Redundancy Management for P2P Storage

Chris Williams, Philippe Huibonhoa, JoAnne Holliday, Andy Hospodor, Thomas Schwarz
Department of Computer Engineering,
Santa Clara University, Santa Clara, CA 95053, USA
phuibonhoa@gmail.com; chris-williams@comcast.net;
[jholliday ,ahospodor, tjschwarz]@scu.edu

Abstract

P2P storage systems must protect data against temporary unavailability and the effects of churn in order to become platforms for safe storage. This paper evaluates and compares redundancy techniques for P2P storage according to availability, accessibility, and maintainability using traces and theoretical results.

1 Introduction and Related Work

The initial wave of peer-to-peer (P2P) networks provided file-sharing and prompted the emergence of applications such as distributed internet storage [19, 13, 22] and internet backup [6, 16, 15]. P2P storage applications still need to handle the typical P2P problem of unreliable peers and of churn – the process of peers leaving and entering the system continuously. In contrast to other P2P systems, P2P storage systems place even more emphasis on reliable data availability, but can be more selective in the peers that are used. Furthermore, the selected peers are usually connected by a high bandwidth network. In this paper, we elide important P2P storage problems such as search, allocation, load distribution, and peer selection in favor of investigating the choice of a redundancy scheme that combines fast access, availability, and low costs of maintenance.

To achieve the required level of data availability, we store data redundantly. We typically generate redundancy with an m/n code. These linear codes break an object into m equal-sized chunks and generate additional parity chunks to bring the total number of chunks to n. Any m out of the n chunks suffice to reconstruct the object. Replication is formally a $1/n$ code.

We can use a m/n code ($m \neq 1$) in two different ways. First, erasure coding breaks a single object, such as a back-up of part of a file system, into m chunks. In the second approach, bucketing, we take m objects of similar size and calculate additional $n - m$ parity objects. Erasure coding must retrieve data from at least m different peers to construct our object. Bucketing needs to access only a single peer, the one with the complete copy. However, if that one is unavailable, it needs to access at least m other peers and transfer m times more data than the size of our object. In the following, we investigate replication, bucketing, erasure coding, and hybrid schemes, which combine replication with erasure coding and bucketing.

Several recent studies [2, 5, 20, 22] compare replication and erasure coding. Erasure coding wins over replication in terms of object availability with limited resources such as storage space [2, 22]. However, Rodrigues and Liskov [20] argue that access costs make erasure coding less attractive – a concern addressed by our hybrid schemes. Blake and Rodrigues [5] show that replacing copies on peers that have left the system poses a big burden for P2P systems. Our focus is P2P storage with the underlying assumption that the selected peers are both reliable and connected by high-speed internet connections, which lessens the strength of the Blake-Rodrigues argument in *this* case.

Due to space considerations, we only measure the impact of churn on the maintenance of data in P2P storage in our traces. We refer the reader instead to the work of Datta and Aberer [9] as well as that of Godfrey et al. [12] for a more complete discussion of maintenance.

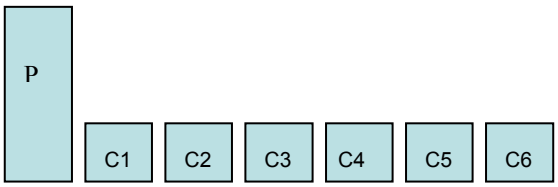

Figure 1: Erasure Coding with primary copy (P), m=3, n=6 (C1-C6).

2 Redundancy Schemes

Data stored on unreliable servers is protected by redundancy. The simplest scheme is *Replication* (Rep) where we store the same object in n different locations. Simple *Erasure Coding* (EC) uses an m/n code. It breaks an object into m chunks and generates additional $n - m$ parity chunks. To access an item, we need only to gather any m chunks and decode. Churn, the constant joining and leaving of node in a P2P system, forces us to constantly replace data stored on lost peers somewhere else. Using EC, we have to

access and read m surviving chunks, reconstruct the data, and generate replacement chunks. To streamline this cumbersome process, we can store the original data elsewhere and generate replacement chunks preferably from it. Following Rodrigues and Liskov [20], we call this *Erasure Coding with Primary Copy* (EC/1P), (see Figure 1). Data accesses in EC/1P are also simpler since we can directly access the stored object. We also consider a scheme EC/2P where we mirror the primary copy.

Bucketing (BUCK) attempts to combine the advantages of replication and EC without the disadvantages of an increased storage footprint. BUCK emulates a RAID Level 6 reliability stripe, where each data item is stored in its entirety in a bucket and a peer holds one bucket. A typical bucket contains several data items. All buckets are roughly the same size. Note that data buckets might not be completely filled. We group m data buckets into a reliability stripe and add to them k parity buckets (Figure 2). If possible, the data is accessed directly from the data bucket containing the item. As any m data buckets in a reliability stripe are sufficient to reconstruct any data item, we can still satisfy requests to data stored on dead or unavailable peers and reconstruct lost buckets in the background.

