Reliability Mechanisms for Very Large Storage Systems

Qin Xin[†] *qxin@cs.ucsc.edu*
Ethan L. Miller[†] *elm@cs.ucsc.edu*
Thomas Schwarz, S.J.[‡] *tschwarz@calprov.org*
Darrell D. E. Long[†] *darrell@cs.ucsc.edu*
Scott A. Brandt[†] *scott@cs.ucsc.edu*
Witold Litwin[*] *Witold.Litwin@dauphine.fr*

[†]*University of California, Santa Cruz*
[‡]*Santa Clara University*
[*]*Université Paris 9 Dauphine*

Abstract

Reliability and availability are increasingly important in large-scale storage systems built from thousands of individual storage devices. Large systems must survive the failure of individual components; in systems with thousands of disks, even infrequent failures are likely in some *device. We focus on two types of errors: nonrecoverable read errors and drive failures. We discuss mechanisms for detecting and recovering from such errors, introducing improved techniques for detecting errors in disk reads and fast recovery from disk failure. We show that simple RAID cannot guarantee sufficient reliability; our analysis examines the tradeoffs among other schemes between system availability and storage efficiency. Based on our data, we believe that two-way mirroring should be sufficient for most large storage systems. For those that need very high reliability, we recommend either three-way mirroring or mirroring combined with RAID.*

1. Introduction

System designers have been making computer systems with better performance, lower cost, and larger scale over the last 20 years, but have not emphasized fault tolerance, high availability and reliability in most designs. Siewiorek and Swarz [23] noted four reasons for an increasing concern on fault tolerance and reliability: harsher environments, novice users, increasing repair costs, and larger systems.

As storage systems scale up, the use of more devices increases both capacity and bandwidth, but it also makes disk failures more common. In petabyte-scale file systems[1], disk failures will be a daily (if not more frequent) occurrence—data losses with this frequency cannot be tolerated. Moreover, disk rebuild times are becoming longer as increases in disk capacity outpace increases in bandwidth [8], lengthening the "window of vulnerability" during which a subsequent disk failure (or a series of subsequent disk failures) causes data loss. At the same time, the increase in the total capacity and bandwidth of a storage system increases the likelihood of nonrecoverable read errors. Standard disk drives have nonrecoverable read error rates of 1 in 10^{14}–10^{15} bits, making it likely that a large storage system will suffer several such errors per hour even when no disk has completely failed.

We investigate the reliability issues in a very large-scale storage system built from Object-Based Storage Devices (OBSDs) [25] and study these two types of failures: disk failures and nonrecoverable read errors. An OBSD is a network-attached storage device [7] that presents an interface of arbitrarily-named objects of variable size. In our OBSD storage system, files are broken up into objects, and the objects are then distributed across many devices. This OBSD system is expected to store 2 PB of data, with additional storage capacity for redundancy information. In such a large-scale distributed storage system with thousands of nodes, there is a high likelihood that at least one node will be down at any given time. Simply using RAID is not reliable enough because it takes too long to rebuild a disk. Assuming the RAID must still provide data to clients while rebuilding, it would take more than a day to rebuild a 500 GB disk at 5 MB/second. With five disks in a RAID, the chance of a second failure during a one day rebuild is about 0.1%, resulting in a Mean Time To Data Loss (MTTDL) from a 2 PB system of less than three years.

Data for a single file may be distributed across hundreds

[1] A petabyte (PB) is 10^{15} bytes.

This paper will appear in the *20th IEEE Symposium on Mass Storage Systems*, held jointly with the *11th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, CA, April 2003.

of individual OBSDs. This distribution must be done in a decentralized fashion so that there is no central bottleneck when a single file is accessed by thousands of clients. Individual OBSDs can manage low-level storage allocation themselves, leaving the problems of allocating data to OBSDs and replication to the higher-level file system. If an OBSD consists of multiple drives, we can use RAID or other redundancy mechanisms within the OBSD to safeguard data. This approach is feasible and could increase the availability of a single OBSD. However, it is expensive because of the need for custom hardware and software to manage the redundancy and the need for a higher-speed network to connect the collection of drives. Additionally, RAID hardware and software are often complex and have greater chance of failures. If the non-disk hardware or software in an OBSD fails, the data is unavailable, though recovery time from such faults is often lower than that for rebuilding a failed disk. An alternate approach would be to put the OBSD software on each drive and attach it directly to the network. This approach might be less expensive in the long run, but would require more redundancy between OBSDs to provide a comparable level of reliability.

Instead, in our work, the higher-level file system employs redundancy. We investigate two mechanisms: keeping two or three copies of each object (mirroring), and using RAID-style parity across objects. To do this, we group objects into redundancy sets and add parity of the objects to the sets as in a software RAID. We use Lazy Parity Backup (LPB) to convert multiple copies of static objects into parity objects, which will be described in Section 4.2.2.

Two OBSDs are "related" if they store objects that are copies of each other or are part of the same redundancy set. We assume that two OBSDs are related at most once by making use of optimal data allocation algorithms [9] to ensure that the storage system is optimally "declustered" [2]. This distributes the work of reconstructing the objects on a failed OBSD optimally through the system. To speed up this reconstruction, we employ Fast Mirroring Copy (FMC), where the reconstructed objects are stored on different OBSDs throughout the system. This will repair an OBSD failure in minutes instead of days. We will discuss the FMC scheme in detail in Section 4.2.1.

To combat nonrecoverable read errors, we store a signature of an object with the object. When we read an object, we recalculate the signature and compare it with the stored signature. This flags bad objects and ensures that failed reads are recognized, allowing the system to correct the error in the data.

2. Related Work

There has been some research beyond RAID [4] in reliability and recovery for large-scale systems, though most of it has focused on the use of storage in wide-area systems. For example, OceanStore [10, 26] is designed to have a very long MTTDL (Mean Time To Data Loss), but at the cost of dramatically increasing the number of disk requests per block written. Pangaea [20] allows individual servers to continue serving most of their data even when disconnected; however, this approach is designed for wide-area systems, and does not permit files to be striped across dozens of servers for higher bandwidth. Microsoft's Farsite project [1, 5] investigated the issue of replication for systems built from relatively unreliable commodity workstations. They focused on reliability, investigating replica placement strategies that take server reliability into account. However, Farsite is not designed for high-bandwidth applications, and must deal with servers that are less reliable than those in a single large-scale file system.

