Autonomous Replication for High Availability in Unstructured P2P Systems*

Francisco Matias Cuenca-Acuna, Richard P. Martin, Thu D. Nguyen
{*mcuenca, rmartin, tdnguyen*}*@cs.rutgers.edu*
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

Abstract

We consider the problem of increasing the availability of shared data in peer-to-peer systems. In particular, we conservatively estimate the amount of excess storage required to achieve a practical availability of 99.9% by studying a decentralized algorithm that only depends on a modest amount of loosely synchronized global state. Our algorithm uses randomized decisions extensively together with a novel application of an erasure code to tolerate autonomous peer actions as well as staleness in the loosely synchronized global state. We study the behavior of this algorithm in three distinct environments modeled on previously reported measurements. We show that while peers act autonomously, the community as a whole will reach a stable configuration. We also show that space is used fairly and efficiently, delivering three nines availability at a cost of six times the storage footprint of the data collection when the average peer availability is only 24%.

1 Introduction

Peer-to-peer (P2P) computing is emerging as a powerful paradigm for sharing information across the Internet. However, we must address the problem of providing high availability for shared data if we are to move P2P computing beyond today's sharing of music and video content. For example, providing high availability is clearly critical for P2P file systems. Another practical motivating example is Google's cached pages: it is extremely annoying to find that a highly ranked page is not available because the server is currently down. These cached pages in effect increase the availability of web content.

At the same time, recent measurements suggest that P2P communities are fundamentally different from current servers [2, 21]; for example, Saroiu et al. report an average peer availability of only 24% [21]. Such low availability implies that providing practical availability for shared data, say 99-99.9%, which is comparable to today's web services [15], would be prohibitively expensive storage-wise using traditional replication methods. Yet, requiring that data be moved or re-replicated as peers leave and rejoin the online community would be prohibitively expensive bandwidth-wise. For example, replicating a file 6 times (i.e. 1 original copy plus 6 replicas) when the average node availability is only 24% would only raise its availability to 85%. Moreover, if a thousand nodes were to share 100GB (700GB after replication), the bandwidth required to keep all replicas online as members join and leave the online community would be around 4GB per node per day.

We are thus motivated to study how to improve the availability of shared data for P2P communities where individuals may be disconnected as often as being online. In particular, assuming that peers eventually rejoin the online community when they go offline, we address the question: is it possible to place replicas of shared files in such a way that, despite constant changes to the online membership, files are highly available without requiring the continual movement of replicas? To answer this question, we propose and evaluate a distributed replication algorithm where all replication decisions are made autonomously by individual members using only a small amount of loosely synchronized global state. To maximize the utility of excess storage, we assume that files are replicated in their entirety only when a member hoards that file for disconnected operation. Otherwise, files are replicated using an erasure code [24]. While the use of an erasure code is not novel, we show how to avoid the need for tracking the placement of specific fragments. In particular, we show how to increase the availability of a file by adding new random fragments rather than regenerating individual fragments that have been lost.

We deliberately study a very weakly structured system because tight coordination is likely difficult and costly in large distributed and dynamic communities [14]. Fundamentally, our approach only depends on peers managing their individual excess storage in a fair manner and having approximate data about replica-to-peer mapping and average peer availability. The need for information on replica-to-peer mapping is obvious. With respect to peer availability, without this information, a replication algorithm cannot differentiate between peers with very different availabilities and thus it may under- or over-replicate files. Beyond this loosely synchronized global state, all decisions are local and autonomous. Of course, one can increase the level of coordination between peers to increase the efficiency of the system in utilizing storage and bandwidth. In essence, we seek to conservatively estimate the amount of storage required to provide a practical availability of 99.9%, which, as already mentioned, is comparable to today's web services.

*PlanetP was supported in part by NSF grants EIA-0103722 and EIA-9986046.

Our contributions include:

- demonstrating that it is possible to increase availability of shared data to practical levels, e.g., 99.9%, using a decentralized algorithm where members operate autonomously with little dependence on the behaviors of their peers.
- showing that the presence of a small number of highly available members can significantly reduce the replication necessary to achieve practical availability levels. In particular, our simulation results show that a community based on Saroiu et al.'s measurements [21], where the average availability is only 24% but members' availability differ widely, can achieve a minimum file availability of 99.8% if the excess storage is at least 6 times the size of the shared data set (for brevity, we refer to this as 6X excess storage), whereas a community with average availability of 33%, but where all peers look alike, requires 9X or more excess storage to achieve the same availability.
- presenting the detailed design of one particular decentralized and autonomous replication algorithm. Our scheme does not build or maintain any distributed data-structures such as a hash table. Instead, our approach relies only on peers periodically gossiping a very compact index and availability information [7]. In addition to the results just mentioned, for a community with a higher average peer availability of 81%, we can achieve 99.8% minimum availability with only 2X excess storage capacity.

2 Background: PlanetP

We begin by briefly describing PlanetP [6, 7], a gossiping-based publish/subscribe infrastructure, since we assume its use for maintaining the loosely synchronized global data needed for our algorithm[1]. Members of a PlanetP community publish documents to PlanetP when they wish to share these documents with the community. PlanetP indexes the published documents, along with any user-supplied attributes, and maintains a detailed inverted index *local* to each peer to support content searches. (We call this the *local index*.)

