High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two

Charles Blake Rodrigo Rodrigues
cb@mit.edu rodrigo@lcs.mit.edu
MIT Laboratory for Computer Science

Abstract

Peer-to-peer storage aims to build large-scale, reliable and available storage from many small-scale unreliable, low-availability distributed hosts. Data redundancy is the key to any data guarantees. However, preserving redundancy in the face of highly dynamic membership is costly. We use a simple resource usage model to measured behavior from the Gnutella file-sharing network to argue that large-scale cooperative storage is limited by likely dynamics and cross-system bandwidth — not by local disk space. We examine some bandwidth optimization strategies like delayed response to failures, admission control, and load-shifting and find that they do not alter the basic problem. We conclude that when redundancy, data scale, and dynamics are all high, the needed cross-system bandwidth is unreasonable.

1 Introduction

Recent systems (CAN, Chord, Pastry, or Tapestry [7, 11, 8, 13]) enable peer-to-peer lookup overlays robust to intermittent participation and scalable to many unreliable nodes with fast membership dynamics. Some papers ([1, 6]) express a hope that, with extra data redundancy, *storage* can inherit scalability and robustness from the underlying lookup procedure. More work still ([5, 9]) *implies* this hope by using robust lookup as a foundation for wide-area storage layers, even though this complicates other desirable properties (e.g., server selection).

This paper argues that trying to achieve all three things — scalability, storage guarantees, and resilience to highly dynamic membership — overreaches bandwidth resources likely to be available, regardless of lookup. Our argument is roughly as follows. Simple considerations and current hardware deployment suggest that idle upstream bandwidth is the limiting resource that volunteers contribute, not idle disk space. Further, since disk space grows much faster than access point bandwidth, bandwidth is likely to become even more scarce relative to disk space.

We elaborate this argument in the next section using a generic resource usage model to estimate conservatively the costs associated with maintaining redundancy in systems built from unreliable parts. Section 3 adapts our model to accommodate hosts which are temporarily unavailable but have not lost their data. Section 4 discusses other issues such as admission control or load-shifting, hardware trends, and the importance of incentives. Along the way we use numbers from Gnutella, a real peer-to-peer system, to highlight how bandwidth contributions are the serious limit to scaling data. We conclude in Section 5.

2 A Simple Model

In this section we consider the bandwidth necessary for reliable peer-to-peer storage. We present a simple analytic model for bandwidth usage that attempts to provide broad intuition and still apply in some approximation to currently proposed systems.

2.1 Assumptions

We assume a simple redundancy maintenance algorithm: whenever a node leaves or joins the system, the data that node either held or will hold must be downloaded from somewhere. Note that by *join* and *leave* we mean really joining the system for the first time or leaving forever. We do not refer to transient failures, but rather the intentional or accidental loss of the contributed data. Section 3 elaborates this model to account for temporary disconnections that may not trigger data transfers. We also assume there is a static data placement strategy (i.e., a function from the current membership to the set of replicas of each block).

We make a number of simplifying assumptions. Each one is *conservative* — increased realism would increase the bandwidth required. Note that any storage guarantee effectively insists that the probability of not getting a datum is below some threshold. The time to create new nodes must therefore consider the worst-case accidents of data distribution and other variations. Therefore, the fact that we perform an average case analysis makes our model conservative.

We assume identical per-node space and bandwidth contributions. In reality, nodes may store different amounts of data and have different bandwidth capabilities. Maintaining redundancy may require in cer-

tain cases more bandwidth than the average bandwidth. Creating more capable nodes from a set of less capable nodes might take more time. Average space and bandwidth therefore conservatively bound the worst case which is the relevant bound for a guarantee.

We assume a constant rate of joining and leaving. As with resource contributions, the worst case is a more appropriate figure to use for any probabilistic bound. The average rate bounds the maximum rate from below, which is again conservative. We also assume independence of leave events. Since failures of networks and machines are not truly independent, more redundancy would really be required to provide truer guarantees.

We assume a constant steady-state number of nodes and total data size. A decreasing population requires more bandwidth while an increasing one cannot be sustained indefinitely. It would also be more realistic to assume data increases with time or changes which would again require more bandwidth.

