2019

A Scalable Nearline Disk Archive Storage Architecture for Extreme Scale High Performance Computing

Hampton Walker Haddock
University of Alabama at Birmingham

A SCALABLE NEARLINE DISK ARCHIVE STORAGE ARCHITECTURE FOR EXTREME SCALE HIGH PERFORMANCE COMPUTING

by

HAMPTON WALKER HADDOCK

PURUSHOTHAM BANGALORE, COMMITTEE CHAIR
MATTHEW CURRY
ROBERT HYATT
SIDHARTH KUMAR
ANTHONY SKJELLUM

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2019

A SCALABLE NEARLINE DISK ARCHIVE STORAGE ARCHITECTURE FOR EXTREME SCALE HIGH PERFORMANCE COMPUTING

HAMPTON WALKER HADDOCK

COMPUTER SCIENCE

ABSTRACT

Parallel file systems that exploit Redundant Arrays of Inexpensive Devices (RAID) as the mechanism for greater resilience are primarily intended to provide high bandwidth and low latency. Quantifying and studying the trade-offs among reduced run time (bandwidth and latency), resilience (availability and integrity), and cost (energy and capital) is important. For instance, distributing the checksums of RAID systems appears in conflict with the canonical parallel access patterns in high performance computing such as long sequential reads, random access, and checkpoint operations. Choices consequently have to be made between performance, concurrency, latency, energy, capital, integrity and availability of the data for normal operation as well as during recovery of a failed device. New strategies are emerging for exascale storage that create additional layers in the storage hierarchy. These strategies are primarily designed to take advantage of the economics of cloud storage technologies and especially the benefits of erasure coding. The Los Alamos National Laboratory has implemented a "Campaign" layer placed below the traditional Parallel File System where longer latencies and lower bandwidth can be traded for lower cost and higher capacities. In addition, Burst Buffers are now being used on top of the traditional Parallel File System to provide higher bandwidth and lower latency for petascale and beyond. These new layers are specialization over the Parallel File system based on trade-offs between cost and performance.

In this dissertation we analyze the requirements of the HPC storage space and identify special problems in the archive layers. We leverage the GPGPU to provide erasure coding on large stripe sizes to increase performance and availability. We also show that data confidentiality can be provided along with erasure coding on GPGPU reducing the overall cost of data protection for nearline disk archive storage.

Keywords: GPU, erasure code, storage, high performance computing, encryption

DEDICATION

To my wife, my children and their children.

ACKNOWLEDGMENT

I would like to thank the National Science Foundations (NSF Grant Nos. ACI-1541310, CNS-0821497, CNS-1229282, CCF-1562306, OAC-1642078, and CCF-1822191) for providing the funding to UAB to acquire the parallel file system that enabled a large portion of our research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

I would like to thank Sandia National Laboratories, especially Matthew Curry, for mentoring me on the art of erasure coding on the GPU platform as well as providing insight into the real issues of high performance storage systems. This material is based upon work supported by Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivations

With the compute core density increasing per node in the past decade in High Performance Computing (HPC), a trend which will likely continue for the next decade, Input/Output (IO) bandwidth requirements per node have also increased. This increase in computing power is putting pressure on the storage capacity and bandwidth throughout HPC systems. To minimize the time required for applications to complete input-output (IO) operations for initialization, checkpoint/restart (CR), and application-specific IO, it is necessary to provide IO bandwidth to the compute nodes that is much higher than today's petascale supercomputers. The stated requirements for the exascale initiative is for applications to run five times faster than they do on today's 200 Peta Floating Point Operations Per second (FLOPs) systems; scaling the IO bandwidth for this requirement results in IO bandwidth speeds that are five times the speed of these systems. Los Alamos National Laboratory (LANL) has introduced Burst Buffers (BB) as an intermediate tier between the compute nodes and the Parallel File System (PFS) [3]. These BBs use Solid State Disks (SSD) and Nonvolatile RAM (NVRAM) to provide high speed storage to meet the faster IO requirements. The BBs enable the applications to complete the IO operations in an acceptable time for Checkpoint/Restart (CR) or application recording task and get back to making forward progress on the application problem solution.

Both BB and PFS file systems are more expensive than slower object and tape archive systems. BBs and PFSs belong at the top of the storage pyramid having minimal size due to cost but providing greater performance. This is analogous to the memory hierarchy in the CPU. What are the optimal choices for the investment in BBs, PFS,

and archive systems to meet the requirements of the HPC system? Current choices are based on experience. The Trinity supercomputer has a memory capacity of 2 PB and the BB capacity is two times the amount of RAM in the compute nodes, 4 PB, with an IO bandwidth of 4 TB/s. The PFS storage capacity for Trinity is 25 times the storage capacity of the BBs, 100 PB, with an IO bandwidth of 1.2 TB/s [4, 5]. Requirements also provide the constraints that the lifetime of the data in the CPU memory and the BBs is hours, and the lifetime in the PFS is weeks. These working set data are the ones being used by current compute jobs that are running on the cluster. The BBs are referred to as tier one in the storage pyramid and the PFS are referred to as tier two. Data that is needed to be kept for longer periods of time may be stored on lower tiers of the storage pyramid where higher latency and lower bandwidth may be tolerated but with greater integrity and availability requirements due to the longer life of the data residence. Below the PFS a third tier has been identified that can store data for a period of time that the research project is actively computing so that data can be quickly moved to the PFS and BB when needed for computation and also so that data can be moved to the third tier relatively quickly when it not necessary to keep it available for computation or analysis. Gary Grider's team has dubbed this tier storage the "campaign" storage layer [6]. This campaign storage layer is a strong candidate for lower cost cloud type storage that provides availability with erasure coding. Tier two storage is typically provided by products like Lustre [7], which uses hardware Redundant Array of Inexpensive Disks (RAID) 6 arrays or Grid RAID [8] and fault tolerant pairs of object storage servers which are much more expensive than cloud based erasure coded storage which uses standard components. An example of this storage hierarchy is shown in Figure 1.1. Since campaign storage is an object storage system that is intended to provide a nearline short-term archive between the PFS and tape archive systems, we call this type of system Nearline Disk Archive (NLDA).

Another problem is the increasing trend in the amount of data that must be preserved for the far distant future, practically forever. Tape has been the preferred solution for

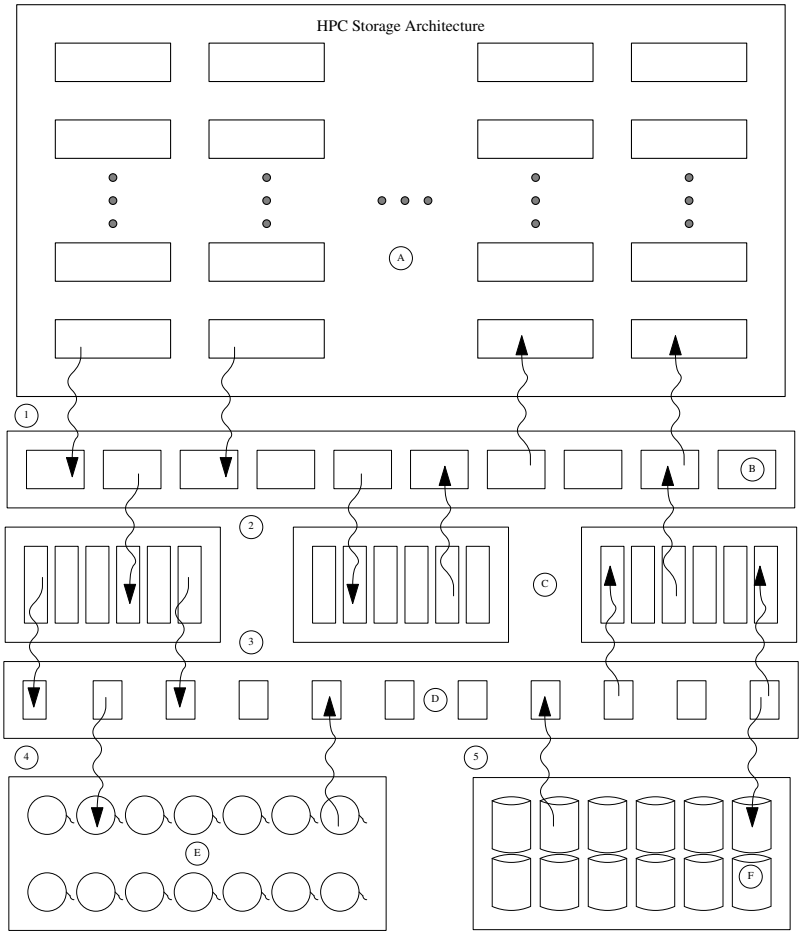

FIGURE 1.1: HPC Storage Architecture — Modern HPC Storage architecture showing the use of Burst Buffers (B) and Nearline Disk Archive (F). Compute nodes (A) have a data life measured in hours and need very high bandwidth to storage for Exascale. This is provided by the Burst Buffers (B) at hundreds of TB/s (1). PFS (C) can provide TB/s bandwidth (2). Hierarchical Storage Management (HSM) moves data between PFS (C), Tape Archive (E) and/or NLDA (F) using File Transfer Appliances (FTA) (D) at data rates of 100s of GB/s (4) and (5). Data lifetime on Burst Buffers (B) is hours; data lifetime on PFS (C) is weeks; data lifetime on NLDA (F) is campaign duration, months; and data lifetime on tape (E) can be forever.

archive storage which costs about $0.01 per GB [9] but bandwidth due to the cost of tape handling systems (robotics) is a significant budget factor at refresh time. Increasing disk drive capacities and falling cost per GB along with the advent of erasure coding technologies being used for low cost cloud storage can provide a viable competitor. One group of researchers have developed models for comparing the cost and performance of various archive storage implementations between hard disk drives, solid state drives and tape [10]. They have concluded that tape and hard disks for archive storage will remain competitive for years to come. The choice depends on the frequency of access to the files. This tends to reinforce the use of a nearline disk archive storage for data while the access is relatively frequent but the use of tape archive when the data needs to be retained for a long time without frequent access (possibly never).

Large HPC users such as National Nuclear Security Administration (NNSA) have begun to implement these new layers of HPC storage into their infrastructure. An important obstacle for the use of the cloud object storage technology is the large investment in applications that depend on the Portable Operating System Interface (POSIX®) file system to perform IO. In addition, the users of these systems are very accustomed to organization their data in the POSIX® file system. These deeply rooted uses of POSIX® require that HPC storage must have a capability to provide POSIX® interfaces for the near to medium future and possibly longer. At the same time, some in the industry are already declaring the death of the POSIX® file system [11] and provide good reasons why users should refactor their applications to use the much faster object Application Programming Interfaces (API). However, there are very practical reasons for the tree structured organization of artifacts, mostly because they are more natural for people to understand; humans are able to process information better when there are no more than about seven objects to track [12]. LANL has created a highly scalable POSIX® file system dubbed MarFS (Mar is taken from the Spanish word for sea). MarFS provides an extremely scalable POSIX® metadata architecture that can meet the LANL requirements of billions of files per directory and trillions of files in the storage system. The MarFS

also handles files that are zetabytes big and one byte small. John Bent, of Seagate, argues that the four layer storage stack will again converge to two layers consisting of BB like storage services with low latency that are close to the computing elements that are backed by slower and cheaper storage services that can store artifacts for long periods of time [11, 13, 14].

Many data storage use cases require protection such as restricting access to authorized users only. At a minimum, the Health Insurance Portability and Accountability Act (HIPAA) requires protection of data at rest [15]. The Department of Defense and National Security Agency have included encryption at rest as a requirement in their Capability Gold Standard [16, 17]. An effective and acceptable control for meeting these requirements is encryption. There are standards that specify the sufficient strength of these encryption algorithms and their use such as the Federal Information Processing Standards (FIPS) [18]. Some organizations have determined that very strong encryption is required to protect data of a very sensitive nature to be able to defeat brute force attacks and even the threat quantum computing methods to break the encryption and reveal the data [17]. Providing encryption for the protection of data requires a very systematic process to generate and manage the encryption keys [19–21]. The acquisition of the keys by unauthorized users will render the control ineffective and can be used to reveal the data by the unauthorized user. The loss of the encryption keys will render the data unavailable to the owners of the data. The concerns about the impact of data availability include the risks from the availability of the encryption keys. These risks are mitigated through well engineered key management systems [19–21].

1.2 Goals and Metrics

In HPC, IO presents much opportunity for parallel processing. HPC storage systems have exploited parallel IO for decades. The most popular medium for storing data persistently and economically where it may need to be accessed frequently has been the hard disk drive. While hard disk drive capacities have continued to increase over the

years, the rate that data can be read or written has not grown at the same rate. Modern hard disk drives are now on the order of 14 TB with bandwidth around 200 MB/s.

In summary, the goals for our scalable nearline object storage architecture for exascale and beyond is as follows:

1. Increase the ratio of usable storage to raw available storage through erasure coding improving utilization
2. Improve performance by achieving erasure coding and repair operation at rates greater or equal to other data flow constraints
3. Exploit parallelism by writing data to multiple media targets e.g., disk drives concurrently
4. Improve reliability by meeting or exceeding the Mean Time To Data Loss (MTTDL) requirements
5. Minimize capital costs by storing data on economical off the shelf systems
6. Minimize energy costs by reducing data movement and using efficient erasure coding and encryption computing

1.3 Dissertation Statement

HPC storage architecture has several levels of performance ranging from very high throughput for moving data into and out of powerful computing units to lower throughput where data is not frequently accessed. Burst Buffers provide much higher throughput than PFSs by using SSD devices which have several times the bandwidth of rotating disks although at higher cost. PFSs that exploit Redundant Arrays of Inexpensive Disks (RAID) as the mechanism for greater resilience are primarily intended to provide high bandwidth and low latency. Tape archive throughputs are much lower than burst buffers and parallel file systems but store data for long life times, possibly forever. We study an intermediate storage layer between the parallel file systems and tape archive, nearline disk archive,

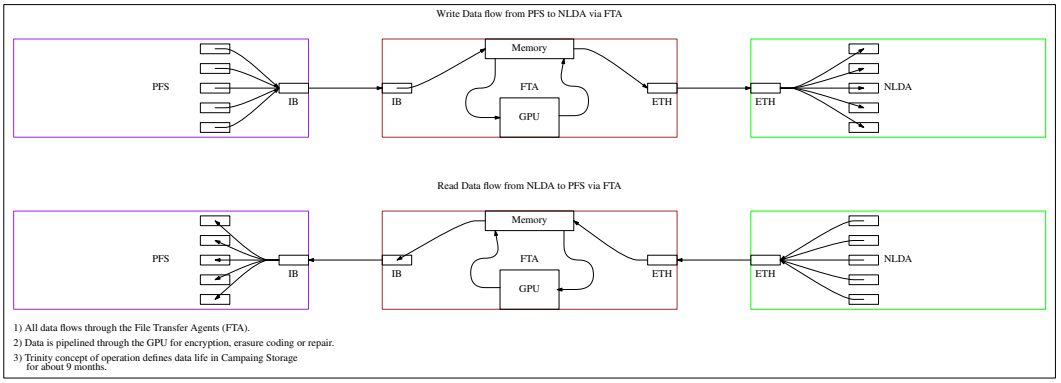

FIGURE 1.2: Data flows between PFS and NLDA — In the top illustration, data flows from the PFS to the NLDA via the FTA where erasure coding and encryption are performed. In the bottom illustration, data flows from the NLDA to the PFS via the FTA where decryption and erasure repair is performed when required.

where constrained requirements provide interesting opportunities for advancements. We design an architecture for nearline disk archive storage that is more performant, of lower cost, more energy saving, more secure, and more reliable than existing designs

1.4 The Approach

In this dissertation, we analyze the function and data flows in nearline disk archive storage (NLDA) systems. These systems are based on object storage technology that has provided economical performance for cloud providers. Based on this analysis, we design a more performant and efficient architecture for extreme scale HPC by decoupling erasure coding and encryption from the object storage system. This new design provides concomitant benefits of reduced data communications costs, more tolerance of media failures, reduced erasure repair costs, and greater enforcement of confidentiality. We exploit the high performance of the Gibraltar erasure coding library [22] and modern general purpose graphical processing unit (GPU) to encode or repair large stripe sizes with high bandwidth [23] which lends to higher performance in the object storage system [24]. Data movement to/from archive in HPC data centers is usually performed by File Transfer Appliances (FTAs) as depicted in Figure 1.2. This architecture provides for the use of resource management systems employed by the HPC system to perform the data movements as a job which is submitted to the queuing system of the resource

manager. These jobs can be chained with compute or other analysis jobs so that the data movement can occur before the other work is able to run. Jobs can be submitted to move data from the higher performance storage systems to the archive by submitting a job for this type of work. The FTAs enable interactive data movement as well so users can manually issue the commands to move data to archive or from archive. Because all data movements traverse the FTAs, this presents an opportunity to perform operations on the data during the transit. Partitioning streams of data into stripes and performing erasure coding over these chunks or shards of the data that is moving to the NLDA is one such operation. Repairing erasures in stripes of data that are being moved from the NLDA to PFS is another such operation. Indeed, data can also be encrypted while moving from PFS to NLDA and decrypted when moving from NLDA to PFS.

