RobuSTore: A Distributed Storage Architecture with Robust and High Performance

Huaxia Xia*
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093, USA
hxia@ucsd.edu

Andrew A. Chien
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093, USA
achien@ucsd.edu

ABSTRACT

Emerging large-scale scientific applications require to access large data objects in high and robust performance. We propose RobuSTore, a storage architecture that combines erasure codes and speculative access mechanisms for parallel write and read in distributed environments. The mechanisms can effectively aggregate the bandwidth from a large number of distributed disks and statistically tolerate pear-disk performance variation. Our simulation results affirm the high and robust performance of RobuSTore in both write and read operations compared to traditional parallel storage systems. For example, for a 1GB data access using 64 disks, RobuSTore achieves average bandwidth of 186MBps for write and 400MBps for read, nearly 6x and 15x that achieved by a RAID-0 system. The standard deviation of access latency is only 0.5 second, about 9% of the write latency and 20% of the read latency, and a 5-fold improvement from RAID-0. The improvements are achieved at moderate cost: about 40% increase in I/O operations and 2x-3x increase in storage capacity utilization.

1. Introduction

Existing and emerging large-scale scientific applications and data-intensive applications require dramatically higher levels of performance from distributed storage systems. These applications involve accessing to massive data collections with objects as large as 10 gigabytes, and sharing of these data collections for collaboration amongst hundreds of or thousands of widely distributed users.

Distributed storage systems with both high and robust performance are critical to these applications. Throughout, we use the term *robust* to mean low variation in data-access latency. High performance is essential for these applications to access their large data objects. These objects are in the size of gigabytes or even larger, so transfer rates of hundreds of MBps or even multiple GBps are required to achieve interactive, real-time data accesses. Robust performance is important for both user interaction and resource scheduling. Distributed storage systems are essential to provide high performance access for hundreds of or thousands of distributed users concurrently.

One major challenge for distributed storage systems is per-disk performance heterogeneity and variation. First, different sites may have quite different disk types, which may lead to different performance. Furthermore, performance may vary by as much as 100-fold even for the same disk type depending on cache status, disk layout, physical contiguity, and disk head seeking distance. Finally, since distributed storage systems are usually shared by many users, the dynamic competitive workloads lead to dynamic network and disk access behaviors. With such high-degree heterogeneity and variation, simple parallel storage schemes cannot perform well, even with replication.

We propose new storage architecture RobuSTore, which combines erasure coding and speculative accesses together. RobuSTore uses erasure codes to add symmetric redundancy for striping; with such layouts, clients can use speculative parallel access and decoding of the fast-returning blocks to both increase performance, and reduce performance dependence on stragglers (lower variability). As a result, RobuSTore can efficiently aggregate large number of distributed storage devices to deliver robust, high access performance.

Our main contribution is in three folds. First, we propose the RobuSTore idea that combines erasure codes and speculative parallel disk access to improve data access performance and to reduce disk performance variation; Secondly, we analyze the different choices for erasure coding and speculative access, providing a guideline for the RobuSTore implementation; Finally, we model different parallel storage systems, and evaluate them using detailed simulation to prove the significant performance advantages of RobuSTore.

The remainder of the paper is organized as follows. In Section 2, we describe the problem and the assumptions. Section 3 presents the RobuSTore approach and describes the RobuSTore design choices. We evaluate the RobuSTore approach in Section 4. In Section 5 and 6 we present the related work, summarize the paper and discuss the future work.

SC07 November 10-16, 2007, Reno, Nevada, USA

* Huaxia Xia is now affiliated with Google Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA. Email correspondence: huaxia@google.com.

2. Background

The problem we are solving is how to achieve robust and high storage performance in distributed shared systems for large data accesses. High performance means high access bandwidth, or, low access latency for a fix-size data object. Robust performance means low variation of access latency.

Our study is based on the following facts: large number of distributed clients, high variation of per-disk performance, large data objects with rare update operations, abundant storage and network resources, and advanced coding theory.

The first two facts make our problem challenging. To support large number of distributed clients, the storage system needs to be a large distributed one to allow concurrent high-bandwidth accesses. In distributed shared storage systems, per-disk performance may have high variation due to the heterogeneity and shared accesses. This disqualifies the simple parallel local filesystems as the solution. Figure 1 shows an example depicting the limitation of simple parallel filesystems. In the example, two replicas of an eight-block file are striped across four disks and a read operation needs to get at least one copy for each of the eight blocks. However, due to the performance variation, the parallel read has to wait for the blocks from the slowest disk. This leads to both high access latency and high variation of the latency.

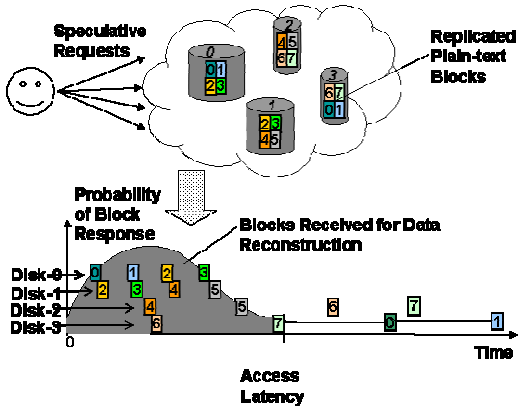

Figure 1. Conventional Parallel Storage. The four disks have different performance, with two replicas of eight blocks.

The other three facts allow us to explore new techniques for the solution. Derived from data-intensive scientific applications, our workloads are dominated by write-once and read-only accesses in the size of gigabytes each. Further, new technologies have been improving network bandwidth, CPU speed, and disk capacity rapidly, so that we may trade these resources for robust and high performance. For example, the low-cost optical transmission and Dense Wavelength Division Multiplexing (DWDM) technique enables individual fibers to carry 100's of 10-Gbps "lambdas", providing wide-area networks with private 10Gbps or even 40-Gbps connections [1, 2]. CPU speed doubles every 18 months and disk capacity doubles every 12 months [3]. Finally, the recent research on erasure codes [4, 5] suggests that LDPC codes can achieve high coding bandwidth and near-optimal coding efficiency.

Erasure codes are a large set of coding algorithms that use a software-based approach to add data redundancy for reliable data transfer [4-8]. In general, an erasure code transforms a message of K symbols into a message with N (N>K) symbols in such a way that the original message can be recovered from a subset of those symbols. In coding theory, N is called *code word length*. The ratio of K/N is called the *rate* of the code, denoted R (0<R<1); the ratio of the redundant data is *degree of data redundancy*, denoted as D = N/K –1 = 1/R –1. Special cases are the *rateless codes*, which can transform K-symbol messages into a practically infinite number of code symbols. Another important attribute is *reception overhead*, which defines the reconstruction efficiency. An erasure code has reception overhead of ε if (1+ε)K encoding symbols are required to reconstruct the original K symbols. Codes with zero reception overhead are called optimal codes; and those with very small reception overhead are called near-optimal codes. A code with lower reception overhead requires a smaller number of symbols for decoding; however, it usually has a higher cost in CPU time, as we will show in this paper.