2.2 Data Maintenance Model

Consider a set of N identical hosts which cooperatively provide guaranteed storage over the network. Nodes are added to the set at rate α and leave at rate λ, but the average system size is constant, i.e. $\alpha = \lambda$. On average, a node stays a member for $T = N/\lambda$.

Our data model is that the system reliably stores a total of D bytes of unique data stored with a redundancy expansion factor k, for a total of $S = kD$ bytes of contributed storage. One may think of k as either the replication factor or the expansion due to coding. The desired value of k depends on both the storage guarantees and redundant encoding scheme and is discussed more in the next section.

We now consider the data maintenance bandwidth required to maintain this redundancy in the presence of a dynamic membership. Note that the model does not consider the bandwidth consumed by queries, and therefore we present a conservative bandwidth estimate.

Each node *joining* the overlay must download all the data which it must later serve, however that subset of data might be mapped to it. The average size of this transfer is S/N. Join events happen every $1/\alpha$ time units. So the aggregate bandwidth to deal with nodes joining the overlay is $\frac{\alpha S}{N}$, or S/T.

When a node *leaves* the overlay, all the data it housed must be copied over to new nodes, otherwise redundancy would be lost. Thus, each leave event also leads to the transfer of S/N bytes of data. Leaves therefore also require an aggregate bandwidth of $\frac{\lambda S}{N}$, or S/T. The total bandwidth usage for all data maintenance is then $\frac{2S}{T}$, or a per node average of:

$$B/N = 2\frac{S/N}{T}, \text{ or } BW/node = 2\frac{space/node}{lifetime} \quad (1)$$

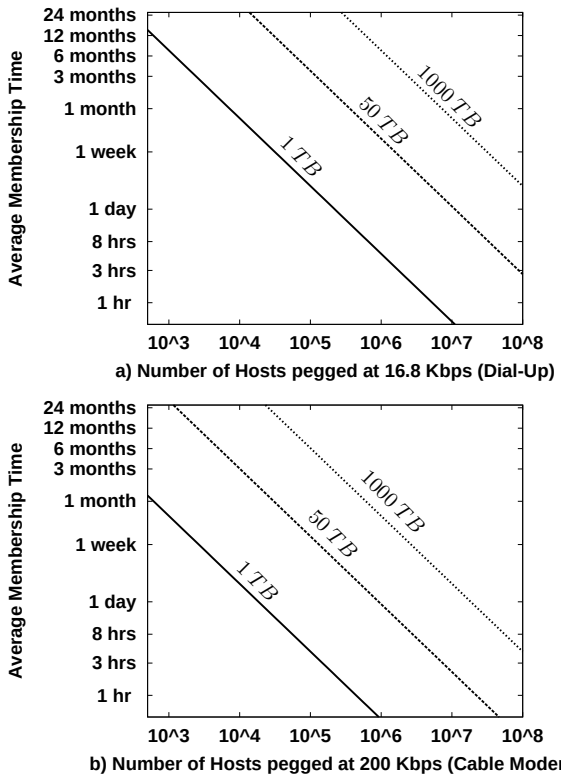

Figure 1. Log-Log plots for the participation requirements of a) dial-up and b) cable modem networks. Plotted are thresholds below which various amounts of unique data will incur over 50% link saturation just to maintain the data. These use a redundancy $k = 20$.

2.3 Understanding the Scaling

Figure 1 plots some example "threshold curves" in the lifetime-membership plane. This is the basic participation space of the system. More popular systems will have more hosts, and those hosts will stay members longer. Points below a line for a particular data scale require data maintenance bandwidth in excess of the available bandwidth. We plot thresholds for maintenance *alone* consuming half the total link capacity for dial-ups and cable modems. The data scales we chose, $1\ TB$, $50\ TB$, and $1000\ TB$, might very roughly correspond to a medium-sized music archive, a large music archive, and a small video archive (a few thousand movies), respectively.