We also consider schemes where each data bucket is mirrored (BUCK/1R) or even triplicated (BUCK/2R). These hybrid schemes satisfy more reads in a single access.

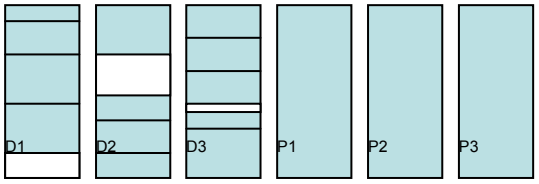

Figure 2: Bucketing, m = 3, n = 6.

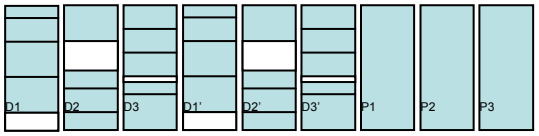

Figure 3: Bucketing with Replication.

3 Theoretical Results

In order to compare our schemes, we derive formulae for the stretch (the ratio of storage actually used over the size of the data stored) and the access costs in the case where peers are available with constant probability p. We first calculate formulae for the availability of our data. Throughout, we use n for the group size, m for the number of peers necessary to recover data, and we abbreviate $q = 1 - p$. We also denote the cumulative binomial probability by

$$B(m,n,p) = \sum_{\nu=m}^{n} \binom{n}{\nu} p^{\nu} q^{n-\nu}$$

For comparison purposes, we select a high level of data availability and calculate the storage overhead as the stretch, defined to be the ratio of the size of storage used in order to provide this availability over the size of the user data items. We use our formulae to numerically determine the stretches for a data availability level of 99.99% (four nines) using m = 7 for all schemes but replication (Figure 4). Certainly, other levels of availability such as two nines or five nines, could be presented in a longer paper.

For ease of reading, the schemes appear in the same order within the legend as they do in the graph at high levels of availability. We observe the much larger stretch required by replication for low to medium peer availability. The hybrid schemes (EC/1P, EC/2P, BUCK/1R, BUCK/2R) converge to the space efficiency of EC and BUCK for low peer availability. However, the hybrid schemes follow the trend to the higher stretch of replication for higher peer availability. Always, BUCK has lower stretch than EC, mainly because we define our availability as the probability that a given object is available. If, for instance, only one bucket survives then the data in that bucket is considered available. In EC, a single chunk is not sufficient to reconstruct data. For much lower object availability levels and very low peer availability, replication can provide better object availability than bucketing and chunking [17] for the same stretch.

Table 1: Availability of a stored data item.

REP	$1-q^n$
EC	$B(m,n,p)$
EC/1P,2P	$1-q^l \cdot (1-B(m,n,p)$
BUCK	$1-q(1-B(m,n-1,p))$
BUCK/1P,2P	$1-q^l(1-D(m,m-1,n-lm,l,p))$

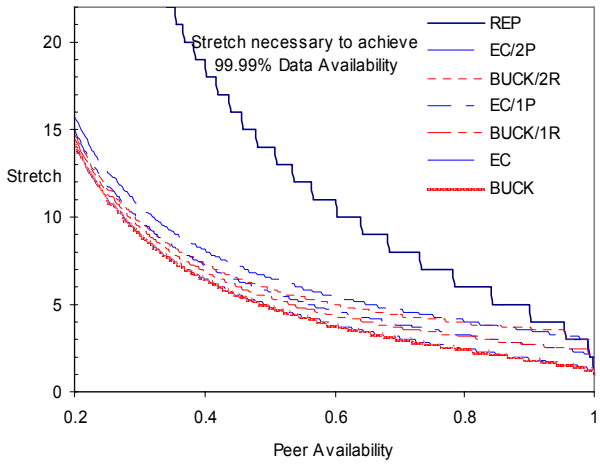

Figure 4: Stretches for various schemes. (m =7).

We now turn to modeling data accesses. As it would be impossible for us to give a complete taxonomy of data access organizations, we use two simple measurements. First, we measure the difficulty to locate peers from which to transfer data. In a typical, P2P system based on Distributed Hash Tables (DHT) with N peers, clients can access a peer with a given ID in O(log N) hops. Rather than counting the hops, we just consider this one access, and then count the number of accesses (pings) it takes to locate enough peers to get the desired data. We assume that the client will ping peers until it finds either a single peer that contains the data or enough peers to reconstruct it. Our model presupposes that client peers cache the Internet Protocol (IP) addresses of their storage sites. The distributed hash table locates the storage peers only if the peer or peers at cached addresses do not respond or refuse to honor the request. While not valid in general (e.g. if a random walk strategy is used), we believe that our assumption is valid in P2P *storage* schemes. Even if the assumption is wrong, our numbers still give a rough idea of the difficulties of accessing data for each scheme.

