UC San Diego
Technical Reports

Title
RobuSTore: Robust Performance for Distributed Storage Systems

Permalink
https://escholarship.org/uc/item/04j1k2xf

Authors
Xiz, Huaxia
Chien, Andrew A

Publication Date
2006-03-13

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

RobuSTore: Robust Performance for Distributed Storage Systems

Huaxia Xia and Andrew A. Chien
Computer Science and Center for Networked Systems
University of California, San Diego

Abstract *Emerging large-scale scientific applications involve massive, distributed, shared data collections (petabytes), and require robust, high performance for read-dominated workloads. Achieving robust performance (low variability) in storage systems is difficult. We propose RobuSTore, a novel storage technique, which combines erasure codes and speculative access to reduce performance variability and increase performance. RobuSTore uses erasure codes to add flexible redundancy then spreads the encoded data across a large number of disks. Speculative access to the redundant data from multiple disks enables application requests to be satisfied with only early-arriving blocks, reducing performance dependence on the behavior of individual disks.*

We present the design and an evaluation of RobuSTore which shows improved robustness, reducing the standard deviation of access latencies by as much as 5-fold vs. traditional RAID. In addition, RobuSTore improves access bandwidth by as much as 15-fold. A typical 1 GB read from 64 disks has average latency of 2 seconds with standard deviation of only 0.5 seconds or 25%. RobuSTore secures these benefits at the cost of a 2-3x storage capacity overhead and ~1.5x network and disk I/O overhead.

1 Introduction

Existing and emerging large-scale scientific applications and data-intensive applications have required dramatically higher-levels of performance from distributed storage systems. These applications are far from isolated anomalies; they are emerging in virtually every area of science, engineering, and commerce, including biology [1], high-energy physics [2], geology [3], and astronomy [4]. For example, one effort in the OptIPuter project [5] is to construct custom indices across 10GB 3-D images in a collection of 1,000,000 images (10Petabytes of data) and to enable interactive, real-time data exploration for collaborative visualization on a large-scale tiled display consisting of 55 panels, 100M pixels, 300Gbps network bandwidth [6].

Typical properties of large-scale scientific applications include: (1) massive data collections with objects as large as 10 GB each and collections larger than tens of petabytes; (2) distributed data sources and owners; (3) large read-dominated workloads (100s of megabytes to 10s of gigabytes per read); (4) thousands of users, and (5) a need for robust, high performance. Throughout, we use the term *robust* to mean low-variability in access latency.

Performance robustness is rarely studied for storage systems; however, it is very important for many large-scale applications. For example, interactive applications require the access latency to be robust; in parallel processes, robust performance can reduce the synchronization overhead; robust performance also makes it easy to predict resource requirement and to schedule resources efficiently. Robust performance is the major goal of RobuSTore. At the same time, we must maintain high access bandwidth and efficient storage space utilization.

Due to the limited single disk bandwidth, efficient aggregation of large number of disks is required to meet the performance requirements. Existing parallel storage systems, like RAID, PVFS, Lustre, etc, provide good performance in homogenous local cluster environment via data striping and parallel accessing. However, the usage of local private storage is not realistic for our scientific applications due to the gigantic datasets and the distributed shared usages. These applications require the aggregation of large number of distributed disks shared by multiple universities/laboratories. Disks in such environments have highly heterogeneous and variable performance, which prevents conventional parallel accesses from achieving high, robust performance since the accesses may have to wait for the slow disks.

In our new storage architecture RobuSTore, we propose the idea of combining erasure coding and speculative accesses together, which enables high, robust performance to distributed storage collections. RobuSTore uses erasure codes to add continuous 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.

Special support from coding algorithm, metadata service, admission controlling, and security schemes are required for the high-performance shared accesses. We describe a system framework which realizes the RobuSTore idea, enabling systematic exploration of the design space.

We then evaluate the RobuSTore idea via detailed simulation, exploring a wide range of possible system parameters. These studies show when and by how much the RobuSTore idea reduces performance variability, increases absolute performance, and incurs overheads on

storage space, disk accesses, and network bandwidth. Our simulation results prove the promise of the RobuSTore idea.

RobuSTore is motivated by rapid increasing of network bandwidth, disk capacity, and CPU speed, but only slow increase on disk bandwidth, which makes it valuable to trade those resources for better storage performance. For example, the development of 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 (even 40Gbps) connections [7]. Numerous research infrastructures with private, high-speed "lambdas" have been deployed (OptIPuter [5], National LambdaRail [8], Netherlight [9], and the Global Lambda Interchange Facility (GLIF) [10]). The doubling period for CPU speed is 18 months according to Moore's Law, and 12 months for disk capacity [21].

RobuSTore is part of the OptIPuter Project [1] exploring configurable optical networks with 100's Gbps to support large-scale scientific applications. Network issues, such as lambda configuration, protocols, etc. are addressed by other parts of the project DVC [22], LambdaRouter [23], CEP [24], and GTP [25].

The remainder of the paper is organized as follows. In Section 2, we describe the problem and present the RobuSTore approach. We describe the RobuSTore design in Section 3, and our simulation-based evaluation in Section 4. Section 5 surveys related work and Section 6 summarizes results, describing directions for future work.

2 Robust, High Performance Distributed Storage

2.1 The Problem

Numerous emerging, large-scale scientific applications involve massive distributed data collections, and demand distributed storage systems that deliver robust, high performance for their read-dominated workloads. The distributed nature of the scientific communities using these applications implies that their needs will be met by distributed collections of disk arrays. Supporting many users, at these high performance levels requires robust and efficient aggregation of disk performance.

For the distributed storage in such systems, disk performance varies greatly due to the federated, evolving nature of such infrastructures. Sources of performance variation or heterogeneity include: heterogeneous disk types (deployed over time at each site), irregular on-disk data layout (due to unique disk history and local/distributed use interaction), and differences in disk load (varied competitive access). As disk technology has improved, read bandwidth has increased, so heterogeneous disk collections often include individual disks with 20-fold read bandwidth variation [21]. In modern storage devices, disk performance is sensitive to disk layout [26, 27], as physical contiguity, seek distance, and rotational latency can vary read bandwidth by 100-fold. Finally, sharing causes access streams to be interleaved which can incur additional seeks which can also cause as much as 100-fold variance in performance.

Traditional parallel filesystems use data striping and replication to aggregate disk performance. However, they only achieve good performance with nearly homogeneous collections of disks and often only when layout is tightly controlled [18]. Further, best performance is often only achieved for exclusive uses. Because parallel filesystems deliver performance with careful layout scheduling and control, a key limitation is that they do not perform well in the face of uncontrolled performance variation; even if replication is used. For example, if access to a single data block or disk is slow, an entire large request can be delayed. Even if data are replicated, late arriving disk responses can still delay completion of the larger request (see Figure 1).