There are two basic points to take away from these plots. First, short membership times create a need for enormous node counts to support interesting data scales. E.g., a million cable modem users must each provide a *continuous month* of service to maintain $1000\ TB$ even if no one ever actually queries the data! Second, this strongly impacts how fast the storage of such a network

can grow. At a monthly turnover rate, each cable modem must contribute less than $1\,GB$ of unique data, or $20\,GB$ of total storage. Given that PCs last only a few years and a few years ago $80\,GB$ disks were standard on new PCs, $20\,GB$ is likely about or below current idle capacity.

Figure 1 uses a fixed redundancy factor $k = 20$. The actual redundancy necessary depends on T, N, probability targets for data loss or availability. Section 3 examines in more detail the necessary k for both replication-style and erasure coded redundancy for availability.

3 Availability and Redundancy

This section expands our model to include hosts that are transiently disconnected and estimates redundancy requirements in more detail.

3.1 Downtime vs. Departure

So far our calculations have assumed that the resources a host contributes are always available. Real hosts vary greatly in availability [3, 4, 10]. The previous section shows that it takes a lot of bandwidth to preserve redundancy upon departures. So it helps to distinguish true departures from temporary downtime, as in [3].

Our model for how systems distinguish true departures from transient failures is a membership timeout, τ, that measures how long the system delays its response to failures. I.e., the process of making new hosts responsible for a host's data does not begin until that host has been out of contact for longer than time τ.

Counting offline hosts as members has two consequences. First, member lifetimes are longer since transient failures are not considered leaves. Second, hosts serve data for a *fraction* of the time that they are members (or a fraction of members serve data at a given moment). We define this fraction to be the availability, a.

Since only a fraction of the members serve data at a time, more redundancy is needed to achieve the same level of availability. Also, the effective bandwidth contributed per node is reduced since these nodes serve only a fraction of the time. Thus, the membership lifetime benefits gained by delayed response to failures are offset by the need for increased redundancy and reduced effective bandwidth. To understand this effect more quantitatively we must first know the needed redundancy.

3.2 Needed Redundancy: Replication

First we compute the data expansion needed for high availability in the context of replication-style redundancy. Note that availability implies reliability since lost data is inherently unavailable.

Average lifetime now depends on timeout: $T = T_\tau$. System size and availability also depend on τ, and $N_\tau = N_0/a_\tau$, by our definition of availability.

We wish to know the replication factor, k_a, needed to achieve some per object unavailability target, ϵ_a. (I.e., $1 - \epsilon_a$ has some "number of 9s".)

$$\begin{aligned}\epsilon_a &= P(\text{object } o \text{ is unavailable}) \\ &= P(\text{all } k_a \text{ replicas of } o \text{ are unavailable}) \\ &= P(\text{one replica is unavailable})^{k_a} \\ &= (1 - a_\tau)^{k_a}\end{aligned}$$

which upon solving for k_a yields

$$k_a = \frac{\log \epsilon_a}{\log(1 - a_\tau)} \approx \frac{\log 1/\epsilon_a}{a_\tau} + O(a^2) \qquad (2)$$

We can now evaluate the tradeoff between data maintenance bandwidth and membership timeout. We account for partial availability by replacing B with $a_\tau B$ in Equation (1). Solving for B/N and substituting Equation (2) gives:

$$B_\tau/N_\tau = \frac{2k_a D}{N_\tau a_\tau T_\tau} = \frac{2D}{N_\tau a_\tau T_\tau} \frac{\log \epsilon_a}{\log(1 - a_\tau)} \qquad (3)$$

To apply Equation (3) we must know N_τ, a_τ, and T_τ, which all depend upon participant behavior. We estimate these parameters using data we collected in a measurement study of the availability of hosts in the Gnutella file sharing network. We used a methodology similar to a previous study [10], except that we allowed our crawler to extract the entire membership, therefore giving us a precise estimate of N_τ. Our measurements took place between April 11, 2003 and April 19, 2003.

Figure 2 suggests that discriminating downtime from departure can lead to a factor of 30 savings in maintenance bandwidth. It seems hopeless to field even $1\,TB$ at high availability with Gnutella-like participation.

3.3 Needed Redundancy: Erasure Coding

A technique that has been proposed by several systems is the use of erasure coding [12, 2]. This is more efficient than conventional replication since the increased intra-object redundancy allows the same level of availability to be achieved with much smaller additional redundancy. We now exhibit the analogue of Equation (2) for the case of erasure coding.