Our second measure is the minimum amount of data that needs to be transferred to retrieve the data. (Some P2P systems are more profligate in their retrieval than others.) While replication and EC only download the requested data, this is not always true for Bucketing. If a data bucket with the requested data is not available, then m complete buckets must be accessed and transferred in order to reconstruct the desired data.

We now present the access costs associated with our various schemes. As before, p stands for the probability of a peer being available, q is $1 - p$, and m and n are the parameters of the erasure correcting code. S is the expected number of pings.

Replication searches for one replica after the other:

$$S_{\text{Repl}} = p + 2qp + 3q^2 p + \ldots (n-1)q^{n-2} p + nq^{n-1} = \frac{1-q^n}{p}$$

Erasure Coding (EC) uses exactly s pings if the first s–1 pings provide m–1 available peers and the next access is successful. The sole exception is $s = n$ pings, which happens if we have not found m available peers in n–1 trials. In this case, our data retrieval is either unsuccessful or uses all n peers in the reliability stripe. It appears that our formula cannot be simplified:

$$S_{\text{EC}}(n,m,p) = \sum_{\mu=m}^{n-1} \mu \binom{\mu-1}{m-1} p^m q^{\mu-m} + n(1 - B(m, n-1, p)))$$

In *Erasure Coding with one primary copy* (EC/1P) n peers store data chunks and an additional peer stores the primary copy. We use one ping if the primary copy is available (with probability p). Otherwise, (with probability q) we proceed, as with Erasure Coding, by looking for m available peers among n possible peers in the reliability stripe:

$$S_{\text{EC/1P}} = p \cdot 1 + q \cdot (1 + S_{\text{EC}}(n,m,p))$$

Similarly, if we duplicate the primary copy to be more likely to satisfy reads from that copy, then we use 1 ping, if the first primary copy is available (probability p), 2 pings, if the first one is down and the second is available (probability pq), and otherwise two more than with Erasure Coding (probability q^2).

$$S_{\text{EC/2P}} = 1 \cdot p + 2 \cdot p \cdot q + (2 + S_{\text{EC}}(n,m,p)) \cdot q^2$$

In *Bucketing* (BUCK), we access the data directly with probability p when the peer with the desired data is available. Otherwise, we must collect data from m of the n – 1 remaining peers.

$$S_{\text{BUCK}} = 1 \cdot p + (1 + S_{\text{EC}}(n-1, m, p)) \cdot q$$

An exact expression for S for Bucketing with Replication is difficult to derive because of the accounting for replicas of data buckets. Merely being able to find m available peers is not sufficient, because sometimes a peer carries redundant user data items that do not contribute to the ability to reconstruct data. We achieve our numerical results in this case using an upper and a lower bound, separated by at most 2%.

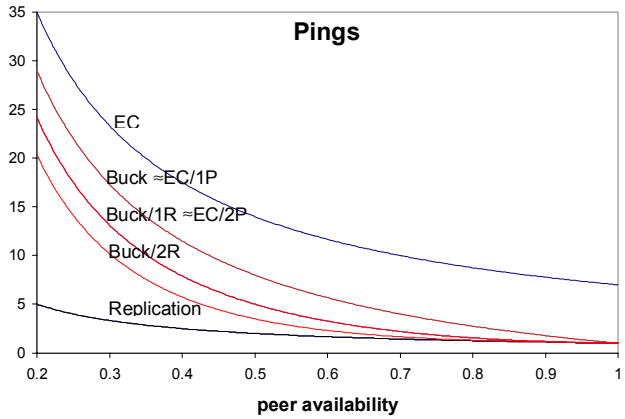

Figure 5: Access cost in pings for various schemes

We show the numerical results in Figure 5 for 99.99% data availability. With perfect peer availability, all schemes require a single ping to access data, except EC, that requires m pings. Not surprisingly, replication requires the least number of pings and erasure coding the most. The addition of primary copies to erasure coding (EC/1P, EC/2P) dramatically improves ping performance. In fact, for very high levels of peer availability (90%), their performance almost equals that of replication. BUCK and EC/1P and BUCK/1P and EC/2P have almost identical ping performance.