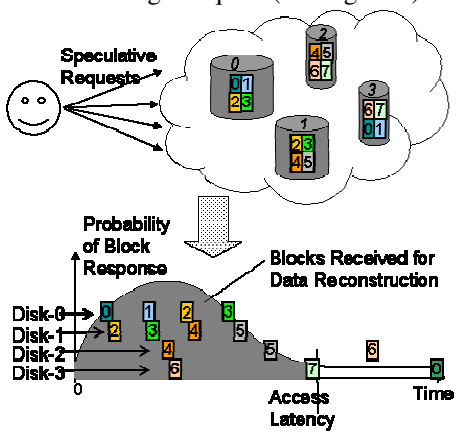

Figure 1.Aggregating Disks with Data Striping: 8 blocks, 2 replicas. Disk performance is varied.

To meet the needs of emerging large-scale scientific applications, we need a new set of technologies for aggregating the performance of disk collections. These technologies must

- tolerate high variation in the performance of individual disks,
- achieve robust access latency (low variance) for large requests, and
- deliver high performance.

In subsequent sections, we describe the RobuSTore approach which meets these goals.

2.2 RobuSTore: Erasure codes and Speculative access

The key idea in RobuSTore is to add redundancy to stored data and then exploit it to aggregate statistically the disk access bandwidths, producing an earlier and lower-variance in access latency for large requests. Erasure

codes allow the systematic introduction of redundancy, so many combinations of returning blocks can be used to assemble the desired data. This redundancy reduces the dependence of the large request on the performance of any individual disk request. Further, we employ speculative access, requesting a redundant set of blocks, increasing the chances that a set of blocks which makes decoding possible (and thereby completion of the large request) will be received quickly. The decoding freedom provided by erasure codes is depicted in Figure 2. RobuSTore only requires the first set of returned blocks to decode the data, and need not wait for later blocks.

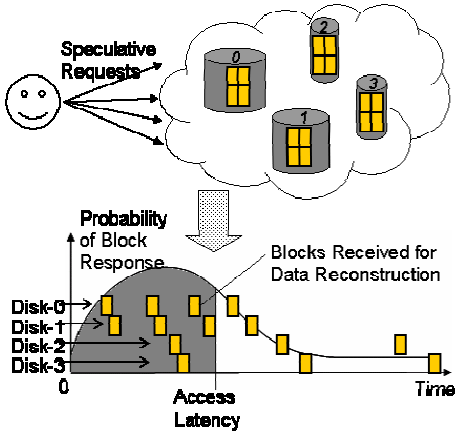

Figure 2.RobuSTore Data Access with Similar Storage. Reconstruction with first arriving coded blocks. Disk performance is varied.

We use erasure codes to provide a continuum of redundancy, enabling a flexible tradeoff of resources against robustness and high performance. In general, we encode N source data blocks into M erasure-coded blocks for any M>N; and these coded blocks contains near-uniform redundancy for all the N original blocks.

During decoding, the structure of erasure codes allows reconstruction of original data from almost any N(1+δ) coded blocks, where δ is zero or a small number decided by the choice of coding algorithms, and (1+δ) is called *reception overhead*. In our current design and experiments, we use LT Codes (modified to guarantee decidability [32]) with parameters that enable decoding with almost any set of blocks 1.4 times larger than the original data. We will discuss the choice of erasure codes in Section 3.

RobuSTore combines the decoding flexibility of erasure codes with speculative accesses. A read client in RobuSTore requests redundant coded blocks in parallel. In general, it receives over a range of time as shown in Figure 2. Benefiting from the decoding flexibility, the client can then reconstruct the original data using a set of early-returning blocks.

We do not discuss write operations in this paper, as our workloads are read-dominated. Mixed workloads will be explored in future RobuSTore studies as in [33-35].

2.2.1 Analyzing Benefits of Erasure Codes

Erasure codes provide greater reordering freedom than replication. For example, if we replicate an N block file 3 times, only a smaller number of subsets of the resulting 4N blocks are sufficient to reconstruct the original data. This is because each block has 4 copies; one of which is required in any subset which enables reconstruction. In contrast, if we encode the N block file into 4N blocks using Luby Transform (LT) codes, we can reconstruct with many more subsets. More formally, for replication the probability reconstruction with M random blocks for an N block file replicated 3 times is:

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

For an LT encoding file, using our encoding settings and equivalent expansion, the probability of reconstruction with M random blocks for an N block file expanded to 4N total blocks is:

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

See the Appendix for the full analysis. In practice, nearly 3N blocks are needed for replication versus about 1.5N blocks for LT coded (see Figure 3).

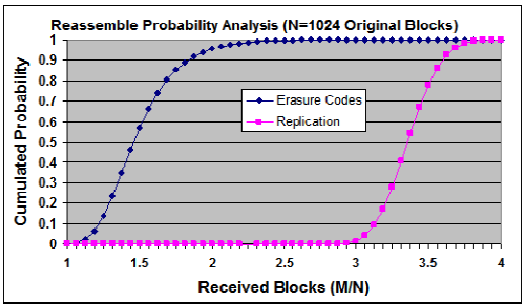

Figure 3.Cumulative Probability of Reassembly of N Original Blocks from M random Blocks

2.2.2 RobuSTore Research Questions

The degree of redundancy encoded and aggressiveness of speculative access are critical for RobuSTore performance. However, many other important research questions are also critical to understanding RobuSTore's potential to improve robustness and performance.

- With realistic disk variability, what improvements in robustness are possible?
- What coding parameters and access strategies are most effective?
- What performance improvement can be expected? With different numbers of disks? Block sizes?
- What capacity overhead, additional disk I/O's, and network bandwidth are required for a particular performance benefit?

We explore these questions in following sections.

3 RobuSTore Design

The RobuSTore system framework is flexible, enabling broad exploration of the design space. We describe the components of a RobuSTore system then discuss the key system architecture, component, and other design issues

3.1 RobuSTore System Architecture

RobuSTore storage system architecture includes four key components: client, metadata server, and storage servers (see Figure 4).

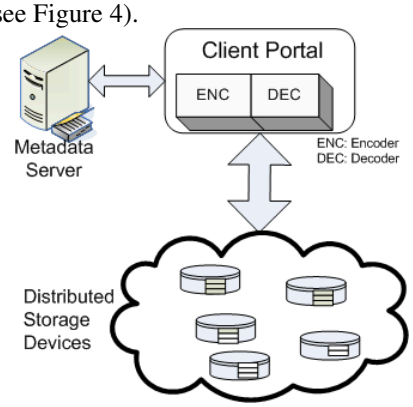

Figure 4.RobuSTore System Architecture

Client Portal perform many distributed filesystem functions: accessing metadata, planning layout, encoding data, sending requests to storage servers, and decoding/assembling the completed response. The client includes an erasure code implementation to implement the encoding and decoding. It also includes a layout planner which places data on disk servers to meet specified QoS requirement.