In addition, PlanetP implements two major components to enable community-wide sharing: (1) an infrastructural gossiping layer that enables the replication of shared data structures across groups of peers, and (2) a content search, ranking, and retrieval service. The latter requires two data structures to be replicated on every peer: a membership directory and a content index. The directory contains the names and addresses of all current members. The global content index contains term-to-peer mappings, where the mapping $t \rightarrow p$ is in the index if and only if peer p has published one or more documents containing term t. The global index is currently implemented as a set of Bloom filters [3], one per peer that summarizes the set of terms in that peer's local index. All members agree to periodically gossip about changes to keep these shared data structures weakly consistent.

To answer a query posed at a specific node, PlanetP uses the local copy of the global content index to identify the subset of peers that contain terms relevant to the query and passes the query to these peers. The targeted peers evaluate the query against their local indexes and return URLs for relevant documents to the querier. PlanetP can contact all targets in order to retrieve an exhaustive list or a ranked subset to retrieve only the most relevant documents.

Using simulation and measurements obtained from a prototype, we have shown that PlanetP can easily scale to community sizes of several thousands [7]. The time required to propagate updates, which determines the window of inaccuracy in peers' local copies of the global state, is on order of several minutes for communities of several thousand members when the gossiping interval is 30 seconds. Critically, PlanetP dynamically throttles the gossiping rate so that the bandwidth used in the absence of changes quickly becomes negligible.

3 Autonomous Replication

In our replication approach, each member of a community *hoards* some subset of the shared files entirely on their local storage, called the member's hoard set, and *pushes* replicas of its hoard set to peers with excess storage using an erasure code. We propose such a structure to support disconnected access to shared data. In loosely organized applications such as current file sharing, hoarding is uncoordinated and entirely driven by members' need for disconnected access. In more tightly organized applications, such as a file system, the application can coordinate the division of the shared data set among individuals' hoard sets, in essence dividing the responsibility for ensuring the availability of the shared data.

To simplify our description, we introduce a small amount of terminology. We call a member that is trying to replicate an erasure-coded fragment of a file the *replicator* and the peer that the replicator is asking to store the fragment the *target*. We call the excess storage space contributed by each member for replication its *replication store*. (Note that we do not include replication via hoarding as part of the replication store.) We assume that each file is identified by a

[1]Although we have assumed PlanetP as the underlying infrastructure for this study, one could also use infrastructures based on distributed hash tables to the same effects [20, 22, 17]. The main advantage of using PlanetP is that updates to the global state are automatically propagated, where groups of updates occurring close together are batched for efficiency. If we instead use a DHT-based system, we would have to poll nodes periodically for updates even if there were no changes.

unique ID. Finally, when we say "the availability of a fragment," we are referring to the availability of the file that the fragment is a piece of.

Given this terminology, the overall algorithm is as follows:

- Each member advertises the unique IDs of the files in its hoard set and the fragments in its replication store in the global index. Each member also advertises its average availability in the global directory.
- Each member periodically estimates the availability of its hoarded files and the fragments in its replication store.
- Periodically, say every T_r time units, each member randomly selects a file from its hoard set that is not yet at a target availability and attempts to increase its availability; the member does this by generating a *random* erasure coded fragment of the file and pushes it to a randomly chosen target.
- The target accepts and saves the incoming fragment if there is sufficient free space in its replication store. If there is insufficient space, it either rejects the replication request or ejects enough fragments to accept the new fragment. Victims are chosen using a weighted random selection process, where more highly available fragments are more likely to be chosen.

Our goal in designing this algorithm is to increase the availability of all shared files toward a common target availability while allowing peers to act completely autonomously using only a small amount of loosely synchronized global data. Given a replication store that is very large compared to the set of documents being shared, we know that this approach will work [19]. The interesting questions become what is the necessary ratio of the replication store to the size of the document set and what happens when the replication store is not sufficiently large for the community to achieve the target availability for all files. We explore these questions in Section 4. In the remainder of this section, we will discuss our use of erasure coding, how to estimate file availability, our replacement policy, and the resiliency of our approach to misbehaving peers.

3.1 Randomized Fragmentation and Replication

We use the Reed Solomon (RS) erasure coding in a novel way to support autonomous member actions. The basic idea of any erasure code is to divide a file into m fragments and recode them into n fragments, where $m < n$, in such a way that the file can be reassembled from any m fragments with an aggregated size equal to the original file size [18].

To date, given (n, m), most uses of erasure codes generate all n fragments and, over time, detect and regenerate specific lost fragments. This approach has three significant disadvantages for highly dynamic environments: (i) as the average per-member availability changes over time, files' availability will change given a fixed n; to maintain a target availability, it may thus be necessary to change n, necessitating the re-fragmenting and replication of some (perhaps all) files; (ii) an accurate fragment-to-peer mapping is required for the regeneration of fragments lost due either to peers leaving the community permanently or ejection from the replication store; and (iii) it must be possible to differentiate accurately between peers temporarily going offline and leaving the community permanently to avoid introducing duplicate fragments, which reduces the effectiveness of erasure coding.