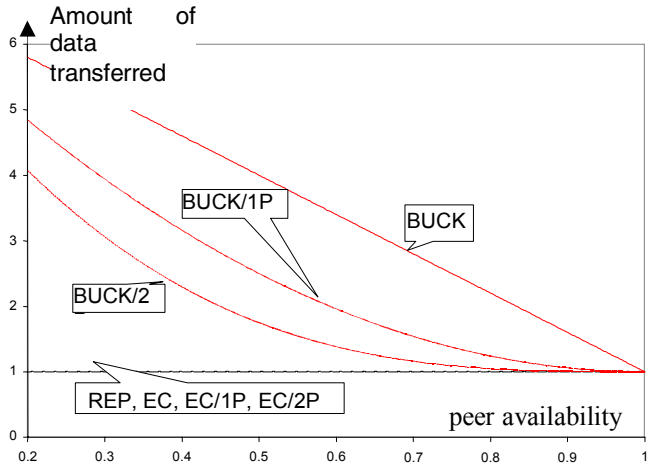

Figure 6: Data transfer overhead

Figure 6 highlights the disadvantage of bucketing schemes, which must fetch additional and unrelated data to reconstruct the requested data. The x-axis gives peer availability, while the y-axis gives the amount of data transferred. We see that with BUCK/2R these additional transfers are small (<10%) for peer availability levels greater than 75%. However, if peers are only available 20% of the time, then BUCK transfers almost six times more data than REP or any of the EP schemes.

4 Experimental Evaluation and Simulation

Of course, our analytic models cannot accurately reflect the variety and diversity of peer availability in an actual P2P system. We therefore used traces extracted from live P2P systems to evaluate the efficacy of the various schemes. We again chose a data availability level of 4 nines, then determined the necessary stretch and access costs associated with each scheme.

A P2P storage scheme must recognize and replace under-performing peers as well as peers that have permanently left the system. Timeouts are a simple and proven approach. A peer currently storing data that has been unavailable for a period T – the time-out value – is replaced by a random available peer. Incidentally, we have experimented with more sophisticated replacement and selection algorithms, and our results confirm others [10,12] that random replacement is a good strategy. We consider here only this simple time-out scheme with three levels of T.

We model a static scheme by using an infinite timeout level, $T = \infty$. Timeout level T = 1 hour is clearly aggressive and replaces any peer not responding within a single time period. Timeout level T = 24 hours was intended to represent a real world case where a peer would be unavailable for many time periods before replacement.

Our evaluation used different traces from Godfrey's collection. The first one is Farsite, which gives the availability of 51,663 desktop PCs within Microsoft for 35 days beginning July 6, 1999 [1, 10]. The overall average availability for the Farsite network was approximately 81% with 9% of all peers being available for the entire trace. The second trace, Overnet, collected the availability of 2400 peers in the Overnet P2P system gathered during a membership crawl over a 7 day period [3]. The overall average uptime for the Overnet network is approximately 15% with none of the hosts being available for the entire trace.

We report our results for an availability level of 99.99% (four nines). We note that our experiments for 99% (two nines) did not lead us to different conclusions. We first compare simulated and theoretical expected values for stretch using the Farsite traces in Table 2. Here, we chose m=7 and 4 nines of availability for different timeouts. The experimental values are consistently less than those predicted by the formulae, even though a model with time-out value $T = \infty$ should be accurate.

The discrepancy has a simple explanation: Peers are not independent with respect to availability. In the Farsite trace, for example, peer availability is highest during the day and lowest during nights and weekends. Secondly, as is well-known [3, 18, 21], random replacement of failed peers by currently available peers tends to select peers with higher availability over time. As a result, the availability of peers actually used in our scheme is on average higher than the general population.

Table 2: Simulated vs. Theoretical Stretch (Farsite)

Scheme	T=1	T=24	T=∞	Theoretical
REP	2.00	3.00	4.00	6.00
EC	1.43	1.72	2.15	2.42
EC/1P	2.15	2.43	2.91	3.14
EC/2P	3.00	3.29	3.72	4.00
Buck	1.29	1.58	2.00	2.28
Buck/1P	2.15	2.29	2.58	3.14
Buck/2P	3.00	3.15	3.29	3.85

4.1 Churn Costs

Churn, the data movement caused by replacing peers that have timed out and are thus considered unavailable, is difficult to model analytically. Even if we were to assume a completely homogenous set of peers, churn costs depend on the frequency of peers changing state from unavailable to available and the timeout value. While it is possible to model the steady state for a homogeneous set of peers, we doubt its applicability to practical systems. Therefore, we

used the traces of P2P systems to evaluate churn. We measure churn as the total amount of data transferred per hour normalized against the total amount of user data. A churn of 0.1 thus signifies that on average, we move 10% of the user-stored data around the network in order to maintain the current state.