Our NLDA achieves the following six objectives:

Erasure Coding — High performance erasure coding with a GPU enables stripes to have many times more data shards, K, and checksum shards, M, than with CPU based erasure coding. We use K to refer to the number of shards we divide a block of data into in order to produce a stripe of data for storage. We also compute M checksum shards using an erasure coding algorithm that is maximum-distance separable (MDS) [25] that we store by concatenating to the K shards producing a stripe that has $N = K + M$ shards. This increases the storage efficiency ratio by making $K \gg M$.

Lazy Repair — Data lifetime on nearline disk archive is expected to be relatively short (months not years) [7] due to service level agreements (SLA) that are recommended. Erasure coding redundancy can be set to exceed this lifetime making immediate repair unnecessary. When users read data, data can be repaired at that time when the data is transiting the FTA for the purposes of the user, avoiding the need to read data solely for repair as is traditionally done in RAID and erasure coding systems. Because the repairs occur at high performance, the user experience is not noticeably affected.

Encryption — While data is being moved to NLDA and erasure coding is being performed, the data can also be encrypted in the GPU without additional data movement costs. Decryption is performed when the data is being moved from the NLDA to the PFS. While data repair is not always necessary when transferring stripes of data from the NLDA, if it is encrypted, it will have to flow through the GPU for decryption.

Higher Performance — FTAs need to move data at high bandwidth between PFSs and NLDAs. FTAs can be ganged together to provide a linear speedup in the data bandwidth. Moving large files can be accomplished in P/N where P is the time for a single FTA to transfer the data and N is the number of FTAs. The bandwidth provided by an FTA will be the $MIN(BW_{\text{PFS}}, BW_{\text{NLDA}}, BW_{\text{GPU}})$ where BW_{PFS} is the bandwidth provided by the PFS to the FTA, BW_{NLDA} is the bandwidth provided by the NLDA to the FTA, and BW_{GPU} is the bandwidth of the erasure coding and encryption on the GPU using the Gibraltar library.

Reduced Capital Costs — By moving the erasure coding and encryption functions from the object storage system to the FTAs we reduce the compute and memory requirements of the object storage servers (OSSs). Since the ratio of OSSs to FTAs should be much greater than one in a typical HPC data center, this results in a reduction in the cost of the servers for storage; also, the energy costs of operating these servers is also reduced. Since modern GPUs cost is on the order of HPC grade multi-core processors, by offloading the erasure coding and encryption to the GPU, the high performance can be attained with servers on the same order of cost as without the GPU.

Reduced Energy Consumption — Erasure coding and encryption of large stripes with high data shard count and high checksum shard count is more energy efficient on the GPU than using the Intel® CPU.

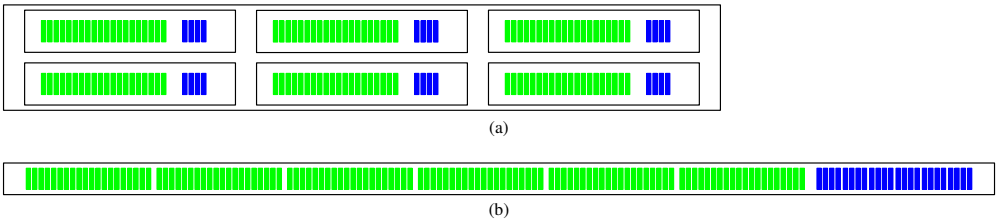

FIGURE 1.3: Erasure code stripe comparison — Comparison between two stripe sizes. In a) The current Trinity configuration, k=20, m=4, 24 shards per stripe set, total storage available is $k \times shard\ size = total\ bytes\ per\ stripe$. In b) our example configuration $k' = 120$, $m' = 24$, 144 shards per stripe set, total storage available is $k' \times shard\ size = total\ bytes\ per\ stripe$. Where k' is $6 \times k$ in our example.

In order to understand these six objectives with our NLDA architecture that we use, we show in Figure 1.3, which illustrates the typical stripe configurations for object storage systems and the large stripe sizes that we propose. In the Campaign Storage system used by Trinity [3], the stripe sizes are $K = 20$ and $M = 4$ providing a storage efficiency ratio of $K/M = 5$. In our NLDA, we are able to create stripes that are much larger; using the same storage efficiency ratio with six times the number of shards, we will have $K = 120$ and $M = 24$. Data capacities of the stripes are also increased by the same amount, six in this example. Because the NLDA may be used for archiving petabyte sized files, larger stripe sizes reduce the number of storage elements that must be tracked in the metadata. Assume that our NLDA has a 60 PB usable capacity. Adding enough capacity for erasure code redundancy, we would need $1.2 * 60 = 72$ TB of raw storage. Using 10 TB disk drives, we will need 7,200 disk drives. Modern storage systems like Ceph distribute objects uniformly across all of the disks in the cluster. A petabyte file will require many more than one million objects to store the shards of data in the stripes so we are guaranteed that our data will be distributed across all of these 7,200 disks. Assuming a 5% annual disk failure rate, the $EC(20, 4)$ Trinity configuration would lose one to two shards in the first year and have a remaining redundancy of two to three shards. In our NLDA configuration with $120x24$ we would lose seven to eight shards and have a remaining redundancy of 16 to 17 shards. The latter configuration provides a much higher margin of safety to prevent data loss and enables the policy of Lazy Repair.

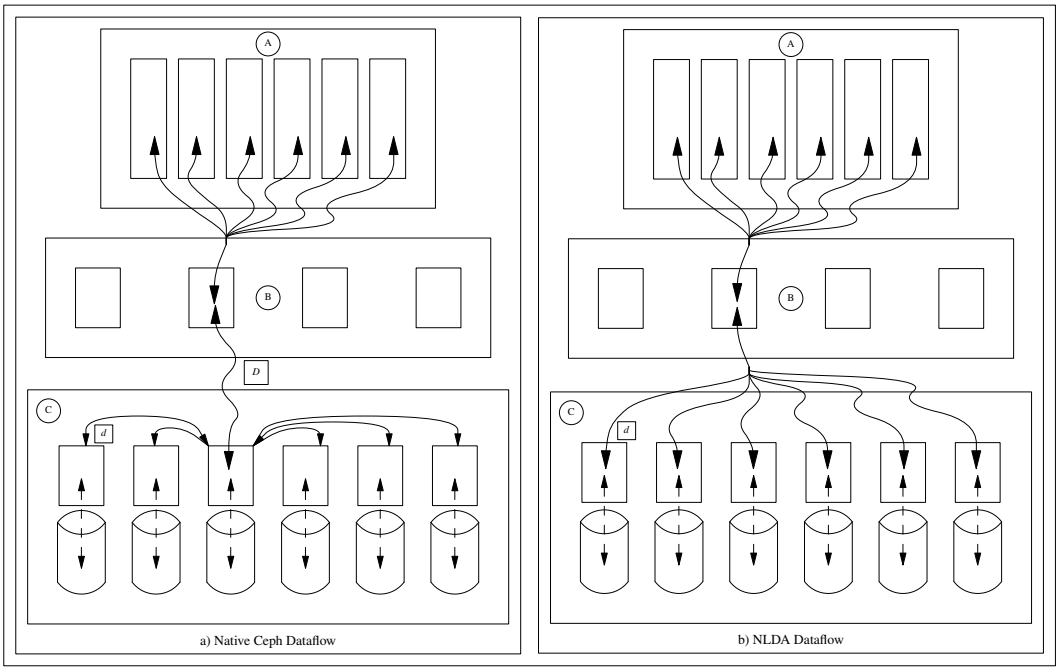

FIGURE 1.4: Comparison between Ceph and Nearline Disk Archive erasure coding — In a), with Ceph, erasure coding is done on the object storage server. In b), with NLDA, erasure coding is performed on the FTA. The PFS is depicted as A in the illustration; the FTAs as B and the Object Storage as C. In the left illustration, all of the data in a stripe moves between the FTA and a specific OSS where it is erasure coded or repaired; the shards are then copied to or from other OSSs. In the right illustration, all of the data in a stripe is erasure coded on the FTA before the data shards and the parity shards are copied directly to the OSS where it is stored. In the left figure the total data moved is $D + \frac{D \times (K+M-1)}{K}$ bytes, where D is shown in the square box and represents the total number of bytes of data in a stripe and d is shown in the smaller square box representing $\frac{D}{K}$, while in the right figure, the total data moved is $\frac{D \times (K+M)}{K}$ bytes, resulting in a reduction in communications of $D - \frac{D}{K}$.

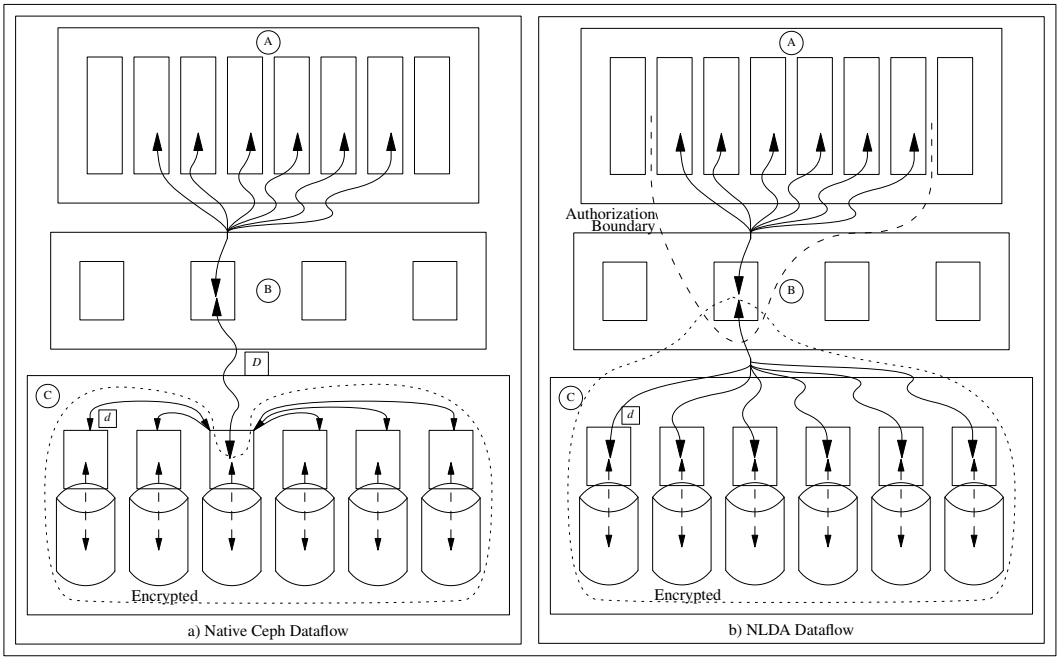

FIGURE 1.5: Comparison between Ceph and Nearline disk archive encryption — Ceph encryption is done on the object storage server. With NLDA, encryption is performed on the FTA.

Present day object storage systems, like Ceph, perform erasure coding in the storage clusters on the OSSs. This is a reasonable compromise in most cases, but with HPC storage, where FTAs are used as data movers, it is not the optimal solution. When erasure coding is performed on the OSSs, it raises the compute and memory requirements for these nodes. Additionally, data will have an extra network transit to make due to this design as shown in Figure 1.4. In our NLDA design, we decouple the erasure coding from the storage system and perform this service on the FTA nodes. In the 60 PB server described previously, having 30 disks per server, would require 240 OSSs. In contrast, there would only need to be about 60 FTAs according to the requirements for the Campaign Storage used on Trinity [7]. In the NLDA architecture, much lower powered OSSs would be required which will reduce the capital costs and the operating costs. Assuming that each FTA could process one GB/s giving a total capacity of 60 GB/s for the full set of FTAs, the one PB file could be written in 4.6 hours.

Currently, encryption is a feature provided by the object storage system. As shown in Figure 1.5 the encryption is performed on the OSSs as the data are written to the object

store. In this mode of operation, the security enclave includes the object store. This type of protection would not be sufficient for environments where some users are not authorized to have access to some of the data that is stored in the object store. System owners would have to choose between using some other archive system for their data that provided the necessary confidentiality controls or encrypt the data before sending it to the archive. By performing the encryption on the FTA, the protection can be provided in an effective manner. Since the FTA will be a cluster resource for a user to perform data movement tasks, the transfer will be performed at the user's authorization level and by a process with the user's authorizations. Since the FTA can be provided in a secure way, it can be logically within the authorization boundary, and when operated in this manner, with encryption being performed on the FTA, the data being sent within an erasure coded stripe to the object store will have confidentiality provided at the level of encryption selected by the user and system operators.

1.4.1 Methodology

We conducted our measurements on Dell R730 servers with an NVIDIA® K40m GPU installed in each FTA. Table 1.1 lists the common configuration of our cluster. The FTAs and Object Storage Servers (OSS) are interconnected via the 10 Gb/E network fabric implemented with a Dell Force 10, S4810P switch. Each node has two SFP+ 10 GbE interfaces which are bonded together to provide 20 Gb/s total bandwidth. There are four OSS nodes, two FTA nodes configured as shown with a third FTA node having dual Intel® Xeon® E5-2660 v3 @ 2.60GHz. All nodes have Hyperthread-enabled and run 40 threads. We are using 48 six TB SAS disk drives on each OSS in the test configuration, a total of 192 disks, to provide about one PB of total storage in the Ceph file system. The one Gb/s network is used for management and control.

TABLE 1.1: Dell R730 File Transfer Nodes.

CPUs	2x Intel® Xeon® E5-2650 v3 @ 2.3 GHZ (Hyperthread-enabled: 40 threads)
RAM	128 GB 2133 MT/s RDIMM
Network	Intel® X520 DP 10Gb DA/SFP+, I350 DP 1Gb Ethernet
System Drives	2x 300 GB 10K SAS 2

With this configuration, we implement a Ceph storage cluster to measure the bandwidth performance for writing and reading data into the NLDA from/to an HPC PFS. The IO with the PFS is simulated in our software by creating buffers and writing data patterns into them. This provides a convenient way to provide a consistent source by eliminating variables that are outside the scope of these measurements. We use this configuration to measure a baseline performance using the native Ceph configuration with erasure coding performed on the OSSs. We also use this configuration to measure the Intel® ISA-L erasure coding library using our FTA method. Last, we measure the bandwidth using the FTA method with the Gibraltar library.

We measure encryption using standard AES libraries as a baseline. We then measure encryption using our implementation of AES on CUDA® that we integrated with the Gibraltar library. We compare these results to show that performing encryption with erasure coding on the GPU is more performant than standard methods.

To measure the effect of lazy erasure repair on MTTDL, we use data provided by Backblaze.com to develop stochastic models of disk failure. We then simulate the impact of disk failures based on these models on several configurations to measure the MTTDL. We simulate a standard configuration using immediate erasure repair to establish a baseline. We compare the baseline with our implementation where we delay repair and measure the MTTDL.

1.5 Contributions

The contributions of this dissertation include the following:

- NLDA Architecture with EC and Encryption on data in transit.
- Erasure coding of large stripe sizes with Gibraltar [22, 26–28] on FTA.
- Combining AES encryption with Gibraltar erasure coding library [22, 26–28].
- Provide a plausible example of a lazy repair policy.