3. RobuSTore Approach

In this section, we present the RobuSTore idea of combining erasure codes and speculative access and explain why this approach is feasible and why it can improve read and write performance, both robustness and bandwidth.

3.1 RobuSTore Idea

The key idea of RobuSTore is to combine erasure codes and speculative access to aggregate a large number of distributed storage devices. RobuSTore uses erasure codes to add symmetric data redundancy, and stripes the coded data blocks across a large number of distributed disks. With such layouts, clients can speculatively write and read the coded blocks in parallel and complete the access using the fast returned blocks.

Erasure codes provide high flexibility on data access. First, they introduce symmetric data redundancy. An erasure code transforms *K* data blocks into *N* (N>K) coded blocks. The coded blocks contain symmetric data information in such a way that the original data can be recovered from a flexible subset of those blocks. Some near-optimal erasure codes, such as LT codes [4], allow the reconstruction using any (1+ε)K-coded blocks, providing significant higher data read flexibility than plain-text replication. Another important feature of LT codes is rateless encoding. Rateless erasure codes can generate a practically infinite number of coded blocks; statistically, any subset of these blocks of a certain size will provide the same level of data redundancy.

RobuSTore uses speculative access to exploit the erasure coding flexibility. The basic idea of speculative access is to initiate read or write requests for more data blocks than needed from a large number of disks, to wait for the requests to be processed in parallel by the disks, and then to cancel the requests once enough blocks have been confirmed as completed. The speculative access for writes and reads are slightly different. To write a K-block data and use a factor of λ (λ>1) spaces, writing clients would first encode the data into N' blocks where N'>λK, exploiting the rateless feature of the erasure codes. They would then send requests to many disks and transfer coded blocks to them in parallel, then cancel the ongoing writing once N=λK blocks have been confirmed as written success. To read the data,

reading clients would request all the blocks from the disks, then continue receiving them in parallel until enough blocks have been received. Benefiting from the decoding flexibility, clients can then reconstruct the original data using the sets of early-returning blocks.

At a high level, our approach is trading storage space and network bandwidth for low and robust access latency. This is valuable because many large-scale applications have relatively plentiful storage space and network bandwidth, and low and robust access latency is more difficult and more important to achieve.

To give a quantitative sense of how much flexibility erasure codes provide, we theoretically analyzed the number of blocks required for data reconstruction in both erasure-coded schemes and plain-text replicated schemes. We cite the conclusion here and present the full analysis in Appendix. Assume we have a K-block file and use four times its storage space. If using a plain-text replicated scheme, each block has four copies and at least one copy for each block should be retrieved to reconstruct all the data completely. The probability of successful reconstruction with M random replicated blocks is:

$$P(M) = \binom{4K}{M}^{-1} \sum_{i=1}^{N} (-1)^{N-i} \binom{K}{i} \binom{K}{i}$$

In contrast, if we encode the K blocks into 4K blocks using a typical LT code in which the average encoded-node degree is about five, we can reconstruct the original data from M random coded blocks with the following probability:

$$P_c(M) = \sum_{i=1}^{K} (-1)^{K-i} \binom{K}{i} \left(\frac{i}{K}\right)^{5M}$$

In practice, about 3K blocks are needed in a replicated scheme versus about 1.5K blocks in an erasure-coded scheme (see Figure 2).

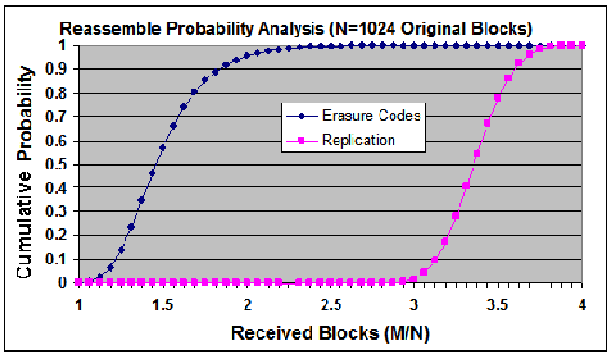

Figure 2. Cumulative Probability of Reassembly of Original Blocks. Assume there are N=1024 original blocks and 4096 coded or replicated blocks.

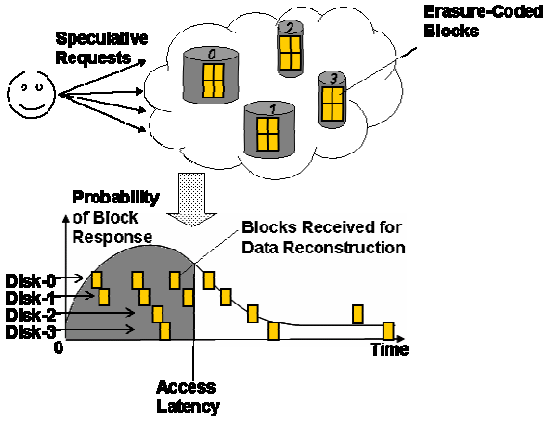

Figure 3. Advantage of Using Erasure Codes and Speculative Access. The four disks have different performance; Assume eight original blocks are encoded into sixteen coded blocks.

By combining erasure codes and speculative access, RobuSTore can tolerate late-arriving blocks and reduce the dependence of a request on any individual disk, and hence achieve robust and high performance. Figure 3 provides an example depicting this advantage in read access. In the example, an eight-block of data is encoded into 16 blocks which are spread across four disks. We assume the data reconstruction needs eight coded blocks, although the number could be slightly larger if we use near-optimal erasure codes. Read clients first send requests to all four disks for all the blocks. The disks then transfer the data blocks back to the clients at different speeds. Once the clients receive eight blocks, they cancel the rest of the accesses, reconstruct the original data, and complete the access with a high-average bandwidth. Furthermore, if any of these first eight blocks are lost or delayed due to any reason, the clients only need to receive one more block and complete the overall access with only slightly longer latency.

3.2 Accesses in RobuSTore

We explain the access procedures of write, read, and update in RobuSTore briefly.

Figure 4 depicts the basic write and read processes. The write operations are in dark circles. In step 1, clients first access the metadata server to open the file, and plan layouts based on disk map information and application QoS requirements. The clients then encode the data to generate redundant coded blocks, and transfer the blocks to the selected servers in parallel to the encoding, shown as step 2 and step 3 in the figure. Once enough data blocks are committed to the servers, the clients cancel the uncompleted write operations, register the data structure with the metadata server, and close the file to complete the write access.