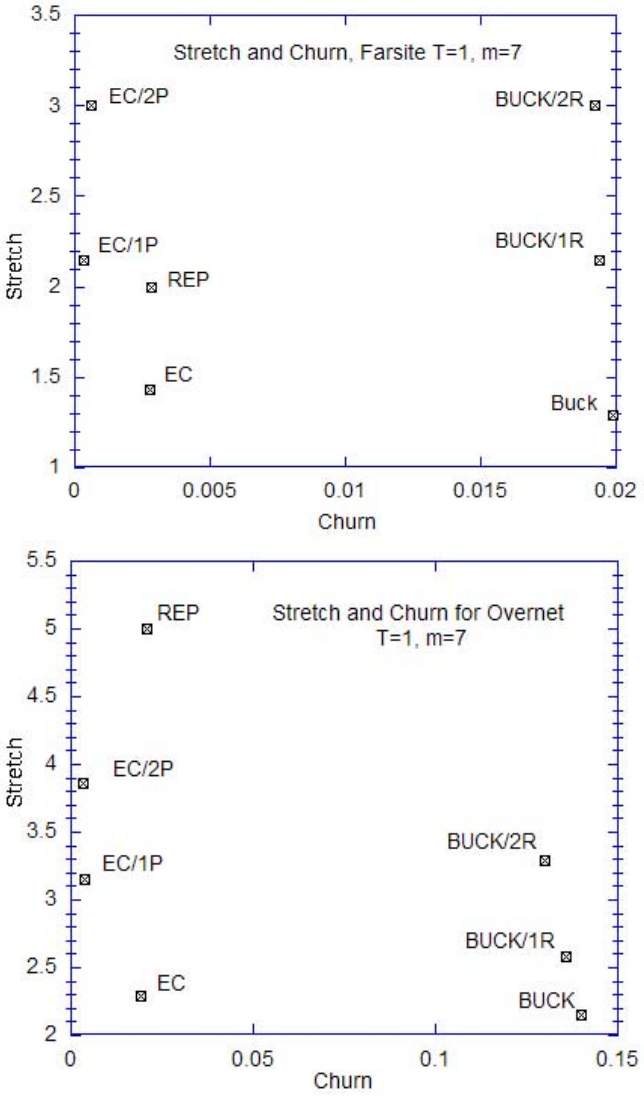

Figure 7: Effect of Stretch and Churn in Farsite and Overnet

Figure 7 shows stretch and churn cost in Farsite and Overnet for the various schemes. Farsite only needs two replicas in the replication scheme at the aggressive timeout value 1. The two primary copies in EC/2P and the triplication of data buckets in BUCK/2R are overkill and these schemes far exceed our availability threshold of 99.99%. EC/1P has a slightly worse stretch than REP, but this is because the minimum configuration, one primary copy and 7 chunks, just misses the availability mark of 99.99%. All bucketing schemes cause considerably more churn. Since Farsite peers are highly available, we typically only have to replace data on a single peer. EC/1P and EC/2P deal best with this situation. If the primary copies have survived, we generate a new chunk from the primary and send it to the replacement peer for a cost of $1/m$. Replacing a peer in BUCK is done at one of the surviving buckets, we read $m-1$ other buckets and write to the replacement peer, for a total transfer cost of m times the contents of the bucket. Consequentially, BUCK and its variants have considerably more churn costs in both traces.

As expected, the Overnet trace, Figure 7, with relatively low peer availability, had considerably higher stretch and churn than Farsite. An aggressive timeout and replacement policies force early replacements of typically a single peer. As before, EC/1P and EC/2P have the least amount of churn cost. REP and EC, as before, have slightly higher churn costs and BUCK and its variants fares more poorly. The cost of churn should diminish as replacement picks more highly available peers.

Table 3: Churn Costs in Overnet and Farsite, First and Last Quintile, Timeouts 1hr and 24 hrs

	Farsite		Overnet	
Scheme	T=1	T=24	T=1	T=24
BUCK 1st quint.	0.0336	0.0058	0.1474	0.0069
BUCK 5th quint.	0.0198	0.0027	0.1302	0.0095
BUCK/1R 1st q.	0.0329	0.0058	0.1424	0.0069
BUCK/1R 5th q.	0.0196	0.0027	0.1273	0.0096
BUCK/2R 1st q.	0.0324	0.0059	0.1358	0.0068
BUCK/2R 5th q.	0.0192	0.0028	0.1219	0.0096
EC 1st quintile	0.0048	0.0008	0.0202	0.0009
EC 5th quintile	0.0028	0.0004	0.0180	0.0014
EC/1P 1st quint.	0.0005	0.0001	0.0049	0.0005
EC/1P 5th quint.	0.0003	0.0000	0.0039	0.0008
EC/2P 1st quint.	0.0011	0.0002	0.0039	0.0003
EC/2P 5th quint.	0.0006	0.0001	0.0033	0.0004
REP 1st quintile	0.0049	0.0008	0.0245	0.0011
REP 5th quintile	0.0029	0.0004	0.0211	0.0014

Table 3 contrasts the churn costs in the first and the last quintile of the trace. We obtained these values with 10,000 runs. We notice the beneficial effects of increasing the timeout. We also notice that Overnet churn costs increase from the first quintile to the last when using $T = 24$, mainly

because the timeout value (1 day) is so close to one fifth the length of the trace (of 7 days). This problem is endemic to using timeout schemes on finite traces. Just as a cold cache has mandatory misses, so our timeout scheme can only replace underperforming peers after a warm-up period.