To overcome these limitations, we choose $n >> m$ but do *not* generate all n fragments. When a member decides to increase the availability of a file, it simply generates an additional *random* fragment from the set of n possible fragments. If n is sufficiently large, the chances of having duplicate fragments should be small, thus maximizing the usefulness of every fragment generated in spite of not having any peer coordination. In this manner, it is easy to dynamically adjust the number of fragments generated for each file to reflect changes in the community.

RS is particularly suited to our proposed use because the cost of generating each fragment is independent of n. The fundamental idea behind RS is that a polynomial of degree $m - 1$ in a Galois field $GF(2^w)$ is uniquely defined by any m points in the field. In order to create an erasure code for the blocks $D_1, ..., D_m$ of a file, we need a polynomial p such that $p(t_1) = D_1, ..., p(t_m) = D_m$. Once we have this polynomial, it is easy to create up to $2^w - m$ additional points $p(t_i) = D_i,\ i > m$ such that the polynomial can be reconstructed from any combination of m items from the set $\{(t_1, D_1), ..., (t_m, D_m), ..., (t_i, D_i), ...\}$. Observe that, given the polynomial, the generation of any one fragment is independent of n as desired. According to Rizzo [18], files can be encoded/decoded on a Pentium 133Mhz at 11MB/s. Moreover using w's up to 16 is quite feasible, which translates into a 0.006 probability of having collisions for the environments studied in Section 4.

3.2 Estimating the Availability of Files

To provide a global mapping of file placement, whenever a member m hoards a file f, it constructs a term that is a concatenation of the word *File_* and a hash of f's content *Hash*(f), and insert the mapping *File_Hash*$(f) \rightarrow m$ into the global index. Likewise, if m accepts a fragment of file f for storage, it inserts the mapping *Frag_Hash*$(f) \rightarrow m$ into the global index. These mappings are of course removed if m stops hoarding f or evicts the fragment of f that it previously accepted. We further assume that peers advertise their average online and offline times in the global directory.

Given the above information in the global index, we can identify the set of peers hoarding any file f, $H(f)$, and those that contain a fragment of f, $F(f)$. Then, assuming that peers' behaviors are not correlated [2, 4], the availability of f, $A(f)$, can be estimated as 1 minus the probability that all nodes in $H(f)$ are simultaneously offline *and* at least $n-m+1$ of the nodes in $F(f)$ are also offline; m is the number of fragments required to reconstruct f and n is re-defined as $n = |F(f)|$. In general, since every peer in the system may have a different probability for being offline, say P_i for peer i, it would be too expensive to compute the exact file availability. Thus, we instead use the following approximation that uses the average probability of being offline (P_{avg}) of peers in $F(f)$:

$$A(f) = 1 - \prod_{i \in H(f)} P_i \sum_{j=n-m+1}^{n} \binom{n}{j} P_{avg}^{j} \left(1 - P_{avg}\right)^{n-j} \tag{1}$$

where

$$P_i = \frac{\textit{avg. offline time}}{(\textit{avg. online time} + \textit{avg. offline time})}$$

$$P_{avg} = \frac{1}{n} \sum_{i \in F(f)} P_i$$

Note that equation 1 assumes that $H(f)$ and $F(f)$ do not intersect and that all n fragments reside on different peers. We ensure this by allowing a peer to either hoard an entire file or store only a single fragment of that file.

In addition, we are assuming a closed community, where members may go offline but do not permanently leave the community. Currently, we assume that members will be dropped from the directory if they have been offline for some threshold amount of time. Thus, when members permanently leave the community, the predicted availabilities of files may become stale. Since we periodically refresh the predicted availability of files, this should not become a problem.

Finally, note that equation 1 does not account for the possibility of duplicate fragments; as already argued, however, we can make the chance of having duplicate fragments quite small and so the impact should be negligible.

3.3 Replacement

When a target peer receives a replication request, if its replication store is full, it has to decide whether to accept the incoming fragment, and if it does, select other fragments to evict from its replication store. Because we are trying to equalize the availability of all files across the system, we would like to eject fragments with the highest availability. However, if we use a deterministic algorithm then multiple peers running the same algorithm autonomously may simultaneously victimize fragments of the same file, leading to drastic changes in the file's availability. Thus, we instead use a weighted random selection process, where fragments with high availability have higher chances of being selected for eviction.

Our replacement policy is as follows. We first compute the average number of nines in the availability of all fragments currently stored at the target. Then, if the incoming fragment's number of nines is above 10% of this average, we simply reject it outright. Otherwise, we use lottery scheduling [23] to effect the weighted random selection of victim fragments. In particular, we create a set of tickets and divide them into two subsets with the ratio 80:20. Each fragment is assigned an equal share of the smaller subset. In addition, fragments with availability above 10% of the average are given a portion of the larger subset. The portion given to each fragment is proportional to the ratio between its number of nines and the sum of the number of nines of all such fragments. The notion of different currencies makes this division of tickets straightforward. For example: if a target node has three fragments with availabilities 0.99, 0.9, 0.5 or 2, 1, .3 "nines" respectively[2] then the average availability in nines plus 10% is 0.76. Now if we have 100 lottery tickets the first fragment will get 67+6.6 tickets from the first and second pool respectively, the second fragment will get 13+6.6 tickets and the third fragment will get 0+6.6 tickets. Overall, the probability of each fragment being evicted will be 0.73, 0.19 and 0.06 respectively.

