Erasure Coding vs. Replication: A Quantitative Comparison

Hakim Weatherspoon and John D. Kubiatowicz

Computer Science Division
University of California, Berkeley
{hweather, kubitron}@cs.berkeley.edu

Abstract. Peer-to-peer systems are positioned to take advantage of gains in network bandwidth, storage capacity, and computational resources to provide long-term durable storage infrastructures. In this paper, we quantitatively compare building a distributed storage infrastructure that is self-repairing and resilient to faults using either a replicated system or an erasure-resilient system. We show that systems employing erasure codes have mean time to failures many orders of magnitude higher than replicated systems with similar storage and bandwidth requirements. More importantly, erasure-resilient systems use an order of magnitude less bandwidth and storage to provide similar system durability as replicated systems.

1 Introduction

Today's exponential growth in network bandwidth, storage capacity, and computational resources has inspired a whole new class of distributed, peer-to-peer storage infrastructures. Systems such as Farsite[2], Freenet[4], Intermemory[3], OceanStore[8], CFS[5], and PAST[7] seek to capitalize on the rapid growth of resources to provide inexpensive, highly-available storage without centralized servers. The designers of these systems propose to achieve high availability and long-term durability, in the face of individual component failures, through replication and coding techniques.

Although wide-scale replication has the potential to increase availability and durability, it introduces two important challenges to system architects. First, system architects must increase the number of replicas to achieve high durability for large systems. Second, the increase in the number of replicas increases the bandwidth and storage requirements of the system.

This paper makes the following contributions: First, we briefly quantify the availability gained using erasure codes. Second, we show that erasure-resilient codes use an order of magnitude less bandwidth and storage than replication for systems with similar *mean time to failure* (MTTF). Third, we show that employing *erasure-resilient codes* increase the MTTF of the system by many orders of magnitude over simple replication with the same storage overhead and

repair[1] times. The contributions of this work over [3, 12] are the addition of bandwidth as a comparison.

2 Background

Two common methods used to achieve high durability of data are complete replication[2, 7] and parity schemes such as RAID[9]. The former imposes extremely high bandwidth and storage overhead, while the latter does not provide the robustness necessary to survive the high rate of failures expected in the wide area.

An *erasure code* provides redundancy without the overhead of strict replication. Erasure codes divide an object into m fragments and recode them into n *fragments*, where $n > m$. We call $r = \frac{m}{n} < 1$ the *rate* of encoding. A rate r code increases the storage cost by a factor of $\frac{1}{r}$. The key property of erasure codes is that the original object can be reconstructed from *any* m fragments. For example, using an $r = \frac{1}{4}$ encoding on a block divides the block into $m = 16$ fragments and encodes the original m fragments into $n = 64$ fragments; increasing the storage cost by a factor of *four*.

Erasure codes are a superset of replicated and RAID systems. For example, a system that creates four replicas for each block can be described by an ($m = 1$, $n = 4$) erasure code. RAID level 1, 4, and 5 can be described by an ($m = 1$, $n = 2$), ($m = 4$, $n = 5$), and ($m = 4$, $n = 5$) erasure code, respectfully.

Data Integrity: Erasure coding in a malicious environment requires the precise identification of failed or corrupted fragments. Without the ability to identify corrupted fragments, there is potentially a factorial combination of fragments to try to reconstruct the block; that is, $\binom{n}{m}$ combinations. As a result, the system needs to detect when a fragment has been corrupted and discard it. A secure verification hashing scheme can serve the dual purpose of identifying and verifying each fragment. It is necessarily the case that any m *correctly verified* fragments can be used to reconstruct the block. Such a scheme is likely to increase the bandwidth and storage requirements, but can be shown to still be many times less than replication.

3 Assumptions

We assume that replicated and erasure encoded systems consist of a collection of *independently, identically distributed* failing disks – same assumption made

[1] Data is periodically repaired to replace lost redundancy in both replicated and erasure encoded systems.

by both *A Case for RAID*[9] and disk manufacturers – and that failed disks are immediately replaced by new, blank ones[2]. During dissemination, each replica (or *fragment*) for a given block is placed on a unique, randomly selected disk. Finally, we postulate a global sweep and repair process that scans the system, attempting to restore redundancy by reconstructing each block and redistributing lost replicas (or fragments) over a new set of disks – Repair in *RAID*[9] is *triggered* when a disk fails, which is fundamentally different than sweep and repair. Some type of repair is required; otherwise, data would be lost in a couple years regardless of the redundancy. We denote the time period between sweeps of the same block an *epoch*.