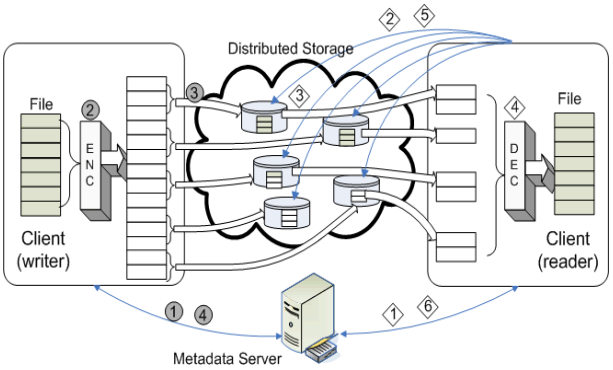

Figure 4. Write and Read Processes in RobuSTore. Dark circles: writes; white diamonds: reads.

The right side of Figure 4 depicts the read process. The numbers with white squares in the figure show the steps in sequence. Similar to the write access process, read accesses start from queries to the metadata server, from which clients obtain storage server information, data structure and location information, and any required locks. To read, the clients request all coded blocks from servers and decode the received blocks in parallel. When enough blocks have been received, the decoding finishes and the original data are reconstructed. At the same time, outstanding requests to the storage servers are cancelled. Finally, the close function notifies the metadata server, releasing read locks and bandwidth reservations on the storage servers.

Update operations are rare in most data-intensive scientific applications and are not our focus in this dissertation, a neat mechanism is still needed to deal with these operations. In RobuSTore, if optimal erasure codes are used, then any minor modification may cause the change of almost all the coded blocks; however, if near-optimal erasure codes are used, the change to one original block only affects a limited number of coded blocks. For example, in the bipartite coding graph of LT codes with 1024 data blocks and 4096 coded blocks, the average degree of data blocks is about 20. In order to change one original block, we need to update at most 20 coded blocks, which is about 0.5% of the total encoded data.

The complete update process is as follows. The clients first get data location information from the metadata server. They can then examine the coding graphs and figure out which coded blocks should be updated. Next, they regenerate those coded blocks, and spread them out to remote disks (not necessary for the disks that store the old coded blocks). Finally, the clients notify the metadata server about the updated blocks and notify the disks to delete the obsolete coded blocks.

3.3 RobuSTore Design Choices

There are many choices for the design, implementation and configuration of a RobuSTore system. Different choices have significant impact on the system performance. We discuss the critical choices for both erasure coding and speculative access.

3.3.1 Choices of Erasure Codes

To deliver high and robust performance on thousands of hard drives, we need erasure codes with low reception overhead, low computation overhead, and long code words. Low reception overhead means high coding efficiency, i.e., only a small number of coded blocks are enough to reconstruct the original data. Low computation overhead allows high bandwidth data encoding and decoding so that we can use an ordinary computer with moderate-speed CPU as a RobuSTore client. A code with a long code word can generate a large number of coded blocks, which brings two benefits to RobuSTore. First, it allows RobuSTore to stripe the encoded data blocks across many disks and to retrieve them from many disks in parallel. Furthermore, long code words allow splitting the original data into more finely grained blocks, which brings more data access flexibility. These three features cannot be optimized at the same time. It is therefore important to choose proper coding algorithms and proper coding parameters.

Optimal erasure codes achieve perfect coding efficiency, but they have a high CPU overhead when using long code words. A rate-R optimal code transforms the original K-block data into N=K/R blocks in such a way that any K-coded blocks suffice to decode the original data. This optimal coding efficiency implies that the information about every original block is mingled into at least N–K+1 coded blocks, since otherwise we can find K coded blocks that are not sufficient to reconstruct the original data. Hence, on encoding, every coded block should be generated by computing at least K(N–K+1)/N original blocks on average, and on decoding, the reconstruction of every original block will need to compute at least K(N–K+1)/N coded blocks on average. Considering N=K/R, the encoding time is at least:

$$N \bullet K(N-K)/N = K(N-K) = \frac{1-R}{R}K^2$$

and the decoding time is at least:

$$K \bullet K(N-K)/N = R \bullet K(N-K) = (1-R)K^2$$

Both encoding time and decoding time is quadratic in K (and thus also quadratic in N). Hence, the encoding and decoding bandwidth is inversely proportional to K. For example, we implement an instance of Reed-Solomon codes and test its performance of encoding and decoding 16 MB data, which is shown in Table 1.

Table 1. Coding Bandwidth of Reed-Solomon Codes. Tested on 2.4GHz Intel Xeon.

K(# original blocks)	N(# coded blocks)	Enc Bandwidth (MBps)	Dec Bandwidth (MBps)
4	8	112.2	99.5
8	16	53.3	60.8
16	32	26.8	31.3
32	64	13.7	15.9

Near-optimal erasure codes make a good trade-off between reception overhead and computation overhead. They require only a few more than optimal coded blocks for reconstruction, but can usually support long code words with low CPU overhead. For example, LT codes [4] use sparse bipartite coding graphs in which each coded block is the parity of a few data blocks. Assuming the average degree of coded block is d_e, we

need d_e–1 XOR operations to recover each data block. If there are K data blocks, a successful decoding requires that each data block is covered by at least one of the received coded blocks. Since each coded block can only cover about d_e random data blocks, the minimum number of coded blocks required for reconstruction is:

$(1+K/(K-1)+K/(K-2)+\ldots+K) / d_e = K\ln K / d_e$

To achieve good reception overhead, d_e should be close to $\ln K$. In another word, the decoding bandwidth of LT is approximately inverse proportional to $\ln K$.

Among different near-optimal erasure codes, LT codes are most suitable for our system for a number of reasons. First, they are rateless, which allows redundancy to be decoupled from other system-design issues, such as the number of storage servers used, and also allows adaptive writing. Second, LT codes use only one level of bipartite structure and block-XOR operations, so that they can be implemented with high coding throughputs. Third, their structure allows the coding process to be overlapped with data I/O, effectively eliminating the critical path time of coding.

Considering these factors, we chose LT codes with K=128~1024 and N=512~4096 in RobuSTore. In our previous work [9], we optimized the LT design and tuned the implementation. The decoding bandwidth of our LT implementation is about 400MBps with less than 0.5 reception overhead on one AMD Opteron processor.

3.3.2 Choices of Speculative Access

During the speculative access, RobuSTore trades the abundant disk and network resources for better performance. However, it is important not to use the resources abusively. We discuss the choice of number of disks and the degree of data redundancy.

The number of disks should be decided by total access performance requirement and estimated per-disk bandwidth. When we stripe the data to multiple disks, we can write and read the disks in parallel to aggregate the performance of multiple disks. Therefore, the number of disks should be no less than the expected total access bandwidth divided by the average disk bandwidth. For example, if the average remote disk bandwidth is 20 MBps, we need to access about 64 disks to saturate a client network with 10 Gbps (1.2 GBps).