Metadata Server maintains both file information (size, organization, and location) and storage server information (available space, performance, connectivity, current load, etc).

Storage Servers provide data storage at block level (erasure-coded block, presumed to be larger than disk blocks). Servers may be single disks or arrays, and each implements local admission and access control. Servers typically have variable performance due to heterogeneity in hardware, data layout, or load.

3.2 Basic RobuSTore Operations

Basic RobuSTore operation interfaces include:

```
open(filename, read/write,QoS_options);
read(fd,buffer,offset,length);
write(fd,data,offset,length);
close(fd);
```

Figure 5 depicts the basic read process: open, read and close. The client opens the file, obtaining storage server and block location information from the Metadata Server, and obtaining any required locks. It also negotiates admission (bandwidth) and access with each storage server. To read, the client requests all coded blocks from servers and does LT decoding (in parallel). When enough blocks have been received, the decoding completes and the original data is returned. Simultaneously, outstanding requests to the storage servers are cancelled. Close notifies the Metadata Server releasing read locks, and also releases and bandwidth reservations on the storage servers.

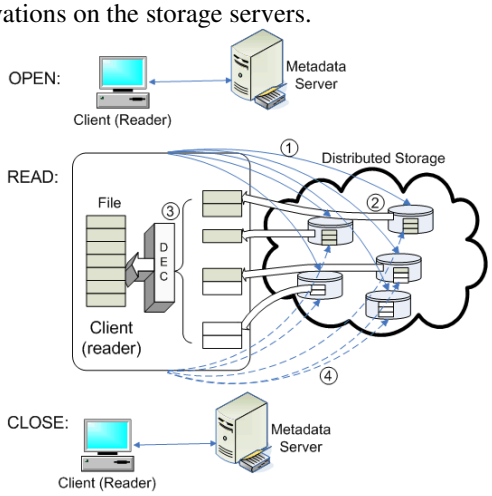

Figure 5.The Read Process in RobuSTore

Writes are similar. On write, the client accesses the Metadata Server, creates the file, and based on disk map information and application QoS requirements, it plans a layout -- an encoding and distribution across storage servers. The client then allocates the needed storage on the servers, and encodes and writes the data appropriately using specified LT coding parameters. After the data is committed on the servers, the client registers the data location and file structures with the Metadata Server. To modify a file, the client will write a new version of the file then update the metadata, this method avoids data inconsistency. Since our workloads are read-dominant, write performance is not studied in this paper.

3.3 Key Design Decisions

The RobuSTore design reflects the major choices in architecture and component implementation. General design principles include modularity, performance, scalability, and simplicity. We summarize the major design decisions below.

Coding: The coding modules (ENC/DEC) are logically in the client, critical if the composition of variable performance is to encompass network variability. If desired, a separate machine/cluster to save client CPU or improve coding speed. Because the clients manage the coding (and store information about it in the Metadata Server), the storage servers need not be aware of coding.

While there are many different erasure codes, such as Reed-Solomon [28, 29], Raptor [36], Tornado [37], etc. RobuSTore uses modified Luby Transform (LT) codes [30, 32]. The Luby Transform codes are part of a general class of low-density parity check codes [31] and use block-XOR operations based on sparse bipartite graphs.

LT codes have a number of advantages. First, they are "rateless", allowing redundancy to be decoupled from other system design issues, such as the number of storage servers used. Second, LT codes use irregular bipartite graphs and block-XOR operations, enabling fast encoding and decoding throughputs [32]. Third, LT codes allow decoding to be overlapped with receiving data from storage servers, masking decoding time.

However, as defined, pure LT codes do not guarantee the original data can be recovered from a finite number of coded blocks. To fix this, we adapted LT codes by generating coding graphs which guarantee decodability [32], and built a fast implementation which improved decoding bandwidth ten-fold 30MByte/s to 320 MByte/s on a Intel 2.4Ghz Xeon CPU. Our experiments show that the memory bandwidth is the bottleneck to coding performance. Our work suggests that multi-processors of the future or processors with higher memory bandwidth (such as AMD Opteron or AMD Athlon 64) will surpass a coding bandwidth of 1GByte/s.

Metadata Service: A simple central metadata server minimizes update and synchronization costs at some penalty in scalability. Reliability and scalability can be addressed by well-known replication and failover techniques [14, 38].

The metadata server maintains object-level information, so the storage servers function as object servers. The benefits of object-based storage systems are well recognized [20, 39, 40], and this approach supports our future extension RobuSTore to federated storage servers in the Grid [41].

Admission and Access Control: Because RobuSTore aggregate distributed resources which may be shared with local or other distributed use, central admission control is not possible. In the RobuSTore framework, each storage server implements local admission control (for QoS) and access control (for security). Clients must negotiate access on file Open. There are many good candidates for the distributed filesystem security, for example, credential-based access control [42] is flexible in the environments with a large number of servers and users.

4 Evaluation

We use a detailed discrete-event simulation to evaluate the RobuSTore idea across the configuration space, varying access strategy, the number of storage servers, data size, block size, network latency, and degree of redundancy.

4.1 Methodology

We compare RobuSTore and conventional approaches with different strategies for data layout, redundancy, and access.

Data Layout and Redundancy: Possible data layout methods include: (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. They are depicted in Figure 6.

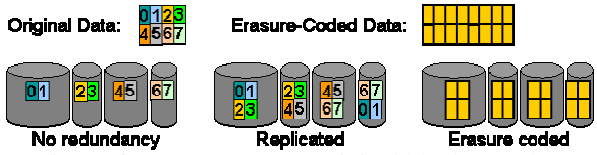

Figure 6.Data Layouts. 8 original blocks; 2x data redundancy in replicated and coded layouts

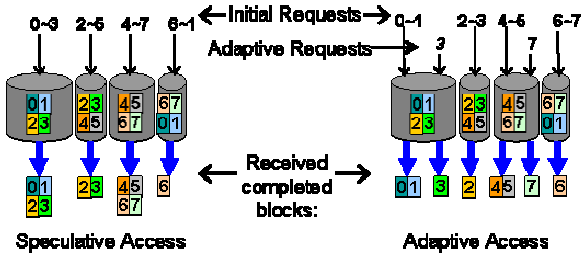

Figure 7.Access Strategies. Disk performance is varied.

Access Strategies: Possible data access strategies are: (1) speculative access, i.e., request all redundant blocks at once in the beginning of the access and cancel the requests once enough blocks have been received; (2) adaptive access, in which the client dynamically requests the unreceived bytes to adapt to disks' performance. Examples are shown for replicated layout in Figure 7 (Request cancellation is not depicted).