Muntz and Liu [17] proposed declustering a disk array of n disks by grouping the blocks in the disk array in reliability groups of size g. Menon and Mattson [15] proposed distributed sparing, where the spare disk is broken up and distributed through the array. In such an array, the reads for a data rebuild are distributed and so are the writes (to the spare space). Long [14] described an efficient algorithm for managing the consistency of mirrored disks.

Other researchers have studied the question of how much replication is really necessary as well as techniques to reduce that level of replication. The Recovery Oriented Computing project [18] is trying to reduce the recovery time in order to gain higher availability and lower total cost of ownership. Castro and Liskov [3] propose a secure replication system to tolerate Byzantine faults and narrow the window of vulnerability. Litwin and Schwarz [12] present a family of linear hashing methods for distributing files. The algorithms reduce the number of messages and scale to the growth or shrink of files efficiently. Schwarz [21] has built a Markov model to estimate system availability; we will apply this model to the scalable file system we are designing.

3. Nonrecoverable Errors

By storing vast quantities of data on commodity disks, we will reach the point where the built-in error detection (and correction) of the disks no longer prevents nonrecoverable errors. For example, error rates of one nonrecoverable error of 1 in 10^{15} bits are common. A single drive running at 25 MB/s would experience such an error about once a year. In a large system with 10,000 disks, however, such

an error will occur once per hour *somewhere* in the system. While the rate may be too small to worry about in a typical commercial database where human errors far outnumber machine-generated errors, there are applications where very large amount of data need to be stored without any data corruption, *e.g.* large-scale simulation data files for the United States Department of Energy.

3.1. Signature Scheme

We have designed a scheme that detects and corrects small errors in blocks stored on disk in a large-scale system. Our scheme consists of two components: a signature scheme that *flags* corrupted data, and a RAID 5-like [4] mechanism that creates groups of blocks spread across different disks and stores the parity of the block on yet another disk. This redundancy allows us to reconstruct any corrupted block of data.

To flag corrupted data, we associate a *signature* with each data block. The signature is a fixed-length bit string that is calculated from the contents of the block. A signature resembles a hash function. If even a bit in a block has changed, the signature should also change. Since a signature is much smaller than a block, collisions are possible. A good signature scheme minimizes the probability that two random blocks have the same signature and minimizes the probability that a plausible change in a block (e.g. inversion of a bit) leads to a change in the signature. We calculate the signature when we store a data block and store the signature on the same disk separately from the data block. When we read the data block, we recompute the signature and compare it with the previously stored signature value. If the two signature values agree, we conclude that the block is correct; otherwise, we flag an error. This error is most likely caused by an incorrect block, but could also be the result of a corruption of the signature file. Since the signature is much smaller than the block, the error is likely to have occurred in the block. Because of its relatively small size, it is possible to use an error correcting code or storage scheme for the signature file.

To correct data we introduce redundancy into the storage scheme. This can be done by mirroring or triplicating data, using RAID Level 5 parity, or using erasure correcting codes such as Even-Odd (which takes n blocks and adds to them two blocks such that any n survivors among the $n+2$ blocks suffice to reconstruct all $n+2$) or Reed-Solomon block codes [2, 19, 21]. This redundancy must be on different disks from the data it protects to guard against disk failure, but it might be possible to keep it on the same disk if it only must guard against nonrecoverable errors.

3.2. Galois Power Signatures of Storage Objects

Storage objects are made up of blocks, which in turn are strings of symbols. Symbols are merely bit-strings of length f, with f being either 8 or 16. Because of the byte-oriented character of computer processing, f should be a multiple of 8, but since we use tables of size 2^f, f should not be too large either.

3.2.1. Galois Fields Galois fields [11] are a well known algebraic structure consisting of a finite set, for our purposes the set of all bit-strings of length f, and two operations, addition and multiplication. We use the usual symbols for these operations. We also have two special elements, the zero, denoted by 0 and in our case represented by the bit-string $00\ldots00$ of f zeroes, and the one, denoted by 1 and represented by the bit-string $00\ldots01$. The same algebraic laws involving these two elements and the two operations hold as for calculations in the real or complex numbers. For example, $ab+ac=a(b+c)$ and $0a=0$. Mathematically, Galois fields are determined by the number of elements in them. Since there are 2^f different bit-strings of length f, we denote our Galois fields by $\mathsf{GF}(2^f)$. Addition in $\mathsf{GF}(2^f)$ is the exclusive-or of the bit strings. This gives a new rule: $\forall a \in \mathsf{GF}(2^f)$, $a+a=0$, so that every element is its own additive inverse.

Multiplication is more complicated to implement. Our implementation uses logarithm and antilogarithm tables with respect to a so-called primitive element α. An element α is primitive if all powers α^i, $0 \le i \le 2^f-1$ are different. In consequence, each non-zero element β can be written as $\beta=\alpha^i$ as a uniquely determined power i $(0 \le i \le 2^f-1)$. We then write $i=\log_\alpha(\beta)$ and $\beta=\mathrm{antilog}_\alpha(i)$. Primitive elements exist in abundance within a Galois field. The product of two non-zero elements β and γ is

$$\beta \cdot \gamma = \mathrm{antilog}_\alpha(\log_\alpha(\beta)+\log_\alpha(\gamma)).$$

In this formula, the addition is taken modulo 2^f-1 and the logarithm and antilogarithm tables take up 2^f-1 entries of size f bits.