The choice of data redundancy is a tradeoff between write and read performance. Data redundancy affects both the writing performance and the reading flexibility. First, higher data redundancy means to generate and write more coded blocks into the storage system, leading to lower writing performance. On the other hand, more coded blocks provide higher flexibility in choosing which blocks to read, which allows RobuSTore to better adapt to disk performance variation and achieve higher read performance.

A good choice of data redundancy should be barely enough to allow each disk to have enough blocks to send during a read access. For example, as depicted in Figure 5(a), when we write data into many disks, a different number of blocks are written to different disks due to disk performance heterogeneity. Assume there are H disks (here H=8), and disk *i* has bandwidth B_{iw}. The average writing bandwidth, then, is:

$$\overline{B_w} = \frac{\sum_{i=1}^{H} B_{iw}}{H}.$$

If the data redundancy is D, the total number of coded blocks is N=(D+1)K. The amount of data written to disk *i* is then:

$$F_{iw} = B_{iw} \cdot T = B_{iw} \cdot \frac{N}{H \cdot \overline{B_w}} = \frac{N \cdot B_{iw}}{H \cdot \overline{B_w}}.$$

When clients speculatively read the data blocks, the dynamic disk performance might be quite different from what it had been when writing the data, as depicted in Figure 5(b). Assume disk *i* has bandwidth B_{ir}, then the average reading bandwidth is:

$$\overline{B_r} = \frac{\sum_{i=1}^{H} B_{ir}}{H}.$$

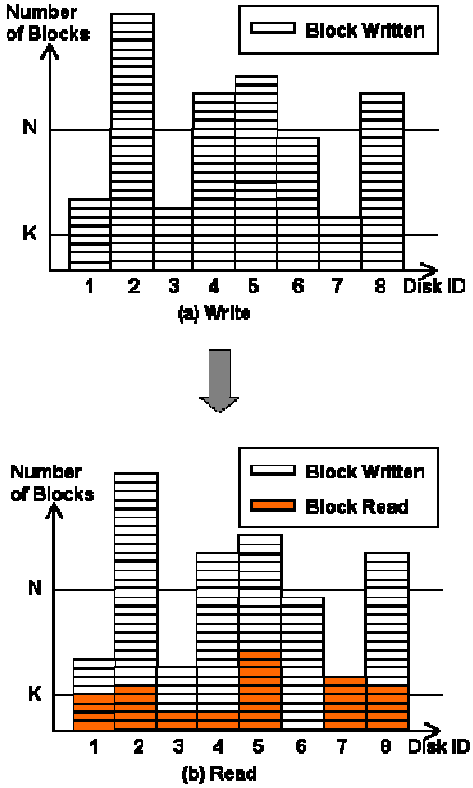

Figure 5. Tolerate Dynamic Disk Performances.

Non-optimal erasure codes have positive reception overhead ε, meaning that they require (1+ε)K coded blocks to construct the K data blocks. If there are enough blocks on every disk to read, the amount of data read from disk *i* are:

$$F_{ir} = B_{ir} \cdot T = B_{ir} \cdot \frac{K(1+\varepsilon)}{H \cdot \overline{B_r}} = \frac{K(1+\varepsilon) \cdot B_{ir}}{H \cdot \overline{B_r}}$$

To guarantee every disk has enough blocks, the following should be satisfied:

$$F_{ir} \le F_{iw}, \forall i \in [1, H]$$
$$\Leftrightarrow \frac{K(1+\varepsilon)\cdot B_{ir}}{H \cdot \overline{B_r}} \le \frac{N \cdot B_{iw}}{H \cdot \overline{B_w}}, \forall i \in [1, H]$$
$$\Leftrightarrow D = \frac{N}{K} - 1 \ge (1+\varepsilon)\cdot \frac{\overline{B_w} \cdot B_{ir}}{\overline{B_r} \cdot B_{iw}} - 1, \forall i \in [1, H]$$

When the number of disks is large, statistical theory tells that $\overline{B_r} \cong \overline{B_w}$. Therefore, the required data redundancy is:

$$D = (1+\varepsilon)\cdot \max_{1 \le i \le H} \frac{B_{ir}}{B_{iw}} - 1 \cdot$$

B_{ir}/B_{iw} is the performance variation of each individual disk.

The performance variation of hard drives may be up to factors in the tens of or even hundreds with the existence of different access contentions. This is a strict boundary of D; however, a very large D is not required in practice. First, it is rare for the disks to be that heavily loaded since we always select the most lightly loaded disk upon which to write new data, and can usually achieve reasonably high B_{iw}. Second, if only a few disks have insufficient number of blocks to read, they will not have a significant impact on overall performance.

In a specific case, if we write same amount of data to each disk, i.e., $F_w = N/H$, then we can get the following using a similar analysis process:

$$D = (1+\varepsilon)\cdot \max_{1 \le i \le H}(B_{ir} / \overline{B_{ir}}) - 1 \cdot$$

In the next section, out experiments show that data redundancy of two to three is enough to provide the best performance.

4. Performance Evaluation

In this chapter, we study the advantages of RobuSTore over traditional parallel storage schemes. We evaluate these storage systems using detailed software simulation, and simulate the systems across a wide range of configurations, including different numbers of storage devices, network properties and degrees of data redundancy.

4.1 Experimental Design

In this section, we describe the storage schemes for comparison, simulation design, metrics, workloads, and experiment configurations in detail.

4.1.1 Storage Schemes for Comparison

We evaluate the RobuSTore scheme by comparing it against conventional parallel storage schemes. The conventional schemes are RAID-0, RRAID-S, and RRAID-A, which are different from RobuSTore in terms of the data layout mechanism or access mechanism.

Data Layout and Redundancy: Possible data layout mechanisms are depicted in Figure 6: (1) split the data into blocks, and distribute them to many disks; (2) split the data and distribute the blocks with replication; (3) split the data, encode the blocks, and distribute these redundant coded blocks to many disks.

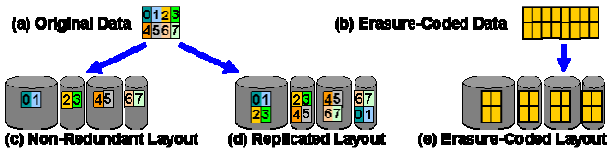

Figure 6. Data Layouts. 8 data blocks; 1x data redundancy in replicated and coded layouts.

Access Strategies: Possible data access strategies are shown in Figure 7: (a) speculative access, i.e., request redundant blocks at once in the beginning of the access and cancel the requests once enough blocks have been received; (b) adaptive access, in which the client dynamically requests the unreceived bytes.