The intuitions behind our replacement policy are as follows. First, we reject the incoming fragment if it will simply become a target for eviction the next time a replication request is received by the target peer. Without this condition, we will simply be shifting fragments around without much effect. Our threshold for this outright rejection may seem rather low; at some cost of bandwidth, if we were less aggressive at rejecting fragments, perhaps over time, the system can reach a better configuration. However, our experimentation shows that while this threshold affects bandwidth usage, it does not significantly affect the overall replication[3]. Since we are targeting environments where bandwidth may be a precious commodity (see Section 4), we decided that an aggressive threshold was appropriate.

Next, we penalize over-replicated files heavily for the number of nines in their availability, making it highly probable that a fragment of an over-replicated file will be evicted. We use the number of nines rather than the availability itself because it linearizes the differences between values, i.e. the difference between 0.9 and 0.99 is the same as that between 0.99 and 0.999.

3.4 Optimizations at Replicators

While the critical decisions rest at the targets in our algorithm, replicators can implement several optimizations to increase the convergence rate, including favoring files that

[2] $\textit{No. nines} = -\log_{10}(1 - \textit{availability})$

[3] We experimented with various ratios between the two pools as well as the threshold for rejecting a fragment outright. Space constraints prevent us from showing these results. However, as long as these parameters reflect the two motivating intuitions, changes had very limited impact.

have low estimated availability and trying to find peers with free space in their replication store rather than causing an eviction at a peer whose replication store is full. We refer the interested readers to [5] for details on these optimizations.

3.5 Resiliency to Misbehaving Peers

We believe that our current scheme is tolerant of a small number of buggy peers or members that are trying to take advantage of peers' replication stores for use as remote storage. First, consider buggy peers that might corrupt fragments in their replication stores. Our use of an erasure code includes embedded error detection so that a corrupted fragment cannot harm the integrity of the file; rather it would only lead to wasted space and lower than expected availability for the corresponding file.

Next, consider buggy or greedy members that attempt to push already highly available files or push files at a higher than agreed upon rate. Because replacement decisions are made at the targets, such actions will waste bandwidth and computing power but otherwise should not affect the efficiency of how the aggregated replication store is used.

Buggy or greedy peers can also provide misleading information along two dimensions: their average availability and whether they have a particular file. The combination that may be most damaging to our algorithm is when a peer advertises very high availability and that it is hoarding a significant portion of the shared data collection. In this case, the community is likely to over-estimate the availability of a significant number of files and so do not replicate them sufficiently. This frees up remote storage for use by the faulty/greedy peer. However, it is actually not in the self-interest of a member to do this because he would get numerous requests for the files he claims to be hoarding; this member is thus giving up bandwidth resources in trying to utilize remote peers' storage. In addition, it is relatively easy to detect when a peer is faulty in this manner by verifying claims from peers that seem to have overly large amount of replicated data.

4 Evaluating the Achievable Availability

In this section we present a simulation-based study of our replication scheme's impact on file availability. We begin by describing our simulator and three distinct communities that we will study. Then, we present simulation results and discuss their implications.

4.1 Experimental Environment

We have built an event driven simulator for this study. To achieve reasonable simulation efficiency, we made several simplifying assumptions. First, we assume that all members of a community attempt to replicate files from their hoard sets at synchronous intervals. Second, we do not simulate the detail timing of message transfers and the full PlanetP gossiping protocol that is used to replicate the directory and the global index. In order to account for potential data staleness due to the latency of gossiping, we reestimate file availability and reassign tickets for the various lotteries only once every 10 minutes[4]. This batching of computations actually serves two purposes. First, it forces peers to work on outdated and inaccurate information (far more inaccurate than the latency provided by PlanetP for the sizes of communities studied here [7]). Second, it simulates a practical real implementation since estimating files' availability and giving tickets to every file and fragment is not a computationally trivial process.

Finally, we always fragment files using a Reed Solomon code with a fixed m set to 10. This means that all fragmented files can be reassembled with 10 fragments, regardless of its size. Although in practice the algorithm could vary m to achieve the best trade-off between fragment size, download latency, and space utilization, we fixed it to simplify the analysis.

4.2 Simulated Communities

We define three distinct P2P communities to assess the impact of replication under different operating environments representing different styles of communities. The first community is representative of a very loosely coupled community sharing mostly multimedia files; the second resembles a corporate department; and the third models a distributed development group. The environments vary in the following ways: (i) The per peer mean uptime and downtime; we model peer arrival to and exit from the online community as exponential arrival processes based on the given means; (ii) the number of files per node; (iii) the number of hoarded replicas per file; (iv) the amount of excess space on each node, and (v) the file sizes. Table 1 summarizes the parameters of each of the communities.

For all three communities, we vary the amount of excess space provided by each member across several different simulations. In all case we refer to excess space as a proportion of the number of bytes in members' hoard sets. In all scenarios, the number of files and nodes has been scaled to allow us to run experiments within reasonable times. As shall be seen, since nodes provide excess space based on what they share, the total number of files and nodes will not affect the algorithm. In all experiments, we assume a target file availability of three nines (99.9%).