4 Availability

Availability gained using erasure codes is a result of exploiting the statistical stability of a large number of components. The availability of a block can be computed as follows

P_o probability that a block is available
n total number of fragments
m number of fragments needed for reconstruction
N total number of machines in the world
M number of currently unavailable machines

$$P_o = \sum_{i=0}^{n-m} \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}} \tag{1}$$

where the probability a block is available is equal to the number of ways in which we can arrange unavailable fragments on unreachable servers multiplied by the number of ways in which we can arrange available fragments on reachable servers, divided by the total number of ways in which we can arrange all of the fragments on all of the servers.

With a million machines, ten percent of which are currently down, simply storing two complete replicas provides only two nines (0.99) of availability. A rate $\frac{1}{2}$ erasure coding of a document into 32 fragments gives the document over eight nines of availability (0.999999998), yet consumes the same amount of storage and bandwidth, supporting the assertion that *fragmentation increases availability*.

[2] We are ignoring other types of failures such as software errors, operational errors, configuration problems, etc., for this simple analysis.

5 System Comparison

We use the same system size (*total blocks*) and write rate ($\frac{wBlocks}{s}$) to compare systems based on replication to that of erasure codes. In this section we make three comparisons. First, we fix the *mean time to failure* (MTTF) of the system and *repair epoch*. Second, we fix the storage overhead and repair epoch. Finally, we fix the MTTF of the system and the storage overhead.

We compare replicated and erasure encoded systems (denoted by x) in terms of total storage S_x, total bandwidth (leaving the source or entering the destination) BW_x, and the total number of disk seeks required to sustain rate (repair, write, and read) D_x. We do not compare reads when considering storage and bandwidth because the amount of data required to read a block is the same for both systems; that is, m fragments is equivalent to one replica in storage and bandwidth requirements.

5.1 Fix MTTF and Repair Epoch

In this subsection we compare replicated systems to erasure encoded systems that have the same fixed system MTTF and repair epoch.

Assuming that we store and do not delete data, the total system size in terms of the total number of fixed size blocks B that will be reached throughout the systems lifetime can be computed using the total number of users N as follows

$$B = N \cdot \frac{wBlocks}{s} \cdot \text{total seconds}$$

More generally, given a system size (defined by the number of users) we focus on answering the question, what are the resources required to store data in a system long-term. We define the *durability of the system* to be the expected MTTF of losing *any* block is sufficiently larger than the expected lifetime of the system given some number of users. That is

$$MTTF_{system} = \frac{MTTF_{block}}{B} \gg \text{total seconds}$$

We derive how to compute storage, bandwidth, and disk seeks required by solving the following equations.

$$\begin{aligned} S_x &= \text{total bytes stored in system } x \\ BW_x &= BW_{x_{write}} + BW_{x_{repair}} \\ D_x &= D_{x_{write}} + D_{x_{repair}} + D_{x_{read}} \end{aligned}$$

S_x is the total storage capacity required of the system x, BW_x is a function of the bandwidth required to support both writes and repair of the total storage

every repair epoch, and D_x is the number of disk seeks required to support repair, writes, and reads. The repair bandwidth is computed by dividing the total bytes stored by the repair epoch. Next, we compute the storage for both systems

$$S_{repl} = b \cdot R \cdot B$$
$$S_{erase} = \frac{b}{m} \cdot n \cdot B = b \cdot \frac{1}{r} \cdot B$$

where R is the number of replicas, $r = \frac{m}{n}$ is the rate of encoding, and b is the block size. We now compute the bandwidth in terms of storage as follows

$$BW_{repl} = b \cdot R \cdot N \cdot \frac{wBlocks}{s} + \frac{S_{repl}}{e_{repl}}$$
$$BW_{erase} = b \cdot \frac{1}{r} \cdot N \cdot \frac{wBlocks}{s} + \frac{S_{erase}}{e_{erase}}$$

where e_x is the repair epoch of system $x \in \{replica, erasure\}$. We show now that the bandwidth due to only the original data being written and repaired can be expressed as *DataRate*

$$DR_x = N\frac{wBlocks}{s} + \frac{B}{e_x}$$

Further, we compute the number of disk seeks required to support writes, repair, and reads. db_{sz} is the size of a disk block.

$$D_{repl} = (R \cdot N \cdot \tfrac{wBlocks}{s} + R \cdot \tfrac{B}{e_{repl}} + 1) \cdot \tfrac{b}{db_{sz}}$$
$$D_{erase} = (n \cdot N \cdot \tfrac{wBlocks}{s} + n \cdot \tfrac{B}{e_{erase}} + m) \cdot \tfrac{b}{m \cdot db_{sz}}$$

The above equation states that the number of disk seeks required is dependent on the number of replicas (or total number of fragments), throughput, system size, repair epoch, the number of replicas (or fragments) needed to reconstruct the block, and the number of replicas (or fragments) that can fit in a disk block.