There are many different combinations of layout and access methods. Since adaptive access is not necessary for layout without redundancy or layout with erasure coded blocks, we evaluate the following four schemes including RobuSTore:

- **RAID-0:** No data redundancy + speculative access
- **RRAID-S:** Replication + speculative access
- **RRAID-A:** Replication + adaptive access
- **RobuSTore:** Erasure coding + speculative access

4.1.1 Workload & Performance Metrics

Workload: Since our focus is on supporting the needs of applications with large-read dominated workloads [1-3]. In these applications, the size of each data object ranges from 100s MB to 10s GB, but may be 100s GB or larger in the future. We study access performance for single 128MB, 256MB, 512MB, and 1GB reads. Data objects larger than 1GB are presumed to be accessed by multiple 1GB reads.

Performance Metrics:

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

Access Bandwidth: The delivered bandwidth for a single read is the request size divided by the access latency, including connection, disk, data transfer, and decoding time. We interpret access bandwidth to be a measure of delivered performance corresponding to our goal of "high performance".

Disk IO Size/Data Size: Several of the approaches make speculative accesses to the I/O system to improve performance, but at a cost of increased network and disk usage. We compute this increased system cost as a ratio over the original data size.

4.1.2 System Models

The system simulation involves modeling for each of the critical system components. The models need to be reasonable for reality approximation and easy for simulation.

Metadata Service, Access Scheduler, and Admission Controller: Each access to these services by the client is modeled as a constant latency of five milliseconds, with no inline overhead. Because our benchmarks do not include any open/close operations, the overhead for access to the Metadata Service and Admission Controller is never incurred.

Network: As bandwidth is presumed plentiful [5], the network is modeled as a constant latency of 1 millisecond round-trip latency (for the baseline, but varied in some other experiments). Each storage server access in RAID-0, RRAID-S, or RobuSTore involves one round-trip-time (RTT), while RRAID-A involves multiple RTTs.

Storage Server/Disk: To model storage server performance, we use a single disk model for each. This model employs the DiskSim toolkit [43], a state-of-the-art discrete event disk simulation. DiskSim parameters define bus, controller, cache, and disk (rotation rate, sector-level disk structure), which can be customized to model a wide range of commercial disks. It has been used extensively in the study of new file system and storage techniques [44-46].

In our experiments, DiskSim is configured based on measurement of a 120GB IBM Deskstar 7K400 disk (ATA-100, 7200 rpm). We use one DiskSim process to simulate each storage device. We model disk performance diversity which may arise from different disk types, layout, and sharing with the single mechanism of block layout. To generate this diversity, we used the DiskSim *synthetic workload generator*, varying *blocking factor* (average number of sectors per request) and *probability of sequential accesses* (contiguity of sequential requests) parameters to model disks with different accumulated layouts (due to different file system histories, and irregular sharing). The configuration and measured average bandwidth for each of the 128 disks is shown in Table 1.

The resulting 100-fold performance (0.52MB/s to 53MB/s bandwidth) approximates a shared distributed storage environment with many sources of variability, as discussed in Section 2.1. Although only single-disk nodes are modeled, they represent the variability characteristic of higher-performance nodes (like disk arrays), with only the difference of absolute performance.

Table 1.Disk Configuration and Performance

Disks' IDs	Blocking Factor	Prob of Seq Access	Avg BW (MB/sec)
0-7	8	0	0.52
8-15	16	0	0.76
16-23	32	0	1.3
24-31	64	0	2.5
32-39	128	0	4.7
40-47	256	0	8.3
48-55	512	0	14.3
56-63	1024	0	21.4
64-71	8	1	3.6
72-79	16	1	6.9
80-87	32	1	9.3
88-95	64	1	12.7
96-107	128	1	16.8
104-111	256	1	29.8
112-119	512	1	53.0
120-127	1024	1	53.0
Average			14.9

Erasure Coding: Coding and decoding performance is critical for RobuSTore. In each simulation, we first run the LT Codes algorithms to generate an LT coding graph (a bipartite graph connecting original blocks and coded blocks); then randomly select coded blocks and feed them to LT decoding algorithm to find out the required number of blocks for complete decoding. The simulation of reading process succeeds once enough blocks are received. We use following LT Codes parameters to generate the coding graph: $C=1.0$ and $\delta=0.5$ (C and δ are the parameters for the distribution of node degrees in LT coding graph: when C increases, there will be more low-degree nodes, and δ has opposite impact. More details are available in [30]). Previous work with LT code implementations [32] shows that the reception overhead is typically 1.4x (i.e., 1.4 times of number of blocks need to be received for a successful decoding), and decoding speeds of 200~300MByte/s are possible with typical processors, so we use that rate to compute code/decode time: Since the decoding process can be overlapped with data reception, extra latency is only incurred for decoding the last block. For example, for blocks of 1 megabyte, we add a constant latency of 5 milliseconds.

4.1.3 System and Experiments Configurations

All of our experiments involve subsets of a wide-area storage system with 128 servers each with a commodity network connectivity (>1 Gigabit/s) and a client with a high bandwidth connectivity (> 10 Gigabit/s).

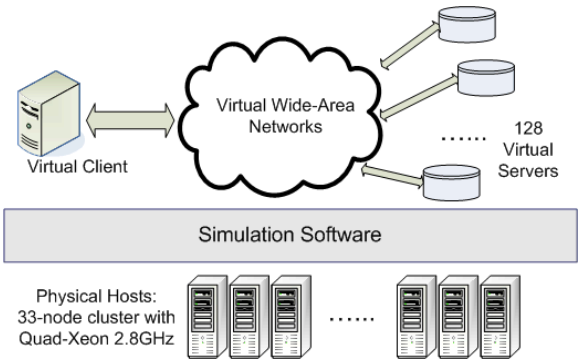

Figure 8.Experiment System Configuration

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.

For each configuration we run 100 trials and present the average and the standard deviation; in each trial, disks are randomly selected if the experiment uses less than 128 disks.

4.2 Results

We simulate the four storage scheme over five configuration dimensions. Each of the figures below shows the results along one dimension with all other four dimensions configured following the baseline above.

4.2.1 Robustness Improvement

We first compare the robustness of the four storage schemes.

Robustness vs. Number of Disks

When we vary the number of disks from 2 to 128, the standard deviation of latency changes as depicted in Figure 9. This figure uses log-scale y-axis to distinguish the details.

RAID-0 suffers because it exploits no redundancy, and the performance is thus subject to the slowest disk.

RRAID-S explores the replicated data to hide the slow disks; however, it reads the blocks on same disk in fixed order, so its performance depends on 1) *intra-disk block ordering*, for example, if a replica of a block is stored as the last block on a fast disk, it will be read last from the disk and has less contribution on hiding slow disks; and 2) *inter-disk block mapping*, i.e., which disk a block is stored on, if the replicas of same block are all on slow disks, they cannot hide the slow disks, as depicted in Figure 1. RRAID-S has the highest variability due to the combination of these factors.