File-sharing (FS). The first community that we study is modeled using two sources of data: (i) Saroiu et al.'s reported statistics collected from the Gnutella and Napster

[4]This simulated time unit is only relevant when compared with the gossiping interval, which is assumed to be 30 seconds. This 30 seconds interval can be thought of as the unit time. For example, if we map this unit to 60 seconds, then all time periods reported in our results would simply be multiplied by two.

IEEE COMPUTER SOCIETY

	File Sharing (FS)	Corporate (CO)	Workgroup (WG)
No. Members	1,000	1,000	100
No. Files	25,000	50,000	10,000
Files per Member Dist.	Weibull(sh:1.93,mean:25)	Weibull(sh:6.33,mean:50)	Weibull(sh:0.69,mean:100) or Uniform
File Size Dist.	Sigmoid(mean:4.3MB)	Lognormal(μ:12.2 σ:3.43)	Constant 290Kb
Hoarded Replica Dist.	mod. Pareto(sh:0.55 sc:3.25)	None	None
Node Availability	24.9% (average)	80.7% (average)	33.3% (fixed)

Table 1. *Parameters used to simulate each environment.*

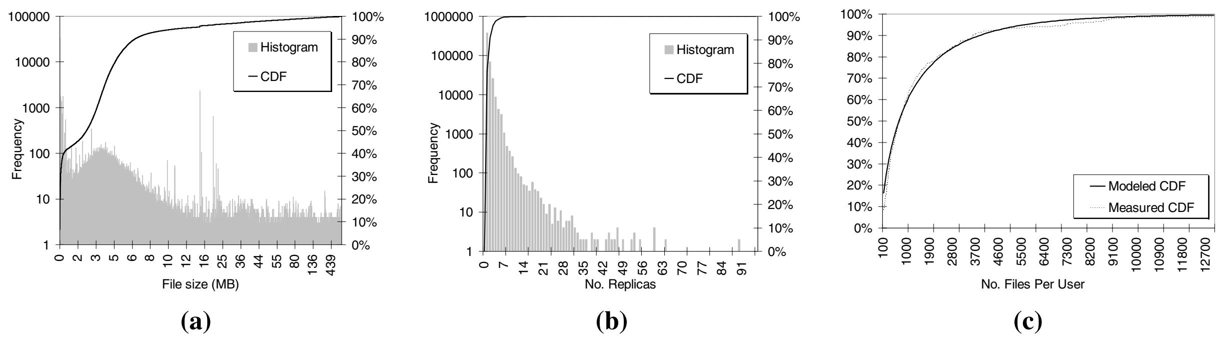

Figure 1. *(a) CDF and histogram of file sizes observed on DC. (b) CDF and histogram of replicas per file observed on DC. (c) CDF of the number of files per user observed on DC and a fitted 2 parameter Weibull distribution (shape:0.69, scale:1167, mean:1598 files per user).*

communities [21], and (ii) data collected from a local DirectConnect [9] community (DC) comprised of University students. In particular, for this community, we simulate 1,000 peers sharing 25,000 files. The number of files per peer is modeled using a Weibull distribution approximating Saroiu et al.'s reported actual distribution. File sizes are modeled using a Weibull distribution approximating our measurements of the local student community, which is shown in Figure 1(a). We cross these data sets because Saroiu et al. do not report the latter. Since the communities serve essentially the same purpose, we believe that data from one is likely to be representative of the other (we verified this by checking characteristics that were measured in both studies). The number of hoarded replicas per file was also taken from the local community and is shown on Figure 1(b).

Finally, we use Saroiu et al.'s reported Gnutella uptimes and availabilities to drive members' arrival to and departure from the online community. We made two assumptions because of insufficient data: (i) we correlate availability with average length of online times; that is, the most highly available members also have the longest average online times. This correlation is intuitive because it avoids the pathological case of a highly available member constantly and rapidly flipping between being online and offline. (ii) we correlate the file distribution with availability, where the most highly available members have the most files. This correlation is motivated by measurements on our local student community, and suggests that "server-like" peers typically hold the largest portion of the shared data.

Corporate (CO). The second community represents what we might expect to see in a corporate or university environment. In addition to being a significantly different environment than FS, studying this environment also allow us to compare our results with Farsite [4]. Farsite has somewhat similar goals to ours but takes a significant, almost inverse approach (see section 5 for more details). Thus, this community is modeled using data provided by Bolosky et al. [4]. All the parameters shown in Table 1 were taken directly from Bolosky et al.'s work with the exception of the absolute number of nodes and total number of files, which were scaled down to limit simulation time.

Workgroup (WG). Finally, the third configuration tries to capture a geographically distributed work group environment, such as a group of developers collaborating on an open-source project. The idea is to evaluate the impact of not having any member with server-like behaviors, i.e., members with large hoard sets that are (relatively) highly available. In particular, we simulate 100 peers sharing 10,000 files of 290KB each—this is the average size of non-media files found in [11]. The number of files per user is distributed either uniformly or according to a distribution approximating our measurements of the local P2P network, shown in Figure 1(c). Peers will follow a work schedule with a mean of 1 hour online followed by 2 hours offline. Again, arrival is exponentially distributed.