With an erasure-coded redundancy scheme, each object is divided into b blocks which are then stored with an effective redundancy factor k_c. The object can be reconstructed from any available m blocks taken from the stored set of $k_c b$ blocks (where $m \approx b$). Object availability is given by the probability that at least b out of $k_c b$ blocks are available:

$$1 - \epsilon_a = \sum_{i=b}^{k_c b} \binom{i}{k_c b} a^i (1 - a)^{k_c b - i}.$$

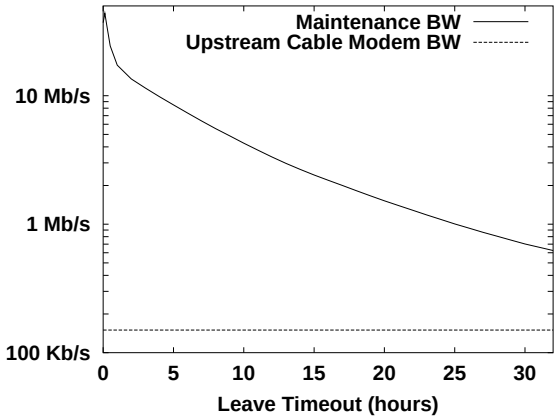

Figure 2. Per node bandwidth to maintain $1\ TB$ of unique data at 6 nines of per-object availability with the system dynamics of $33,000$ Gnutella hosts. Bandwidth is lessened by longer delays responding to failures, but remains quite large in terms of home Internet users. Each host contributes only about $3\ GB$.

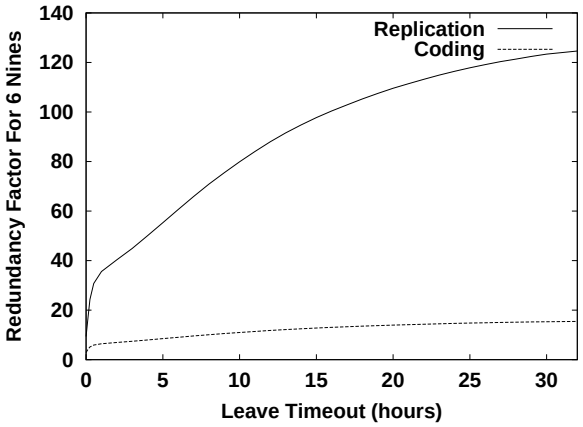

Figure 3. This graph shows that decreased availability from delayed response to failure causes a marked increase in the necessary redundancy. While coding beats replication, the bandwidth savings are only a factor of 8 for our Gnutella trace.

Using algebraic simplifications and the normal approximation to the binomial distribution (see [2]), we get the following formula for the erasure coding redundancy factor and then expand it in a Taylor series:

$$k_c = \left(\frac{\sigma_\epsilon \sqrt{\frac{a(1-a)}{b}} + \sqrt{\frac{\sigma_\epsilon^2 a(1-a)}{b} + 4a}}{2a} \right)^2 \quad (4)$$

$$\approx \frac{\sigma_\epsilon^2}{4b}(1+q)^2 q^{-\frac{1}{2}}\left(\frac{1}{a} - \frac{1}{q} + O(a)\right) \quad (5)$$

$$where \quad q = \sqrt{1 + \frac{4b}{\sigma_\epsilon^2}}.$$

σ_ϵ is the number of standard deviations in a normal distribution for the required level of availability, as in [2]. E.g., $\sigma_\epsilon = 4.7$ corresponds to six nines of availability.

Figure 3 shows the benefits of coding over replication when one uses $b = 15$ fragments. Rather than a replication factor of 120, one can achieve the same availability with only 15 times the storage using erasure codes, for large values of τ, an 8-fold savings. This makes it borderline feasible to store $1\ TB$ of unique data with Gnutella-like participation and about $75\ Kbps$ while-up per node maintenance bandwidth. Utilization is correspondingly *lower* for the same amount of unique data. Only $500\ MB$ of disk per host is contributed. This is surely less than what peers are willing to donate.