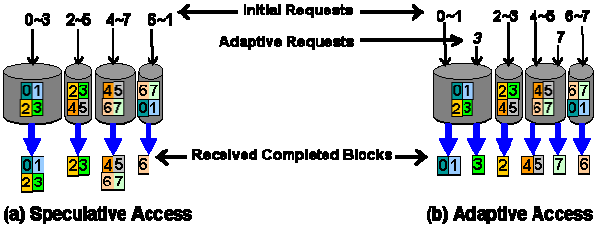

Figure 7. Access Mechanisms. Disk performance is varied.

We evaluate the following four combined schemes: (1) RAID-0: No data redundancy + speculative access; (2) RRAID-S: Replication + speculative access; (3) RRAID-A: Replication + adaptive access; and (4) RobuSTore: Erasure coding + speculative access.

4.1.2 Simulator Design and Configuration

We simulate an environment with one client and 128 storage servers connected by wide-area networks.

Each storage server is simulated using one DiskSim [10] process. DiskSim processes are configured to have different disk-level block layouts and background workloads such that individual disk performance varies from 0.52MBps to 53MBps. This represents the performance variability in shared distributed storage environments with many sources of variability, as discussed in Section 1.

The virtual client models all other overheads for metadata access, server connection, network latency, and block decoding. The metadata access and server connection are assumed to take constant time; network latency to each server is configured constant from 1ms to 100ms. For block decoding, since it can be pipelined with data receiving, extra latency is only incurred for decoding the last block; we model it as a constant 5 ms overhead. We assume sufficient network bandwidth and CPU power.

In the experiments, we study the four storage schemes (RAID-0, RRAID-S, RRAID-A, and RobuSTore) along five system parameters: number of disks, data size, block size, network latency, and degree of redundancy. In each experiment, we vary only one parameter, and compare to a fixed baseline. The baseline is a typical SAN configuration: to access 1GB data from 64 disks, 1ms network round-trip time (RTT), 1MB block size and 4x data redundancy, except for RAID-0 which always has 1x data redundancy.

4.1.3 *Workloads and Metrics*

Since our focus is on supporting the needs of applications with large workloads [11-13], we use synthetic workloads with sequences of large-size accesses. In these applications, each data object is from 100s of MB to 10s of GB, and with the potential to increase to 100s of GB or larger in the future. We study access performance for single 128 MB, 256 MB, 512 MB, and 1 GB accesses. Data objects larger than 1 GB are presumed to be accessed by multiple 1 GB accesses. There are both read sequences and write sequences accesses and sequences with mixed read and write operations.

Moreover, to simulate disks shared by multiple applications, we generate competitive background workloads for each disk. The background workload is a sequence of random accesses arriving in a certain interval. By varying the interval of the background workload, we can simulate different degrees of disk sharing.

In our experiments, we measure RobuSTore and other conventional storage systems in three metrics.

Variation of Access Latency: A critical RobuSTore goal is robust performance, i.e., minimum performance variation. We formalize this for access latency by computing the standard deviation over a set of one hundred accesses. Naturally, smaller standard deviations correspond to higher degrees of robustness.

Access Bandwidth: While robust performance is the major goal of RobuSTore, we must also maintain high access bandwidth for the requirement of accessing large datasets. The delivered bandwidth for a single read or write is the original data size divided by the access latency, including connection, disk, data transfer, and coding time. We interpret access bandwidth to be a measure of delivered performance corresponding to our goal of "high performance".

I/O Overhead: The benefits of aggressive access to redundant copies can yield performance benefits, but it also increases network and disk I/O costs. We measure this increased I/O cost using the ratio between the additional bytes sent over networks and the original data size:

$$\text{I/O Overhead} = \frac{\text{Bytes sent over networks - Original data size}}{\text{Original data size}}.$$

Note that the bytes sent over networks may be more than the bytes read from disks if some bytes are read from the filesystem cache.

We measure both read performance and write performance in these three metrics.

4.2 Experiment Results

We simulate the four storage schemes over five configuration dimensions. Due to space limitation, we only show the comparison in various degrees of data redundancy. We vary data redundancy from 0 to 900% (10 times the storage spaces used) to simulate its performance impact. Because the RAID-0 scheme always has zero redundancy, there is no curve for RAID-0 in the following graphs; its performance is represented by the zero-redundancy point in RRAID-S or RRAID-A.

First, we show the results of write accesses in Figure 8. In the schemes of RAID-0, RRAID-S, and RRAID-A, a write operation uniformly writes the same number of blocks to each disk, the write bandwidth is very low because it is limited by the slowest disk. RobuSTore achieves much higher bandwidth since its speculative writing can efficiently utilize the capability of all the disks. As shown in Figure 8(a), when the data redundancy is 300%, the write bandwidth in RobuSTore is about 186 MBps, while RRAID-S and RRAID-A only deliver bandwidth of 7.5 MBps. It is 30 MBps for RAID-0 (with zero redundancy).

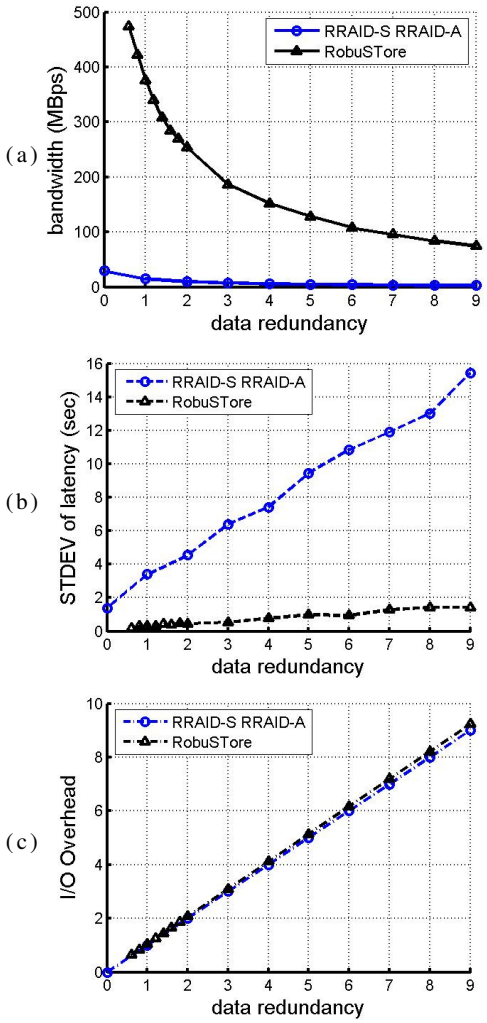

Figure 8. Write Performance vs. Data Redundancy. RAID-0 is at the zero-redundancy point.

The standard deviation of write latency is more than 10 times better in RobuSTore than in RRAID-S and RRAID-A, as shown in Figure 8(b). For example, when the data redundancy is 300%, the standard deviation is 0.5 seconds for RobuSTore and 6.4 seconds for RRAID-S and RRAID-A.