1.6 Broader Impacts

This dissertation should have impact on HPC professionals and researchers that are interested in investigation new approaches to large data storage systems, use of GPU acceleration for erasure coding and encryption for data storage systems, and cost reduction strategies for data storage. Storage system developers and early adopters will also find this dissertation interesting in that it may provide an intriguing approach to low cost, high performance archive storage and reduce the demands on longer term archive storage systems.

1.7 Outline

The remainder of the dissertation is organized as follows: Chapter 2 presents background information on HPC file systems, HPC archive systems, object file systems, erasure coding, and encryption. Chapter 3 describes practical and theoretical principles concerning nearline archive disk storage systems. Chapter 4 provides the detail design of our NLDA, experimental design and the evaluation of the results, while in Chapter 5 we give our conclusions and discuss the future work.

CHAPTER 2

BACKGROUND

2.1 Introduction

Persistent storage and sharing of data have been a problem in computer science from the beginnings. Data sizes have grown due to advances in sensor technology, record keeping in healthcare, business, science, government, the invention of social media services such as Twitter, high definition cameras on smart phones, email, telephone messages, high resolution imaging and many other reasons. Commodity storage costs have declined from $1/MB for rotating disk storage in 1994 to less than $0.03/GB today. Processing power has increased from one million instructions per second in the 1980's to over two billion instructions per second on many cores today. The methods for writing and reading the information stored on media have changed from proprietary interfaces early in the digital age to interfaces based on standards such as the ST-506 standard based on the Seagate ST-506 disk drive, Small Computer System Interface (SCSI), Integrated Device Electronics/Advanced Technology Attachment and the Serial Attached SCSI [29, 30]. Media is available as hybrid electromechanical devices which include tape and rotating disk drive using magnetic oxides and more recently on solid state devices that use NAND flash memory [31].

But the persistent storage of data should have a high-level abstraction for application programs, availability, integrity, access control and performance according to the requirements of the task at hand. Locally attached media provides storage that is directly accessible to programs running on the computer to which the media is attached. Distributed storage is made available to one or more computers by systems that provide

storage services over a communications medium. The Network File System by Sun Microsystems is an example of such a storage service [32]. Storage Area Networks [33] and Network Attached Storage [34] are other technologies for connecting storage to compute systems to provide applications with access to their data. These types of problems have been the interest of distributed file systems research for several decades and have motivated a large number of contributions to the computer science literature [35–51]. Recent needs to store and process "Big Data" has stimulated research and development in high performance distributed file systems e.g., Lustre File System [52], Hadoop Distributed File System [47] and Ceph High-Performance Distributed File System [53].

The world is moving towards exascale computing which is putting pressure on high performance distributed storage systems to keep up. One group of researchers at the DKRZ in Germany have provided a detailed analysis of the challenges to keep pace with the exascale march in the storage area. In their publication they forecast that their storage requirements will increase 300 times from their 2015 capacity to their exascale capacity by 2020 while the compute capability increases by a much higher factor. At the same time, the power budget for storage will increase 14 to 50 times where storage is 25% of the compute power consumption [54].

2.2 Foundations of Computer File Systems

2.2.1 File Standards

A required feature of today's operating systems include standard interfaces to stored data which are referred to as files. Indeed, the UNIX® and Plan 9 operating systems treat all interfaces for programs as files. In 1985 Richard Stallman recommended to the IEEE P1003 group to name the standard for UNIX® type operating systems POSIX® for Portable Operating System Interface standards [55]. The first release of the standard was in 1988 and now it has evolved into the IEEE Std. 1003.1, 2013 also known as POSIX®.1. This standard includes specifications for file and directory operations which

are concerns of this research [56].

However, the POSIX® file interface presents a performance problem with the hierarchical directory tree that serializes operations to the tree data structure including add, remove and update operations. Ceph, Lustre, and GPFS, among others, have addressed the serialization for metadata management for the POSIX® file system by providing multiple metadata servers allowing the workload to be distributed when there are no data dependencies. As long as the load on the file system is spread out over the hierarchical directory tree, sub-trees can be handed off to the other members of the metadata servers to balance workload. The redistribution of the work comes at a cost, however, because locality is often lost and the cost of management and process migration are also expensive [49, 53, 57]. An approach to providing intelligent management of the Ceph metadata services has been developed that uses the domain specific language, Mantle, to write policies which program the Ceph Metadata Server load balancing mechanism [58].

2.2.2 File System Abstractions

In the domain of high performance computing the principle media for storing data are high speed cache memory devices, random access memory, non-volatile memory devices, solid state disk devices, rotating disk devices and tape. The later media, tape, is beginning to be overtaken by rotating disk but is still a large budget item in high performance computing centers where it is supported by expensive robotics [2, 54, 59]. In the beginning of file system development these media were treated as blocks of n bytes, usually 512 bytes or 1024 bytes in the interfaces to the lower-level drivers that put to or get from the kernel services that managed the medium.

In the early 1990's researchers proposed a higher-level abstraction for interfacing to these medium. There are two abstractions for file systems to persist data to these medium in high performance systems: block and object. Object storage systems were presented in research on network attached storage devices by Gobioff in 1997 [39]

followed by a more complete concept of object storage by other researchers [40, 60]. The object file abstraction has been used in important high performance file systems including GPFS [49], Panasas Parallel File System [50], Lustre [52], Ceph [53] and others. However, the Google File System (GFS) continues to use the block abstraction because of the more simple design and the decision to use block sizes of 64 MB [48]. GFS is used successfully in Map Reduce type computation at Google similar to the way it is done in Hadoop [47, 61] where Hadoop Distributed File System (HDFS) uses 128 MB blocks for the chunks of files that are replicated across the storage nodes of the cluster.

The introduction of the object abstraction to the storage medium has been very important. Researchers have compared the typical file system layers and functions between block based IO and Object based IO [1]. The illustration in Figure 2.1 shows that the interface to the media has been pushed down into the object module. The interface for the file system has a higher-level of abstraction relieving the complexities of managing blocks on many types of devices on many types of operating systems from the higher-level file API. The object abstraction is an important feature to support high performance file systems. Object based file systems today implement interfaces as defined in the SNIA and ANSI-T10 standards [62].

Another important performance improvement for high performance parallel file systems is the separation of the concerns of metadata and file data storage functions. The former requires many small atomic operations to be performed to record the state of every file that is stored in the file system where the latter is, in most cases, a single write of the contents of the file and many reads of the content having a higher ratio of data to calls to the file system. Attributes that are stored in the metadata for each file include the location the data are stored, the creation time, the modification time, the owner, access permissions, etc. Mutual exclusion must also be maintained when modifying file data which requires that a lock service be provided. Figure 2.2 illustrates the separation of concerns of metadata and file data in a typical distributed file system using object storage.

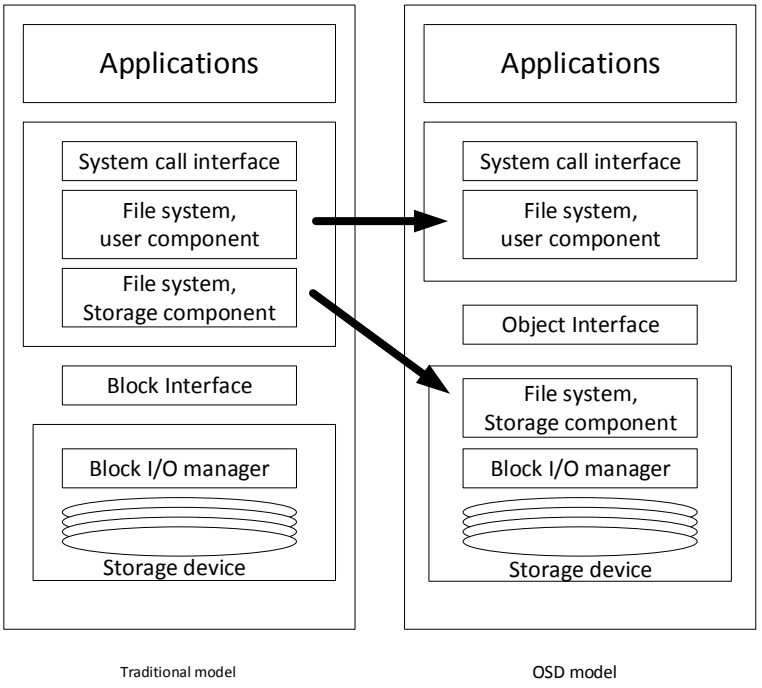

FIGURE 2.1: Conventional block based file systems compared to object based file systems [1].

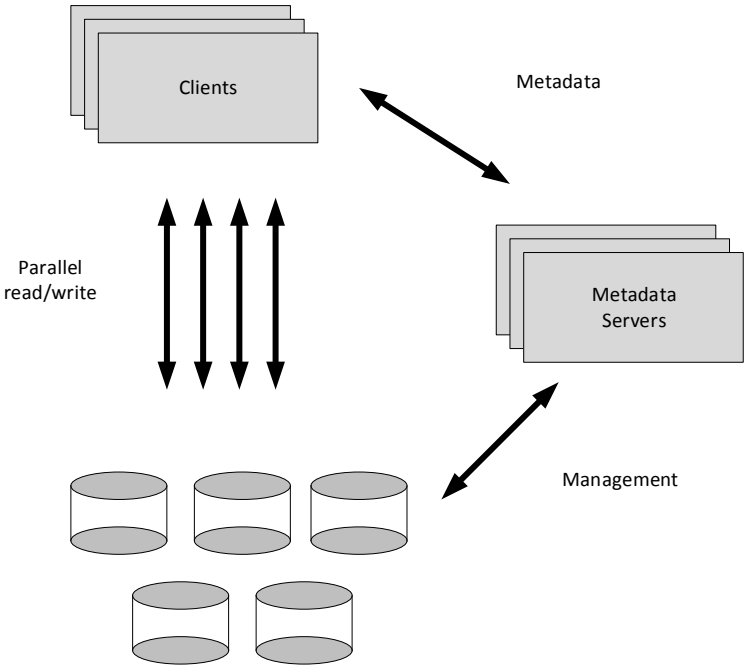

FIGURE 2.2: Distributed file system using Object Storage Devices [1] [2].

But the cost of performance and complexity required to maintain consistency of the metadata due to synchronization which is required to have strong ACID properties can be relaxed in many use cases. In these cases where the main requirements are access control, reliability, high performance and lowest cost, the object interface along with additional features is sufficient. Amazon Web Services introduced S3 [63] as a member of their cloud product suite and the OpenStack group has developed Swift [64, 65]. The OpenStack Nebula product includes a simple object based interface for storage named Cumulus [66] and Microsoft provides a product named Azure Blob [67]. All of these provide a simple application programming interface (API) which provides strong access controls, allows for unlimited file size, user choices of availability, user specified attributes as key-value pairs, etc. The API provides the concept of an *account* to which the user must authenticate. Objects are stored in the service using a *name* which is a concatenation of the *account* and a string representing the user identification of the object which can look like a UNIX® path, e.g., data/experiment-1.

Other works to address the problem of many parallel instances writing output products to a single file have been studied and implemented. Early solutions include Vesta and the IBM AIX® Parallel I/O File System product [44, 68, 69] and more recent work has been done with the HDF5 file system in the Department of Energy (DoE) Fast Forward Storage and I/O Project [70, 71]. These approaches treat files as multiple segments instead of a single stream as in POSIX®, much like columns of a matrix which fits nicely into these structures. Combining multiple segment file systems with distributed file system components can enable N to one write performance that approaches N to N performance.

2.3 Distributed File Systems

2.3.1 Coda

Coda was developed at Carnegie Mellon based on five years of research on the Andrew File System (AFS) [35] and retains some main features including scalability, a few trusted servers with many clients, client caching of files, token based authentication and end-to-end encryption [36]. The authors wanted to address resilience and to standardize the file interface [37].

Coda introduced replication of files among the servers to mitigate the threat of network failures or server failures. It also includes the feature of caching files that a user has recently opened on each workstation which also provides some protection against network or server failures within a short window of time. When a file with mutations is closed, the client writes the entire file to all servers holding a replica. This reduces the work burden on the servers and uses the client to perform the replication work.

Coda implements a POSIX® compatible file interface using the Coda driver. All files stored on the Coda file servers are seen under the global mount point which is /coda on Unix operating systems. Metadata is also cached to each workstation and is stored as files on the Coda servers. File creation and updates are passed immediately to the servers. The Coda servers implement a callback cache coherency protocol to invalidate cached artifacts on workstations when an update occurs so that the next use of the file will require the client getting a fresh copy of the file.

The main goals of Coda are availability and providing simple file sharing services for a large group of Unix clients. The major performance benefit is provided by the local caching of files and metadata on each workstation under the assumption that the read to write ratio is high and that there is not a great deal of write conflicts among users. Coda implements parallel communication between clients and servers using multicast in the form of MultiRPC [72].

2.3.2 Swift

Swift [38] was the first experiment in distributing chunks of data from a file across multiple storage nodes over a network. The goal of Swift was to use parallelism to enable I/O to provide data flow rates to match the needs of applications. When applications are waiting for data due to the limit on the maximum transfer rate of the storage devices (i.e., disks, tapes), then nearly linear scaling can be achieved by striping file data across multiple nodes and reading or writing in parallel, a scatter and gather approach. In Swift, objects are stored by *storage agents*. Clients interact with *storage mediators* which generate a *transfer plan* that describes where the data is stored or will be stored. Clients use the *transfer plan* to interact with the *distribution agent* to carry out the operation. In Swift, the *striping unit* is determined by the *storage mediator* according to the bandwidth SLA with the client. Other responsibilities of the *storage mediator* include the storage of cryptographic keys for authentication and access control with the *storage agents*, the configuration that determines replication levels or erasure coding. (Note: this is not the same product as OpenStack Swift [64]).

Swift included the concepts for resilience in the prototype design [38]. In a later work, the author demonstrated an implementation of Swift that provided distributed a RAID levels 0, 4 and 5 using XOR parity [73]. The Swift library provides an application interface that presents a set of UNIX® file system operations to the client. The client is not aware of the underlying implementation of the storage mechanism. The *storage mediator* maintains metadata in memory and stores a copy within the persistent storage system for restart and recovery. Persistence of metadata must be fault tolerant, loss of metadata will result in the loss of the capability to read or mutate existing files. The Swift file system was able to demonstrate near linear scaling for reads and N to N writes.

2.3.3 NASD

NASD provided important research work in distributed file systems which led to the standardization of the object storage systems[39]. The main problem that the authors

were trying to solve was to store data over an untrusted network using network attached storage devices. These prototype devices provided access controls to data objects stored by using a central authentication service and a ticket system. Clients could then use the tickets obtained from the central authentication service to prove their authorization to use the resource. A capabilities architecture was included in the design of NASD which included a capability key and capability arguments to convey all of the information necessary to grant a request implemented as a cryptographic function. The NASD enabled reliability by allowing clients to replicate data across storage devices on the network. The research provided a proof of concept for storing data in an untrusted network. Prototype file systems were implemented to demonstrate the capability and performance using NFS and AFS. NASD is the fore runner to more modern object based parallel file systems which continue to use the capability architecture and the network attached object storage agents. The research showed that the I/O scaled linearly as disks were added to the network but the overhead of the parallel file system limited the performance [40].

2.3.4 zFS

zFS is an extension of an earlier IBM distributed file system, DFS [41] extending it to use object storage. zFS was designed to scale to thousands of network clients and to be built from commodity components. zFS contributes a cooperative cache and distributed transactions for atomic metadata updates. The goals of zFS are high performance with near linear scaling, off the shelf components and a distributed global cache. zFS achieves scalability by separating the control and data concerns. Storage management is encapsulated in the Object Storage Devices (OSDs) freeing the file system from the storage management details. zFS stores files and directory objects in the OSDs. zFS provides a *Lease Manager* which provides a locking service with timeouts to provide atomicity and protect data integrity. A POSIX® compliant interface is provided by providing the hierarchical tree directory services. The metadata for the directory

is stored in its own objects in the OSDs. Atomicity in directory operations is attained through the use of a *Transaction Server*. zFS keeps recently used pages in local caches on nodes in the cluster and uses a cached copy to fill requests for any node in the cluster avoiding a read from disk [42].

zFS depends on the underlying Object Storage Service to manage striping and resilience mechanism. zFS presents a POSIX® compatible interface. zFS provides a hierarchical tree directory service that is stored in memory for speed and journaled to the object store for resilience. All operations on the directory structure are handled by the *Transaction Server* which obtains all leases required for the transaction. If a failure occurs during the transaction, the state is rolled back to the last consistent state. zFS performs on par with other distributed file systems in most cases. Because of its strong POSIX® compliance it does not perform as well for the N to one write case as do other file systems that relax POSIX® compliance [42]. (Note: The OpenZFS product which derived from the Sun Microsystems ZFS product is not a parallel file system. It is a host based file system, although it does provide for software RAID within the file system.)