Note that all of this is for maintenance only. It would be odd to engineer such highly availabile data and not read it. An actual load is hard to guess, but, as a rule of thumb, one would probably like maintenance to be less than half the total bandwidth. So, the total load one might expect would be greater than $150\ Kbps$ or just at the limit of what cable modems can provide.

This is also not very much service. Only $5,000$ of the $33,000$ Gnutella hosts were usually available. If all these hosts were cable modems, the aggregate bandwidth available would be about $500\ Mbps$, or $250\ Mbps$ if half that is used for data maintenance. By comparison, the same level of service could be provided by five reliable, dedicated PCs, each with a few $300, $250\ GB$ drives and $50\ Mbps$ connections up 99% of the time.

4 Discussion

This section discusses other issues related to the agenda and design of large-scale peer-to-peer storage.

4.1 Admission Control, Load-Shifting

Another strategy to reduce redundancy maintenance bandwidth is to attempt to not admit highly volatile nodes or very similarly shift responsibility to non-volatile hosts. Fundamentally, this strategy weakens how dynamic and peer-symmetric the network one is envisioning. Indeed, a strong enough bias converts the problem into a garden variety distributed systems problem — building a larger storage from a small number of highly available collaborators.

As Figure 4 shows, in Gnutella, the 5% most available hosts provide 29 of the total 72 service years or 40%. The availability of these 6,000 nodes is about 40% on average. If one is generous, one may also view this 5%-subset of more available hosts as a fairer model of the behavior of a hypothetical population of peer-to-peer participants. We repeated our analysis of earlier sections

using just this subset of hosts with a one day membership timeout. The resulting bandwidth requirement is $30\ Kbps$ per node per unique-TB using coding. Using delayed response, coding, and admission control together enables a 1000-fold savings in maintenance bandwidth over the bleak results at the left edge of Figure 2.

The total scale of this storage remains bounded by bandwidth, though. If the 6,000 best 5% of Gnutella peers each donated $3\,GB$ each then a total of $3\,TB$ could be served with six nines of availability. These hosts would each use $100\ Kbps$ of maintenance bandwidth whenever they were participating. Assuming the query load was also about $100\ Kbps$ per host, cable modems would still be adequate to serve this data. The same service also could be supported by 10 universities, each using $\frac{1}{3}$ of the typical OC3 connections and a $\$1,500$ PC.

Stricter admission control rapidly leads to a subset 967 Gnutella hosts with 99.5% availability. This surpasses even observed enterprise wide behavior [4]. The cost of this improvement is a reduction in service time by 10-fold. The real service reduction will depend on the correlation between availability and servable bandwidth. Ideally, this correlation would be strong and positive.

If the per-node bandwidth of the best hosts is roughly 10-fold the per node bandwidth of the excluded hosts then the total service is only cut in half by using just good nodes. Stated in reverse, leveraging tens of thousands of flaky home users only doubles total data service. This fact is further backed up by a simple back-of-the-envelope calculation. Two million cable modem users at 40% availability can serve about as much bandwidth as 2,000 typical high availability universities allowing half their bandwidth for file sharing.

4.2 Hardware Trends

The discussion so far suggests that even highly optimized systems can achieve only a few GB per host with Gnutella-like hosts and cable-modem like connections. However, hardware trends are unpromising.

Year	Disk	Home access		Academic access	
		Speed (Kbps)	Days to send	Speed (Mbps)	Time to send
1990	60 MB	9.6	0.6	10	48 sec
1995	1 GB	33.6	3	43	3 min
2000	80 GB	128	60	155	1 hour
2005	0.5 TB	384	120	622	2 hour

Table 1. Generous bandwidth estimates suggest distributing local disk will get harder. Disk increased by 8000-fold while bandwidth increased only 50-fold.

A simple thought experiment helps us realize the implications of this trend. Imagine how long it would take to upload your hard disk to a friend's machine. Table 1 recalls how this has evolved for "typical" users in recent times. The fourth and sixth columns show an ominous trend for disk space distributors. Disk upload time is getting larger quickly. If peers are to contribute meaningful fractions of their disks their participation must become more and more stable. This supports the main point of this paper: synchronizing randomly distributed, large-scale storage is expensive now, dynamic membership makes it worse, and this situation is worsening.