The I/O overhead in write operations is proportional to data redundancy because a write operation needs to write every byte of the redundant data. RobuSTore may incur slightly more overhead due to the usage of a speculative writing mechanism, as shown in Figure 8(c).

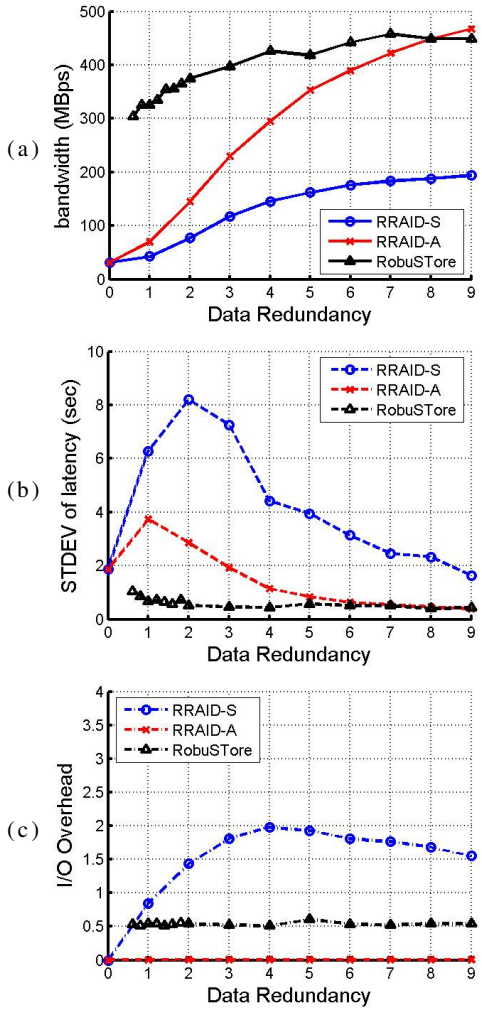

Figure 9. Read Performance vs. Data Redundancy, with Unbalanced Data Striping in RobuSTore.

Next, we study the follow-up read accesses. In RobuSTore, speculative writing may cause unbalanced data striping across multiple disks. RAID-0, RRAID-S, and RRAID-A have no such issue since they always use balanced data striping.

The simulated results are depicted in Figure 9. Figure 9(a) shows that higher data redundancy leads to higher read bandwidths in all the storage schemes. RobuSTore delivers must higher read bandwidth than RRAID-S and RRAID-A, except that RRAID-A with very high redundancy can also deliver similar performance. At 3x redundancy, RobuSTore delivers about 400MBps bandwidth; while RRAID-S and RRAID-A only achieve 117MBps and 228MBps respectively, and RAID-0 has only 31 MBps.

Figure 8(b) shows that RobuSTore achieves the lowest standard deviation of latency. In RRAID-S and RRAID-A, the variation comes from disk speed, intra-disk block ordering (in RRAID-S), and inter-disk block mapping. When they use higher data redundancy, their robustness will potentially suffer less from disk speed variation and inter-disk block mapping, while suffering more from intra-disk block ordering. RAID-0 only suffers variation from the slowest disk. Due to the combination of these factors, RRAID-S and RRAID-A with small redundancy have worse robustness than RAID-0, and gradually get better as redundancy increases. In RobuSTore, as long as the fast disks have enough data blocks, they can hide the slow disks effectively. It needs only 1x~2x data redundancy to obtain most of this robustness benefit. When data redundancy is more than 2x, the standard deviation of latency is only about 0.5 seconds, or 25% of the average access latency.

The I/O overhead results are shown in Figure 8(c). RAID-0 has no speculative access, so it incurs no additional costs, and has zero I/O overhead. RRAID-A costs just a little bit more than zero overhead, as it only generates additional accesses when they are clearly needed. When data redundancy is increased, both RRAID-S and RobuSTore increase the requested data size in proportion. For RobuSTore, the access is completed as long as a certain number of coded blocks are received, so the final I/O overhead is mainly decided by the reception overhead of LT Codes. However, in RRAID-S, high data redundancy lets the client receive more duplicated data blocks, leading to high I/O overhead up to 200%.

To make the comparison fairer, we also study the RobuSTore read access with balanced data striping. In this case, RobuSTore uses exact same amount of per-disk spaces with RRAID-S and RRAID-A.

The results are depicted in Figure 10. For RobuSTore, the bandwidth increases rapidly and approaches the best performance when the redundancy is higher than 200%. RRAID-S and RRAID-A benefit less from high redundancy because their structured data replication cannot adapt to read more blocks from the faster disks as flexibly as in RobuSTore. The variation of RobuSTore access latency, as shown in Figure 10(b), is the least among the four storage schemes. Figure 10(c) shows that RobuSTore has about 50% I/O overhead due to the requirement of extra blocks for decoding, similar to the results in Figure 9.

4.3 Summary of the Evaluation

The simulation results show that RobuSTore provides best performance in terms of access bandwidth and robustness, and it only incurs moderate overheads. For example, to write 1GB data with 3x redundancy on 64 disks, RobuSTore achieves 186MBps, which is six times of RAID-0 (with zero-redundancy) and 25 times of RRAID-S/RRAID-A; the standard deviation of latency is only 1/13 of RRAID-S/RRAID-A. To read 1GB data from 64 disks, RobuSTore achieved an average bandwidth of over 400 MBps, nearly 15x that achieved by a baseline RAID-0 scheme. At the same time, RobuSTore achieves standard deviation of access latency of only 0.5 seconds, less than 25% of the total access latency. The RobuSTore IO overhead is about 50% and storage space overhead is 2x–3x.

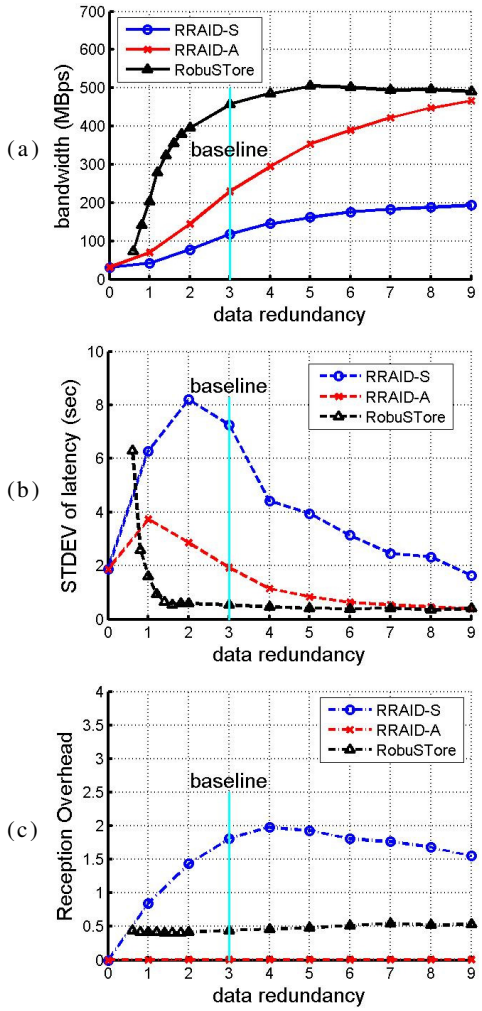

Figure 10. Read Performance vs. Data Redundancy, with Balanced Data Striping.