2.3.5 pNFS

pNFS is an extension to NFSv4 that adds parallel concepts. In the base NFS design there is a single point of mediation for all file operations between the client and server. pNFS separates the metadata and data concerns. The metadata concerns are still between the client and the NFSv4 server but the data concerns are between the client and the data servers which can be distributed. pNFS adds a class of *layout drivers* that understands how to access the blocks of data on the storage servers. pNFS can be implemented on any of the many types of systems that support NFSv4 [43]. Currently pNFS is a part of the NFSv4.1 specification with published RFC specifications for block [74, 75] and object storage interfaces [76].

The Block specification of pNFS does not address resilience of the stored state on the underlying disk storage, this must be provided by RAID and multi-path mechanism

that are typical for NFS file servers. The Object specification of pNFS does address the capability of OSDs to provide resilience using replication, RAID and erasure coding with Reed-Solomon [77] encoding scheme [76]. pNFS provides protocols for network file services and requires a top-level file system to provide the file interfaces. pNFS adds the layout driver layer which provides an abstraction to various configurations of block level access in the storage network. pNFS provides the same single mediation access to metadata that is provided by NFSv4. Concepts around the parallel distribution of data are provided by the top-level file system interface that is provided.

pNFS provides increased performance by separating the control and data concerns. Control operations are mediated by the single filer as in standard NFS but the system can be configured to use many data servers. Because the flow of data are usually much greater than the flow of control information, the distribution of data flow over many data servers can provide an increase in performance over the standard NFS single mediation configuration.

2.3.6 Vesta

Vesta is a distributed parallel file system that is focused on short to medium term persistent storage for parallel applications. Data are distributed across multiple I/O nodes to allow for parallel execution of input and output processing. Vesta extends the file as a sequence of bytes into a sequence of segments. These segments are distributed among a specified number of I/O nodes using a hashing algorithm [44].

Vesta does not provide any resilience mechanisms, it depends on the underlying I/O node to provide this feature using lower-level storage systems such as provided by SAN or Direct attached RAID devices [44]. Vesta presents its own interface which allows applications to configure and control parallelism for optimal performance. It also provides a generic POSIX® wrapper that provides a basic configuration of its interface to allow for applications to use the file system without adding the additional complexity. The Vesta file interface is similar to object storage in that the management

of the persistence is pushed to the I/O nodes. It includes a concept of two dimensional files which are compatible with the storage of matrices in rows and columns. Vesta provides subfiles and cells which abstract the storage from the application. Subfiles are file partitions that are usually accessed by the different members of a parallel compute process and cells are the segments that subfiles are partitioned in to based on the Basic Striping Unit (BSU) [44].

2.3.7 Parallel Virtual File System (PVFS)

The Parallel Virtual File system provides fast IO to compute clients of an multi-computer cluster by striping files across many IO nodes. Metadata about files stored in PVFS are managed by a single node which include properties such as owner, access control, etc. This metadata is stored in regular PVFS files that are striped across the cluster. PVFS presents a native API, POSIX® and MPI-IO API to users for their applications. The current version, PVFS2 provides an object interface to storage [45, 46]. PVFS is able to provide near linear scaling for read and write performance. Data reliability depends on the underlying object storage provider.

2.3.8 Hadoop Distributed File System (HDFS)

The Hadoop file system was developed at Yahoo and contributed to the Apache Software Foundation. Hadoop is packaged with a suite of other components that enable it to perform MapReduce style applications with optimum performance. HDFS distributes data in 128 MB blocks across *DataNodes* and uses replication for data durability. Metadata is managed by a *NameNode* which keeps an in memory copy of all of the files, various properties, and the DataNodes on which the blocks are stored. The NameNode stores a data structure on disk that contains the metadata of the Hadoop file system and writes a journal when changes are made.

The key strength of the Hadoop Distributed File System is the feature of reporting the DataNodes where data are stored to applications. Applications can schedule their

computation on the DataNode that has a copy of the data. When an application writes data, it contacts the NameNode and receives a list of DataNodes where the data are to be stored. The client application then sets up a pipeline and writes data to the first node in the list. The second node receives a replica from the DataNode that received the first write block. This pipeline of operations is continued until the data are replicated across the complete list of DataNodes that were provided by the NameNode. Hadoop clusters including their Hadoop Distributed File System component can scale to large numbers of servers. One researcher from Yahoo has reported that they have Hadoop clusters that contain over 25,000 nodes which have over 25 PB of data stored. The largest Hadoop cluster they operate is 3,500 nodes [47].

2.3.9 Lustre

Lustre is a scalable, high performance, parallel, distributed file system that evolved from research on the Coda project at Carnegie Mellon [37]. The name is derived from Linux Cluster and it is an object based storage system. The major components of Lustre are the clients, the Object Storage Targets called OST and the Metadata server (MS). The clients run the Lustre file system. The MS in Lustre is a performance limiting hot spot but clients read and write directly to the OSTs which are distributed across all of the storage servers in the cluster each of which is referred to as an OSS [78].

Lustre does not provide resilience at the application but depends on the underlying storage subsystem. It is very important to store metadata and data on storage that is configured for high availability in Lustre. Storage for metadata should be on fast components such as RAID1 or RAID10. In addition, due to the MTTR of today's high capacity disks (greater than 1TB) one parity disk in RAID is no longer sufficient to protect against a second failure during the reconstruction of a failed disk. For the data storage RAID6 or methods that provide more redundancy should be provided. A recommended configuration for data storage is RAID6 with six data disks and two parity disks [78].

Lustre presents a POSIX® compatible file system interface and has a loadable device driver for the operating systems that it supports. Lustre separates the concerns of metadata from file data and provides metadata services via the Metadata Server (MS). Lustre stores metadata on file systems that are mounted on the MS. Loss of metadata will result in loss to the underlying file data because the system will no longer be able to look up a path and filename and provide the information describing where the data are stored.

Lustre is a high performance file system and has a great number of small to large cluster installations. The performance is derived from striping data across many Object Storage Servers (OSS) which contain one to many Object Storage Targets (OST). Lustre stores data in objects so that it does not have to manage the details of data storage to the underlying devices. The performance is gained by adding the data bandwidth of the number of stripes that are configured for the specific file.

2.3.10 GFS: Google File System

The Google File System was designed and implemented by Google to address their specific needs for reading and writing very large amounts of chronologically ordered data that are organized into records. This property is characterized by most file mutation occurring by appends to files. The only constraint they have is that each writer is able to complete the append of the record atomically. Because of these relaxed requirements, the GFS does not implement a standard file system API. GFS does organize files hierarchically and use path names for identification. GFS supports operations like *create, delete, open, close, read* and *write* which are usually found in standard file APIs such as POSIX®. They have also added a *snapshot* operation and a *record append* operation. Through a lease mechanism along with the specialized *record append* operation, GFS is able to provide strong ACID properties. The architecture consists of a single *master* and many *chunk servers* and *clients*. The master does not store state but constructs metadata on startup by chatting with chunk servers. Changes to metadata are written to a log on the local master disk with a replica stored on another chunk server to

protect in the event of a system failure. The master stores metadata in memory to provide fast metadata operations for clients [48]

GFS provides resilience by

1. replicating chunks over chunk servers,
2. check summing over chunks and
3. writing metadata to multiple copies of a transaction log.

The GFS interface is non-standard but offers all of the POSIX® operation along with *snapshot* and *record append*. These mechanisms provide a greatly simplified ordering of mutations from multiple clients which results in speed and correctness. The *snapshot* operation allows an application to quickly make a point-in-time checkpoint of a file for consistency. GFS maps from a PATH/filename name space to a list of chunks that are stored on chunk servers. Chunks are 64 MB blocks that are allocated as needed by a dialog between the master and the chunk servers. The master maintains an even distribution of chunks over the set of chunk servers which include replicas. Changes to metadata are written to an operation log which is replicated to chunk servers for resilience. Operations to metadata are serialized through a locking mechanism. Performance is achieved by storing metadata in memory data structures and using an efficient lease mechanism. Only control communications occur between master, clients and chunk servers, data flows are between clients and chunk servers.

The performance of GFS is excellent cases of N to one and N to N because of the fast metadata operations and distributed data. If more parallelism is required, the number of replicas can be increased. The number of metadata operations and the size of the metadata data structures are reduced by having 64 MB chunk sizes. The record append case provides write performance for multiple writers in the N to one case that approaches that of a single writer to a file.

2.3.11 GPFS: IBM's General Parallel File System

GPFS, General Parallel File System, which is also known as Spectrum Scale was developed by IBM [49, 79]. GPFS provides a POSIX® file system interface to clients. The interface provides byte-range locking and distributed lock management based on tokens with journaling. GPFS also provides a MapReduce interface for Hadoop applications and implements the OpenStack Object Store Swift interface for storing and retrieving large object data using the simple application programming interface. GPFS provides for high scalability and performance by distributing metadata management across the storage nodes in the cluster. Reliability is provided by erasure coding providing greater than two parity strips per stripe, typical configurations have stripes containing six data strips and four parity strips to survive the loss of up to four strips. GPFS provides scalability to thousands of storage nodes and linear bandwidth scaling for parallel reads. Distributed locking and metadata management helps N to one writes scale.

2.3.12 Panasas

The Panasas file system provides a POSIX® file system on a distributed cluster of object storage nodes enabling good performance for high performance computing. Panasas provides the file system as a kernel driver for Linux operating systems. It also provides an NFS and CIFS file system for use by clients that cannot use the Panasas FS components. Panasas consists of object storage servers which provide and object interface to storage on the server consisting of off the shelf disk drives. These servers manage the data on the disk using a specialized OSDFS which provides the essential services to store data on the disk sectors and provides high performance. Panasas FS provides reliability with replication and RAID5. The stripes for the reliable storage are configured as pools across the storage servers and the parity for each stripe is computed by the client when writing data. Parity computation is performed on a per file basis which allows the Metadata Server to handle the reconstruction of parity when an OSD fails. By striping data over multiple storage servers and distributing parity computation

to the clients, Panasas is able to provide excellent performance with linear scaling of bandwidth [50, 51].

2.3.13 Ceph

Ceph is a distributed high performance file system that decouples metadata from data and provides a deterministic function for mapping metadata to data location, CRUSH – Controlled Replication Under Scalable Hashing [80]. Ceph is an object storage system that uses peer to peer sharing of a compact hierarchical description of the cluster configuration and a replication policy. This innovation distributes the computation to determine replica placement to any member of the cluster, including clients and removing the serialization of determining data placement from a centralized metadata service. The CRUSH algorithm uses rule sets to define policies on data placement and result in evenly distributed storage of data across all of the OSDs in the cluster. The rules enforce availability policies, for example, replicas must not be in the same rack or other defined failure domain in the data center.

Ceph implements the data storage layer of file systems with the library librados which exposes an interface to the Ceph object store. Typical block based file systems can access the Ceph cluster object storage via Reliable Autonomic Distributed Object Store (RADOS) Block Device, a driver for Linux kernels based on librbd [81]. The library librbd provides an effective solution for legacy file systems to take advantage of the Ceph storage cluster in much the same way that a block based interface is provided by other SAN technologies. Because legacy file systems store metadata and data together, using librbd for this purpose does not provide any significant performance benefits. Recent research at CERN has provided performance data for Ceph at scale to 30PB [82].

Ceph has also included CephFS as a feature in the product which provides a POSIX® compatible, high performance distributed file system. CephFS provides an additional service, the Metadata Service (MS), which provides the POSIX® compatible file name space features as well as the management of atomicity for operations, i.e., file creation,

file deletion, file renaming, attribute changes, permissions, locks, etc. CephFS consults the MS to provide the client with the layout for the file that is being operated upon. The layout informs the client where the file data is stored in the cluster. The layout specifies the number of strips in a stripe of the file where the stripe spans a number of objects that are configured. This feature distributes the client data access for reading and writing across as many of the Object Storage Devices (OSDs) that are necessary to achieve the performance objective. Ceph MS stores metadata in the Ceph Object Storage Cluster. Configuring the metadata storage pool on fast devices with with replication provides the best performance and reliability. Currently Ceph storage pools containing solid state devices can provide the fastest performance although at higher cost than traditional disk devices. Because metadata is typically less than 2% [78] of the size of the data stored, this cost amortization may be justifiable. Ceph also provides a Hadoop File System, HDFS, as a a plugin to CephFS.

CephFS provides a separation of metadata and file data concerns. Metadata services for CephFS are provided by the Ceph MS component. Metadata are stored in a dedicated Ceph storage pool to provide fast and resilient services. The MS enables a POSIX® compatible file system interface and enables a true distributed file system [83]. CephFS is nearing production quality and will provide multiple MS nodes to distribute metadata services across the cluster which will increase scalability and reduce metadata operation hot spots. Recent research by Michael Sevilla, *et al.* has demonstrated a kernel plugin system for Ceph metadata operations which implements agents to schedule MS workload over a pool of MS nodes according to several properties. This work may provide optimal metadata services for Ceph which will improve the scalability [58].

2.4 Tape and Virtual Tape Archive Systems

Magnetic tape storage has been a part of the information technology landscape for nearly 70 years [84]. IBM announced their IBM Model 726 Tape Unit in 1952 to provide an alternative to punched cards for data storage. Beginning in 1997, the storage industry

established the Linear Tape-Open standard (LTO) which has dominated the magnetic tape for archive systems ever since. The current generation of magnetic tape is the LT0-8 which has a raw capacity of 14 TB and a bandwidth of nearly 400 MB/s [85, 86]. NERSC has recently added a third tape library and currently stores 180 PB, growing at the rate of 3 PB/month [87].

Another archive technology that emerged around 2000 was the Virtual Tape Server (VTS) which presented an interface to disk storage systems that looked like the tape archive systems that had dominated the landscape. IBM announced their IBM Virtual Tape Server in 1952 that would emulate up to 256 tape drives on their direct access storage device (DASD). VTS which the research authors call TLFS is shown to be faster than tape libraries [88]. Also, a hybrid disk and tape archive system has been proposed which provides an easier to use interface for the users, provides the performance of disk storage and the economics of tape storage [89].

2.5 Modeling Storage Systems

Modeling and measuring the performance of computer systems has been important throughout their existence. IBM published a book in the System Programming Series in 1978 on the methodology of evaluating system performance [90]. We have found some research that focused on modeling and performance of real-time systems [91]. Integer Programming was used for finding the optimal location to store shared files among computers that were linked together based on the minimum cost of storage and transmission [92].

Recent research has been published on methods for the modeling of storage systems using Integer Programming. A feasibility study for using a Ceph storage system as an archive tier for the University of Aukland. The authors used Integer Programming to determine if the product could provide the minimal cost and meet the capacity and bandwidth requirements of the University [93]. They used their institution cost rates for operating expenses in the model including rack space costs and power consumption

TABLE 2.1: Parameters Used for Archive Model [10]

Parameter	Value	CAGR
Initial Capacity [95]	1 PB	30%
Minimum Read Bandwidth	100 MB/s	30%
Minimum Write Bandwidth	100 MB/s	30%
Cost of Electricity [96]	$0.11/kwh	1.3%
Minimum Data Read Yearly [97]	3-75%	-
Data Scrubbing	Monthly or Annually	-
Length of Simulation	25 years	-

with functions based on the U size of the equipment and the wattage consumed by the equipment respectively. They included a scalar variable for the number of years of operations in these functions. A subsequent research paper was published by the same group about their experience with bench marking a storage system and comparing it with the Integer Programming model [94].

