system will simply wait and propagate the blocks for those hosts when they become available. Until they become available, the system can store those blocks on currently available hosts.

Of course, the file will not achieve maximum availability until all of its blocks propagate to their hosts. However, to compensate for time-of-day effects, the system will have chosen a stretch factor c to achieve the desired file availability assuming the worst host availability distribution for the day. Consequently, the system will provide at least the desired file availability just with those hosts that are instantaneously available at the time the file is introduced into the system.

Alternatively, when the system tries to place a block on a host for block placement, it can determine whether the host is unavailable. If a host is unavailable, the system can randomly choose another host on which to place the block. This approach assumes that host availability is identically and independently distributed, in which case host selection should still appear to be a random subset from the original host availability distribution.

Finite storage: Although we use erasure coding as an efficient mechanism to achieve redundancy and random placement to evenly distribute storage across hosts in the system, host storage capacity is finite. As

a result, when the system randomly chooses hosts on which to place blocks, it is possible that some hosts chosen do not have the capacity to store those blocks. The system can deal with this problem in similar ways as with the block propagation issue.

First, as with the block placement issue, when the system chooses a host for block placement the host can explicitly inform the system that it is unable to store the assigned block. The system can then randomly choose another host. Assuming that the amount of storage per host is not correlated with its availability, host selection again should still appear to be a random subset from the original host availability distribution.

Second, the system can further overestimate the stretch factor by a small amount to compensate. In this case, hosts that have reached their storage capacity can silently drop blocks assigned to them. The larger stretch factor will distribute more blocks than necessary, compensating for the fact that some of them will be dropped. The increase in stretch factor will depend on how frequently hosts reach their storage capacity.

8 Conclusions

In this paper, we have analyzed techniques for providing availability in peer-to-peer systems. We have developed a probabilistic model for reasoning about the storage overhead required to deliver a specified level of availability. Our model can accommodate any stationary distribution for host availability and is therefore appropriate for accommodating the heterogeneity in conventional peer-to-peer systems. We show that under such challenging host availability distributions, erasure coding is highly efficient since it requires considerably less storage than conventional replication to provide a given level of file availability. However, we also demonstrate that erasure coding experiences a dramatic phase transition as hosts are removed from the system and therefore can degrade very quickly if additional redundancy is not added to the system as persistent failures occur. Finally, we address some of the practical tradeoffs in implementing replication strategies in a peer-to-peer system and show how to accommodate deviations from the pure form of our model without unduly compromising availability. In conclusion, we have shown that high levels of availability are attainable, even in peer-to-peer systems with pervasive failure, in exchange for a modest overhead in redundant storage.

References

[1] 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. In *Measurement and Modeling of Computer Systems*, pages 34–43, 2000.

[2] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM*, pages 56–67, 1998.

[3] Y. Chen. A prototype implementation of archival intermemory. In *Proceedings of the Fourth ACM International Conference on Digital Libraries*, 1999.

[4] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)* , 2001.

[5] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, 2000.

[6] R. Durrett. *Probability: Theory and Examples*. Brooks/Cole Publishing Company, 1991.

[7] edonkey homepage, http://edonkey2000.com.

[8] Gnutella homepage, http://www.gnutella.com.

[9] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *Proceedings of the Fourth International Workshop on the Web and Databases (WebDB '2001)*, June 2001.

[10] Kazaa homepage, http://www.kazaa.com.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, 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. In *Proceedings of ACM ASPLOS*, 2000.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *To appear in Proceedings of 16th ACM International Conference on Supercomputing*, 2002.

[13] A. D. R. Marc Waldman and L. F. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[14] Napster homepage, http://www.napster.com.

[15] V. Pless. *Introduction to the theory of error-correcting codes*. John Wiley and Sons, 3rd edition, 1998.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[17] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[18] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN*, 2002.

[19] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[20] Swarmcast homepage, http://sourceforge.net/projects/swarmcast.

[21] H. Weatherspoon and J. Kubiatowicz. Erasure coding v/s replication: a quantitative approach. In *Proceedings of the First International Workshop on Peer-to-peer Systems*, 2002.

[22] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB-CSD-01-1141, U. C. Berkeley, April 2000.

A Appendix

We show the derivation of the value of the stretch factor for the erasure coded case in this appendix. We restate Equation 8 here, and go on to show the derivation.

$$1/c = \mu_W - k\sigma_W$$

We know that $\mu_W = \mu_H$ and $\sigma_W^2 = \mu_H(1-\mu_H)/cb$. Putting this into the equation 8, we get

$$1/c = \mu_H - k\sqrt{\frac{\mu_H(1-\mu_H)}{cb}}$$

This gives us a quadratic in terms of $\sqrt{c}$, solving which and eliminating impossible roots, we get

$$c = \left(\frac{k\sqrt{\frac{\mu_H(1-\mu_H)}{b}} + \sqrt{\frac{i^2\mu_H(1-\mu_H)}{b} + 4\mu_H}}{2\mu_H} \right)^2$$