Finally, a replicated system can be compared to a similar erasure encoded system with the following bandwidth, storage, and disk seek ratios

$$\frac{S_{repl}}{S_{erase}} = R \cdot r \tag{2}$$
$$\frac{BW_{repl}}{BW_{erase}} = \frac{R \cdot DR_{repl}}{\frac{1}{r} \cdot DR_{erase}} = R \cdot r \tag{3}$$
$$\frac{D_{repl}}{D_{erase}} = \frac{(R \cdot DR_{repl} + 1) \cdot \frac{b}{db_{sz}}}{(n \cdot DR_{erase} + m) \cdot \frac{b}{m \cdot db_{sz}}} \approx R \cdot r \tag{4}$$

We make the abstract numbers concrete using the following parameters as appropriate. Bolosky *et. al*[2] measured that an average workstation produces

$35\frac{MB}{hr}$ of data. We associate a workstation with a user. We set $b = 8$kB blocks, $db_{sz} = 8$kB disk blocks, $N = 2^{24}$ users, $e_{repl} = e_{erase} = 4$ months, and $MTTF_{system} > 1000$ years. As a consequence of the former parameters we calculate $B = 10^{17}$ total blocks; hence, $MTTF_{block} = 10^{20}$ years. Finally, using the analysis described in [12] and reprinted in Appendix A, we solve for the number of replicas and rate and compute that $R = 22$ and $r = \frac{32}{64} = \frac{1}{2}$ satisfy above constraints, respectively.

Applying these parameters to equations 2, 3, and 4 we produce the following result

$$\frac{BW_{repl}}{BW_{erase}} = 11$$
$$\frac{S_{repl}}{S_{erase}} = 11$$
$$\frac{D_{repl}}{D_{erase}} = 11$$

These results show that a replicated system requires an order of magnitude more bandwidth, storage, and disk seeks as an erasure encoded system of the same size.

5.2 Fix Storage Overhead and Repair Epoch

The same formulas from subsection 5.1 above can be used to verify durability of system calculations presented in [3, 12]. For example, using our simple failure model presented in section 3 and parameters in section 5.1, we set repair time of $e_{repl} = e_{erase} =$ *four* months, $R =$ *two* replicas[3], and rate $r = \frac{32}{64}$. Both the replicated and erasure encoded systems have the same apparent storage overhead of a factor of *two*. Using Appendix A, we compute the $MTTF_{block}$ of a block replicated onto two servers as 74 years and the $MTTF_{block}$ of a block using a rate $\frac{1}{2}$ code onto $n = 64$ servers as 10^{20} years! It is this difference that highlights the advantage of erasure coding.

5.3 Fix MTTF and Storage Overhead

As a final comparison, we can fix the MTTF and storage overhead between a replicated and erasure encoded system. This implies that the storage and bandwidth for writes are equivalent for these two systems. In this case erasure encoded systems must be repaired less frequently, and hence, require less repair bandwidth.

[3] In section 5.1 $R = 22$ to attain the same durability

For example, we can devise systems that have a $MTTF_{block} = 10^6$ years, a factor four storage overhead, $B = 1000$ blocks, and a $MTTF_{system} = 1000$ years. The replicated system meets the above requirements using $R =$ four replicas and a repair epoch of $e_{repl} =$ one month. The erasure encoded system meets the same requirements using an $r = \frac{16}{64} = \frac{1}{4}$ code and a repair epoch of $e_{erase} = 28$ months. The replicated system uses 28 times more bandwidth than erasure encoded system for repair.

If, instead, the $MTTF_{block} = 10^{20}$ years, $B = 10^{17}$ blocks, $MTTF_{system} = 1000$ (as described in subsection 5.1), and still using a factor of *four* storage overhead, the erasure encoded system meets the requirements using an $r = \frac{16}{64} = \frac{1}{4}$ code and a repair epoch of $e_{erase} = 12$ months, but a replicated system with $R = 4$ replicas would have to repair all blocks almost instantly and continuously.

6 Discussion

The previous section presented the advantages of erasure codes, but there are some caveats as well. Two issues that we would like tob highlight are the need for intelligent buffering of data and the need for caching.