A recent study about the performance, cost and reliability of various media for archive storage was done using simulation. The authors developed a model based on several parameters and ran the simulation over 25 years making assumptions about the *compound annual growth rate* (CAGR) and the technology progress of the various media [10]. The parameters they used are shown in Table 2.1. Their model was informed by their research in working archive workload studies. They developed the parameters by analyzing the cost of the selected parameter of a multi-year period to determine the CAGR. Their model consists of two parts: 1) the Storage Group Class, and 2) an Archive Class. The Storage Group Class consists of the Archival System Components which include media, drives, robotics, enclosures, and networking. This class stores the configuration facts of the Storage Groups along with lifetime attributes to aid in determining when the component has reached its useful life or has failed. The Archive Class aggregates the Storage Group Classes and runs the steps of the simulation over the time of the experiment, 25 years in the paper's example. The simulation begins with a one PB archive and grows it to 25 PB at the rate of one PB per year. Storage

components are replaced as they fail and upgraded to newer technologies when the earlier components are no longer supported. The projected costs of the components are used at the price/performance expected at the time in the future when they are brought into the model. Assumptions were made that caused the cost of hard disk technology to be projected at a much lower cost in the future which seems to contradict their argument that the CAGR of hard disk storage has not met the prediction of Kryder's Law [98, 99]. They also made assumptions that the cost of SSD technology would decrease at the highest rate. The media lifetimes were given as Tape: 30 years, Optical Disc: 50 years, HDD: 10 years, and SSD: 10 years. They presented an argument that SSDs would actually last 15 years as archival storage since the data are Write Once, Read Many (WORM). The bottom line of their analysis shows that tape, optical, and disk will have about the same total cost of ownership (TCO) over the 25 year simulation. The study showed that disk will use significantly more electrical power than the other media within this TCO. The TCO for SSD systems was over 2.5 times the other media included in the simulation. Under the constraint that energy consumption should be minimized in modern data centers, then it would be prudent to use smaller amounts of disk storage and more tape storage in the archive over the next 25 years. At some time in the future, the SSD technology may become the most economical and would become the storage medium of choice for archive.

We found one study that analyzed the cost of erasure coding for exascale storage [100]. They chose to emphasize erasure coding on the OSSs in order to relieve clients from the load of the erasure coding computation. In the study, they model the component costs of this traditional server-based erasure coding scheme in detail which provides an excellent understanding of the data transfer costs and the computational burden on the OSSs. In another study the authors extended an analytical model which studied the reliability of RAID 5 in order to understand the impact of irrecoverable read errors (IREs) on RAID 6 systems [101]. They used Markov Chains to compute the probability that the read of a third data chunk would fail in a RAID 6 reconstruction due

to an IRE. They concluded that declustered RAID 6 will increase the Mean Time To Data Loss MTTDL due to IREs by a factor of 150 over standard RAID 6. They showed that the MTTDL increases by a factor of two when scrubbing is performed annually and that daily scrubbing will increase the MTTDL by a factor between 27 and 170 over the case where data are not scrubbed. The conclusions were based on the assumption that data parity would be recomputed when an error was detected by scrubbing and that the rebuild would occur within a few seconds. While these researchers used the term IRE to refer to these types of errors, other literature uses the term unrecoverable read error (URE). We will use the appropriate term depending on the context.

A group of researchers have developed a tool to model the I/O path for parallel file systems that uses a functional representation [102]. The tool has been helpful for analyzing the functional components of file systems especially for determining where to insert instrumentation for monitoring performance. The tool also helps in identifying bottle necks and other constraints. The tool was developed in the style of The Unified Modeling Language (UML) [103], the Model-Driven Architecture (MDA) [104], Systems Analysis and Design (SAD) [105], and The Systems Modeling Language (SysML) [106]. The modeling language represents logical functions as nodes and the interactions between logical functions as edges. There are annotations that can be applied to the model to inform the user about the logical function or the interaction. For example, the name of a file would be a node, the persistent data structure that stores facts about the file, an inode, would be represented as a node, and the linkage between the file name and the inode would be shown as an edge. The modeling language includes hints such as dashed lines to represent redundancy.

In another study the authors presented their analysis of the expenses for building and operating exascale storage systems [54]. The study identified the requirements for operating the DKRZ facility when the processing capacity reached exascale circa 2025. They built models based on the past growth rates of various components of concern for data storage including the data volumes, data velocity, and the long term preservation of

data. This analysis enabled them to identify bottle necks that would need attention in order to operate the data center at that scale. A fact that became very obvious in their study was that compute performance was increasing at a 20x rate while storage capacity was only increasing at a 5x rate. This observation uncovered the storage crisis which required solutions. They further analyzed techniques to reduce storage requirements using: recomputing results, data deduplication, and data compression. Recomputing results is challenging because results are dependent to the machine and software that was used for the original answers. Changing software libraries and hardware will likely make it impossible to recompute results. Data deduplication is not so useful for scientific applications but could be used as a tool to identify users that keep multiple copies of the same artifacts in storage. This problem could be addressed with user education. Data compression provides promise to reducing the TCO for storage but may best be left to the users. Selection of the compression method will need the domain expert's input to determine where lossy compression will not affect the quality of the data products.

Los Alamos National Laboratory (LANL) performed developed several models using linear programming and several different solver tools to determine the optimal mix of nearline disk archive storage and tape. Their findings were not as expected. Two models that were homogeneous disk based archive and homogeneous tape based archive were both much more expensive, on the order of $35M through 2026. However when they combined the two models, they ended up with an investment of $14M. They reasoned that the bandwidth constraints drove up the costs of the tape only solution due to the high cost of robotics and tape drives while this constraint also caused an over provisioning of disk drives. By mixing the models, the combined bandwidth was sufficient to allow the disk provisioning and tape provisioning to track closer to their estimated data storage requirements. Also, the cost of storing data on tape is much lower than the cost of providing power to disk drives and the added maintenance for the number of disk drives to satisfy their complete data archive requirements. The modeling included the resources to migrate data from old tapes to newer, higher capacity tapes [59].

2.6 Storage System Reliability

Protection of data stored in data systems is a key requirement. Users demand that storage providers are able to deliver their data to them when they ask for it. SLAs may specify the measurement of determining success for meeting this requirement such as "five 9's" which means with a likelihood of success of 99.999% [107].

Since the redundant array of inexpensive disks was introduced in 1988 that made systems more resilient against data loss, research has continued to provide more techniques for improving availability of data and improving performance [108]. The principle methods for mitigating the loss of data due to media or system failure has been replication, RAID and erasure coding. The design choices between these methods must be balanced between the higher cost of storage for replication of n x r where n is the size of the data and r is the number of replicas plus one or the computational cost of parity generation for RAID and erasure coding. Of course the trivial case of RAID are no replicas or one replica also known as striping and mirroring respectively.

Erasure coding provides a higher degree of durability, i.e., the storage system can survive the loss of a greater number of disks, using less additional storage than replication [109, 110]. The types of erasure codes that we study are called maximum-distance separable (MDS) [25] codes which can provide for recovery as long as K shards are available out of $K + M$ previously computed shards. We use K to refer to the number of shards we divide a block of data into in order to produce a stripe of data for storage. We also compute M checksum shards using an erasure coding algorithm that is MDS that we store by concatenating to the K shards producing a stripe that has $N = K + M$ shards. This stripe will have a storage efficiency ratio of $\frac{N}{K}$. Erasure coding can provide a higher order of redundancy by generating more than two parity disks has been heavily studied by James Plank [111, 112, 112–115]. Matthew Curry showed how erasure codes could be computed and decoded using GPUs [22, 26–28]. Another consideration for data resilience is locality. Storage subsystems that replicate data or store parity on direct

attached medium can provide data storage services at a lower communications cost as compared to storage systems that distribute replicas or parity throughout a set of storage nodes that are connected over a high speed network. Especially in the case of reconstruction parity for RAID 1, RAID 2 or erasure coding where the minimum set of data or parity strips must be assembled in memory to recompute the missing data or parity. Facebook studied the problem of parity reconstruction for erasure coded storage [116]. There is strong evidence that using erasure coding with commodity hardware for durability in high performance computing is more economical and faster than dedicated storage subsystems [117, 118]. Microsoft has chosen to implement the storage systems in their Azure cloud service using erasure coding [119]. A thorough treatment of performance measurement for erasure coding has been given in other research [120]. The power efficiency of erasure coding has been discussed in other research [121]. DACO is a design that proposes a scheme where remote code is executed by disk drive controllers to update parity directly on the media saving on the data transfer costs that are usually associated with updates to erasure coded stripes [122]. The DACO article provides a very interesting description of the internal workings of the hard disk drive.

One research work has been done that uses large stripe sizes to archive data at thousands of endpoints on the internet [109]. In this research, they use a combinatorial method to count the number of parity shards to include in their stripes and compute the likelihood of failure. In their discussion, they determine that the time to failure using two way replication to be 54 years but the time to failure using $K = 32$ and $M = 32$ onto 64 endpoints to be 10^{20} years. In this model they assumed that all failures were independent but gave some examples how smarter decisions could be made in their system to choose storage endpoints that were less likely to have common threats, e.g., power outage, tornadoes, earthquakes, etc.

Previous work has been done where clients performed the erasure coding and stored the stripes on the storage nodes that were on the same network [123]. They determined that using one KB shards would be the optimal configuration because erasure coding

costs increased as the shard sizes increased. They reported their performance topping out at 200 MB/s during writes from up to 60 clients using 8 storage servers. Read performance became better with the number of shards in the stripe topping out at nearly two GB/s with 32 shards and 60 clients reading. In another research project a design is presented for an erasure coded distributed storage system where clients perform the erasure coding. The authors measured an average write bandwidth of 186 MB/s and an average read bandwidth of 400 MB/s. In this paper, they reasoned using combinatorial computations to show that the likelihood of failure using replication was greater than with erasure coding [124].

One researcher has studied how using random nodes for erasure coding in a distributed system can reliably create erasure coded data stripes [125]. This article provides a very detailed mathematical argument about the storage network and the proofs that the data can be recovered as long as there are K shards that are available.

One group of researchers performed an extensive measurement of erasure coding performance [126]. They reported that erasure coding performance began to flatten out at shard sizes greater than 1,000 bytes due to the increase in cache misses but performance increased with larger packets due to more reuse of data during the coding computation, a trade off between these two resources. They also measured the difference between $GF(w^4)$ and $GF(2^{32})$ and found that it had no performance effect with their Jerasure [112] implementation as long as the CPU cache was large enough. They reported a maximum erasure coding rate of 1.2 GB/s using $RS(6,2)$ and $RS(14,2)$ but it dropped to 400 MB/s with $RS(12,4)$. Repair performance was nearly as fast as encoding since they use the same algorithm. While they reported performance for the range of Galois field sizes, we only mentioned the performance for the $GF(2^8)$ size which we are most interested in, especially since the larger sizes result in lower performance.

Storage services for high performance computing systems can be provided by storage area networks (SAN) which provide data resilience and high speed communications over specialized networks. Some of the high performance distributed file systems rely on

these types of storage providers where the responsibility for data resilience is handled by the SAN [44, 52]. These file systems can also be configured to provide resilience against the loss of data serving nodes by providing multi-path connections to the SAN storage. The SAN subsystems present storage volumes to the storage servers in the form of LUNs which are logical volumes of media blocks formed by the SAN subsystem that have the resilience properties that have been specified by the administrator. Under the threat of network or server failures file systems must consider the importance of consistency. Pessimistic replication and optimistic replication are other strategies that address these problems [127].

Lazy Recovery as a means of reducing the erasure repair workload impact on storage systems has previously been studied [128]. The authors developed a Markov chain model of MTTDL using knowledge of the reliability of the disks in their analytical system. In their work, they showed that relaxing the urgency of erasure repair in an erasure coded storage system using Reed-Solomon [77] K=10, M=4, that there should be no reduction in MTTDL but they were able to reduce the repair bandwidth by a factor of four. Research in reliability mechanisms for very large object storage systems also used lazy parity backup (LPB) to generate parity over data that was relatively static [129] after the replicas had been written to the object store.

The detection of latent failures in disk drive media, especially for large petabyte storage systems, is important for preventing data loss. A technique called "scrubbing" has been studied extensively and shown to reduce data loss [130, 131]. The system maintains a signature of each data element stored in the system which is checked during the scrubbing process by reading the data element, computing the signature and comparing it to the value originally saved. If the signature does not match, the data element is flagged for repair.

Erasure codes for cloud file systems have been studied to learn how to reduce I/O for recovery and degraded reads [132]. Degraded reads occur when a stripe needs recovery to repair erasures when the user has requested to read the data from the file system. The

study compared the impact of I/O for various erasure coding algorithms. They concluded that shard sizes should be larger and that stripe sizes of 567 MB would be more preferred over the 64 MB stripe sizes that are currently popular. They were able to achieve recovery rates of 16 MB/s with $K = 6$ and $M = 2$.

Erasure coding has also been performed on Field Programmable Gate Arrays FPGA devices [133]. The authors developed an OpenCL-Based Erasure Coding program and measured the performance on several hardware platforms. The authors found that FPGAs were able to maintain a high bandwidth as the number of shards were increased above 30. Their experiments showed that both GPU and multi-core CPU implementations were faster than accelerated processing units (APU) and CPU implementations.

Research has been done that analyzed the likelihood of losing data with the Cloud-RAID 6 implementation, a way of striping data across multiple cloud service providers [134]. They use the methods described in "Complex System Reliability", the element level coverage (ELC) and the fault level coverage (FLC) models [135], to calculate the likelihood of failure. The ELC model considers the element level failure and the FLC model considers the fault level coverage of a system. The probabilities are time based and use the assumption that failures are independent and the rate of occurrence is constant. The authors ignore the other types of media failures such as UREs because they are looking at the storage members as being provided by the service provider. Their analysis is only with the system survivability for storing data with multiple cloud service providers while using a RAID 6 configuration, i.e., each shard of a stripe is stored with a different cloud service provider. Based on several time based failure rates and recovery time rates, they use the Binary decision diagram (BDD)-based combinatorial models for their analysis. They describe a BDD as an acyclic graph which has a root, i.e., a tree data structure, in which the binary structure is based on the Shannon decomposition rule.

One study examined a design for increasing the reliability of storage systems that replicate data [136]. They state that mirroring alone can provide 99.9% reliability but because users require higher reliability, they posit that the mirroring can be supplemented

by generating erasure coded parity across the objects stored in the system, increasing reliability with a lower storage costs than having a third replica of the data. They propose to store the parity in NVRAM storage that is distributed across the system to reduce the overhead of writing to disk. Because the target storage system for this technology does not restrict data updates, i.e., the data stored is not immutable, the approach is effective. However, generating the parity requires that all of the data on the storage system must be read to compute the parity. When new data is added to the storage system, the parity has to be updated, requiring a read of the parity and the generation of the new code which includes the new data. When data are changed, the parity has to be read and updated with the changed data. One of their approaches only generates a single set of parity for each protected set of objects, the use of XOR coding is simple and practical. Combining one replica and one parity is equivalent to providing the capability of surviving the loss of two data units. Another approach was to increase the number of parity shards per stripe by using Reed-Solomon [77] encoding; since their method formed parity stripes with on the order of 3,000 objects, the use of a $GF(2^8)$ was not sufficient so they developed a $GF(2^{16})$ Reed-Solomon implementation to meet the MDS [137]. Also given in this paper was a Markov model to predict the MTTDL where they showed that their solution increased the time to failure by a factor of 4,000. The authors use a population sampling approach to predict the reliability of their approach. This method decouples the determination of failure from the complex failure of storage system elements and provides a meaningful metric within the context of the SLAs.

In another work [138, 139] the author argues that the MTTDL metric that has been in use since RAID was introduced nearly 30 years ago [40, 140] is not relevant. He gives several reasons that it is not useful and provides a list of properties that a desired metric should possess. A new metric, NOMDL_t, is provided and explained. He concludes by comparing the NOMDL_t with four other storage system reliability metrics. NOMDL_t is computed with a Monte Carlo simulation which can be performed using the High-Fidelity Reliability (HFR) Simulator. NOMDL_t does provide a relevant and meaningful metric

for determining storage reliability in terms of data loss likelihood over a given mission time for the system based on the parameters used in the Monte Carlo simulation.