4.3 Other Replication Algorithms

To evaluate the impact of the amount of information assumed to be available to drive the replication algorithm, we

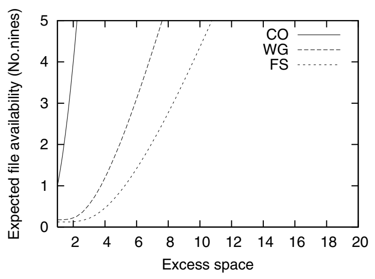

Figure 2. *Achieved file availability in number of nines vs. excess storage in terms of x times the size of the entire collection of hoarded files.*

will compare our algorithm against one called OMNI that assumes centralized knowledge, where replication is driven by a central agent that tries to maximize the minimum file availability. Comparing against OMNI quantifies the effect of autonomous actions using loosely synchronized data as opposed to having perfect, centralized information.

OMNI is implemented as a hill-climbing algorithm that tries to maximize the availability of the least available file on every step. OMNI assumes centralized/perfect knowledge of peer behaviors, file and fragment distributions, and the ability to replicate a fragment on any member at any time. This algorithm was motivated by Bolosky et al.'s work [4], who shows that in an environment like CO it produces allocations close to the optimal.

We have also compared our algorithm against one that does not use information on file and peer availability to drive replication. This comparison quantifies the effect of having approximate knowledge of current file availability. While these comparisons show that having some estimate of file and peer availability is critical, we do not show them here because of space constraints. It is intuitive that having some knowledge of availability, even if only approximate, should provide a clear win over having no availability information. We refer the interested readers to [5] for more details on this comparison.

4.4 Availability Improvements

Availability of Peers vs. Excess Storage Needed. We begin our evaluation considering two factors affecting file availability: the size of the aggregate replication store, and members' average availability. In particular, we use equation 1 as a simple analytical model.

Figure 2 shows the file availability plotted as a function of available excess space for mean peer availability equal to that of the three communities. As expected from [4, 10], only a small amount of excess space is required to achieve very high file availability in the CO community: 3 nines availability can be achieved with 1.75X excess capacity while almost 5 nines can be achieved with 2X excess capacity. Achieving high availability is much more difficult for FS-type communities because many peers only go online briefly to locate and download information: around 8X of excess capacity is required for 3 nines availability. As shall be seen later, however, this model is pessimistic because using average peer availability looses the importance of server-like peers that are online most of the time. Our simulation results will show that only around 1.5X and 6X of excess capacity are needed to achieve 3 nines availability on CO and FS respectively.

We have also used this model to study the effect of replicating entire files across hoard sets instead of using erasure coding [5]. These results show that replication across hoard sets has little effect on availability except in the case where peer availability on average is very high, validating our decision to use erasure coding for replication.

Overall Availability. We now consider simulation results for the three communities discussed earlier. Figure 3 shows the CDFs of file availability for all three communities; Table 2 pulls out some of the more interesting points in these CDFs. In this figure, and for the remainder of the paper, all presented file availability are computed using equation 1, using the final placement of files and fragments, with P_{avg} computed using the *expected* availability of each peer. An alternative approach would have been to present the actual measured availability; that is, divide the observed amount of time that a file was accessible by the total simulated time. However, this estimation is often optimistic because it is difficult to simulate a sufficiently long period of time to observe significant downtime for highly available members. We also studied several other estimators which we do not discuss here; we refer the interested reader to our technical report [5] for a discussion and comparison of these estimators.

As expected from our analytical study, the CO community has little trouble in achieving high file availability because the average availability of members is high. At 2X excess storage capacity, over 99% of the files have higher than 3 nines availability, with the minimum file availability of 2.73 nines (or 99.8%).

For the FS community, we see that using around 6X excess storage capacity, we achieve 3 nines availability for 99% of the files, with the minimum availability being 2.9 nines. This is significantly less than the 8X predicted by our model. Because of the presence of server-like peers that are highly available, files that are hoarded by these peers achieve high availability from the beginning without using extra storage. Further, any fragment stored at these peers increases the availability of the corresponding file significantly and reduces the total number of fragments needed.

Observe that for both CO and FS, there are subsets of files that achieve much greater availability than the average. Interestingly, this is not related to the fairness properties of our algorithm. Rather, it is because these files are naturally replicated on very highly available peers through hoarding.

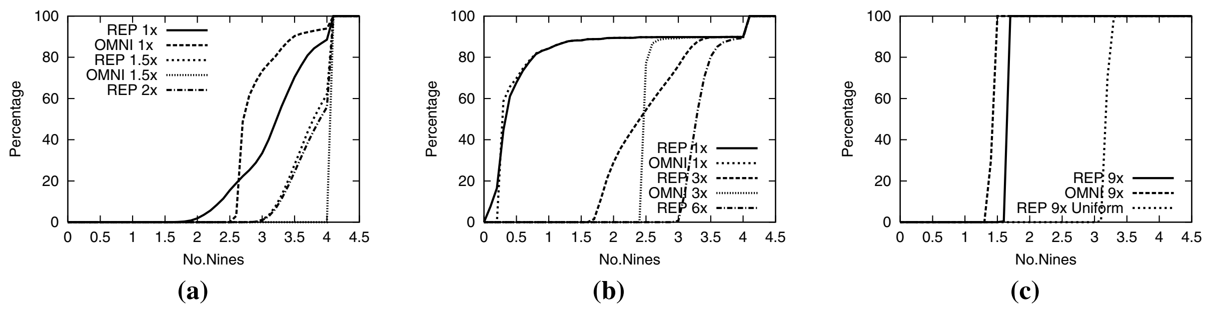

Figure 3. *CDFs of file availability for (a) the CO community with 1X, 1.5X and 2X, (b) the FS community with 1X, 3X, and 6X, and (c) the WG community with 9X excess storage capacities. REP refers to our replication algorithm as described in Section 3. OMNI is as described in Section 4.3.*

