Replication Strategies for Highly Available Peer-to-Peer Storage

Ranjita Bhagwan, David Moore, Stefan Savage, and Geoffrey M. Voelker

Department of Computer Science and Engineering
University of California, San Diego

1 Introduction

In the past few years, peer-to-peer networks have become an extremely popular mechanism for large-scale content sharing. Unlike traditional client-server applications, which centralize the management of data in a few highly reliable servers, peer-to-peer systems distribute the burden of data storage, computation, communications and administration among thousands of individual client workstations. While the popularity of this approach, exemplified by systems such as Gnutella [3], was driven by the popularity of unrestricted music distribution, newer work has expanded the potential application base to generalized distributed file systems [1, 4], persistent anonymous publishing [5], as well as support for high-quality video distribution [2]. The widespread attraction of the peer-to-peer model arises primarily from its potential for both low-cost scalability and enhanced availability. Ideally a peer-to-peer system could efficiently multiplex the resources and connectivity of its workstations across all of its users while at the same time protecting its users from transient or persistent failures in a subset of its components.

However, these goals are not trivially engineered. First-generation peer-to-peer systems, such as Gnutella, scaled poorly due to the overhead in locating content within the network. Consequently, developing efficient lookup algorithms has consumed most of the recent academic work in this area [9, 11]. The challenges in providing high availability to such systems is more poorly understood and only now being studied. In particular, unlike traditional distributed systems, the individual components of a peer-to-peer system experience an order of magnitude worse availability – individually administered workstations may be turned on and off, join and leave the system, have intermittent connectivity, and are constructed from low-cost low-reliability components. One recent study of a popular peer-to-peer file sharing system found that the majority of peers had application-level availability rates of under 20 percent [8].

As a result, all peer-to-peer systems must employ some form of replication to provide acceptable service to their users. In systems such as Gnutella, this replication occurs implicitly as each file downloaded by a user is implicitly replicated at the user's workstation. However, since these systems do not explicitly manage replication or mask failures, the availability of an object is fundamentally linked to its popularity and

users have to repeatedly access different replicas until they find one on an available host. Next-generation peer-to-peer storage systems, such as the Cooperative File System (CFS) [1], recognize the need to mask failures from the user and implement a basic replication strategy that is independent of the user workload.

While most peer-to-peer systems employ some form of data redundancy to cope with failure, these solutions are not well-matched to the underlying host failure distribution or the level of availability desired by users. Consequently, it remains unclear what availability guarantees can be made using existing systems, or conversely how to best achieve a desired level of availability using the mechanisms available.

In our work we are exploring replication strategy design trade-offs along several interdependent axes: Replication granularity, replica placement, and application characteristics, each of which we address in subsequent sections. The closest analog to our work is that of Weatherspoon and Kubiatowicz [10] who compare the availability provided by erasure coding and whole file replication under particular failure assumptions. The most critical differences between this work and our own revolve around the failure model. In particular, Weatherspoon and Kubiatowicz focus on disk failure as the dominant factor in data availability and consequently miss the distinction between short and long time scales that is critical to deployed peer-to-peer systems. Consequently, their model is likely to overestimate true file availability in this environment.

2 Replica granularity

Systems like Gnutella employ whole file replication: files are replicated among many hosts in the system based upon which nodes download those files. Whole file replication is simple to implement and has a low state cost – it must only maintain state proportional to the number of replicas. However, the cost of replicating entire files in one operation can be cumbersome in both space and time, particularly for systems that support applications with large objects (e.g., audio, video, software distribution).

Block-level replication divides each file object into an ordered sequence of fixed-size blocks. This allows large files to be spread across many peers even if the whole file is larger than what any single peer is able to store. However, downloading an object requires that enough hosts storing block replicas are available to reconstruct the entire object at the time the object is requested. If any one replicated block is unavailable, the object is unavailable. For example, measurements of the CFS system using six block-level replicas show that when 50 percent of hosts fail the probability of a block being unavailable is less than two percent [1]. However, if an object consists of 8 blocks then the expected availability for the *entire object* will be less than 15 percent. This dependency is one of the motivating factors for the use of erasure codes with blocking replication.

Erasure codes (EC), such as Reed-Solomon [7], provide the property that a set of k original blocks can be reconstructed from any l coded blocks taken from a set of ek coded blocks (where l is typically close to k, and e is typically a small constant). The addition of EC to block-level replication provides two advantages. First, it can dramatically improve overall availability since the increased intra-object redundancy can tolerate the loss of many individual blocks without compromising the availability

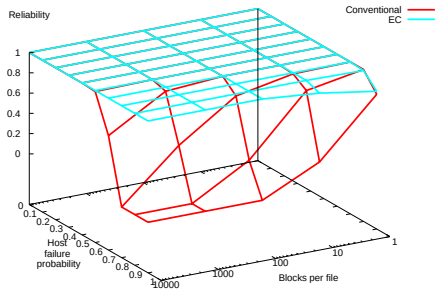

Figure 1: Reliability as a function of the number of blocks per file and host failure probability.