RRAID-A mitigates the dependency on intra-disk block ordering by accessing blocks selectively, but it is still dependent on inter-disk block mapping.

For small number of disks (<8), the replicated schemes RRAID-S and RRAID-A have comparable robustness to RobuSTore. As the data redundancy rate is fixed to 4x and few disks are used, the system is essentially performing whole-file replication and suffers a low-level of inter-disk dependence.

RobuSTore uses erasure-coded blocks and has greater flexibility on use all the stored blocks. Its performance variability is from the overall bandwidth of all the disks and is not related to intra-disk ordering or inter-disk mapping at all. RobuSTore has the lowest performance variability for systems with more than a few disks (>8). The standard deviation of access latency on 64 disks for RAID-0, RRAID-S, RRAID-A and RobuSTore are 1.9, 7.3, 1.9, and 0.5 seconds respectively; and they are 0.63, 3.8, 1.1, and 0.13 seconds on 128 disks. RobuSTore improves robustness for up to 5x compared to RAID-0, and more than 15x compared to RRAID-S. This lower variability of RobuSTore demonstrates the benefits of erasure-coding and the resulting order-freedom.

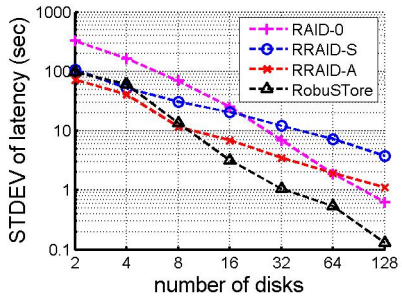

Figure 9.Robustness vs. Number of Disks.

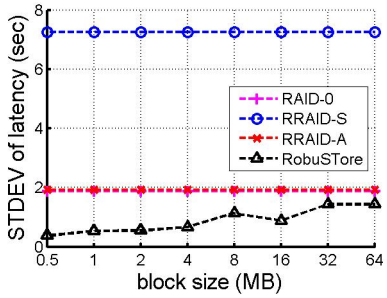

Figure 10.Impact of Block Granularity.

Robustness vs. Block Granularity

Block granularity affects the behavior of RobuSTore and has no impact on other schemes. In RAID-0, RRAID-S and RRAID-A, data blocks are replicated in plain-text, so an access can use fractions of data blocks and ignore block boundaries. However, in RobuSTore, decoded blocks are not replicated and only whole blocks can be applied to block-XOR operations for decoding. We vary block size from 0.5MB to 64MB, and present the results in Figure 10.

The STDEV of access latency in RobuSTore increases as block size grows. This is because larger block size decreases the rate of reading the blocks, so if any block is delayed, the client will potentially wait longer time for the next arriving block, i.e., the access latency is more sensitive to block delay.

Robustness vs. Network Latency

Because distributed storage networks may have a variety of latencies, we vary network latency between client and storage servers from 1ms (machine room or Metro) to 100ms (intercontinental). Our results in Figure 11 show that network latency has neglectable impact on performance robustness. In RAID-0, RRAID-S, and RobuSTore, their accesses only involve one-time read request, so the impact of varying network latency is at most one RTT on total access latency. RRAID-A uses adaptive access strategy and involves multi-RTT on total access latency. Considering that the total access latency is 2~30 seconds, the variability from network is much less than that from disks in our model.

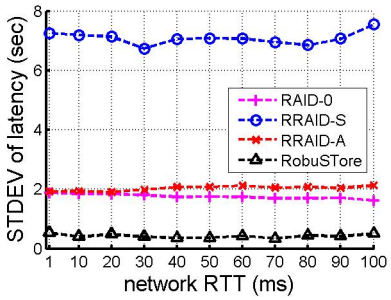

Figure 11.Robustness vs. Network Latency.

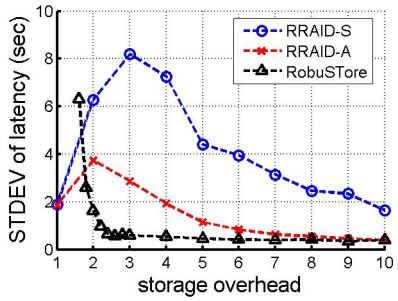

Figure 12.Robustness vs. Data Redundancy.

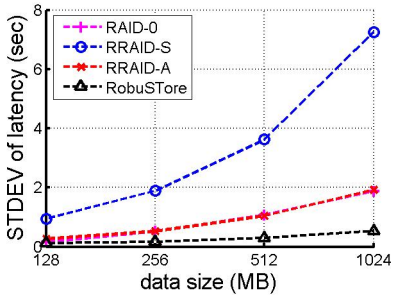

Figure 13.Robustness vs. Data Size.

Robustness vs. Redundancy

Intuitively, we can increase data redundancy to hide slow disks. We vary storage overhead from 1x to 10x (900% overhead) to quantify its performance impact (see Figure 12). Because RAID-0 scheme does not include replication, its performance is represented by the 1x point on the RRAID-S curve.

In RRAID-S and RRAID-A, the variability 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 variability and inter-disk block mapping, while more from intra-disk block ordering. RAID-0 only suffers variability from the slowest disk. Due to the combination of these factors, RRAID-S and RRAID-A have worse robustness than RAID-0 when redundancy is small, 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. RobuSTore achieves the best performance robustness, and needs only 2-3x data redundancy to obtain most of this benefit. When using more than 4x storage space, the standard deviation of latency is only ~ 0.5 seconds or about 25% of the average access latency.

Robustness vs. Data Size

A driving OptIPuter application interactively accesses high-resolution 3-D brain images; the images range in size from 100s MB to 10s GB, but may be 100sGB or larger in the future. Images larger than 1GB are presumed to be accessed by multiple 1GB reads; smaller images have relatively higher access overhead from scheduling, decoding, etc. Figure 13 shows the results for 128MB, 256MB, 512MB, and 1GB reads.

The standard deviations change as expected: less time is required to access smaller data, so the deviation of time also decreases in consequence.

RobuSTore exceeds all the other schemes.

4.2.2 Bandwidth Improvement

While robust performance is the major goal of RobuSTore, we must also maintain high access bandwidth for the requirement of accessing large datasets.

In most of our experiments, RobuSTore delivers significantly better performance than other schemes. We present the results on three dimensions of number of disks, block size, and storage overhead. Other two dimensions, network latency and data size, are either no significant impact or too simple, as we analyzed above.

Bandwidth vs. Number of Disks

We vary the number of disks and present the bandwidth results in Figure 14. RAID-0 exhibits the worst bandwidth, RRAID-S is second worst, and best bandwidth is given by RRAID-A and RobuSTore. This bandwidth gap grows as the number of disks is increased.