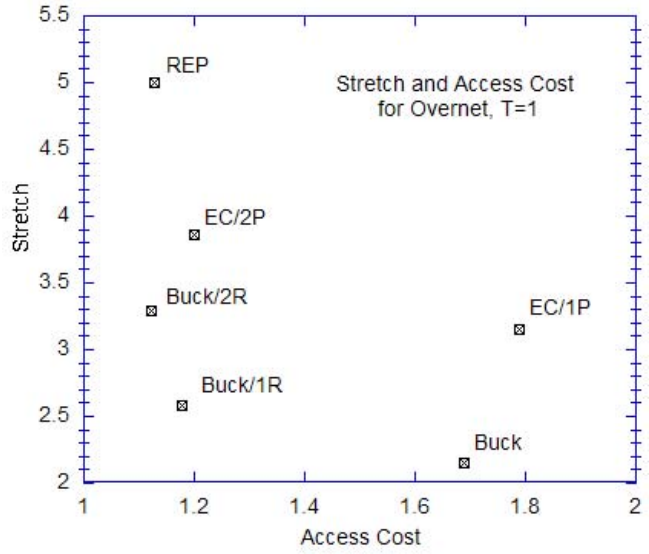

Figure 8: Effect of Stretch and Access in Overnet

4.2 Access Costs

Recall that we distinguish two access costs, the number of pings to find the data and the amount of data that needs to be transferred. Recall also that measuring access costs in pings should be valid in a P2P *storage* system, but is meaningless for some P2P systems. Only Bucketing transfers a larger amount than the requested piece of data, and this only if the peer(s) storing the entire data item are unavailable. We use the number of pings as a measure of the latency users will experience before they start receiving data.

Figure 8 shows the stretch and access cost (number of pings) for Overnet. The EC scheme is not shown because its access cost is so much greater than the others (about 8 under these conditions). The access cost of EC is so high because no single peer can supply the entire data item.

4.3 Sensitivity

Tables 4 and Figure 9 show the influence of the choice of the time-out parameter. In related, unpublished work, the authors found that more sophisticated selection mechanisms than randomly picking an available peer as a replacement can improve the availability of the selected peers over time, but that the improvement is rarely over 20% in availability. Similarly, the choice of the erasure coding parameter *m* also influences stretch and churn costs but the effect is also less pronounced.

Table 4: Complete Overnet and Farsite stretch results

		Overnet			Farsite		
Scheme	T-out	*m*=4	*m*=7	*m*=12	*m*=4	*m*=7	*m*=12
REP	1	5	5	5	2	2	2
	24	12	12	12	3	3	3
	∞	21	21	21	4	4	4
BUCK	1	2.75	2.15	1.92	1.5	1.29	1.25
	24	5.71	4.43	3.67	2	1.58	1.42
	∞	9.25	7.29	6	2.5	2	1.81
BUCK /1P	1	3	2.58	2.42	2.25	2.15	2.09
	24	5.75	4.58	3.92	2.5	2.29	2.17
	∞	9.5	7.29	6.09	2.75	2.58	2.34
BUCK /2P	1	3.5	3.29	3.25	3	3	3
	24	6	5	4.42	3.25	3.15	3.09
	∞	9.5	7.58	6.34	3.5	3.29	3.17
EC	1	2.75	2.29	1.92	1.5	1.43	1.25
	24	5.75	4.43	3.75	2	1.72	1.5
	∞	9.25	7.29	6	2.5	2.15	1.84
EC/1P	1	3.5	3.15	2.84	2.25	2.15	2.11
	24	6.5	5.41	4.67	2.71	2.43	2.34
	∞	10	8.15	6.84	3.25	2.91	2.67
EC/2P	1	4	3.86	3.67	3	3	3
	24	7.25	6.15	5.5	3.5	3.29	3.25
	∞	10.75	8.91	7.75	4	3.72	3.51