3.2.2. Galois Power Series Signatures (GPSS) Our signature consists of n components of length f. We first define a single component. In the following, we assume that a page (a block) has at least n signatures and that $n < 2^f-1$. We write the page as a series of l symbols $P=p_1p_2p_3\ldots p_l$ l, each containing f bits. Let β be an element of $\mathsf{GF}(2^f)$. We define

$$\mathrm{sig}_\beta(P)=\sum_{\mu=1}^{l} p_\mu \beta^{\mu-1}$$

and call it the β signature of page P. The page signature $\mathrm{sig}_\beta(P)$ is a Galois field and hence a bit-string of length f.

Now, let α be a primitive element of $\mathsf{GF}(2^f)$. We define the *n-fold α-signature* to be

$$\mathrm{sig}_{\alpha,n}(P) = \left(\mathrm{sig}_{\alpha}(P), \mathrm{sig}_{\alpha^2}(P), \ldots, \mathrm{sig}_{\alpha^n}(P)\right).$$

It is a vector of n symbols, *i. e.*, a bit-string of length nf. We tabulate a list of important properties of $\mathrm{sig}_{\alpha,n}$. Useful properties of the single symbol signature are also contained in Schwarz, *et al.* [22], and the proofs are in Mokadem [16] and Litwin and Schwarz [13] and forthcoming publications.

1. The probability that two (uniformly distributed) random pages have the same n-fold α-signature is 2^{-nf}.
2. Any change in any n or less page symbols changes $sig_{\alpha,n}$.
3. If we concatenate page P of length l with page Q of length m then $sig_{\beta}(P|Q) = \mathrm{sig}_{\beta}(P) + \beta^l \cdot \mathrm{sig}_{\beta}(Q)$. Thus, $\mathrm{sig}_{\alpha,n}$of the concatenation can be calculated from the $\mathrm{sig}_{\alpha,n}$of the concatenated pages and their length.
4. If we change page P by modifying r characters starting at position s and if we call Δ the string of changes $(\Delta = (\delta_1, \delta_2, \ldots, \delta_r)$ with $\delta_i = p_i^{\mathrm{new}} - p_i^{\mathrm{old}})$ then $\mathrm{sig}_{\beta}(P^{\mathrm{new}}) = \mathrm{sig}_{\beta}(P^{\mathrm{old}}) + \beta^{s-1} \cdot \mathrm{sig}_{\beta}(\Delta)$. Thus, $\mathrm{sig}_{\alpha,n}$can be calculated from the old page signature, the signature of the delta-string, and the location of the change.
5. If we form the parity of a number of pages (made to be of equal length by padding with zeroes if necessary) then $\mathrm{sig}_{\alpha,n}$ of the parity page is the parity of the $\mathrm{sig}_{\alpha,n}$ of the pages. This is even true if the parity is the generalized parity in a p+q erasure-correcting generalized Reed-Solomon code. This property allows us to check through the signatures whether the parities are in sync with the pages that they protect.

These properties are highly useful, but any signature scheme having them cannot be cryptographically secure. Since cryptographically secure signature schemes such as SHA-1 always have a performance disadvantage, algebraic signatures such as GPSS have much to recommend them.

3.2.3. Implementation The speed of signature calculation is crucial in many applications. If we were to implement the signature formulæ naively, the performance would be quite tolerable, but there is still room for improvement. First, we interpret the page symbols directly as logarithms. This saves a log table look-up. The logarithms range from 0 to $2^f - 2$ (inclusively) with an additional value for $\log(0)$, which is set to $2^f - 1$. Next, the signature calculations form the product with α^i, which has i as the logarithm. One does not need to look this value up either. Finally, we dispense with the awkward addition modulo $2^f - 1$ by doubling the antilogarithm table. We give sample pseudo-code in Figure 1. We experimented with two Galois fields, with $f = 8$, and with $f = 16$ [16]. We used standard 1.8 GHz Pentium 4 machines, already considered relative slow in 2003. Results show that using the larger Galois field ($f = 16$) leads to up to 5% improvement. It appears that further substantial performance improvements of 20–50% may occur for the smaller field at least, by forcing the load of the entire antilogarithm table into cache. This can be done using the PREFETCH command for Intel-compatible microprocessors [6].

```
#define SYM Symbol
SYM signature (pagelen, SYM page_{0...pagelen-1})
  SYM sig ← 0
  for i ← 0 to pagelen − 1
    if page_i ≠ 2^f − 1
      sig ← antilog(i + page_i) ⊕ sig
  return sig
```

Figure 1. Pseudo-code for the single symbol signature calculation. The antilog function is calculated by table lookup, as described in Section 3.2.1.

4. Disk Failures

Protecting against nonrecoverable disk errors guarantees that individual data blocks are protected, but does not provide a good protection mechanism to deal with the failure of an entire disk. As discussed in Section 4.1, a 2 PB storage system will experience about one disk failure per day. Since disk capacity is increasing at a faster rate than disk bandwidth, the time needed to rebuild an entire disk is becoming longer, lengthening the "window of vulnerability" during which a subsequent disk failure is likely to cause data loss.

We investigated several redundancy mechanisms and developed mechanisms for fast recovery in a large-scale storage system, resulting in lower risk of data loss at an acceptable storage overhead.

4.1. Redundancy Mechanisms

We assume that our storage system holds 2 PB of data, and is built from 500 GB disk drives. Such drives are not

Table 1. Cost and overhead of different reliability mechanisms.

Method	Cost ($ Millions)		Storage Efficiency
	2002	2005	
Mirror 2	2	0.2	50%
Mirror 3	4	0.4	33%
RAID 5+1	3	0.3	40%

yet available, but will be by 2004–2005, assuming that current growth rates in disk capacity [8] continue. Note that a storage system containing 2 PB of data will require more capacity for redundancy. The ratio between data capacity and total storage capacity is the *storage efficiency*. We further assume that our disks have a mean time to failure (MTTF) of 10^5 hours. This is significantly shorter than that specified by drive manufacturers, which is typically 10^6 hours, but is longer than the 50,000 hours reported by sites such as the Internet Archive [24]. For simplicity, we assume the failure rates of the disks in the system are identical and independent, though this may not be true if many disks are from the same manufacturing batch and are running in the same environments.