	Min	1%	5%	Avg		Min	1%	5%	Avg
CO 1X	1.2083	1.92	2.17	3.20	FS 3X	1.5944	1.69	1.75	2.55
CO 1X OMNI	2.5976	2.60	2.60	2.90	FS 3X OMNI	2.4337	2.43	2.43	2.63
CO 1.5X	1.5829	2.99	3.14	3.91	FS 6X	2.9199	3.01	3.05	3.36
CO 1.5X OMNI	4.0000	4.00	4.00	4.19	WG 9X	1.5208	1.61	1.61	1.61
CO 2X	2.7357	3.01	3.17	4.02	WG 9X OMNI	1.3518	1.35	1.35	1.41
FS 1X	0.0005	0.01	0.06	0.78	WG 9X Uniform	3.0001	3.11	3.11	3.14
FS 1X OMNI	0.2587	0.26	0.26	0.79					

Table 2. *Some specific points from the graphs shown Figure 3. In particular, we report the minimum and average availability and the 1% and 5% points, corresponding to the minimum availability of 99% and 95% of the most available files respectively.*

If we look at the number of fragments generated per file, we would see that most of these files were never fragmented at all, or had only a few fragments, which are useless since files need at least 10 fragments to increase availability. This constitutes a key difference between our assumed environment, and thus the approach to replication, and a file system such as Farsite. In the latter environment, the "excess" hoarded copies would be collapsed into a single copy and thus increasing the fragment space; however, our view of hoarded copies as user-controlled copies prevents us from taking that approach.

For the WG environment, we observe an interesting effect. When files and excess storage are uniformly distributed among peers, 9X excess capacity is enough to achieve 3 nines availability, as predicted. If files and excess capacity are non-uniformly distributed, however, then average availability drops sharply. This degradation, also observable on OMNI, arises from the small number of peers in the community and our assumed per-peer coupling between the size of the hoard-set and the replication store. In this case, peers with the most files to replicate also have the most excess storage, which is not useful to them. Further, since there are no high-availability peers that are up most of the time, it is not easy for replicators to find free space on the subset of peers with large replication stores. This points to the importance of even a few server-like peers that provide both excess capacity as well as high availability in communities similar to FS.

Comparing Against Having Centralized Knowledge. Figure 3 also shows that having centralized information allows OMNI to: (i) achieve the target availability with less excess space, and (ii) increase the minimum availability when the target availability cannot be achieved for all files. While the former is not insignificant—for example, for CO, OMNI was able to achieve 4 nines availability with only 1.5X excess storage and 2.43 nines availability with only 3X excess storage for FS—we believe the latter is a more important difference. With the continual rapid growth in capacity and drop in price of storage, having sufficient excess space on order of 2-9X does not seem outrageous. However, increasing the minimum file availability may be critical for applications like file systems. It is interesting to note, however, that occasionally, REP can outperform the heuristic-based OMNI; for example, for WG 9X where documents are distributed according to a Weibull distribution, REP achieves a minimum availability of 1.5 where as OMNI can only achieve 1.35.

One main reason why we cannot approach OMNI more closely for minimum availability in most instances, is that members with low availability cannot maintain high availability for files in their hoard sets when there is insufficient excess storage. Over time, peers that are more available win the competition for excess storage because they have the advantage of being able to push their hoarded files more often.

IEEE COMPUTER SOCIETY

This is a direct consequence of our desire for autonomous peer actions. One potential approach to alleviate this problem is for more highly available peers to also hoard these files so that they can "lend a hand" in pushing replicas.

Bandwidth Usage. Finally, we study the amount of bandwidth used after a community has reached stability. To put bandwidth in perspective, we will use the average number of bytes transferred per hour per node over the average file size. In essence, this metric captures the rate of replication that continues because, lacking centralized knowledge, the system is not able to discern that it should cease trying to increase availability—there is just not sufficient excess storage.

For CO, when the excess space is only 1X, REP continues to replicate on average 10 files per hour per node. If we increase the amount of excess storage to 2X the average amount of files replicated drops to zero, recall from figure 3(a) that 2X is enough to reach the target availability of 3 nines. Similarly, in FS the number of files replicated per hour per node goes from 241 to 0 as we increase the excess space from 1X to 3X (figure 3(b)).

This points out an important advantage of being able to dynamically estimate file availability: when the available excess storage is not too much less than what is required to achieve the target availability, our approach is relatively stable. This implies that while it is important to set the target availability to what the community can support—for example, it is not reasonable to target three nines availability when there is only 1X excess storage in FS as shown by the large number of files being pushed—it doesn't have to be highly accurate.

5 Related Work

A primary distinction of our work is that it considers availability for a wide range of communities that are distinct from those assumed by previous P2P systems like CFS [8], Ivy [16], Farsite [1], and OceanStore [13]. In particular, we consider communities where the level of online dynamism is high and there may be considerable offline times for members operating in disconnected mode.