Erasure coding used to mitigate lost data in storage units comes from Forward Error Correction in Algebraic Coding Theory [25, 111, 141]. John Kerl provides a very thorough review on Galois Fields by showing how these special fields are derived from integral domains and giving us the acronym "FEPUI", the meaning is shown in Equation 2.1 [141] where **PID** is a principle ideal domain and **UFD** is a unique factorization domain. Galois fields have as members, symbols, which can be determined from $\mathbb{Z}$ modulo a prime integer p. For erasure coding, we set $p = 2$ so that the members of $GF(2) = \{0, 1\}$. We then create polynomials of the form $ax^n + bx^{n-1} + \cdots + x + 1$ in the field $GF(2^n)$ where the coefficients $a, b, c, \cdots$ are from $GF(2)$. But, since $a, b, c, \cdots \in \{0, 1\}$, we usually write $x^n + x^{n-1} + \cdots + x + 1$ where the coefficient of x^n is certainly 1 and not 0, or we would not have included the term, hence, the leading coefficient of the highest degree is one, a property known as **monic**. Kerl thoroughly explains finite fields including the process of generating logarithms to perform multiplication and division (inverse multiplication) using tables, a technique that trades memory space for computation in erasure coding implementations. All of this provides us a set of positive integers that have the properties required for computing linear combinations over a vector space as shown in Figure 2.3. By over sampling our data and creating a null space, we are able to compute any vector in the space, hence our ability to regenerate a lost vector. The number of vectors that can be regenerated depends on the size of the null space. To tolerate the loss of n vectors, we will need a null space with n vectors.

$$\text{fields} \subseteq \text{Euclidean domains} \subseteq \text{PIDs} \subseteq \text{UFDs} \subseteq \text{integral domains} \tag{2.1}$$

James Plank gives an instructive approach to writing programs in C to perform erasure coding using the Reed-Solomon method [77] giving examples for fields in 4, 8,

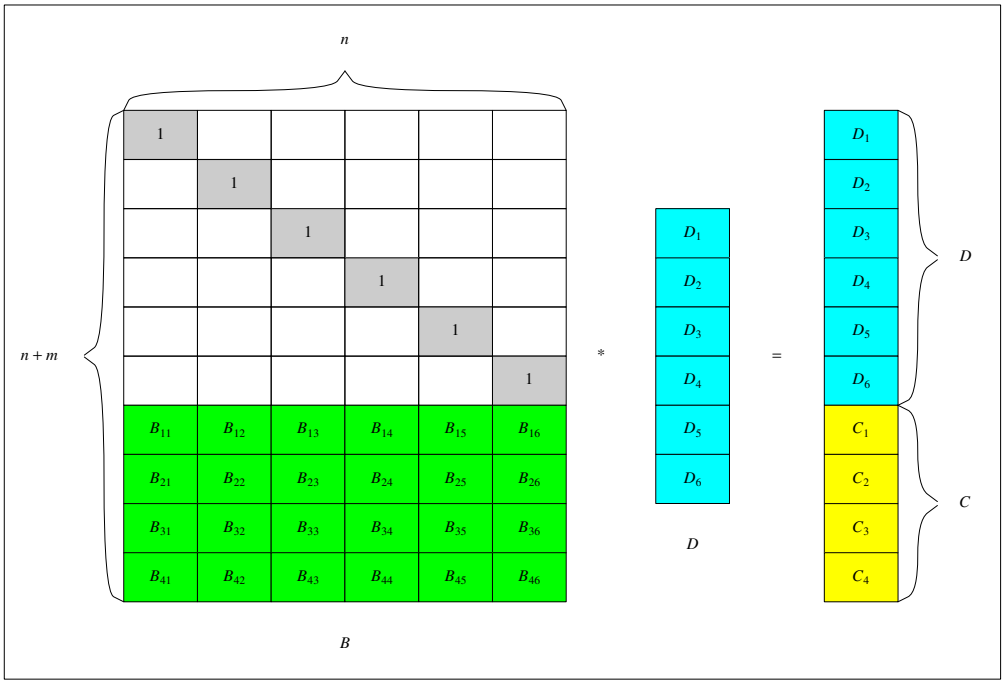

FIGURE 2.3: Erasure Code Matrix — Erasure coding is produced by linear combinations over finite fields.

and 16 space where he shows how to generate the log and inverse log tables that are used for Reed-Solomon erasure coding. Both Kerl and Plank ground their instruction in the number theory from several good sources [25, 142, 143]. Plank gives more examples on computing higher order Galois fields [144]. Another author gives a detailed discussion of the mathematics of RAID-6 [145]. Curry also explains how logarithms over $GF(2^8)$ are used in Gibraltar where the NVIDIA GPU is able to perform many table lookups nearly concurrently to the log and inverse log tables, a performance feature that the GPU has over CPU architectures [22].

The biggest concern with erasure coding is the cost of computing the erasures and the additional network cost incurred due to the parity and the reconstruction of the parity when repairs are required. Kim *et al.*studied these problems with various network bandwidths from one GbE to 100 GbE and found that client overhead increases about 46% with erasure coding [100].

LANL studied the performance of two commercial object storage products. They report the performance writing to the object storage system using a client file system implementation to be 35 MB/s when writing 100 MB files but noted that it dropped off quickly when file sizes approached 1 GB [146].

A study has been done on the performance and power consumption used by erasure coding [147]. They observed that research in erasure coding had mainly focussed on small data sizes that were used by user applications where there were small reads, writes and update. In HPC storage applications where erasure coding would be used, i.e., nearline archive disk storage, the data sizes will be much greater. The selected several SIMD erasure coding libraries for the Intel® CPU platform to perform their experiments. In their experiments, they measured the power consumption of the system under test under idle conditions. They then ran test where they read data from disk, erasure coded the data and then wrote the data back to disk. They used a four TB hard disk drive and a one TB FLASH drive for their tests. The results they measured are shown in Table 2.2. The costs were computed based on the three year storage estimates

TABLE 2.2: Erasure Coding Performance and Power Consumption [147]

Method	K, M	I/O	Coding	Idle	Power Cost
Jerasure 2.0	160,80	10%	11%	79%	$18,218
Jerasure 2.0	160,40	13%	9%	78%	$11,430
Jerasure 2.0	160,32	16%	7%	77%	$10,464

at LANL for Trinity which total 120 PB using the electrical power cost rate of $10 per KWH.

To make it clear when we are talking about the erasure coded data sets, we will call the data sets with their parity "stripes", the data members of the set will be called "data shards", and the parity members will be called "parity shards." We will refer to the number of data shards in the stripe as K and the number of parity shards in the stripe as M; e.g., the notation $RS(K, M)$ will be used to describe a Reed-Solomon erasure coded stripe with K data shards and M parity shards.

2.7 Advanced Encryption System

In 2001, The National Institute of Standards and Technology adopted the Advanced Encryption Standard (AES) as a subset of the cryptographic algorithm developed by Vincent Rijmen and Joan Daemen who submitted their product to NIST during the selection process [148]. Their algorithm is a family of ciphers with various key and block sizes. NIST selected a single block size, 128 bits and thee key lengths: 128, 192 and 256 bits [148]. In 2007, NIST released Special Publication 800-38D which established the Galois Counter Mode (GCM) for AES encryption [149]. GCM provides for authenticated encryption, a feature that computes a hash that can be used to verify the integrity of the data. Because the GCM specifies an algorithm for independently computing the initialization vectors that are used for each block of 128 bits that are encrypted, these can be computed *a priori* and many 128 bit blocks of data can be encrypted in parallel. The Federal Information Processing Standards (FIPS) require that

TABLE 2.3: Comparison of High Performance Distributed File Systems discussed in section 2.3.

Product	Storage	Interface	Performance	Resilience
Coda	Block	POSIX®	Caches file replicas and metadata on each client, clients push updates to each replica	Replication
Swift	Object	Unix Library	Near linear scaling	Replication and XOR RAID
NASD	Object	POSIX®	Linear scaling depending on file system	Replication
zFS	Object	POSIX®, Unix Library	Near linear scaling	Depends on underlying storage provider
pNFS	Block, Object	POSIX®, Unix Library	Near linear scaling	Block depends on underlying storage provider; Object provides replication, RAID and erasure coding
Vesta	Block	Unix Library	2 dimensional file structure for high rate N to 1 write operations	Depends on underlying storage provider
PVFS(2)	Block, Object	POSIX®, MPI-IO, API	Linear scaling	Depends on underlying storage provider
HDFS	Block	HDFS	Super linear scaling (compute and data locality)	Replication
Lustre	Object	POSIX®	Linear scaling	Depends on underlying storage provider
GFS	Block	Unix Library	Linear scaling, excellent N to 1 write	Replication
GPFS	Object	POSIX®, HDFS, Openstack SWIFT	Linear scaling	Replication, erasure coding
Panasas	Object	POSIX®	Linear scaling	Replication, RAID
Ceph	Object	POSIX®, HDFS, Openstack SWIFT	Linear scaling	Replication, erasure coding

the block cipher algorithms that implement AES be approved by an authorized testing lab. Approved cryptographic modules are included in many software and hardware products that are available on the market today. The OpenSSL software library that is included in many Linux distributions and is available Open Source has approved AES algorithms that meet the FIPS requirement. There have been a number of GPU based implementations of AES and research has been done to refine these types of algorithms and measure their performance. However, because modern instruction set architectures have included special machine instructions that accelerate AES encryption, the GPUs are disadvantaged due to the burden of offloading the data to the GPU for encryption.

AES encryption is based on Galois Finite Field arithmetic [148] so it is very similar computationally to erasure coding. Since there are requirements in a number of high performance computing work loads, to provide confidentiality, integrity, and availability, there will be cases where erasure coding, authentication, and encryption will need to be provided. In these cases, where there will be multiple reuses of the data for computation, the cost of offloading data to the GPU will be amortized by a larger factor and becomes a strong competitor with CPU based solutions.

Several research papers have been published on performing AES encryption on GPUs. One research effort has produced an implementation of AES encryption using the NVIDIA® CUDA® programming language [150]. This research implemented several block cipher modes and compared the results with host CPU based encryption performance, the concepts and computer code have been shared as open source software. All of these produced encryption and decryption in the order of MB/s which do not meet our bandwidth requirements. The reason for the lower performance was the smaller block sizes being encrypted due to the use cases the works were scoped to. Also, none of these efforts showed the benefits from data reuse for performing encryption with erasure coding.

2.8 Summary

The fundamental storage unit in the object storage systems is the Object. The Object is a higher-level of abstraction over the primitive storage elements in the persistent media that we use for high performance computing. Hard disk media (rotating disks) and solid state disk media both present groups of bits in the interfaces that are organized in 8-bit bytes. The size of these groupings are referred to as sectors that range in size of 512 bytes to 4,096 bytes in today's products. Traditional file systems assemble these sectors together in blocks, usually about four MiB, to link together to store a file's data in. In Object storage, the details of storage in the lower layers of the file system are encapsulated by the object interface as illustrated in Figure 2.1 in chapter 2. Object Storage systems provide an interface to create an object with a given name and then the user can write data to the object, even specifying an offset into the object where the data begins. Once data is written to an object, it can be read from the object storage system by expressing the object name, the beginning offset and the length of the data to be read.

Object Storage Systems are formed with multiple servers, each of which is configured with persistent storage media. These servers are interconnected by a network having desired performance characteristics. Object Storage Systems differ in how they address objects in the cluster. Ceph uses CRUSH to determine where an object can be found in the cluster. Objects are owned by Object Storage Devices (OSDs) which usually consists of a process that manages a single persistent storage device. There are usually several OSDs deployed on each Object Storage Server (OSS) in the cluster.

The storage devices, called disks from here on out, are connected to the server's memory and compute components via the PCI bus, a Host Bus Adapter (HBA), either a Serial Attached SCSI (SAS) or a Serial ATA (SATA) channel. The SAS bus is most frequently used for HPC storage applications because of the rich control features of the Small Computer Systems Interface (SCSI). Modern archive storage disks have read and write bandwidths on the order of 200 MiB/s. The SAS-2 interface can provide 6 GiB

full duplex bandwidth for data transfer to a disk, up to 600 MiB/s. HBAs interface with enclosure devices that connect several disk drives to the system, enclosures may contain one to many disks, e.g., 60 disks. The HBAs typically provide four SAS channels to connect to the disk enclosures over special cables. This gives an available bandwidth of 24 GiB/s for data to be read or written to the disks in the enclosure. Some HBAs provide two sets of four SAS groups for connecting to two enclosure ports. The HBA usually has eight PCI channels to move data to and from the computer memory, processor or other devices on the PCI bus. With the PCI 3.0 specification, each channel can move 8 GT/s (billion transfers per second) each transfer is 128 bits giving 32 GiB/s. The peak data transfer capability of the HBA is 256 GiB/s.

OSSs are interconnected between themselves and clients via a network fabric. For example, these networks can be Ethernet fabrics with bandwidth ranging between one Gb/s to 100 Gb/s, depending on the system requirements. The network adapters are also connected to the server compute and memory via the PCI bus, usually using eight PCI channels and having the same bandwidth capacity as given above for the HBA.

When using Object Storage for HPC nearline archive, the data will traverse the File Transfer Appliances (FTA). FTAs will be connected to the high performance file systems of the HPC system over the network fabric used for that system, e.g., Infiniband, as well as the Object Storage System fabric discussed previously. The Infiniband bandwidth in today's systems is on the order of 100 Gb/s. The Infiniband adapters are connected to the FTA compute and memory via the PCI bus, usually using eight PCI channels and having a bandwidth capability of 256 GiB/s. The FTAs are also connected to the Object Storage fabric with adapters appropriate for that fabric. When the FTA is configured with GPU devices, these devices are connected to the server via the PCI bus with 16 PCI channels. This gives a maximum bandwidth between the host and the GPU of 512 GiB/s.

CHAPTER 3

THEORY AND PRACTICE

3.1 Introduction

In this section we describe the architecture and performance of our nearline disk archive storage system (NLDS). Because cloud storage has proven to be very economical for highly aggregated services where latency is normal and expected, e.g., Amazon EC2®, Google Cloud Platform™, Rackspace®, and Microsoft® Azure® [119, 151–153], the object technology behind cloud storage should provide the same benefits of cost savings and reliability to the high performance computing (HPC) storage stack [147]. The need for NLDS is due to the scale of HPC computing which requires the expansion of persistent storage due to the large scaling of the main compute and memory to reach exascale. Because of the requirement for greater HPC storage capacity, it presents economic opportunities for specialization. Where higher bandwidth is required on the edge of the compute nodes, this storage need only be a few times the size of the compute memory, so it is economical to use SSD technology here which provides several times the bandwidth and lower latency than rotating disk media even though this media costs about 20 times more. The performance gradient on the lower performance side of the HPC storage is also an opportunity for economic optimization. Los Alamos National Laboratory (LANL) [154] has planned to use 144 PB of cloud object storage between the tape archive and the parallel file system by 2020. Our research is focused on the problems of reliability, performance and energy costs of this cloud type storage used for nearline disk archive storage in HPC. In this chapter, we discuss the problems that we address, and the capabilities of the components. We introduce the ideas of large stripe sizes using a GPU erasure coding to provide availability and encryption to provide confidentiality

while reducing the capital and operating costs of the total storage infrastructure. We articulate our architecture and how we developed it.

We began this research project on HPC Storage to investigate the use of GPU for high performance computing storage applications. The scope of the work included research, prototyping, and ultra-reliable petascale storage for massive, fast, long-term storage. We looked at using erasure coding to provide high availability at economical space savings by trading off higher computation. This research leveraged previous research done by Matthew Curry that produced the Gibraltar GPU Erasure Coding Library [22, 26–28, 155]. The Gibraltar library provided an erasure coding and repair capability using the GPU based on CUDA [156] but did not address erasure coding for the object storage applications. Our first goal was to select an object storage system implementation to use as a prototype to integrate Gibraltar erasure coding. By using Gibraltar and an existing object storage system as a baseline, we were able to investigate the problems that we laid out in chapter 1. After we carried out the research as described in chapter 2, we decided on a Ceph file system for our prototyping. The reasons that we selected Ceph for our prototyping were:

- Ceph already had an erasure coding implementation that was implemented with a plugin architecture. This feature was important to us.
- The system configuration for a Ceph object storage cluster provided the maximum capacity and performance for out budget.
- The software for Ceph was open source and very actively under development.
- There were already large users of Ceph, especially the OpenStack group.