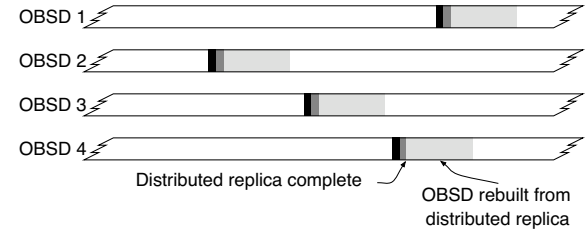
We use the term *redundancy set* to refer to a block group composed of data blocks or objects and their associated replicas or parity blocks. A single redundancy set will typically contain 1 MB to 1 TB, though we expect that redundancy sets will be at least 1 GB to minimize bookkeeping overhead and reduce the likelihood that two redundancy sets will be stored on the same set of OBSDs.

We consider three methods to configure the organization of a redundancy set: two-way mirroring (Mirror 2), three-way mirroring (Mirror 3), and RAID 5 with mirroring (RAID 5+1). In *n*-way mirroring, each data block in the redundancy set is stored *n* times, with each replica stored on a different OBSD. Under RAID 5+1, each OBSD consists of multiple disks organized as a RAID 5, in which a redundancy set is distributed among five disk drives, and each data block is mirrored on two OBSDs. Table 1 summarizes the cost and storage efficiency of the three redundancy schemes using a current disk price of $1/GB and an estimated disk price of $0.1/GB in 2005.

4.2. Fast Recovery Mechanisms

When we use any of the redundancy mechanisms described above, there is still a small chance that the system will lose data. For example, Mirror 3 will fail when two of the three OBSDs in the redundancy set fail and the third fails while the other two are being rebuilt. We can use one

(a) A second OBSD failure within the window of vulnerability from the first causes data loss.



(b) By creating a distributed replica before rebuilding the failed OBSD, the system reduces the window of vulnerability and avoids data loss.

Figure 2. Window of vulnerability

of two mechanisms to deal with this situation: Fast Mirroring Copy (FMC) or Lazy Parity Backup (LPB).

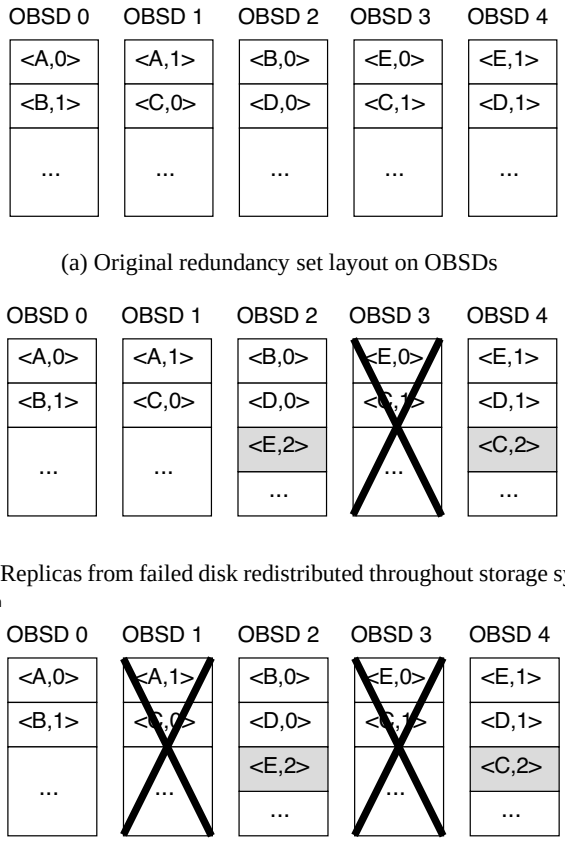
4.2.1. Fast Mirroring Copy Fast Mirroring Copy (FMC) quickly backs up blocks in a redundancy set affected by the loss of a disk. Rather than attempting to immediately rebuild an entire disk, FMC recreates the lost replicas throughout the storage system. When a disk failure is detected, each disk constructs a list of redundancy sets stored locally that have lost a member on the failed disk. For each item in the list, the disk causes the construction of a replacement elsewhere in the storage system. Each OBSD in the system does this in parallel, resulting in a great speedup over rebuilding a single disk. For example, it takes one day to rebuild a failed disk with a capacity of 500 GB at a rate of 5 MB/sec. However, it only takes 100 seconds to recreate a replica for each of the 500 MB redundancy sets that were on the failed disk, as long as all of the copies can proceed in parallel. While we can only achieve this speed if we carefully arrange redundancy sets in the storage array, a random placement still yields rebuild times of minutes instead of tens of hours. Since data is vulnerable to a subsequent failure during data reconstruction, as shown in Figure 2, FMC dramatically increases the longevity of the data in the storage system by allowing the reconstruction to use the parallelism inherent in massive storage systems.

The size of the "window of vulnerability" is determined by the mean time to repair (MTTR) and mean time to failure (MTTF) for an individual disk, where $MTTF \gg MTTR$. If *MTTR* is long, then there will be a higher chance that the other replicas will be lost during the reconstruction of a failed OBSD. One redundancy set with n replicas distributed on n OBSDs will fail if there is an overlap in the failure and reconstruction time among all n OBSDs.

After a failure, FMC creates a new replica for each redundancy set that had a replica on the failed disk. Rather than create all of the replicas on the replacement disk, however, FMC attempts to place all of the new replicas on different disks. Since both the remaining good copies of the affected redundancy sets and the replicas being created are distributed throughout the storage system, the replicas can be created in parallel, dramatically reducing the time necessary to recreate a new replica for the data on the failed disk. We could achieve the same effect by simply keeping more replicas of all data; however, this approach would require extra space. Because FMC can create replicas quickly, it provides nearly the redundancy of an extra replica without actually storing the extra replica until it is needed.