5. Related Work

There has been a wealth of related work on distributed storage and performance aggregation of multiple disks. RAID [14] and parallel file systems ([15-18], etc) aggregate multiple disks, addressing the performance and capacity limitation of single disks or servers. They assume uniform arrays of storage devices in a SAN or LAN environment, without consideration on dynamic performance variability in distributed environments. Some peer-to-peer file sharing systems ([19-21], etc) improve access performance by speculatively fetching from massively replicated data copies. However, the massive replication is expensive in terms of storage overhead and access scheduling. Further, these systems focus on the shared internet where per-node network bandwidth is as low as 1-10 megabits/s.

Numerous storage systems [22-26] exploited erasure codes in their design. However, most of them are focused on data reliability and availability instead of robustness or bandwidth.

A few recent distributed storage systems focused on performance aggregation of heterogeneous disks. Collins and Plank [27] studied the usage of Reed-Solomon Codes and LDPC Codes to improve the bandwidth of wide-area storage systems. However, they assume slow shared networks, bandwidths < 10MByte/s, and small number of blocks (N≤100), and they concludes that Reed-Solomon Codes perform better than or equal to LDPC codes. In contrast, we focus on performance robustness as well as bandwidth; we design RobuSTore for high bandwidth wide-area networks (>10Gbps), and explore a much wider array of design choices in data coding parameters, redundancy, layout and access.

Lumb et al. proposed D-SPTF protocol [28], which can dynamically select a replica server to serve each read request. Our analysis and evaluation in this paper shows that replication-based scheme is less effective in adapting to performance variation than erasure-code-based scheme.

Wu et al proposed adaptive resource selection (ARS) heuristic algorithm for load balancing the servers [29]. By using replicated or erasure-coded data blocks, their design can flexibly choose servers to read and write, which most closely resembles our work in spirit. Our paper explores the problem in a number of different aspects: we study the advantage of erasure coding over replication; we analyze the choice of erasure codes and the proper data redundancy; we assume each server can store multiple data blocks instead of single block, which provides finer-grained load balancing; and our speculative access scheduling is slightly more expensive but more flexible than ARS scheduling.

6. Conclusions

We propose the distributed storage architecture RobuSTore for high and robust storage performance in distributed environments. Achieving high and robust performance in distributed storage systems is an important open research challenge. Traditional network filesystems or local parallel filesystems cannot satisfy these requirements. The performance variation of the individual disks is the major obstacle facing current systems. We propose the RobuSTore idea to address this issue. RobuSTore combines erasure coding and speculative access mechanisms for high and robust storage performance. The erasure coding mechanism encodes the original data into fragment blocks with symmetric redundancy, allowing flexible data striping during write accesses and flexible data reconstruction during read accesses. The speculative access mechanism fully utilizes the available disk bandwidths to read/write redundant fragment blocks from/to heterogeneous distributed disks. We then discussed the critical design choices for erasure coding and speculative access, which gave guidelines for the RobuSTore implementation.

We compare the performance of RobuSTore with three traditional parallel storage schemes and see superior performance from RobuSTore in both write and read accesses. For example, for a 1GB data access using 64 disks with random in-disk data layout, RobuSTore achieves average bandwidth of 186MBps for write and 400MBps for read, nearly 6x and 15x that achieved by a RAID-0 system. The standard deviation of access latency is only 0.5 second, about 9% of the write latency and 20% of the read latency, and a 5-fold improvement from RAID-0. The improvements are achieved at moderate cost: about 40% increase in I/O operations and 2x-3x increase in storage capacity utilization.

We would remind the audience about the limitations of RobuSTore again. RobuSTore is not a general storage system; instead, it is for accessing large data objects on which update operations are rare. On homogeneous storage clusters with no shared access, RobuSTore is not better than traditional parallel file system due to the reception overhead of erasure codes, which will reduce the bandwidth for up to 30%.

While we believe that we have made significant contributions, more advances can be made to improve the RobuSTore design and performance. First, we need erasure codes algorithms that can deliver higher coding bandwidth to match the network bandwidth increase. We achieve around 600 MBps decoding bandwidth with 50% reception overhead using LT Codes on 2.8 GHz AMD Opteron Processor. This is about 7 Gbps network utilization. Higher coding performance may be achieved by more efficient erasure codes, parallel coding algorithms, or dedicated coding hardware. Admission control is another topic that can complete the RobuSTore design. It is important for QoS guarantee and efficient resource sharing.

7. Acknowledgements

Supported in part by the National Science Foundation under awards NSF EIA-99-75020 Grads and NSF Cooperative Agreement ANI-0225642 (OptIPuter), NSF CCR-0331645 (VGrADS), NSF NGS-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from Hewlett-Packard, BigBangwidth, Microsoft, and Intel is also gratefully acknowledged.