Each client in an erasure-resilient system sends messages to a larger number of *distinct* servers than in a replicated system. Further, the erasure-resilient system sends smaller "logical" blocks to servers than the replicated system. Both of these issues could be considered enough of a liability to outweigh the results of the last section. However, we do not view it this way. First, we assume that the storage servers are utilized by a number of clients; this means that the additional servers are simply spread over a larger client base. Second, we assume intelligent buffering and message aggregation. Although the outgoing fragments are "smaller", we simply aggregate them together into larger messages and larger disk blocks, thereby nullifying the consequences of fragment size. These assumptions are implicit in our exploration via metrics of total bandwidth and number of disk blocks in the previous section.

Another concern about erasure-resilient systems is that the time and server overhead to perform a read has increased, since multiple servers must be contacted to read a single block. The simplest answer to such a concern is that mechanisms for *durability* should be separated from mechanisms for *latency reduction*. Consequently, we assume that erasure-resilient coding will be utilized for durability, while replicas (i.e. caching) will be utilized for latency reduction. The nice thing about this organization is that replicas utilized for caching are *soft-state* and can be constructed and destroyed as necessary to meet the needs of temporal locality. Further, prefetching can be used to reconstruct replicas

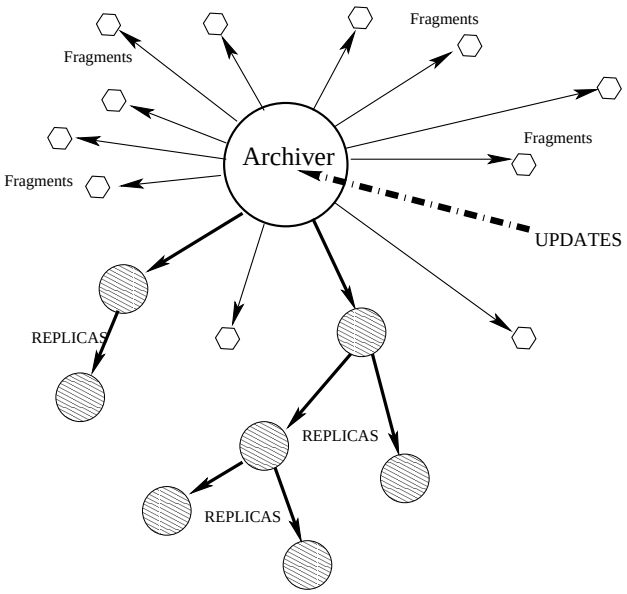

Fig. 1. Hybrid Update Architecture: *Updates are sent to a central "Archiver", which produces archival fragments at the same time that it updates live replicas. Clients can achieve low-latency read access by utilizing replicas directly.*

from fragments in advance of their use. Such a hybrid architecture is illustrated in Figure 1. This is similar to what is provided by OceanStore [8].

7 Future Work

We present some open research issues that affect both replicated and erasure encoded systems alike.

Failure Independence: The most troubling assumption of the previous sections are that failures are *independent* and *identically distributed*. This is not true for all sets of storage servers. We list two possible techniques to address independence. First, most routing overlay networks, such as CAN[11], Chord[13], Pastry[7], and Tapestry[14], provide a location and routing infrastructure that permits fragments to be distributed to geographically diverse locations, eliminating a large class of correlations caused by natural disasters, denial of service attacks, and administrative boundaries. Second, sophisticated measurement and modeling techniques could be used to choose a set of nodes that are maximally independent during fragment dissemination.

Efficient Repair: Action must be taken to maintain replicas (or fragments) despite failure; otherwise, all replicas (or fragments) will be lost. The sweep and repair is simplistic because it assumes that all data in the world is reconstructed on some periodic basis. While this is not entirely implausible (every object is independent and could be repaired in parallel), it does consume many resources.

8 Related Work

The idea of a global-scale, distributed, persistent storage infrastructure was first motivated by Ross Anderson in his proposal for the Eternity Service[1]. To our knowledge the tradeoffs between bandwidth, storage, and disk seeks when comparing a replicated system to an erasure coded system have not been discussed in literature. A discussion of the durability gained from building a system from erasure codes first appeared in Intermemory[3]. The authors describe how their technique increases an object's resilience to node failure, but the system does not incorporate a repair mechanism that would also increase objects durability. More recently, there has appeared a large body of work on the subject of wide-scale, distributed storage. FreeHaven[6] is a system for anonymous publishing that uses an information dispersal algorithm, in a manner analogous to erasure codes.

Our discussion in Section 6 motivated the need for a hybrid system. OceanStore[8] is a distributed storage system that uses the notion of *promiscuous caching*, where replicas are soft-state and only for read benefit, while erasure codes are used for durability.