Figure 3 shows how FMC operates, using an example with Mirror 2, five OBSDs, and five redundancy sets labeled A–E. Each replica stored on an OBSD is identified by a tuple $\langle$*redundancy set*, *replica ID*$\rangle$, where the *replica ID* is used to distinguish different replicas belonging to the same redundancy set. Figure 3(a) shows the initial states of the five OBSDs and the replicas that they contain. If one of the OBSDs fails—OBSD 3 in our example—replicas $\langle E,0\rangle$ and $\langle C,1\rangle$ are lost. When this failure is detected, OBSD 4 immediately copies E to OBSD 2, creating $\langle E,2\rangle$, and OBSD 1 copies C to OBSD 4, creating $\langle C,2\rangle$, as shown in Figure 3(b). As soon as this copy has completed, the data that was on OBSD 3 is protected against another failure. In a storage system with thousands of OBSDs, replication can proceed in parallel, reducing the window of vulnerability from the time needed to rebuild an entire OBSD to the time needed to create one or two replicas of a redundancy set. If another OBSD were to fail after this process completes but before the data was restored to the failed disk, no redundancy set would lose data, as shown in Figure 3(c). In Figure 3(c), OBSD 1 has failed before OBSD 3 has been replaced; even though two disks have failed in a system with two-way mirroring, no data is lost. FMC has created the replica $\langle C,2\rangle$, preventing the loss of data from redundancy set C that might have occurred with normal two-way mirroring.

FMC relies on a mapping of replicas to OBSDs that will guarantee that the replicas of one redundancy set will not be stored on the same OBSD and that two redundancy sets share as few OBSDs as possible. This mapping can be done



(a) Original redundancy set layout on OBSDs

(b) Replicas from failed disk redistributed throughout storage system

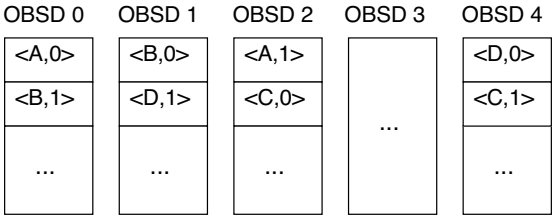
(c) Second failure does not cause loss of data

Figure 3. Fast Mirroring Copy

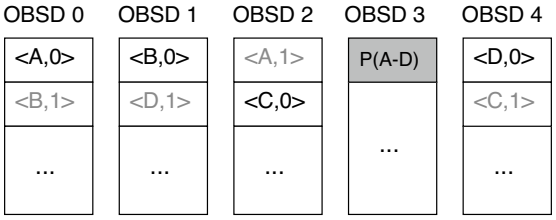
using a table, which will grow linearly with the number of redundancy sets, or with an algorithm such as that described by Honicky and Miller [9], which reduces the storage required at each client by allowing replica placement to be computed on the fly.

4.2.2. Lazy Parity Backup Lazy Parity Backup (LPB) has the same goal—protecting data by replication—but works by creating parity blocks in the background when the associated data blocks have not been modified for some time. This method creates RAID-like structures across OBSDs. If FMC is used for rapidly-changing data, LPB can be used for more static data to gain additional reliability with lower storage overhead. This technique is somewhat similar to one used in AutoRAID [27], but operates on a far larger storage system.

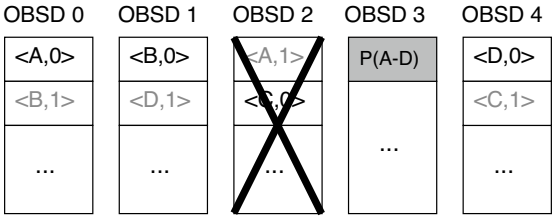
Figure 4 shows how LPB protects data in a small system, similar to that used in Figure 3. $P(d_1$–$d_n)$ represents the parity block for data blocks with object identifiers d_1 to d_n. Figure 4(a) shows the initial data layout, with each of four OBSDs having a primary and mirror replica from



(a) Original data layout on OBSDs



(b) Parity calculated across data on different OBSDs. The second replica from each redundancy set (shown with grayed-out identifiers) can be reclaimed to save space, or can be retained for added reliability.



(c) A single OBSD failure does not cause loss of data.

Figure 4. Lazy Parity Backup

a redundancy set. In the background, the system generates parity across the redundancy sets, resulting in the layout in Figure 4(b). The mirror replica for each redundancy set may be reused if space is at a premium or may be kept to further increase reliability. If an OBSD fails, as depicted in Figure 4(c), no data is lost even if the mirror replicas are no longer available. To further improve overall system reliability, FMC can be combined with LPB to rapidly create a distributed replica of a failed disk protecting data with parity. LPB can use more complex error correcting codes to generate multiple error correction "replicas" for a given set of data, allowing the system to survive multiple OBSD failures.

From the example shown in Figure 4, it may seem that LPB places strict constraints on data placement. In a system with thousands of OBSDs, this is not the case. The only rule that must be enforced is that no OBSD may have more than one element from a stripe. With a stripe width of four in a system with five OBSDs, this is a difficult constraint. However, a stripe width of ten in a system with thousands of OBSDs does not suffer a similar difficulty in laying out data to follow this rule because the stripe width is a small fraction of the number of OBSDs.

Table 2. Parameters for a 2 PB storage system.

Parameter	Value
Z (total data in the system)	2 PB
γ (recovery rate)	10^2 GB/hr
N (number of redundancy sets)	Z/S
$MTTF_{disk}$	10^5 hours
D (disks in an OBSD RAID 5)	5
S (data in one redundancy set)	varies

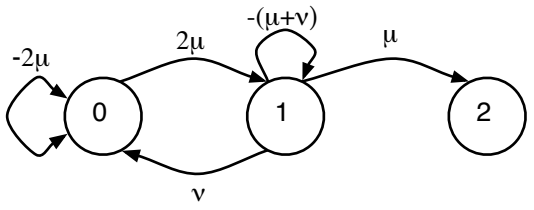


Figure 5. State Transitions for Mirror 2

Consistency problems may arise when we use FMC or LPB to make data more reliable, especially for workloads in which data changes rapidly. We can use a signature scheme or other consistency protocols to maintain data consistency.