RobuSTore is slightly worse than RRAID-A for small numbers of disks (<8) due to the reception overhead of

1.4x; but it performs the best for large numbers of disks. RobuSTore achieves 15x the bandwidth of RAID-0 for 16-128 disks. The access bandwidth to 1GB on 64 disks is as follows: 31 MBps for RAID-0, 117 MBps for RRAID-S, 228 MBps for RRAID-A, and 459 MBps for RobuSTore.

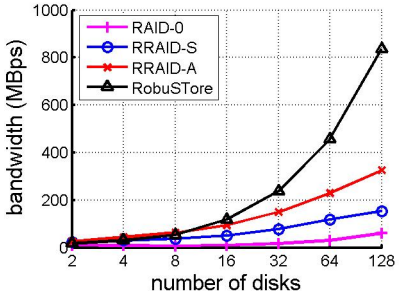

Figure 14.Bandwidth vs. Number of Disks.

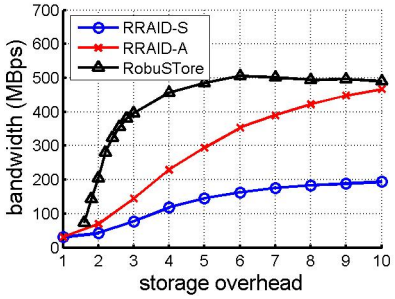

Figure 15.Bandwidth vs. Data Redundancy.

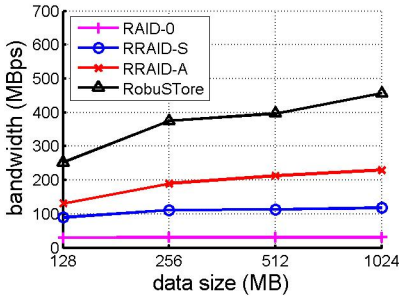

Figure 16.Bandwidth vs. Data Size.

Bandwidth vs. Data Redundancy

Figure 15 shows that increasing data redundancy increases the bandwidth of RobuSTore rapidly, reaching a peak at 6x storage (and over 15x performance). Best performance is achieved when we fully utilize the fastest disks, which requires enough blocks on the fastest disks to keep them busy for the entire access.

The bandwidth of RRAID-S and RRAID-A both benefit less than RobuSTore as data redundancy is increased. This is because their structured redundancy (replication) cannot adapt to reading more blocks from the faster disks as flexibly as in RobuSTore.

Bandwidth vs. Data Size

Figure 16 shows that all four schemes have lower average bandwidth when accessing smaller data. RobuSTore and RRAID-A are affected the most. For RRAID-A, the adaptive scheduling becomes more costly for smaller data. For RobuSTore, it is due to two reasons: (1) connection and coding time become more costly for smaller data; (2) we fix the block size for erasure coding and thus smaller data has fewer blocks, which increases the LT reception overhead (The detailed relationship between number of blocks and reception overhead is beyond the scope of this paper; it is studied in [32]).

4.2.3 Access Overhead

The benefits of aggressive access to redundant copies can yield performance benefits, but it also increases network and disk I/O costs. The number of bytes read from the disks divided by the request size captures the additional network bandwidth and storage access costs.

Overhead vs. Number of Disks

Figure 17 depicts the overhead for all the four schemes. Because RAID-0 has no speculative accesses, it incurs no additional costs, and achieves a ratio of 1. RRAID-A is just a little bit more than 1, as it only generates additional accesses when they are clearly needed. RRAID-S generates a large number of speculative requests, reaching overhead ratios as high as 3x. RobuSTore has cost about 1.4 due to the requirement of extra blocks for decoding. Although RobuSTore also uses speculative access, its use of erasure codes avoids fetching duplicated blocks, showing perfect parallelism.

Overhead vs. Data Redundancy

As data replication is increased, only RRAID-S increases its accesses in proportion. As a result, RRAID-S has increasing overhead as data redundancy increases when compared to the other schemes, as in Figure 18.

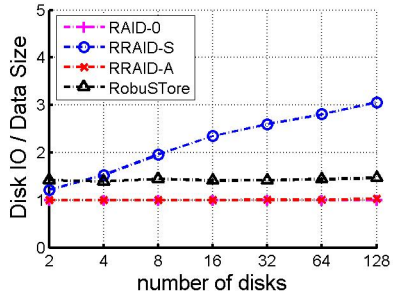

Figure 17.Overhead vs. Number of Disks.

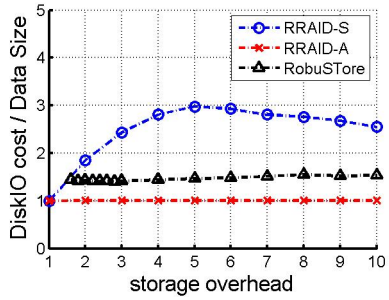

Figure 18.Overhead vs. Data Redundancy.

4.2.4 Evaluation Summary

We compared RobuSTore to RRAID-S, RRAID-A, and RAID-0 across a wide range of system configurations. These simulation results show that RobuSTore reduces performance variability significantly. For a 1GB read using 64 disks, RobuSTore receives standard deviation of access latency of only 0.5 second, less than 25% of the access latency, nearly 5x improvement compared to a baseline RAID-0 scheme. This is in the face of a challenging disk performance heterogeneity spread of 100-fold. At the same time, RobuSTore provides best performance of over 400MB/s bandwidth, nearly 15x that achieved by RAID-0. RobuSTore incurs only moderate I/O costs of about 1.4x and storage overheads of 2-3x.

5 Related Work

While there has been a wealth of work on distributed storage and performance aggregation of multiple disks, we know of no work which focuses on producing storage systems which produce robust, high performance – even in the presence of significant disk and network heterogeneity as demonstrated in our OptIPuter case study. We briefly survey related work in the distributed and parallel filesystems. Other filesystem researchers have used Erasure Codes to provide highly reliable or available data storage. The erasure codes were in fact developed to provide reliable data communication – not storage -- a closely related concept. We discuss each of these in turn.

Distributed file systems typically assume slow wide-area networks, so caching and prefetching are the primary focus, and single server performance is sufficient. Generally, little effort is spent on aggregating the performance of multiple servers. LAN or campus oriented systems include NFS [11], Sprite [13], LOCUS [47], etc. and depend on file caching (and small files with significant access locality) to achieve acceptable performance. While network performance is assumed to be moderately good, single server performance is generally acceptable. A notable exception is Zebra [48]. Wide-area filesystems such as AFS and Coda [12], xFS [49], and Pangaea [50] spend much more effort on enabling partitioned network or disconnected execution (lazy update), and assume slow networks. None of these systems addresses the aggregation of multiple storage devices for high performance; IO bandwidth achieved by one client is usually bounded by the speed of single storage server.