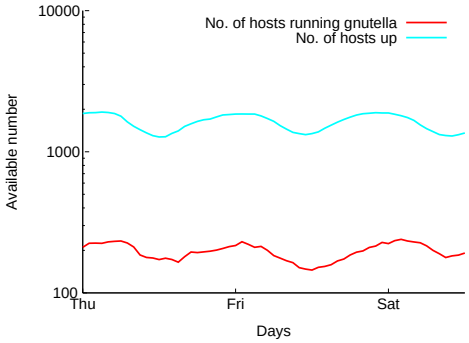

Figure 2: Diurnal patterns in the number of probed hosts available on the network, as well as in the number of these hosts running Gnutella.

of the whole file. Second, the ability to reconstruct an object from many distinct subsets of EC blocks, permits a low-overhead randomized lookup implementation that is competitive in state with whole-file replication. Rather than maintain the location of every replica for each block, a system using EC blocks can simply track the location of each peer holding *any* block belonging to the object.

2.1 Replication and Host Failure

As an initial experiment to explore the trade-off between conventional block replication and block replication with erasure codes, we simulate the replication and distribution of a single file in a idealized system of N hosts. The file is divided into fixed-size blocks and replicated. Replicated blocks are randomly assigned to hosts, each of which has the same uniform failure probability. We then simulate random host failure and determine whether the file is still recoverable from the system assuming an ideal lookup system and perfect network conditions. A file is recoverable if, after the failures, enough of its blocks survive to reconstruct its original contents. We repeat this experiment 100 times and measure the fraction of times that the file is completely recoverable. For the purposes of this experiment, we define this fraction as the reliability of the system.

We use the term storage redundancy to refer to the amount of storage a replication technique uses. In the conventional case, storage redundancy is simply the number of replicas of the file. For the erasure coded case, redundancy is introduced not only by replication, but also by the encoding process. In this case, storage redundancy is the number of replicas times the encoding redundancy. A file consisting of k blocks is encoded into ek blocks, where $e > 1$ is what we call the encoding redundancy (or *stretch factor*). In our simulations, $e = 2$. Comparing strategies when using storage redundancy is more fair than comparing them using number of replicas. Figure 1 shows simulation results of the idealized system as a function of blocks per file k and host failure probability, for a storage redundancy of 20.

From the figure, we see that for host failure probabilities less than 0.5 both replication schemes achieve high reliability. For higher host failure probabilities, the relia-

bilities of the two techniques diverge. Because we abstract away block size, the case where only one block is used for a file corresponds to the use of whole-file replication. Note that the "conventional" curve for this case has the best reliability compared with using more blocks per file. Also, as number of blocks per file increases, the two techniques quickly diverge in reliability. Erasure coded blocks actually increases reliability since the system has more flexibility in choosing among hosts to reconstruct the file. However, conventional block replication decreases in reliability, and is very sensitive to high host failure probability.

The implication of these results is that using conventional blocking and scattering those blocks across a large number of relatively unreliable hosts makes the system less reliable. However, by decoupling exactly which blocks are required to reconstruct a file from the hosts storing replicas of those blocks, erasure coded replication is able to achieve excellent reliability even when the underlying hosts are quite unreliable. And the reliability of using erasure coding increases, rather than decreases, for larger files.

3 Replica placement

For the purposes of file availability, peer-to-peer systems should not ignore the availability characteristics of the underlying workstations and networks on which they are implemented. In particular, systems should recognize that there is wide variability in the availability of hosts in the system. Saroiu and Gribble found that fewer than 20 percent of Gnutella's peer systems had network-level availability in excess of 95 percent [8], while over half of the remainder had availability under 20 percent. Given such a wide variability, the system should not place replicas blindly: more replicas are required when placing on hosts with low availability, and fewer on highly available hosts. Moreover, under many predictable circumstances, peer failures may be correlated. We performed a study of the Gnutella network similar to Saroiu and Gribble's work, and found that the number of hosts running Gnutella is well correlated with time of day, and shows a diurnal pattern, as shown in Figure 2. Also, independent investigations of client workstation availability has shown strong time-zone specific diurnal patterns associated with work patterns [6]. As a consequence, placing replicas in out-of-phase time zones may be a sound replication strategy.

4 Application characteristics

The relationship between when data are requested and the time at which they must be delivered creates several opportunities for optimization based on application characteristics. For example, traditional Unix-like file system applications usually require an entire file object to be delivered to the application buffer cache before the application can make forward progress. However, the order in which this data is delivered and variations in overall delay rarely have a significant impact. In contrast, streaming media workloads typically only require that the data surrounding the current playout point be available, but this particular data must be delivered in a timely fashion for the application to operate correctly.

5 Summary

We are investigating strategies for using replication to design and implement highly available storage systems on highly unavailable peer-to-peer hosts. In particular, we are exploring the availability provided by whole object and blocking replication, and of erasure coding in such systems. In addition, we are investigating how application properties such as object size, timeliness of delivery, and workload properties such as object popularity should influence replication strategies. We are also investigating how replica placement policies can be tuned to compensate for diurnal patterns in host availability, and to take advantage of out-of-phase time zones. Eventually, we plan to implement our results in a prototype system for practical evaluation.

References

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

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

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

[4] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, 2000.

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

[6] D. Moore. Caida analysis of code-red, http://www.caida.org/analysis/security/code-red/, 2001.

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

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

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

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

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