4.3. System Availability

Using Markov models, we compared the mean time to data loss (MTTDL) in a 2 PB of Mirror 2, Mirror 3, and RAID 5+1. The storage system parameters we used in our comparison are listed in Table 2. We use μ as the failure rate and ν as the repair rate in figures of state transitions and the calculations. Here, $\nu \gg \mu$, meaning that mean time to repair a disk is much shorter than mean time between failures.

Figure 5 shows state transitions for one redundancy set using the Mirror 2 mechanism, where μ is the failure rate and ν is the repair rate. State 0 indicates that both OBSDs that contain a copy of the objects in the redundancy set are functioning properly; if either one of them is down, then it goes to state 1. The model goes to state 2 only when the second OBSD is down while the earlier failed one has not yet been rebuilt.

Assuming that there is at most only one failure or repair happening during a very short time interval Δt, we find that

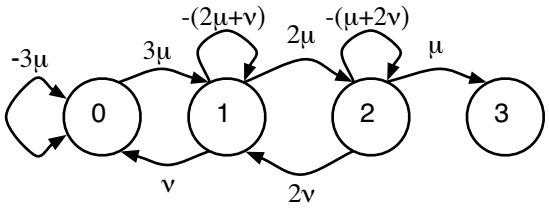


Figure 6. State Transitions for Mirror 3

MTTDL of one redundancy set is

$$MTTDL_{RS-Mirror\,2} = \frac{3\mu+\nu}{2\mu^2} = \frac{\nu}{2\mu^2} + \Delta_{RS-Mirror\,2} \quad (1)$$

The relative error is

$$\frac{\Delta_{RS-Mirror\,2}}{MTTDL_{RS-Mirror\,2}} = \frac{3\mu}{3\mu+\nu} \approx \frac{3\mu}{\nu} \quad (2)$$

Since $\frac{\mu}{\nu}$ is very small (10^{-7} when the size of a redundancy set is 1 GB), the approximate *MTTDL* for one redundancy set under Mirror 2 is

$$MTTDL_{RS-Mirror\,2} = \frac{\nu}{2\mu^2}\,. \quad (3)$$

We then derived the *MTTDL* of one redundancy set when the Mirror 3 mechanism is used from the Markov model shown in Figure 6 in a similar way:

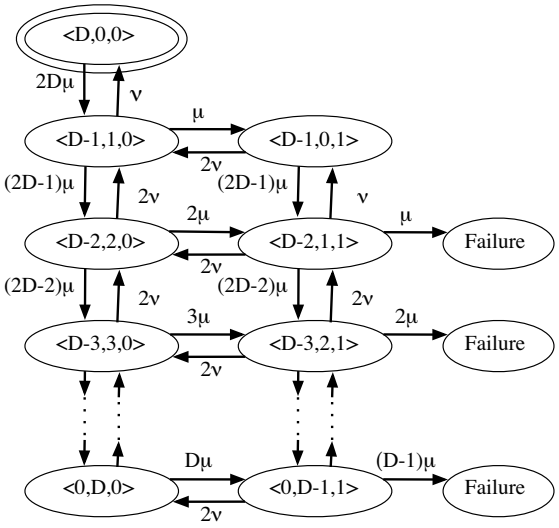
$$MTTDL_{RS-Mirror\,3} = \frac{2\nu^2+7\mu\nu+11\mu^2}{6\mu^3} = \frac{\nu^2}{3\mu^3} + \Delta_{RS-Mirror\,3} \quad (4)$$

$$\frac{\Delta_{RS-Mirror\,3}}{MTTDL_{RS-Mirror\,3}} = \frac{7\mu\nu+11\mu^2}{2\nu^2+7\mu\nu+11\mu^2} \approx \frac{7\mu}{2\nu}. \quad (5)$$

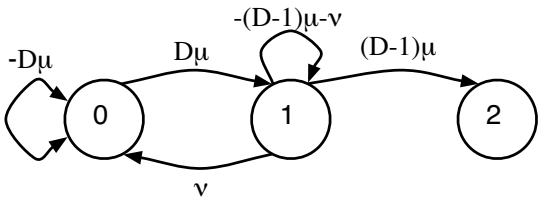
The approximate *MTTDL* for Mirror 3 and the relative error are shown in Equations 4 and 5. From these equations, we see that the approximate *MTTDL* for a redundancy set in Mirror 3 is

$$MTTDL_{RS-Mirror\,3} = \frac{\nu^2}{3\mu^3}\,. \quad (6)$$

The Markov model of one redundancy set under RAID 5+1 is shown in Figure 7(a). In this state diagram, D is the total number of disks in one RAID 5 and $\langle x,y,z\rangle$ indicates that there are x pairs of OBSDs in which both of them are in operation, y pairs of OBSDs in which one of the them are in operation and the other one is down, and z pairs of OBSDs in which neither of the two OBSDs are working. Here, we refer to the two mirrored OBSDs as a *pair*.



(a) State transitions for RAID 5+1



(b) State transitions for RAID 5

Figure 7. Markov models for RAID 5+1

State transitions with the RAID 5+1 mechanism are more complicated than those for Mirror 2 and Mirror 3 since the model goes to the failure state only when two OBSDs in RAID 5 fail and the two corresponding mirroring OBSDs in another RAID 5 fail at the same time. We simplify the Markov model by first deriving the failure rate and repair rate for a pair of OBSDs. We then substitute the failure rate of a mirrored pair of OBSD into the Markov model for a single RAID depicted in Figure 7(b). The derivation is as follows:

$$MTTDL_{RS-RAID5} = \frac{(2D-1)\cdot\mu_{RAID5}+\nu_{RAID5}}{D\cdot(D-1)\cdot{\mu_{RAID5}}^2}. \quad (7)$$

Using Equation 3, we have

$$\mu_{RAID5} = \frac{2\mu^2}{3\mu+\nu} \quad \text{and} \quad \nu_{RAID5} = \nu\,. \quad (8)$$