Parallel file systems [16-20, 48] aggregate multiple disks, addressing the performance and capacity limitation of single disks or servers. Techniques such as managing disk layout, scheduling, parallel access, intelligent and coordinated caching, prefetching, write-back techniques, and even collective I/O can deliver good performance. Traditional parallel file systems assume uniform arrays of storage devices in a system-area network (SAN) or local-area network (LAN) environment. Uniform storage devices provide homogeneous access and relative stable performance; the network environment implies low and uniform communication latency to the storage; centralized control and scheduling is also feasible in such environment to provide better performance. Recently, object-based systems such as Lustre [20] and Panasas [40] use object-based techniques to tolerate heterogeneous object server capabilities. This approach provides a static mechanism for managing performance heterogeneity. However, neither of these systems tolerates dynamic performance heterogeneity in a fashion comparable to RobuSTore.

Similar to RobuSTore, some peer-to-peer file sharing systems (Kazaa [51], BitTorrent [52]) improve access performance by speculatively fetching from massively replicated data copies. However, the massive replication is unstructured and expensive in terms of storage overhead. Further these systems focus on the shared internet where access networks limit per-node bandwidth to 1-10 megabits/s.

There are many possible choices for erasure codes besides of LT Codes, such as Raptor Codes [36], LEC [53], and Online Codes [54]. Plank et al. [55] explored the practical considerations of LDPC codes in storage systems, including coding redundancy, number of blocks, coding graph, etc. Uyeda et al. [32] improved implementation of LT Codes and achieved 300MBps decoding bandwidth.

Erasure codes are mainly used to improve reliability and availability in previous storage systems. Weatherspoon et al. [56] analyzed the impact of erasure coding and replication on availability, concluding that erasure-coded systems can provide higher availability with lower bandwidth and less storage. Numerous storage systems have exploited erasure codes for data reliability and availability (Oceanstore [33], Frangipani [57], Total Recall [58], PASIS [34], Koh-i-Noor [59], and [35], etc.). While some of these systems increase performance opportunistically by exploiting data redundancy, for none of them is this a focus.

Collins and Plank's work [60] studied the usage of Reed-Solomon Codes and LDPC Codes to improve *bandwidth* of wide-area storage systems. However, they only focused on access bandwidth, with no study on performance robustness. Furthermore, they assume slow shared networks, bandwidths < 10MByte/s, and small number of blocks ($N \leq 100$), 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.

Speculative access is used in previous works to hide I/O latency in UNIX file system [61], but it is not used in distributed storage or used with erasure codes.

6 Summary and Future Work

We have proposed RobuSTore, a system with erasure coding for flexible redundancy which exploits the statistical properties of speculative accesses to both improve performance robustness and absolute performance. This idea is a general one, and can enable fundamentally better behaved storage systems – with low standard deviations in access times. To realize the idea, we present a system framework which explains the major architecture and function required to make it workable.

Using discrete-event simulation, we compare the performance of RobuSTore with three traditional schemes which introduce redundancy and speculative access. Extensive simulations which vary number of disks, block size, network latency, and redundancy overhead in a distributed heterogeneous storage network show that RobuSTore provides superior performance robustness, for a 1GB read using 64 disks achieving a standard deviation of access latency of only 0.5 second, less than 25% of the access latency, and a 5-fold improvement. Absolute performance is also improved, achieving average 400MB/s nearly 15x that achieved by a RAID-0 system. Both improvements were achieved on disk heterogeneity of 100x in performance. RobuSTore's improvements are achieved at moderate cost; in this case ~1.4x increase in I/O operations and storage capacity increases of less than 3x.

We have only taken initial steps in evaluating RobuSTore. Natural extensions include: 1) evaluation with a richer set of workloads, varying access sizes and including writes, 2) empirical studies with a RobuSTore prototype, 3) experiments with real applications and real testbeds, 4) study of different algorithms for encoding, and 5) evaluation for multi-user workloads and shared storage servers.

Acknowledgements

Supported in part by the National Science Foundation -- Cooperative Agreement ANI-0225642(OptIPuter), CCR-0331645(VGrADS), ACI-0305390, and Research Infrastructure Grant EIA-0303622. Support from the UCSD Center for Networked Systems, BigBangwidth, and Fujitsu is also gratefully acknowledged.

The authors thank Justin Burke and Frank Uyeda for their comments on the system design and evaluation.

References

[1] "Biomedical Informatics Research Network (BIRN)," http://www.nbirn.net.
[2] "GriPhyN," http://www.griphyn.org.
[3] "The EarthScope Project," http://www.earthscope.org.
[4] "AstroGrid," http://www.astrogrid.org.
[5] L. L. Smarr, A. A. Chien, T. DeFanti, J. Leigh, and P. M. Papadopoulos, "The OptIPuter," *Communications of the ACM*, vol. 46, pp. 58-67, 2003.
[6] A. Johnson, J. Leigh, L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, C. V., N. Schwarz, A. Spale, C. Zhang, and G. Goldman, "LambdaVision and SAGE - Harnessing 100 Megapixels," CSCW Workshop on Human Factors in Advanced Collaborative Environments, Chicago IL, 2004.
[7] "iGrid2005," http://www.igrid2005.org, San Diego, CA, Sep 26-30, 2005.
[8] "National LambdaRail," http://www.nlr.net.
[9] "NetherLight," http://www.netherlight.net.
[10] "The Global Lambda Interchange Facility (GLIF)," http://www.glif.is.
[11] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System," USENIX Summer Technical Conference, Portland, Oregon, 1985.
[12] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," *IEEE Computer*, vol. 23, 1990.
[13] J. K. Ousterhout, A. R. Cerenson, F. Douglis, M. N. Nelson, and B. B. Welch, "The Sprite Network Operating System," *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 1988.
[14] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams, "Replication in the Harp File System," 13th ACM Symposium on Operating Systems Principles, Pacific Grove, CA, 1991.
[15] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," International Conference on Management of Data (SIGMOD), 1988.
[16] P. F. Corbett and D. G. Feitelson, "The Vesta parallel file system," *ACM Transactions on Computer Systems*, vol. 14, pp. 225--264, 1996.
[17] N. Nieuwejaar and D. Kotz, "The Galley Parallel File System," *Parallel Computing*, vol. 23, pp. 447-476, 1997.
[18] P. H. Carns, W. B. L. III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System For Linux Clusters," 4th Annual Linux Showcase and Conference, Atlanta, GA, 2000.
[19] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," FAST 2002, 2002.
[20] C. F. System, "Lustre: A Scalable, High-Performance File System," Lustre File System v1.0 Architecture White Paper from clusterfs.org, 2002.
[21] E. Grochowski and R. D. Halem, "Technological impact of magnetic hard disk drives on storage systems," *IBM Systems Journal*, vol. 42, pp. 338-346, 2003.

