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

Abstract

We are investigating strategies for using replication to design and implement highly reliable peer-to-peer systems. In particular, we are comparing the use of whole object and blocking replication, and pursuing the use of erasure codes with blocking replication as a novel technique for achieving high reliability even for systems primarily composed of hosts with poor availability. In this paper, we briefly present the different replication strategies we are exploring, how those strategies can be influenced by application characteristics and host availability, and some preliminary simulation results.

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 Napster [11] and Gnutella [6], was driven by the popularity of unrestricted music distribution, newer work has expanded the potential application base to generalized distributed file systems [1, 3, 8], persistent anonymous publishing [4, 9], wide-area databases [7], as well as support for high-quality video distribution [5]. The wide-spread 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 [13, 14, 16, 17]. The challenges in providing high availability to such systems is even 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 [15].

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 Napster and 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) [3], recognize the need to mask failures from the user and implement a basic replication strategy that is independent of the user workload.

However, how best to replicate data to build highly available peer-to-peer systems is still an open problem. In our work we are exploring replication strategy design tradeoffs along several interdependent axes:

- *Application characteristics*. The demands made on a storage system, and the utility associated with making a given piece of data available, depend on the application being used. Small whole-file applications, common under Unix-like workloads, have significantly different availability requirements from large-file, streaming media workloads.
- *Replica placement.* Existing systems such as CFS assume that failures are independent and uniformly distributed, although it is well understood that this is not so. Hence, the placement of replicas according to measured and estimated failure distributions can have a significant impact on overall application availability.
- *Replication granularity.* Whole-file replication allows simple and low-overhead implementations of naming and lookup, while block-level replication allows increased performance through parallel downloads and better balancing of large file objects. Finally, block-level erasure coding can enhance overall availability while providing the flexibility of block-level replication and, surprisingly, the minimal state requirements of whole-file designs.

In this whitepaper, we briefly present the different replication strategies we are exploring, how those strategies can be influenced by application characteristics and host failure distributions, and some preliminary simulation results.

2 Application characteristics

Peer-to-peer systems are being used for a wide range of applications, including music and video sharing, wide-area file systems, archival file systems, software distribution. Two key properties of peer-to-peer applications that impact the use of replication are object sizes and timeliness of object download.

First, larger objects take longer to replicate and are more cumbersome to manage as a whole, and naturally motivate the use of block-level replication. However, when conventional blocking is used for large objects, the reliability of the system depends upon an increasingly large number of hosts being available at the same time. As a result, the availability of an object is inversely related to its size: the larger the object, the worse the availability.

Second, the relationship between when data is requested and the time at which it must be delivered. 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 play-out point be available, but this particular data must be delivered in a timely fashion for the application to operate correctly.

3 Replica placement

For the purposes of reliability, 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 [15], 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. For example, independent investigations of client workstation availability has shown strong time-zone specific diurnal patterns associated with work patterns [10]. As a consequence, placing replicas in out-of-phase time zones may be a sound replication strategy.

4 Replica granularity

Gnutella and Napster 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 approach has several benefits. Because individual parts of an object may be named independently, a block-level system may download different parts of an object simultaneously from different peers and reduce overall download time. Also, because the unit of replication is small and fixed, the cost to replicate an individual block can be small and can be distributed among many peers. Finally, block-level representation allows large files to be spread across many peers even if the whole file is larger than what any single peer is able to store.

There are two tradeoffs, however. First, in order to locate individual blocks, a block-level system must maintain state proportional to the product of the number of replicas and the number of blocks in an object. More seriously, though, 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 replicas fail the probability of a block being unavailable is less than two percent [3]. 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 [12] and Tornado [2] codes, provide the property that a set of n original blocks

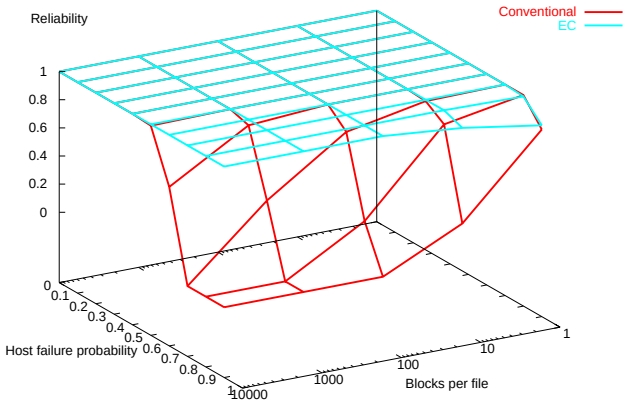

Figure 1: Reliability as a function of the number of blocks per file and host failure probability.

can be reconstructed from any m coded blocks taken from a set of kn (where m is typically close to n, and k 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 of the whole file. For example, for the same storage requirements, one can either duplicate n blocks of an object or code those blocks into $2n$ EC blocks. If the blocks are distributed across $2n$ nodes, then, when using standard block replication, at least one of two hosts for each block must be available to reconstruct the object. When using EC block replication, however, any n of the $2n$ hosts storing EC replicas need be available to reconstruct the object.

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. When a client desires an object it can simply request random blocks until enough data is returned to reconstruct the object. This approach is surprisingly efficient. For an object consisting of n EC-blocks, the expected number of blocks downloaded in this random fashion is O($nlogn$ + c) (or a penalty of only $logn$ unnecessary blocks downloaded).

5 Replication and Host Failure

As an initial experiment to explore the tradeoff 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. For a given storage redundancy, the two blocking techniques use different amounts of storage to replicate a given file. 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. 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 storage redundancy and host failure probability. Redundancy is introduced in the conventional case only through replication. Hence storage redundancy, for the conventional case, is simply the number of replicas in the system. On the other hand, for the erasure coded case, redundancy is introduced not only by replication of the file, but also by the encoding. A file consisting of k blocks is encoded into ek blocks, where $e > 1$ is what we call "encoding redundancy" (or *stretch factor* [2]). So, to take into account both these sources of redundancy, the storage redundancy for the erasure coded scenario is the product of number of replicas and the encoding redundancy. In our simulations, $e = 2$.

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 reliabilities of the two techniques diverge and the magnitude of the divergence depends on the number of blocks per file. 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.

With multiple blocks per file, the number of blocks per file roughly corresponds to the size of the file since we assume a constant block size. For a small number of blocks per file, or small files, the two techniques quickly diverge in reliability. Erasure coded blocks actually increases in 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. For peer-to-peer file systems, where the average file size is small and, hence, the average number of blocks per file is small, the effect is only severe for very high host failure probabilities. However, for peer-to-peer systems that serve large files such as music and video, conventional block replication has poor reliability even for host failure probabilities close to 0.5.

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.

6 Summary

We are investigating strategies for using replication to design and implement highly reliable peer-to-peer systems. In particular, we are comparing the use of whole object and blocking replication, and pursuing the use of erasure codes with blocking replication as a novel technique for achieving high reliability even for systems primarily composed of hosts with poor availability. In addition, we are investigating how application properties such as object size, timeliness of delivery, workload properties such as object popularity, and network properties such as host availability should influence replication strategies. Initial experiments indicate that the use of erasure codes with blocking replication is promising, and we are further exploring their use. Eventually, we plan to implement our results in a prototype system for practical evaluation.

References

[1] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Measurement and Modeling of Computer Systems*, pages 34–43, 2000.

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

[3] 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.

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

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

[6] Gnutella homepage, http://gnutella.wego.com.

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

[8] 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.

[9] 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.

[10] D. Moore. Caida analysis of code-red, 2001.

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

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

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

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

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

[16] 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.

[17] 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.