3.2 Problems

The problem spaces in nearline disk archive storage are rooted in data availability, disk IO bandwidth, capital cost, and energy cost. The availability problem in cloud

storage has been addressed using replication and erasure coding. For replication, the best practice is to use a minimum of three replicas to enable the survival of a failed media and the possibility that there might be a failure in the other copies during the reconstruction. Research on reliability for very large storage systems has shown that the mean time to data loss (MTTDL) increases by about 1 million times between two replicas and three [129]. Other large capacity storage research has included ways to handle data being unavailable due to network failures due to the Byzantine Generals problem [157, 158]. The cost of providing this level of replication is three times the capital and operating costs for the required amount of storage. When a lost member is repaired, the system must copy the full amount of data to a new storage location. This results in network utilization that must be accounted for in the design and acquisition of the system. For example, the failure of an 8 TB disk drive that has a utilization of 60% will require the minimum read and write of nearly five TB of data for the repair. Assuming that a 10 GbE network can move 1 GB/s and the data are repaired sequentially, the time to move five TB is about 1.5 hours to read, and write out for 1.5 hours. With replication, we can have parallelism because the unit of replication in object storage systems is the object and there will likely be many objects stored on the failed disk.

Erasure coding provides an alternative for mitigating the availability problem that trades disk storage space for compute, memory and network bandwidth. For example, using Reed-Solomon [77] with 10 data shards and two parity shards $RS(10, 2)$, the system can provide the same level of reliability but with only 20% more storage space, reducing the needed storage by 60%, i.e., $RS(10, 2)$ requires $1.2/3.0 = 0.40$, 40% of the storage required by replication. From a cost perspective, $RS(10, 2)$ requires 20% additional capital and operation costs as compared with replication which requires 200% additional capital and operational costs. However, the erasure coding to produce the redundant parity shards generates a large compute load for the storage system [100]. Data communication costs requires that sufficient data be read to repair the erasures. In the case of $RS(10, 2)$ we say that $K = 10$ and $M = 2$, so at least K shards must be read

in order to regenerate the erasure(s). In $RS(10, 2)$ we can tolerate the loss of two failed shards and fully recover. The loss of three shards in the same stripe set will result in the loss of the data. The same risk applies to replication in the case of three replicas.

3.3 Failure in storage

Data are written to storage media with an expectation it can be read back again without error in the future. There are several threats to this expected integrity and availability that must be understood and mitigated if the user's expectations are to be met. The stake holders must together decide based on the impact of the loss of data how much risk to take which will in turn determine how much of the budget will be allocated to prevent data loss or to what degree data loss is acceptable. In the context of nearline disk archive for HPC, we will need to provide tens to hundreds of petabytes of storage. With today's disk drives having a capacity on the order of 10 TB each, this will require thousands of disk drives. Disk drive failures include total unit failures due to electronics, servomechanism or media failure. Disk drives also exhibit irrecoverable read errors (IRE) at a predictable level of about one sector in 10^{15} bits read where a sector is 4,192 bytes [1]. The impact of an IRE will be the loss of the data unit that is on the media where the IRE occurred. In the case of object storage, the impact would depend on the size of the object of which this sector was an element. In the case of unit failures, all data stored on the unit will not be available [2].

The practical mitigations to the types of data loss discussed above have been replication and error correcting parity. Replication writes data to two or more storage places as shown in Figure 3.1 so that data should still be available on one device if the other has failed. This approach requires twice the amount of storage for data if one replica is

[1] The legacy size for disk sectors was 512 bytes. With larger disk capacities, the sector size is now standardized at 4,192 bytes (4 KB).

[2] While this is true from a practical perspective, there are companies that specialize in data recovery for those stake holders that did not take heed to mitigate against data loss. These services are very pricey to those that determine the value of the data is worth the cost of the recovery. However, not all data is recoverable even using this method.

created or three times the storage if two replicas are created. With today's large capacity storage devices, three replicas is considered the best practice. Error correcting parity can mitigate against data loss by storing a parity which can be used to recreate the lost data. The advantage is that parity can be generated across large sets of data and uses much less storage than replication to provide an equivalent degree of protection. Additional work is required to produce parity since it is a computation on the data shards using Galois field algebra of a stripe to produce the parity shards that are linear combinations of the data. Fortunately, these computations are independent and provide $\mathcal{O}(\frac{S}{K} \times M)$ concurrency, where S is the number of computers performing the erasure coding. A simple method of providing repair capability for the loss of a single data member is to use the logical exclusive or (XOR) operation on the data. For example as shown in Figure 3.2, a corresponding data sector on three disk drives which are used to store a set of data that fits in two disk sectors, e.g., 1024 bytes (for 512) byte sectors, can be XORed together to produce a third sector of parity. If one of the sectors fails, then the XOR operation can be computed again with the two sectors that can be read correctly to generate the missing third sector of data. In the latter example, the repaired sector would be written back to disk. If the disk unit only had an IRE, then the drive controller would automatically remove the bad sector from the map and replace it with a spare sector for the repaired data to be stored to. To use the same example where a failure was detected and a repair attempted, there is a risk that another failure may occur among the same set of data. If this happens, then the set of data that were stored on the two sectors cannot be recovered. For this reason with today's larger capacity disk drives, two or more parity units are preferred and can be generated using erasure coding. In Figure 3.3 we illustrate an $RS(2, 2)$ configuration where we use erasure coding to produce two shards of parity data using the Reed-Solomon algorithm [77], a similar construction as a RAID 10 but able to tolerate double errors without failure while the RAID 10 can only tolerate a single error without failure. Both provide a storage utilization of 50%. Last, delays in detecting and repairing lost data set members increases the chance of data loss. For the configuration where two parity units are provided for a healthy set of data,

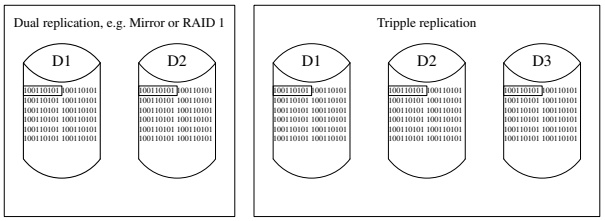

FIGURE 3.1: Data Replication — One common way to mitigate against data loss is with replication. One or more replicas are made of the data which can be used to repair a data loss. Replication also provides the benefit of providing multiple IO paths to the data which can be used to improve performance where the data is read concurrently by multiple consumers.

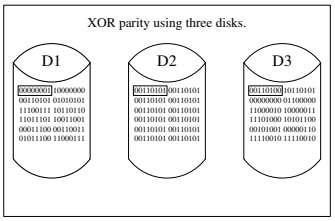

FIGURE 3.2: XOR Coding Parity — Another way to mitigate data loss is by computing an exclusive or (XOR) of the data. This provides the ability to reconstruct a single lost data element in a stripe.

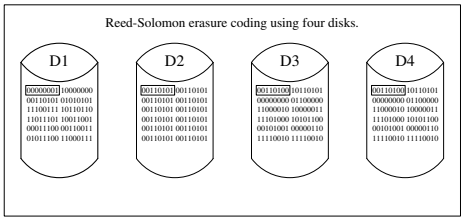

FIGURE 3.3: Erasure Coding Parity — To provide more than one parity shard for a stripe we use erasure coding, such as Reed-Solomon, to compute additional sets of parity data with linear combinations of the data over finite integer fields.

when a member fails, whether detected or not, the set is now only protected with a single parity unit. Since IREs occur on a function of the amount of data read by the system, the chances of an error occurring continue to rise. Total unit failures may also occur within the same set of data which would also reduce the chances of survival [138].

The problem of protecting data from loss is not as hopeless as it sounds as disk drive unit failures range from less than 1% per year to rare worst cases of 25% per year. The norm seems to be more clustered around the 1.25% per year according to a report by Backblaze for 2018 [159]. While IREs seem to be relentless, disk drives are manufactured with thousands of spare sectors to be used when failures are detected. As

long as the system can repair the lost data and write it back, the full integrity can be restored. The remaining problems are the costs of the mitigations. Either we pay for two or three times the storage space to provide replication or we pay to compute parity and use a much smaller amount of additional storage space. Both cases will also have data transfer costs as well because to repair lost data members requires that the good copy of the data be read from its storage location, repairing of the data, and writing it back. In the case of replication, only a read and write of the failed sectors is required but for parity based systems, a full stripe of data must be read to effectively perform the repair. To reuse the earlier example of the failed 10 TB disk drive, with replication, no more than the capacity of the disk drive would need to be recovered but with erasure coded stripes, K times the size of the lost shards on the disk would need to be read to reconstruct the lost data.

It is important to make clear that when storing the shards of data stripes, they do not have to be one to one with a disk unit, e.g., data shard one goes to disk one, etc. It is a much more common practice to uniformly distribute the shards of a stripe across a large set of data storage units, especially for HPC NLDA. When the object storage system places object across the storage system using a uniform distribution, like Ceph does with CRUSH [80], the benefits are balanced work load and a reduction of failure impact due to a more even distribution of failing devices. The shards of a data stripe are stored in objects in the NLDA with an object storing shards for several consecutive stripes, e.g., shard 1 of stripe 1 will be placed in Object.1 beginning at offset 0, shard 1 of stripe 2 will be placed in Object.1 at offset d where d is the size of the shard in bytes, shard 1 of stripe 3 will be placed in Object.1 at offset $2 * d$, etc. After object.1 is filled, Object.2 begins with the next stripe being stored at offset 0 and so forth, as with Object.1. New objects are created to store subsequent stripes of data as required. The number of shards stored in each object is a trade off between reducing the overhead of object creation, reducing the number of objects tracked in the metadata, and the repair cost. When a failure occurs that renders an element of a stripe unavailable, then K

objects must be read to repair the erasure. Having smaller objects results in a smaller amount of data that will need to be read for repair. Having objects with too few shards per object results in greater overhead to create the objects in the object storage system and increases the number of objects that must be tracked in the metadata for the data set. The number of possible different ways to store data stripes in an object storage system is found by computing the combinations C of $K + M$ among the Q disks in the storage system Equation 3.1. If we count the number of objects that can be stored on each disk, we will have even more ways to store a data stripe. Letting the size of an object be B and using the multiplication principle for combinations [160], we will have C_T ways Equation 3.2.

$$C = \binom{Q}{K+M} \tag{3.1}$$

$$C_T = \binom{Q}{K+M} \times \frac{Q}{B} \tag{3.2}$$

For example, given a storage system having objects with $B = 256\,\mathrm{MB}$ of data stored in each, $k = 20$ data shards per stripe, $m = 4$ parity shards per stripe, i.e., $RS(20, 4)$, with $Q = 1,000$ 10 TB disk drives (a total of 10 PB raw storage) provides about 40,000,000 objects in which these shards can be stored as shown in (3.3).

$$\frac{10\,\mathrm{TB}}{256\,\mathrm{MB}} \times 1,000\,\mathrm{disks} = 40,000,000\,\mathrm{objects} \tag{3.3}$$

There are 4.8×10^{52} ways to store the stripes of 24 shards in objects in the storage system as shown in (3.4).

$$\binom{1000}{24} \times \frac{10\,\mathrm{TB}}{256\,\mathrm{MB}} = \frac{1000!}{24! \times (1000-24)!} \times 40000 \approx 4.8 \times 10^{52} \tag{3.4}$$

TABLE 3.1: Combinations of $RS(20.4)$ stripes with 1,000 disks.

$\frac{f}{f_d}$	1	4	12	100	200
0	$9.76E-01$	$9.07E-01$	$7.46E-01$	$7.73E-02$	$4.40E-03$
1	$2.40E-02$	$8.95E-02$	$2.23E-01$	$2.12E-01$	$2.72E-02$
2	$0.00E+00$	$3.17E-03$	$2.92E-02$	$2.74E-01$	$8.00E-02$
3	$0.00E+00$	$4.77E-05$	$2.21E-03$	$2.24E-01$	$1.49E-01$
4	$0.00E+00$	$2.57E-07$	$1.08E-04$	$1.30E-01$	$1.98E-01$

$$P(f_d|f) \approx \frac{\binom{1000-f}{24-f_d}\binom{f}{f_d}}{\binom{1000}{24}} \tag{3.5}$$

$$P(f_d|f) \approx \frac{\binom{1000-f}{144-f_d}\binom{f}{f_d}}{\binom{1000}{144}} \tag{3.6}$$

In Table 3.1 we show the likelihood of having f_d disk failures in a stripe based on the number of failed disks, f, in a system with 1,000 disks. We show the likelihoods of having up to 4 failures in a stripe. The likelihood of having a stripe with f_d bad shards given there are f failed disks is given in Equation 3.5 for an $RS(20, 4)$ configuration. In Table 3.2 we show the likelihood of having f_d disk failures in a stripe based on the number of failed disks, f, in a system with 1,000 disks. We show the likelihoods of having up to 24 failures in a stripe. The likelihood of having a stripe with f_d bad shards given there are f failed disks is given in Equation 3.6 for an $RS(120, 24)$ configuration.

From Table 3.1 and Table 3.2, in the last row of each table, the likelihood of having failures is greater for the $RS(20, 4)$ configuration than for the $RS(120, 24)$ configuration. For the $f_d = 4, f = 200$ case in Table 3.1 compared to the $f_d = 24, f = 200$ case in Table 3.2, the former is four times more likely.

TABLE 3.2: Combinations of $RS(120.24)$ stripes with 1,000 disks.

$\frac{f}{f_d}$	1	4	12	100	200
0	$8.56E-01$	$5.36E-01$	$1.53E-01$	$7.20E-08$	$6.17E-16$
1	$1.44E-01$	$3.62E-01$	$3.13E-01$	$1.37E-06$	$2.70E-14$
2	$0.00E+00$	$9.10E-02$	$2.91E-01$	$1.28E-05$	$5.84E-13$
3	$0.00E+00$	$1.01E-02$	$1.63E-01$	$7.82E-05$	$8.31E-12$
4	$0.00E+00$	$4.15E-04$	$6.08E-02$	$3.52E-04$	$8.74E-11$
5	$0.00E+00$	$0.00E+00$	$1.60E-02$	$1.24E-03$	$7.26E-10$
6	$0.00E+00$	$0.00E+00$	$3.06E-03$	$3.59E-03$	$4.95E-09$
7	$0.00E+00$	$0.00E+00$	$4.26E-04$	$8.72E-03$	$2.86E-08$
8	$0.00E+00$	$0.00E+00$	$4.28E-05$	$1.82E-02$	$1.42E-07$
9	$0.00E+00$	$0.00E+00$	$3.03E-06$	$3.30E-02$	$6.21E-07$
10	$0.00E+00$	$0.00E+00$	$1.44E-07$	$5.30E-02$	$2.40E-06$
11	$0.00E+00$	$0.00E+00$	$4.10E-09$	$7.57E-02$	$8.34E-06$
12	$0.00E+00$	$0.00E+00$	$5.30E-11$	$9.72E-02$	$2.61E-05$
13	$0.00E+00$	$0.00E+00$	$0.00E+00$	$1.13E-01$	$7.46E-05$
14	$0.00E+00$	$0.00E+00$	$0.00E+00$	$1.19E-01$	$1.95E-04$
15	$0.00E+00$	$0.00E+00$	$0.00E+00$	$1.15E-01$	$4.68E-04$
16	$0.00E+00$	$0.00E+00$	$0.00E+00$	$1.02E-01$	$1.04E-03$
17	$0.00E+00$	$0.00E+00$	$0.00E+00$	$8.39E-02$	$2.14E-03$
18	$0.00E+00$	$0.00E+00$	$0.00E+00$	$6.34E-02$	$4.10E-03$
19	$0.00E+00$	$0.00E+00$	$0.00E+00$	$4.45E-02$	$7.33E-03$
20	$0.00E+00$	$0.00E+00$	$0.00E+00$	$2.90E-02$	$1.23E-02$
21	$0.00E+00$	$0.00E+00$	$0.00E+00$	$1.77E-02$	$1.92E-02$
22	$0.00E+00$	$0.00E+00$	$0.00E+00$	$1.00E-02$	$2.84E-02$
23	$0.00E+00$	$0.00E+00$	$0.00E+00$	$5.32E-03$	$3.95E-02$
24	$0.00E+00$	$0.00E+00$	$0.00E+00$	$2.65E-03$	$5.18E-02$

3.4 Requirements of a Nearline Disk Object Storage System

NLDA storage systems that will provide petabytes of storage will take up a significant amount of space in the data center, and will require attention to the power and cooling requirements. Disk drives are heavy and need to be mounted in a safe manner. Bandwidth and availability requirements of the users must be met. LANL currently has 72 PB of Campaign storage and plans to be at 144 PB by 2020 to handle the 4 PB of total system memory for Crossroads. They need 1 GB/s of bandwidth per petabyte of storage. The system will need to store three times the total Trinity system memory per month and store the data for up to one year [154]. Categories:

- Power/cooling/data center
- Network bandwidth and redundancy
- File Transfer Appliances (FTA) and erasure coding
- Object Storage Servers

The requirements will depend on the priorities that the system owner assigns to these categories. Given the weighting of these properties, the goal will be the intersection of the solutions. Design choices will be made by decision makers based on their priorities.