[22] N. Taesombut and A. Chien, "Distributed Virtual Computer (DVC): Simplifying the Development of High Performance Grid Applications," the Workshop on Grids and Advanced Networks (GAN'04), Chicago, Illinois, 2004.

[23] D. J. Bishop, C. R. Giles, and G. P. Austin, "The Lucent LambdaRouter: MEMS Technology of the Future Here Today," *IEEE Communications magazine*, vol. 40, pp. 75-79, 2002.

[24] E. Weigle and A. A. Chien, "The Composite Endpoint Protocol (CEP): Scalable Endpoints for Terabit Flows," IEEE Conference on Cluster Computing and the Grid (CCGrid 2005), 2005.

[25] R. Wu and A. Chien, "GTP: Group Transport Protocol for Lambda-Grids," 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004), Chicago, Illinois, 2004.

[26] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A Fast File System for UNIX," *Computer Systems*, vol. 2, pp. 181-197, 1984.

[27] C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns," Winter USENIX, San Diego, CA, 1993.

[28] J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," *Software -- Practice & Experience*, vol. 27, pp. 995-1012., 1997.

[29] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300--304, 1960.

[30] M. Luby, "LT Codes," IEEE Symp. On Foundations of Computer Science 2002.

[31] R. G. Gallager, *Low Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[32] F. Uyeda, H. Xia, and A. Chien, "Evaluation of a High Performance Erasure Code Implementation," UCSD, Technical Report CS2004-0798, 2004.

[33] J. Kubiatowicz, D. Bindel, and e. al., "OceanStore: An Architecture for Global-Scale Persistent Storage," the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000.

[34] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, "Fault-Scalable Byzantine Fault-Tolerant Services," Symposium on Operating Systems Principles, Brighton, UK, 2005.

[35] M. K. Aguilera, R. Janakiraman, and L. Xu, "Using Erasure Codes Efficiently for Storage in a Distributed System," DSN 2005: The International Conference on Dependable Systems and Networks, Yokohama, Japan, 2005.

[36] A. Shokrollahi, "Raptor codes," Digital Fountain and EPFL, 2003.

[37] M. Luby, "Practical Loss-Resilient Codes," ITW 1998, San Diego, CA, 1998.

[38] J. F. Bartlett, "A NonStop Kernel," the Eighth Symposium on Operating System Principles, 1981.

[39] G. A. Gibson, J. S. Vitter, and J. Wilkes, "Strategic directions in storage I/O issues in large-scale computing," *ACM Computing Surveys (CSUR)*, vol. 28, 1996.

[40] "The Panasas File System," http://www.panasas.com

[41] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Intern. J. High Performance Computing Applications*, vol. 15, pp. 200-222, 2001.

[42] S. Miltchev, V. Prevelakis, S. Ioannidis, J. Ioannidis, A. D. Keromytis, and J. M. Smith, "Secure and Flexible Global File Sharing," USENIX 2003 Annual Technical Conference, San Antonio, TX, 2003.

[43] J. S. Bucy and G. R. Ganger, "The DiskSim Simulation Environment Version 3.0 Reference Manual," Carnegie Mellon University CMU-CS-03-102, January 2003.

[44] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang, "Modeling hard-disk power consumption," 2nd USENIX Conference on File and Storage Technologies (FAST '03), 2003.

[45] J. L. Griffin, J. Schindler, S. W. Schlosser, J. C. Bucy, and G. R. Ganger, "Timing-accurate storage emulation," USENIX Conference on File and Storage Technologies (FAST'02), Monterey, CA, 2002.

[46] A. Riska, E. Riedel, and S. Iren, "Managing Overload Via Adaptive Scheduling," 1st Workshop on Algorithms and Architecture for Self-Managing Systems, San Diego, CA, 2003.

[47] G. J. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Theil, "LOCUS: A Network Transparent, High Reliability Distributed System," the Eighth ACM Symposium on Operating Systems Principles, published in Operating Systems Review 15, Pacific Grove, CA, USA, 1981.

[48] J. H. Hartman and J. K. Ousterhout, "The Zebra striped network file system," *ACM Transactions on Computer Systems (TOCS)*, vol. 13, pp. 274 - 310, 1995.

[49] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS File System," USENIX 1996 Technical Conference, San Diego, CA, 1996.

[50] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam, "Taming Aggressive Replication in the Pangaea Wide-Area File System," OSDI '02, 2002.

[51] "Kazaa," http://www.kazaa.com.

[52] "BitTorrent," http://www.bittorrent.com.

[53] J. A. Cooley, J. L. Mineweaser, L. D. Servi, and E. T. Tsung, "Software-based Erasure Codes for Scalable Distributed Storage," 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, San Diego, CA, 2003.

[54] P. Maymounkov, "Online Codes," Tech Report of NYU, TR2002-833, November 5, 2002.
[55] J. S. Plank and M. G. Thomason, "A Practical Analysis of Low-Density Parity-Check Erasure Codes for Wide-Area Storage Applications," DSN-2004: The International Conference on Dependable Systems and Networks, Florence, Italy, 2004.
[56] H. Weatherspoon and J. D. Kubiatowicz, "Erasure Coding vs. Replication: A Quantitative Comparison," the First International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, 2002.
[57] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System," 16th ACM Symposium on Operating Systems Principles, Saint Malo, France, 1997.
[58] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," the First ACM/Usenix Symposium on Networked Systems Design and Implementation (NSDI), 2004.
[59] M. Isard, M. Manasse, and C. Thekkath, "Koh-I-Noor Project," http://research.microsoft.com/research/sv/kohinoor/.
[60] R. L. Collins and J. S. Plank, "Assessing the performance of Erasure Codes in the Wide Area," DSN-2005: The International Conference on Dependable Systems and Networks, Yokohama, Japan, 2005.
[61] F. W. Chang and G. A. Gibson, "Automatic I/O Hint Generation through Speculative Execution," the 3rd Symposium on Operating Systems Design and Implementation, New Orleans, LA, 1999.

Appendix: Analysis

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

$F_M(N) = 0$, if $N<M/4$,

which satisfies (A.1).

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

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

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

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

$$= \binom{4N}{M} - \sum_{j=1}^{N-1} \sum_{i=j}^{N-1} (-1)^{i-j} \frac{N!}{i!(N-i)!} \cdot \frac{i!}{j!(i-j)!} \binom{4j}{M}$$

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

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

$$= \binom{4N}{M} - \sum_{j=1}^{N-1} \frac{N!}{j!(N-j)!} (-(-1)^{N-j}) \binom{4j}{M}$$

$$= \sum_{j=1}^{N} (-1)^{N-j} \binom{N}{j} \binom{4j}{M}$$

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:

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