Thus, we find the *MTTDL* of one redundancy set under RAID 5+1 is

$$MTTDL_{RS-RAID5+1} = \frac{(3\mu\nu+\nu^2+(2D-1)\cdot 2\mu^2)\cdot(3\mu+\nu)}{D\cdot(D-1)\cdot 4\mu^4}$$

$$= \frac{\nu^3}{D\cdot(D-1)\cdot 4\mu^4} + \Delta_{RS-RAID5+1}\,. \quad (9)$$

The relative error is

$$\frac{\Delta_{RS-RAID5+1}}{MTTDL_{RS-RAID5+1}} \approx \frac{6\mu}{\nu}\,. \quad (10)$$

The *MTTF* for RAID 5+1 is approximately

$$MTTDL_{RS-RAID5+1} \approx \frac{\nu^3}{4D\cdot(D-1)\mu^4}\,. \quad (11)$$

To compare the MTTDL of the three redundancy schemes, we used the following equations which approximate (to within 1%) the MTTDL for each of the redundancy mechanisms. *MTTF* for one redundancy set is just the *MTTF* of a disk, ($MTTF_{disk}$), so we have

$$MTTF_{disk} = \frac{1}{\mu} \quad (12)$$

and *MTTR* (Mean Time To Repair) for a single redundancy set is:

$$MTTR_{RS} = S/\gamma = \frac{1}{\nu}\,. \quad (13)$$

for each of $N = \frac{Z}{S}$ redundancy sets.

For a system with N redundancy sets, since $\frac{1}{MTTDL_{RS}}$ is very small, we have

$$MTTDL_{system} = \frac{1}{1-(1-\frac{1}{MTTDL_{RS}})^N} \approx \frac{MTTDL_{RS}}{N} \quad (14)$$

Using the above equations, we find the *MTTDL* of the whole system for each of three mechanisms is as follows:

$$\begin{aligned} MTTDL_{Mirror\,2} &= \frac{\frac{MTTF_{disk}^2}{2\cdot MTTR_{RS}}}{N} \\ &= \frac{MTTF_{disk}^2\cdot\gamma}{2\cdot Z} \quad (15) \\ MTTDL_{Mirror\,3} &= \frac{\frac{MTTF_{disk}^3}{3\cdot MTTR_{RS}^2}}{N} \\ &= \frac{MTTF_{disk}^3\cdot\gamma^2}{3\cdot S\cdot Z} \quad (16) \\ MTTDL_{RAID\,5+1} &= \frac{\frac{MTTF_{disk}^4}{4\cdot D\cdot(D-1)MTTR_{RS}^3}}{N} \\ &= \frac{MTTF_{disk}^4\cdot\gamma^3}{4\cdot D\cdot(D-1)\cdot S^2\cdot Z} \quad (17) \end{aligned}$$

Using Equations 15, 16, and 17 and Table 2, we calculated the MTTDL for each redundancy mechanism using different sizes for a single redundancy set, as shown in Figure 8. For each mechanism, we show MTTDL for both $MTTDL_{disk} = 10^5$ hours and the manufacturers' claims of $MTTDL_{disk} = 10^6$ hours.

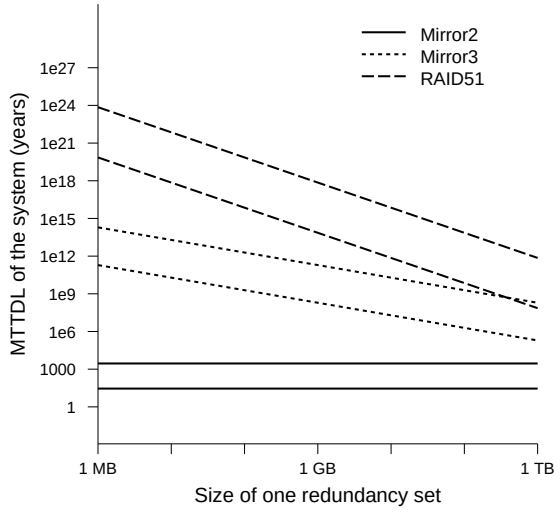


Figure 8. Mean time to data loss in a 2 PB storage system. The upper line for each configuration is for a system built from disks with a 10^6 hour MTTF, and the lower line for each configuration is for a system built from disks with a 10^5 hour MTTF.

4.4. Discussion

Our first result is that, as shown in Equation 15, MTTDL for the system does not vary with the size of a redundancy set. Though larger redundancy sets require more time for recovery, there are fewer of them, and the likelihood that any redundancy set falls as the total number of sets decreases. These two effects are balanced for Mirror 2, so the size of a single redundancy set does not affect overall system MTTDL. For Mirror 3 and RAID 5+1 mechanisms, however, MTTDL decreases as the size of a single redundancy set increases. In both cases, the decrease in reliability due to longer recovery time overwhelms the increased reliability from having fewer, larger redundancy sets.

Figure 8 seems to indicate that smaller redundancy sets provide longer MTTDL. However, this approach has several limitations. First, overall file system bandwidth will decrease if redundancy sets are too small because individual disk transfers will be too small. This sets a lower limit of 256 KB–4 MB for redundancy sets. Second, we assume that redundancy sets fail independently. If there are too many redundancy sets, however, many will share multiple disks, causing correlated failures. Third, the bookkeep-

ing necessary for millions of small redundancy sets will be overwhelming. For all these reasons, we believe it is unlikely that redundancy sets will be much smaller than 200 MB–1 GB.

Disk lifetime is another important factor in calculating MTTDL for the entire storage system. An order of magnitude improvement in $MTTDL_{disk}$ from 10^5 hours to 10^6 hours can improve overall MTTDL by a factor of 100 for Mirror 2, 1000 for Mirror 3, and 10,000 for RAID 5+1. The use of a controlled environment to ensure longer disk lifetimes will result in a major benefit in overall system reliability.

Increasing the recovery rate γ can also improve overall system reliability. Placing a higher priority on disk transfer rate and faster recovery will greatly improve reliability by reducing the "window of vulnerability" during which the system may lose data. Doubling the recovery rate will double the reliability of Mirror 2, but will increase the reliability of RAID 5+1 by a factor of eight.