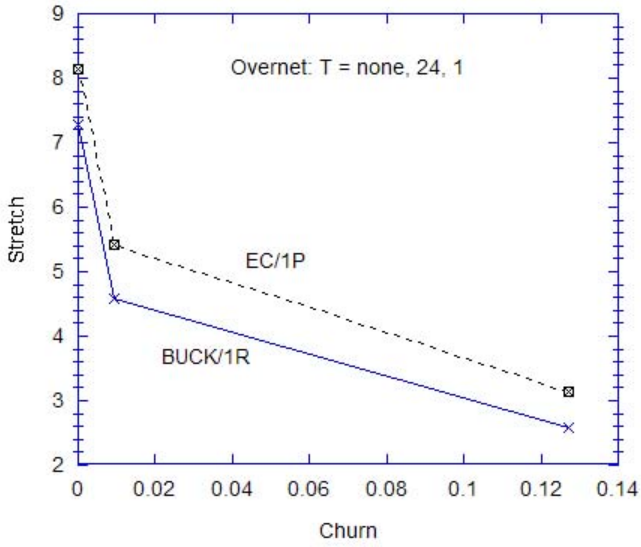

Figure 9: Effect of Timeout in Overnet

4.4 Improving Bucketing

Lazy peer replacement lowers the costs of churn. As bucketing has higher churn costs, we experimented with an "improved" version of bucketing (BUIM), where we add *m* additional parity peers to a reliability stripe. We only replace data on lost parity peers if *m* or more peers need to be replaced. This bulk replacement lowers the data transfer overhead that the original bucketing scheme incurs. Recall

that if we reconstruct a single peer, we access *m* peers, but also have to transfer *m* times the amount of data stored on a peer. Table 5 shows that the increased stretch of BUIM with mirrored or triplicated data peers pays off in significantly lowered churn costs, about a factor of 10 in both Overnet and Farsite.

4.5 Summary

When assessing our various schemes, we observed that Replication has the lowest maintenance costs and is conceptually the simplest scheme. However, only with high peer availability was its stretch comparable to other schemes. Bucketing and Erasure Coding have much lower stretch, with the advantage going to Bucketing when using highly available peers. In general, Bucketing has the highest churn cost and Erasure Coding the highest access cost.

Table 5: Churn Costs in Overnet and Farsite, First and Last Quintile, Timeouts 1hr and 24 hrs

	Farsite		Overnet	
Scheme	*T*=1	*T*=24	*T*=1	*T*=24
BUCK/1R 1st	0.0329	0.0058	0.1424	0.0069
BUCK/1R 5th	0.0196	0.0027	0.1273	0.0096
BUIM/1R 1st	0.0033	0.0005	0.0141	0.0002
BUIM/1R 5th	0.0019	0.0002	0.0133	0.0003
BUCK/2R 1st	0.0324	0.0059	0.1358	0.0068
BUCK/2R 5th	0.0192	0.0028	0.1219	0.0096
BUIM/2R 1st	0.0037	0.0006	0.0183	0.0005
BUIM/2R 5th	0.0022	0.0003	0.0165	0.0007

Not surprisingly, we observe that combining replication with either Bucketing or Erasure Coding lowers the access cost considerably because a complete copy of the data is usually available from a single peer. Our "improved" hybrid bucketing schemes (Table 5) present a further performance compromise. It trades higher stretch than hybrid Erasure Coding for easier access and almost the low churn costs of hybrid Erasure Coding.

5 Conclusions

Replication is simple and offers good performance, but its stretch is reasonable only for very highly available peers. More complex schemes are worth considering when peers exhibit lower availability as in the traces of Farsite and Overnet. Fault-free software for distributed systems is difficult to produce and we caution against adding too much complexity. The most complicated part of a P2P storage scheme implementation lies in the addressing scheme, the monitoring of the peers and the selection of replacement peers, i.e., the handling of churn. An improved redundancy scheme does not require a significant amount of added complexity. Linear Erasure Coding schemes are well known and can be implemented in a separate module. The complexity of updating data (and parity) in any scheme depends on the guarantees given about when the change of data becomes visible to all users. This is a well-known, difficult, but solved problem in distributed systems and well-studied in the literature. The added complexity of Erasure Coding or Bucketing is small in comparison.

In short, we believe that implementing pure erasure coding and bucketing impose rather high maintenance costs (churn) and access costs, even for relatively highly available peers. As a consequence, we advise the use of *hybrid schemes* that combine data Replication with either Erasure Coding or Bucketing. A hybrid scheme achieves reasonably good performance because it typically accesses data items directly (i.e. it uses Replication for both maintenance and data access) and achieves very high data availability at a reasonable stretch level because it uses the more space efficient methods of Erasure Coding or Bucketing to increase the data availability level. Our experiments with traces of P2P systems confirm these conclusions. Additionally, we can reduce the costs of churn in Bucketing by being smart about maintenance, creating proactively "spare" parity buckets so that we can perform bulk maintenance operations less often.