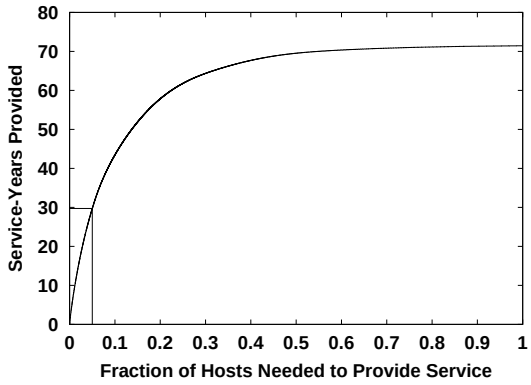

Figure 4. A graph showing how most service time in Gnutella is provided by a tiny fraction of hosts. 5% of hosts provide 40% of the total service time in a three day subset of our trace.

4.3 Incentive Issues

Unlike pioneer systems like Napster and Gnutella, current research trends are toward systems where users serve data that they may have no particular interest in. A good fraction of their outbound traffic might be saturated by access to this data. Storage guarantees exacerbate the problem by inducing a great deal of synchronization traffic above and beyond access traffic. These higher costs may make participation even more capricious than our example of the Gnutella network. Given that stable membership is necessary to reach even modest data scales, participation must be strongly incentivized.

The added value of service guarantees might seem to be one incentive. However, this is not stable since a noticeable downward fluctuation in popularity will make the provided service decline. Another option is having user interfaces which discourage or disallow disconnection, but this is very much against the spirit of a volunteer or donation based system. One reasonable idea is to allow client bandwidth usage to be only proportional to contributed bandwidth. Enforcing this raises many design issues, but since it is an extension to the threat model it seems inappropriate to relegate it to an afterthought.

5 Conclusion

This paper argues that the real scalability problem for robust, Internet-scale storage is not lookup state or procedure, but rather is the *service* bandwidth to field queries and maintain redundancy. For DHT-style systems, maintaining redundancy takes cross-system bandwidth proportional to data scale and membership dynamics. All three properties — redundancy, data, and dynamics — can be high only when cross-system bandwidth is enormous.

The conflict between high availability, large data scales, home user-like bandwidth and fast participation dynamics raises many questions about current DHT research trajectories. In dynamic deployment scenarios, why leverage many nodes to serve data a few reliable ones might? In static deployment scenarios, small lookup-state optimizations may do more harm than good in terms of system complexity and other properties, especially if designers insist on implementing other optimizations in membership state restricted ways. If storage guarantees are inappropriate for large-scale peer-to-peer why worry about lookup guarantees? When anonymity or related security properties are the high priority guarantees, it seems bad to plan on incorporating defenses against threats to these properties as an afterthought.

Acknowledgements

We would like to thank Stefan Saroiu, Krishna Gummadi, and Steven Gribble for supplying the data collected in their study. This research is supported by DARPA under contract F30602-98-1-0237 monitored by the Air Force Research Laboratory, NSF Grant IIS-9802066, and a Praxis XXI fellowship.

References

[1] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Communications of the ACM*, pages 43–48, February 2003.

[2] Ranjita Bhagwan, Stefan Savage, and Geoffrey Voelker. Replication strategies for highly available peer-to-peer storage systems. Technical Report CS2002-0726, UCSD, November 2002.

[3] Ranjita Bhagwan, Stefan Savage, and Geoffrey Voelker. Understanding availability. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, February 2003.

[4] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 2000.

[5] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proc. 18th ACM Symposium on Operating System Principles*, Banff, Canada, October 2001.

[6] John Kubiatowicz. Extracting guarantees from chaos. *Communications of the ACM*, pages 33–38, February 2003.

[7] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM '01 Conference*, San Diego, CA, August 2001.

[8] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-s cale peer-to-peer systems. In *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[9] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. 18th ACM Symposium on Operating System Principles*, Banff, Canada, October 2001.

[10] S. Saroiu, P. K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Computing and Networking 2002 (MMCN'02)*, January 2002.

[11] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM '01 Conference*, San Diego, CA, August 2001.

[12] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[13] Ben Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.