8. References

1. Smarr, L.L., et al., *The OptIPuter.* Communications of the ACM, 2003. **46**(11): p. 58-67.
2. *iGrid2005*. San Diego, CA, Sep 26-30, 2005; http://www.igrid2005.org.
3. Grochowski, E. and R.D. Halem, *Technological impact of magnetic hard disk drives on storage systems.* IBM Systems Journal, 2003. **42**(2): p. 338-346.
4. Luby, M. *LT Codes*. in *Proc. IEEE Symp. On Foundations of Computer Science 2002*. 2002.
5. Shokrollahi, A., *Raptor codes*. 2003, Digital Fountain and EPFL.
6. Plank, J.S., *A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems.* Software -- Practice & Experience, 1997. **27**(9): p. 995-1012.
7. Reed, I. and G. Solomon, *Polynomial codes over certain finite fields.* Journal of the Society for Industrial and Applied Mathematics, 1960. **8**(2): p. 300--304.
8. Gallager, R.G., *Low Density Parity-Check Codes*. 1963, Cambridge, MA: MIT Press.
9. Uyeda, F., H. Xia, and A. Chien, *Evaluation of a High Performance Erasure Code Implementation*. 2004, UCSD.
10. Bucy, J.S. and G.R. Ganger, *The DiskSim Simulation Environment Version 3.0 Reference Manual*. 2003, Carnegie Mellon University.
11. *Biomedical Informatics Research Network (BIRN)*. http://www.nbirn.net.
12. *GriPhyN: Grid Physics Network*. http://www.griphyn.org.
13. *The EarthScope Project*. http://www.earthscope.org.
14. Patterson, D.A., G.A. Gibson, and R.H. Katz. *A Case for Redundant Arrays of Inexpensive Disks (RAID)*. in *International Conference on Management of Data (SIGMOD)*. 1988.
15. Carns, P.H., et al. *PVFS: A Parallel File System For Linux Clusters*. in *4th Annual Linux Showcase and Conference*. 2000. Atlanta, GA.
16. Schmuck, F. and R. Haskin. *GPFS: A Shared-Disk File System for Large Computing Clusters*. in *USENIX Conference on File and Storage Technologies (FAST)*. 2002. Monterey, CA.
17. System, C.F., *Lustre: A Scalable, High-Performance File System*. 2002, Lustre File System v1.0 Architecture White Paper from clusterfs.org.
18. Nagle, D., D. Serenyi, and A. Matthews. *The Panasas ActiveScale Storage Cluster –Delivering Scalable High Bandwidth Storage*. in *ACM/IEEE Conference on Supercomputing, 2004 (SC'04)*. 2004. Pittsburgh, PA.
19. *Kazaa*. http://www.kazaa.com.
20. *BitTorrent*. http://www.bittorrent.com.
21. Corbett, P.F. and D.G. Feitelson, *The Vesta parallel file system.* ACM Transactions on Computer Systems, 1996. **14**: p. 225--264.
22. Abd-El-Malek, M., et al. *Fault-Scalable Byzantine Fault-Tolerant Services*. in *Symposium on Operating Systems Principles*. 2005. Brighton, UK.
23. Kubiatowicz, J., D. Bindel, and e. al. *OceanStore: An Architecture for Global-Scale Persistent Storage*. in *the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2000. Cambridge, MA.
24. Bhagwan, R., et al. *Total recall: System support for automated availability management*. in *the First ACM/Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 2004. San Francisco, CA.
25. Weatherspoon, H. and J.D. Kubiatowicz. *Erasure Coding vs. Replication: A Quantitative Comparison*. in *the First International Workshop on Peer-to-Peer Systems (IPTPS)*. 2002. Cambridge, MA.
26. Aguilera, M.K., R. Janakiraman, and L. Xu. *Using Erasure Codes Efficiently for Storage in a Distributed System*. in *DSN 2005: The International Conference on Dependable Systems and Networks*. 2005. Yokohama, Japan.
27. Collins, R.L. and J.S. Plank. *Assessing the performance of Erasure Codes in the Wide Area*. in *DSN-2005: The International Conference on Dependable Systems and Networks*. 2005. Yokohama, Japan.

28. Lumb, C.R., R. Golding, and G.R. Ganger, *D-SPTF: decentralized request distribution in brick-based storage systems*, in *11th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2004: Boston, MA.
29. Wu, C. and R. Burns, *Improving I/O Performance of Clustered Storage Systems by Adaptive Request Distribution*, in *The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*. 2006: Paris.

Appendix

To better understand the advantage of erasure-coded schemes over replicated schemes, we analyze the number of blocks required for data reconstruction in each scheme. General problem description: Assume we have N original blocks, and we transfer them into 4N output blocks using either replication or erasure coding. Now randomly permute the 4N blocks. What is the probability that we can reassemble the original blocks using the first M output blocks?

A1. Plain-text Replication

The problem is equivalent to the following:

Given: 4N balls with N different colors (four balls per color); randomly pick M balls from them

Want: probability of at least one ball per color.

Assume the number of M-ball sets to have at least one ball per color is $F_M(N)$. Then we have:

$F_M(N)$ = (All sets) – (sets with less than N colors)

$$= \binom{4N}{M} - \sum_{i=1}^{N-1} \binom{N}{i} F_M(i), \quad (\text{Let } \binom{a}{b} = 0 \text{ if } a<b)$$

We will prove the following using induction:

$$\text{(A.1)} \quad F_M(N) = \sum_{i=1}^{N} (-1)^{N-i} \binom{N}{i} \binom{4i}{M}$$

First, since there are only 4 balls per color, we have

$$F_M(N) = 0, \text{ if } N<M/4,$$

which satisfies (A.1).

Now we assume (A.1) is satisfied for any number less than N, then:

$$\begin{aligned}
F_M(N) &= \binom{4N}{M} - \sum_{i=1}^{N-1} \binom{N}{i} F_M(i) \\
&= \binom{4N}{M} - \sum_{i=1}^{N-1} \binom{N}{i} \sum_{j=1}^{i} (-1)^{i-j} \binom{i}{j} \binom{4j}{M} \\
&= \binom{4N}{M} - \sum_{i=1}^{N-1} \sum_{j=1}^{i} (-1)^{i-j} \binom{N}{i} \binom{i}{j} \binom{4j}{M} \\
&= \binom{4N}{M} - \sum_{j=1}^{N-1} \sum_{i=j}^{N-1} (-1)^{i-j} \frac{N!}{i!(N-i)!} \cdot \frac{i!}{j!(i-j)!} \binom{4j}{M} \\
&= \binom{4N}{M} - \sum_{j=1}^{N-1} \frac{N!}{j!(N-j)!} \sum_{i=j}^{N-1} (-1)^{i-j} \frac{(N-j)!}{(N-i)!(i-j)!} \binom{4j}{M} \\
&= \binom{4N}{M} - \sum_{j=1}^{N-1} \frac{N!}{j!(N-j)!} \sum_{k=0}^{N-j-1} (-1)^{k} \frac{(N-j)!}{(N-j-k)!k!} \binom{4j}{M} \quad \text{(let k=i-j)} \\
&= \binom{4N}{M} - \sum_{j=1}^{N-1} \frac{N!}{j!(N-j)!} (-(-1)^{N-j}) \binom{4j}{M} \\
&= \sum_{j=1}^{N} (-1)^{N-j} \binom{N}{j} \binom{4j}{M}
\end{aligned}$$

So (A.1) also fits for N.

Therefore, the probability of picking M balls to include at least one ball per color is:

$$P(M) = \frac{F_M(N)}{\binom{4N}{M}} = \sum_{i=1}^{N} (-1)^{N-i} \frac{\binom{N}{i}\binom{4i}{M}}{\binom{4N}{M}}$$

A2. Erasure-Coded Case

With parameter of C=1.1 and δ=0.5, the average output-node degree in the LT coding graph is about 5. To simplify the analysis, we assume that all output nodes have degree 5 and their neighbors are independently randomly selected from the N original blocks. The number of blocks to reconstruct the original N blocks is about the number of blocks whose neighbors include all the N blocks. So the probability that M coded blocks are sufficient is the probability that 5M neighbors can cover all the N original blocks. Using similar induction as in above section, we can prove that:

$$P_c(M) = \sum_{i=1}^{N} (-1)^{N-i} \binom{N}{i} \left(\frac{i}{N}\right)^{5M}$$