For a system with 2 PB of storage, we believe that Mirror 2 will provide sufficient redundancy at an acceptable cost. MTTDL for such a system will be about 30 years regardless of the redundancy set size, allowing us to use larger redundancy sets to reduce bookkeeping overhead. Mirror 3 and RAID 5+1 can provide much longer MTTDL—up to 2×10^8 years for Mirror 3, and 10^{14} years for RAID 5+1 if the size of one redundancy set is 1 GB. It is also interesting that when the size of one redundancy set gets big, close to 100 GB, the reliability achieved by using Mirror 3 will exceed that achieved by using RAID 5+1. This, however, assumes a 10^6 hour *MTTF* for Mirror 3 and a 10^5 hour *MTTF* for RAID 5+1. Although other schemes provide much greater MTTDL, Mirror 2 is considerably simpler to implement than Mirror 3 and RAID 5+1, and provides good reliability at relatively low cost.

5. Conclusions

The mechanisms proposed and analyzed in this paper show that high reliability is important for very large-scale storage systems. However, many questions still need to be studied. We know the MTTF of the whole system with several reliability mechanisms, but we do not know about the distribution of failures and their variance over a long period of time. Another interesting issue is the impact of various replica placement policies on the reliability mechanisms. We will compare several placement policies and find the tradeoffs between them. We are also concerned about data consistency problems when we update data blocks during the process of fast mirroring copying. We are considering the use algebraic signatures to check whether the state on disk reflects the current state of the data. This protects against improper RAID 5 reconstruction, where we mix old and new data and parity in the same stripe. Signatures may also be used as a low-cost concurrency mechanism.

We have discussed two major sources of data loss in large-scale storage systems—nonrecoverable read errors and disk failures—and have presented mechanisms for dealing with each. By using signatures on individual disk blocks and redundancy across multiple storage devices, we can reduce the risk of data loss. Techniques such as Fast Mirror Copy further decrease the chance of data loss to the point where simple two-way mirroring in a 2 PB file system can still have a mean time to data loss of 30 years without the use of expensive RAID hardware. If this is not sufficiently long, techniques such as three-way mirroring and RAID 5 with mirroring can virtually guarantee that data will never be lost.

Acknowledgments

Our shepherd Jean-Jacques Bedet provided useful feedback and comments on our first draft. We also thank the members of the Storage Systems Research Center at the University of California, Santa Cruz for their help in preparing this paper.

Qin Xin, Ethan Miller, Darrell Long, and Scott Brandt were supported in part by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714. Thomas Schwarz was supported in part by IBM Research Grant 41102-COEN-RSCH-IG-IG09. Witold Litwin was supported in part by research grants and from the European Commission project ICONS project no. IST-2001-32429 and from Microsoft Research.

References

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

[2] G. A. Alvarez, W. A. Burkhard, and F. Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 62–72, Denver, CO, June 1997. ACM.

[3] M. Castro and B. Liskov. Proactive recovery in a Byzantine-fault-tolerant system. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, 2000.

[4] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2), June 1994.

[5] J. R. Douceur and R. P. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. In *Proceedings of the 20th Symposium on Reliable Distributed Systems (SRDS '01)*, pages 4–13, New Orleans, LA, Oct. 2001. IEEE.

[6] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 115–126. ACM, 2001.

[7] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 92–103, San Jose, CA, Oct. 1998.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture—A Quantitative Approach*. Morgan Kaufmann Publishers, 3rd edition, 2003.

[9] R. J. Honicky and E. L. Miller. A fast algorithm for online placement and reorganization of replicated data. In *Proceedings of the 17th International Parallel & Distributed Processing Symposium (IPDPS 2003)*, Nice, France, Apr. 2003.

[10] 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 the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, MA, Nov. 2000. ACM.

[11] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 2nd edition, 1997.

[12] W. Litwin and T. Schwarz. LH*$_{RS}$: A high-availability scalable distributed data structure using Reed Solomon codes. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 237–248, Dallas, TX, May 2000. ACM.

[13] W. Litwin and T. Schwarz. Algebraic signatures for scalable distributed data structures. Technical Report CERIA Technical Report, Université Paris 9 Dauphine, Sept. 2002.

[14] D. D. E. Long. A technique for managing mirrored disks. In *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pages 272–277, Phoenix, Apr. 2001. IEEE.

[15] J. Menon and R. L. Mattson. Distributed sparing in disk arrays. In *Proceedings of Compcon '92*, pages 410–416, Feb. 1992.

[16] R. Mokadem. Stockage de données en utilisant les signatures dans les sdds. Technical Report CERIA Technical Report, Université Paris 9 Dauphine, Sept. 2002.

[17] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th Conference on Very Large Databases (VLDB)*, pages 162–173, 1990.

[18] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. Technical Report UCB//CSD-02-1175, University of California, Berkeley, Mar. 2002.

[19] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software—Practice and Experience (SPE)*, 27(9):995–1012, Sept. 1997. Correction in James S. Plank and Ying Ding, Technical Report UT-CS-03-504, U Tennessee, 2003.

[20] Y. Saito and C. Karamanolis. Pangaea: A symbiotic wide-area file system. In *Proceedings of the 2002 ACM SIGOPS European Workshop*. ACM, Sept. 2002.

[21] T. Schwarz. Generalized Reed Solomon codes for erasure correction in SDDS. In *Workshop on Distributed Data and Structures (WDAS 2002)*, Paris, France, Mar. 2002.

[22] T. Schwarz, R. W. Bowdidge, and W. A. Burkhard. Low cost comparison of files. In *Proceedings of the 10th International Conference on Distributed Computing Systems (ICDCS '90)*, pages 196–201, 1990.

[23] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems Design and Evaluation*. The Digital Press, 2nd edition, 1992.

[24] B. Tofel. Personal communication, Aug. 2002.

[25] F. Wang, S. A. Brandt, E. L. Miller, and D. D. E. Long. OBFS: A file system for object-based storage devices. Submitted to the 2003 USENIX Technical Conference.

[26] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, Massachusetts, Mar. 2002.

[27] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95)*, pages 96–108, Copper Mountain, CO, 1995. ACM Press.