In contrast to our approach of randomly generating erasure coded fragments from a large space of such fragments, OceanStore [13] proposes to replicate files using a fixed number of erasure coded fragments, which are repaired periodically but at a very low rate (on the order of once every few months [25]). In their scenario, nodes have relatively high availability and so replication, and repair of fragments, are motivated mostly because of disk failures. Conversely, in the environments we study, the wide variance of node availability makes it difficult to adopt a similar approach and also requires a more pro-active replication algorithm.

Similarly, Ivy [16] uses a distributed hash table called DHash [8] to store file blocks as well as a fixed number of replicas across a P2P community. To ensure availability, they assume that replicas are refreshed over time. Further, because the location of data in this system depends on the current set of online members, it is difficult for them to adopt a less dynamic placement strategy such as ours. We believe that the constant refresh of data would be too expensive in highly dynamic communities.

We have found that the use of adaptive erasure codes and estimated node availability plays a key role in equalizing the overall file availability. Along these lines, Farsite [4] uses information about node availability to increase minimum file availability. Because its designers are targeting a corporate LAN environment exclusively, however, they do not use erasure coding. In this environment, the average node availability can be quite high so that erasure coding is not as important. Farsite also takes a more aggressive stance toward using network bandwidth to adjust to changing node availability while better utilizing all the available disk resources. Finally, our replication scheme is appropriate for disconnected operation, where members may wish to access part of the shared data while offline. Farsite does not consider such communities.

6 Conclusions and Future Work

In this work, we have addressed the question of increasing the availability of shared files for a range of P2P communities. We have attempted to quantify a conservative estimate of the amount of excess storage required to achieve a practical availability of 99.9% by studying a decentralized algorithm that only depends on a modest amount of loosely synchronized global state. Indeed, our results show that it is exceedingly difficult to achieve this level of availability if individuals do not at least have approximate knowledge of peers' availability and files' current availability, which exactly comprise the global state that we assume are maintained by our infrastructure.

Our results are extremely encouraging because they demonstrate that practical availability levels are achievable in a completely decentralized P2P system with low individual availability. For example, even in a community where the average availability is only 24%, we can achieve a minimum and average availability of 99.8% and 99.95%, respectively, at only 6X excess storage. With today's low cost storage, this degree of replication seems quite feasible. This result demonstrates that such availability levels do not require sophisticated data structures, complex replication schemes, or excessive bandwidth. Indeed, our achieved availability levels are on par with today's managed services [12, 15].

On the other hand, our results demonstrate that the presence of a small fraction of server-like members which are online most of the time is critical. A community where each member is available 33% of the time, giving an average availability of 33% (which is significantly more than

24%), requires 9X excess storage to achieve 99.9% availability even when the content and excess storage is uniformly distributed.

Finally, note that while we have focused on studying how to increase the availability of static content to limit the scope of this work, our approach is not limited to the sharing of read-only data. At worst, updating a file is the same as injecting a new version and letting the replacement mechanism evict the old copies. Applications can greatly optimize this process, however, by propagating updates as diffs. Thus, we leave this as an issue for future work as update propagation and replication is highly application dependent.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[2] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[4] 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 *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2000.

[5] F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen. Autonomous Replication for High Availability in Unstructured P2P Systems. Technical Report DCS-TR-509, Department of Computer Science, Rutgers University, Nov. 2002.

[6] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. In *Proceedings of the International Workshop on Peer-to-Peer Computing (co-located with Networking 2002)*, May 2002.

[7] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*, June 2003.

[8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[9] Direct Connect. http://www.neo-modus.com.

[10] J. R. Douceur and R. P. Wattenhofer. Competitive Hill-Climbing Strategies for Replica Placement in a Distributed File System. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, Oct. 2001.

[11] K. M. Evans and G. H. Kuenning. Irregularities in file-size distributions. In *Proceedings of the 2nd International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2002.

[12] J. Gray. Dependability in the Internet Era. Keynote presentation at the Second HDCC Workshop, May 2001.

[13] 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 the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Nov. 2000.

[14] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*, July 2002.

[15] M. Merzbacher and D. Patterson. Measuring end-user availability on the web: Practical experience. In *Proceedings of the International Performance and Dependability Symposium (IPDS)*, June 2002.

[16] A. Muthitacharoen, R. Morris, T. Gil, and I. B. Chen. Ivy: A read/write peer-to-peer file system. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Aug. 2001.

[18] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, Apr. 1997.

[19] T. Roscoe and S. Hand. Transaction-based charging in mnemosyne: a peer-to-peer steganographic storage system. In *Proceedings of the International Workshop on Peer-to-Peer Computing (co-located with Networking 2002)*, May 2002.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, Jan. 2002.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Aug. 2001.

[23] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Nov. 1994.

[24] H. Weatherspoon and J. Kubiatowicz. Erasure Coding vs. Replication: A Quantitative Comparison. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002.

[25] H. Weatherspoon, C. Wells, P. R. Eaton, B. Y. Zhao, and J. D. Kubiatowicz. Silverback: A global-scale archival system. Technical Report UCB/CSD-01-1139, University of California, Berkeley, Mar. 2000.

IEEE COMPUTER SOCIETY