Other systems with similar goals include PAST[7] and Farsite[2]. PAST is a large-scale peer-to-peer storage utility. Farsite seeks to provide an organizational-scale distributed file system comprised of cooperating, but not trusting, machines. Both rely on replication for durability and availability.

9 Conclusion

In this paper we have described the availability and durability gains provided by an erasure-resilient system. We quantitatively compared systems based on replication to systems based on erasure codes. We showed that the *mean time to failure* (MTTF) of an erasure encoded system can be shown to be many orders of magnitude higher than that of a replicated system with the same storage overhead and repair period. A novel result of our analysis showed that erasure-resilient codes use an order of magnitude less bandwidth and storage than replication for systems with similar MTTF. Finally, if care is taken to take advantage of temporal and spatial locality erasure encoded systems can use an order of magnitude less disk seeks than replicated systems.

References

1. ANDERSON, R. The eternity service. In *Proceedings of Pragocrypt* (1996).
2. BOLOSKY, W., DOUCEUR, J., ELY, D., AND THEIMER, M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. of Sigmetrics* (June 2000).
3. CHEN, Y., EDLER, J., GOLDBERG, A., GOTTLIEB, A., SOBTI, S., AND YIANILOS, P. Prototype implementation of archival intermemory. In *Proc. of IEEE ICDE* (Feb. 1996), pp. 485–495.
4. CLARK, I., SANDBERG, O., WILEY, B., AND HONG, T. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, CA, July 2000), pp. 311–320.
5. DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proc. of ACM SOSP* (October 2001).
6. DINGLEDINE, R., FREEDMAN, M., AND MOLNAR, D. The freehaven project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability* (July 2000).
7. DRUSCHEL, P., AND ROWSTRON, A. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. of ACM SOSP* (2001).
8. KUBIATOWICZ, J., ET AL. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of ASPLOS* (Nov. 2000), ACM.
9. PATTERSON, D., GIBSON, G., AND KATZ, R. The case for raid: Redundant arrays of inexpensive disks, May 1988.
10. PATTERSON, D., AND HENNESSY, J. *Computer Architecture: A Quantitative Approach*. Forthcoming Edition.
11. RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. A scalable content-addressable network. In *Proceedings of SIGCOMM* (August 2001), ACM.
12. RHEA, S., WELLS, C., EATON, P., GEELS, D., ZHAO, B., WEATHERSPOON, H., AND KUBIATOWICZ, J. Maintenance free global storage in oceanstore. In *Proc. of IEEE Internet Computing* (2001), IEEE.
13. STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM* (August 2001), ACM.
14. ZHAO, B., JOSEPH, A., AND KUBIATOWICZ, J. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB//CSD-01-1141, University of California, Berkeley Computer Science Division, April 2001.

A Appendix: Durability Derivation

In this appendix we describe the mathematics involved in computing the *mean time to failure* (MTTF) of a *particular* erasure encoded block.

Considering the server failure model and repair process as described in Section 3, we can calculate the MTTF of a block as follows. First, we calculate the probability that a given fragment placed on a randomly selected disk will survive until the next epoch as

$$p(e) = \int_{e}^{\infty} \frac{l p_d(l)}{\mu} \frac{l-e}{l} dl \tag{5}$$

$$= \frac{1}{\mu} \int_{e}^{\infty} p_d(l)(l-e)dl \tag{6}$$

where e is the length of an epoch, μ is the average life of a disk, and $p_d(l)$ is the probability distribution of disk lives. This equation is derived similarly to the equation for the residual average lifetime of a randomly selected disk. The term $\frac{l-e}{l}$ reflects the probability that, given a disk of lifetime l, a new fragment will land on the disk early enough in its lifetime to survive until the next epoch. The probability distribution $p_d(l)$ was obtained from disk failure distributions in [10], augmented by the assumption that all disks still in service after five years are discarded along with their data.

Next, given $p(e)$, we can compute the probability that a block can be reconstructed after a given epoch as

$$p_b(e) = \sum_{m=rn}^{n} \binom{n}{m} [p(e)]^m [1-p(e)]^{n-m} \tag{7}$$

where n is the number of fragments per block and r is the rate of encoding. This formula computes the probability that at least rn fragments are still available at the end of the epoch.

Finally, the MTTF of a block for a given epoch size can be computed as

$$\text{MTTF}_{block}(e) = e \cdot \sum_{i=0}^{\infty} i[1-p_b(e)][p_b(e)]^i \tag{8}$$

$$= e \cdot \frac{p_b(e)}{1-p_b(e)}. \tag{9}$$

This last equation computes the average number of epochs a block is expected to survive times the length of an epoch.