In the larger picture, our work addresses to some extent the concerns of the work of Blake and Rodrigues [5]. As we have seen, churn costs are manageable in our environment, mainly because our replacement mechanism selects peers with higher availability with good network bandwidth. We achieve scalability, because reliability groups are autonomous. This leaves the problem of locating the data, something that this paper could not address.

In conclusion, our results lead us to recommend the use of replication to obtain simple data access most of the time, but to use additional erasure-code-generated parity to achieve the last nines in the desired availability level. While we can tolerate the observed churn in a P2P *storage* environment, lazy and proactive techniques to reduce churn are advisable. However, in this paper, we did not study the efficacy of these schemes again. Our results suggest a P2P storage system with reasonable maintenance costs, fast access time, desired high availability, and reasonable storage overhead from peers with a much smaller availability level.

REFERENCES

[1] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, R.. Wattenhofer: FARSITE: federated, available, and reliable storage for an incompletely trusted environment. 5th Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[2] R. Bhagwan, D. Moore, S. Savage, G. Voelker: Replication strategies for highly available peer-to-peer storage. Future Directions in Distributed Computing (FuDiCO), 2002.

[3] R. Bhagwan, S. Savage, G. Voelker: Understanding Availability. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03). Peer-to-Peer Systems II, Springer, Lecture notes in computer science vol 2735, pp 256-267.

[4] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, G. Voelker: Total Recall: System Support for Automated Availability Management. Proceedings of the first Symposium on Networked Systems Design and Implementation (NSDI), Mar. 2004.

[5] C. Blake, R. Rodrigues: High availability, scalable storage, dynamic peer networks: pick two. Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX), Lihue, Hawaii, 2003.

[6] L. P. Cox, C. Murray, B. Noble: Pastiche: Making backup cheap and easy. In 5th Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[7] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, R. Morris: Efficient replica maintenance for distributed storage systems. Proceedings, 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06), 2006.

[8] E. Cohen, S. Shenker: Replication strategies in unstructured peer-to-peer networks. ACM SIGCOMM Computer Communication Review, vol. 32(4), October 2002.

[9] A. Datta, K. Aberer: Internet-Scale Storage Systems under Churn – A Study of the Steady-State Using Markov Models. 6th International Conference on Peer-to-Peer Computing, (P2P'06), 2006.

[10] J. Douceur, R. Wattenhofer: Optimizing file availability in a secure peerless distributed file system. Proceedings of 20th IEEE Symposium on Reliable Distributed Systems (SRDS), 2001, pp. 4-13.

[11] Brighten Godfrey: Repository of Availability Traces. www.cs.berkeley.edu/~pbg/availability/

[12] P. Godfrey, S. Shenker, I. Stoica: Minimizing Churn in Distributed Systems, Proceedings of SIGCOMM, 2006.

[13] S. Hand and T. Roscoe: Mnemosyne: Peer-to-Peer Steganographic Storage. Proceedings 1st International Workshop on Peer-to-Peer Systems (IPTPS '02). 2003.

[14] L. Hellerstein, G. Gibson, R. Karp, R. Katz, D. Patterson: Coding techniques for handling failures in large disk arrays. Algorithmica, 12:182-208, 1994.

[15] J. Li, F. Dabek: F2F: Reliable Storage in Open Networks. Proceedings, 5th International Workshop on Peer-To-Peer Systems. 2006.

[16] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, M. Isard: A cooperative internet backup scheme. In Proceedings of the 2003 Usenix Annual Technical Conference, pp. 29-41, San Antonio, Texas, June 2003.

[17] W. Lin, D. Chiu, Y. Lee: Erasure Code Replication Revisited, Proc. 4th International Conference on Peer-to-Peer Computing (P2P'04).

[18] R. Mahajan, M.l Castro, A. Rowstron: Controlling the Cost of Reliability in Peer-to-Peer Overlays. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003. Peer-to-Peer Systems II, Springer, Lecture notes in computer science vol. 2735.

[19] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, J. Kubiatowicz: Maintenance-Free Global Data Storage, IEEE Internet Computing, Vol. 5(5), September/October 2001, pp 40-49.

[20] R. Rodrigues, B. Liskov: High Availability in DHTs: Erasure Coding vs. Replication. 4th International Workshop on Peer-to-Peer Systems (IPTPS'05). Ithaca, New York, USA. February 2005.

[21] E. Sit, A. Haeberlen, F. Dabek, B. Chun, H. Weatherspoon, R. Morris, M. . Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. Proceedings of the Fifth International Workshop on Peer-to-Peer Systems (IPTPS '06), February 2006

[22] H. Weatherspoon and J. Kubiatowicz: Erasure coding vs. replication: A quantitative comparison. Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002.

[23] B. Zhao, L. Huang, J. Stribling, S.. Rhea, A. Joseph, J. Kubiatowicz: Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications, Vol 22(1), January 2004