Disk Bandwidth: Disk bandwidth can be increased by writing to more disks in parallel. Disk bandwidth is limited by interconnect and spindle speed for spinning disks. Faster spinning disks have higher bandwidth than slower spinning disks. However, faster spinning disks are smaller in capacity than slower spinning disks. SSDs have higher bandwidth than spinning disks but cost more per byte. SanDisk has a 1 TB SSD priced at $599 [161] while Seagate's 16 TB disk is priced at $609 [162].

IO Operations Per Second (IOPS): Smaller write or read sizes will increase IOPS. Random reads or writes on spinning disks will reduce IOPS. SSDs perform well on small IO and random operations. For NLDA, larger write sizes result in greater bandwidth.

Stripe Sizes: Larger stripe sizes with shards around 4 to 8 MB result in higher throughput on the GPU.

Our design needs to provide policies and controls for dealing with several threats to availability and performance. Some of these are permanent and will require the system owners to replace failed components within a time frame that meets their SLA. Other failures can be addressed less urgently due to system redundancy or self-healing. These failure modes are:

- Temporary failure of a component that is repaired within the SLA. There is no state change in this failure domain. The system is restored to the same state as before the failure occurred within the scope of the affected objects in the failure domain.

- Permanent failure resulting in the loss of objects. The objects can only be recovered from the erasure coding service. This is mitigated by having a sufficient number of coding shards per stripe. The goal is to avoid recovery until the object is read by the user; reconstructed shards are then written back to the object storage system.

- Availability must always provide a minimum of $K + x \,\text{shards where}\, x$ is required to provide safety during any current recovery from a temporary or permanent failure, including IREs.

- Power/Cooling/Data center: Scope is a single data center with high availability power and cooling to meet or exceed the SLA. Power and cooling are likely dependent so we will derive a cost unit based on the power consumption. Data center space is the unit for cost and should be fixed.

- Network bandwidth and redundancy must meet the bandwidth requirements under failure conditions. Bandwidth to/from the FTAs in total must be greater than the required system bandwidth by the factor of coding redundancy. This bandwidth must be matched by the aggregate bandwidth of the Object Storage Servers (OSS).

- The FTA and erasure coding must meet the bandwidth requirements to/from the upstream file storage. Erasure coding throughput must be greater than or equal to the system bandwidth times the coding redundancy factor. Additional FTAs are provided to meet the bandwidth requirement under FTA failure. The failure domain for an FTA includes the upstream network link, the FTA server and the system network link. These are all temporary failure components. The assumption is that the FTA is a shared-nothing implementation and the work can be restarted on another FTA when failure occurs.

- Object Storage Servers can fail in both ways. Disk failures are probably not temporary and all objects stored on the associated Object Storage Device (OSD) are lost. Failures with the other components of the OSS are assumed to be temporary and can be repaired within the SLA. Failed disk drives will be replaced within the SLA but the objects that were stored on them must be recreated by the erasure coding process and possibly stored on another OSD in the mean time.

3.4.1 Baseline Architecture Design

The design baseline architecture consists of the following assembly of components in standard data center racks:

- **Storage rack** —A Top of Rack switch (TORS) to connect to each OSS at the leaf connection speed of the storage network. The TORS will up link to the storage system backbone fabric at leaf switch up link connection speed. The rack contains O OSSs having D disks of capacity C and bandwidth B.

  - The disks are Field Replaceable Units (FRU) and will be replaced upon failure within the SLA to maintain the system raw capacity. The remainder of the OSS is considered a FRU and failure will be in the temporary category. Disks are transferred from the failed OSS to the replacement OSS and brought back online within the SLA so the system state with regard to the affected OSDs is restored to where it was at the time of failure.
  - At any time there must be storage racks available with capacity greater than or equal to the the object stripe size. A storage rack is a failure domain. A storage rack can become unavailable due to a TORS failure or power failure or cooling failure. Other failures within the rack would be confined to the OSS. Disk failures are the minimal failure unit.

- **Power, Cooling, Data center** —We are assuming that the data center provides power, cooling and physical space according to the SLA. Power is provided to each rack through Power Distribution Units (PDU) and may disrupt service due to a fault on an individual rack. Cooling faults are expected to have broader impact and may result in a temporary system unavailability. Facility detection and protection from fire or other environmental threats are outside of our scope.

- **Network infrastructure backbone** —Each FTA will have two full bandwidth connections to the network backbone. The network will be configured to deliver the required bandwidth during component failures. Redundant bandwidth is provided to the TORSs. Each OSS will only have a single network connection. Failure of a TORS will cause a temporary failure of the entire storage rack. The storage rack will be brought back online after the TORS is repaired and the system state will be restored.

- **Number of storage racks** —The number of storage racks required should be greater than:
  $\text{data shards} + \text{coding shards} + \text{fault tolerance on write}$

TABLE 3.3: Example Nearline Disk Archive Storage System Configuration

6 TB disks 25 PB data storage capacity 120 data shards per stripe and 24 coding shards per stripe
25 PB data + 5 PB coding = 30 PB raw 5,000 6 TB disks = 30 PB raw 5,000 / 144 shards == 35 disks per shard

- In order to maintain the full protection capability of erasure coding, the number of available failure domains to support a write are available must be greater than $K + M$ when the system is online. This is to make sure that we have a place to put all of the shards of a stripe.

For the example storage system configuration given in Table 3.3 we show 35 disks per failure domain with 6 TB disks and a total capacity of 25 PB. A 42 U storage rack can hold more than 33 disks. The following are some possible configurations for the storage options:

1. 1U OSS with 3 disks each, total 11 servers + 1 TORS = 12 U.
2. Put 2 failure domains in each rack. Each with 11 servers, a TORS and two PDUs.
3. Put 3 failure domains in each rack with 3 TORSs and 3 PDUs.

The last option amortizes the cost of a rack and cuts the data center footprint by 1/3 of option one so this would be the most economical configuration. The heat load of 33 servers, 99 disk drives and three TORSs will be less than the 35 KW per rack limit.

3.5 Preliminary Investigation and Measurements

We did some early experiments to reproduce other research and better understand the problems with object storage systems and erasure coding to see how they would fit in the storage solutions for HPC. For the tests and curation of the results we used several tools

provided by the open source community. Running concurrent tests and measurements on multiple systems in our test configuration was enabled by the GNU Parallel tool [163].

3.5.1 Raw Disk IO Measured with *dd* and *fio*

To develop our baseline we wanted to establish accurate ways to measure performance. The UNIX® *dd* command has been around for years and does basic IO. It is often used to report IO throughput and its simplicity eliminates much of the software stack from the measurement. We measured several sizes of write operations and read operations to arrive at the maximum sequential performance for a single disk drive in the Dell C8000 chassis. We also measured the bandwidth using *fio* to compare how these two tools report the same performance property.

TABLE 3.4: Baseline Disk Performance — *dd* Version 8.22, *fio* Version 2.1.10

Bandwidth Single Disk Comparison between *dd* and *fio*			
Op	BS	*dd*	*fio*
write	512 B	26.0 KB/s	30 KB/s
write	4 KB	490 KB/s	7 MB/s
write	8 KB	956 KB/s	17 MB/s
write	128 KB	14.8 MB/s	136 MB/s
write	256 KB	26.4 MB/s	174 MB/s
write	1 MB	**75.2 MB/s**	**194 MB/s**
read	512 B	214 MB/s	50 MB/s
read	4 KB	221 MB/s	84 MB/s
read	8 MB	87 MB/s	112 MB/s
read	128 KB	221 MB/s	217 MB/s
read	256 KB	**220 MB/s**	188 MB/s
read	1 MB	178 MB/s	**241 MB/s**

We measured the performance of the components in the data path so that we will understand what the capability of the system is. This has helped us to analyze the performance and locate the bottlenecks. This information also helps us understand the greatest constraints in the system which will impose an upper bound on the system performance. In Table 3.4 we begin by measuring the bandwidth for various block sizes to our storage disks using the *dd* tool and the *fio* tool. In this measurement we wanted to learn what the read and write performance should be for each disk. With three disks per

server, 1 MB read or writes, we should expect approximately 600 MB/s as the maximum bandwidth per node. We measured disk bandwidth on another system with *dd* and got measurements more consistent with the *fio*. The differences were operating system, SAS or SATA, PCI 3 or PCI2, among other things. See Table 3.8 in section 3.7 for the details of these measurements. The higher bandwidth with the 1 MB block sizes is the behavior that matters for our design.

3.5.2 Bandwidth and IOPS Performance

To determine the performance of the disk subsystem we performed measurements of several disk configurations using *fio*. All of the tests were configured to write a total of 8 GB using 20 jobs with a time limit of 240 seconds. The performance that is most critical for distributed file system performance is the 512 byte read and write but the performance at this size is very low. The larger sizes of operations will demonstrate the maximum capabilities of the system. Table 3.5 shows the bandwidth performance measured with 512 byte, 4 KB, 8 KB, 128 KB, 256 KB and 1 MB. These operation sizes were all measured using the 1, 2, 4, 8, 24, 48 and 96 disk configurations. The disk drives were selected to use IO data paths that provided the greatest resources for the test, e.g., the two disk test used one disk on the first HBA and the second disk on the second HBA. Table 3.6 shows the IOPS that were measured at the same time as the bandwidth shown in Table 3.5. We have shown these results as bar charts in Figures 3.4 and 3.5.

The tests show that the IOPS peak for the 3 KB read operations. Write operations are mediocre with maximums occurring at the two disk configuration where we have one disk drive per HBA. The tests were configured to use the libaio driver, a queue depth of four, direct IO and all writes are appending to the single file per disk (the raw device, there are no file systems, e.g., /dev/sdf). Bandwidth peaks at 3.4 GB/s with the 1 MB reads and writes. The read tests hit this 3.4 GB/s maximum with 24 disks drives. The maximum read bandwidth for eight drives is 3.1 GB/s so the system appears to hit the upper limit for bandwidth around 10 disk drives and adding more disk drives does not

increase the bandwidth. For the smaller size operations, the additional disks do make a difference and the bandwidth increases proportionally to the number of disks for some disk configurations. For others, the 4 K and 8 K writes and the 512 byte read, this doesn't hold.

The 8 KB writes are approximately two times the bandwidth of the 4 KB writes so we can conclude that the system can perform the same number of IOPS for these operation sizes and we get twice the bandwidth by doubling the size of the operation. However, this does not hold between the 512 byte and the 4 KB operation sizes, though. The speed of a one disk 512 byte write is 80 KB/s and the speed of the one disk 4 KB write is seven MB/s which is a factor of 87 times and at 24 disks, the 512 byte write bandwidth is 382 KB/s compared with the 4 KB write bandwidth at 20 MB/s giving a ratio of 52 times. In order for the 512 byte write operation to produce the same bandwidth as the 4 KB operation the system would need to perform 13,671 IOPS but the limit on the Everest node is around 8,000 IOPS which is produced with the 4 KB write operation on two disks where there are two HBAs in the system. This low IOPS performance is a concern because our Red Mountain node is able to deliver more than 24,000 IOPS on 512 byte writes and nearly 22,000 IOPS for 4 KB writes Table 3.8.

TABLE 3.5: Disk IO Bandwidth — Raw disk bandwidth comparison. 1-48 with one Dell C8000 chassis, 96 with two.

Bandwidth Disks							
BS	1	2	4	8	24	48	96
Write							
512 B	30.3 KB/s	61.2 KB/s	121 KB/s	240 KB/s	382 KB/s	732 KB/s	1.4 MB/s
4 KB	7.0 MB/s	32 MB/s	27 MB/s	25 MB/s	20 MB/s	28.8 MB/s	31 MB/s
8 KB	17 MB/s	63.3 MB/s	56 MB/s	50 MB/s	40 MB/s	55.5 MB/s	59 MB/s
128 KB	136 MB/s	228 MB/s	331 MB/s	291 MB/s	489 MB/s	996 MB/s	1.3 GB/s
256 KB	174 MB/s	265 MB/s	392 MB/s	563 MB/s	935 MB/s	1.7 GB/s	2.0 GB/s
1 MB	194 MB/s	366 MB/s	668 MB/s	1.1 GB/s	2.1 GB/s	**3.4 GB/s**	
Read							
512 B	50 MB/s	13 MB/s	23.6 MB/s	60 MB/s	46 MB/s	69.3 MB/s	171 MB/s
4 KB	84 MB/s	117 MB/s	161.6 MB/s	208 MB/s	1.4 GB/s	2.1 GB/s	1.4 GB/s
8 KB	112 MB/s	186 MB/s	213.6 MB/s	368 MB/s	672 MB/s	1.6 GB/s	1.7 GB/s
128 KB	217 MB/s	289 MB/s	525 MB/s	1.0 GB/s	3.2 GB/s	3.1 GB/s	2.1 GB/s
256 KB	188 MB/s	296 MB/s	791 MB/s	1.8 GB/s	**3.4 GB/s**	3.0 GB/s	2.0 GB/s
1 MB	241 MB/s	379 MB/s	1,743 MB/s	3.1 GB/s	**3.4 GB/s**	**3.4 GB/s**	

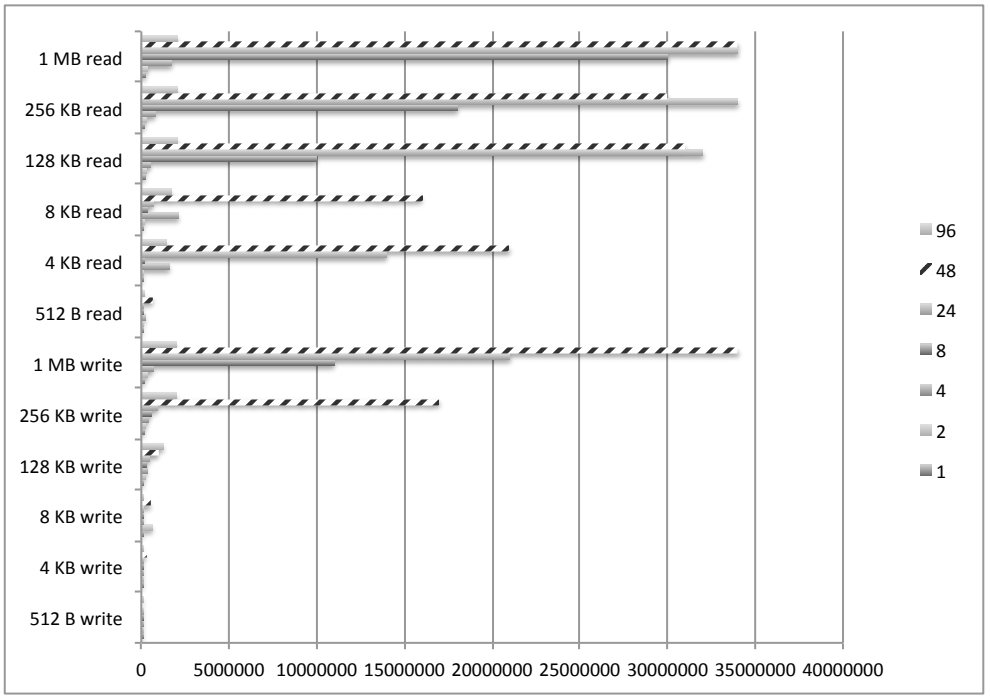

FIGURE 3.4: Disk Bandwidth measured with *fio*

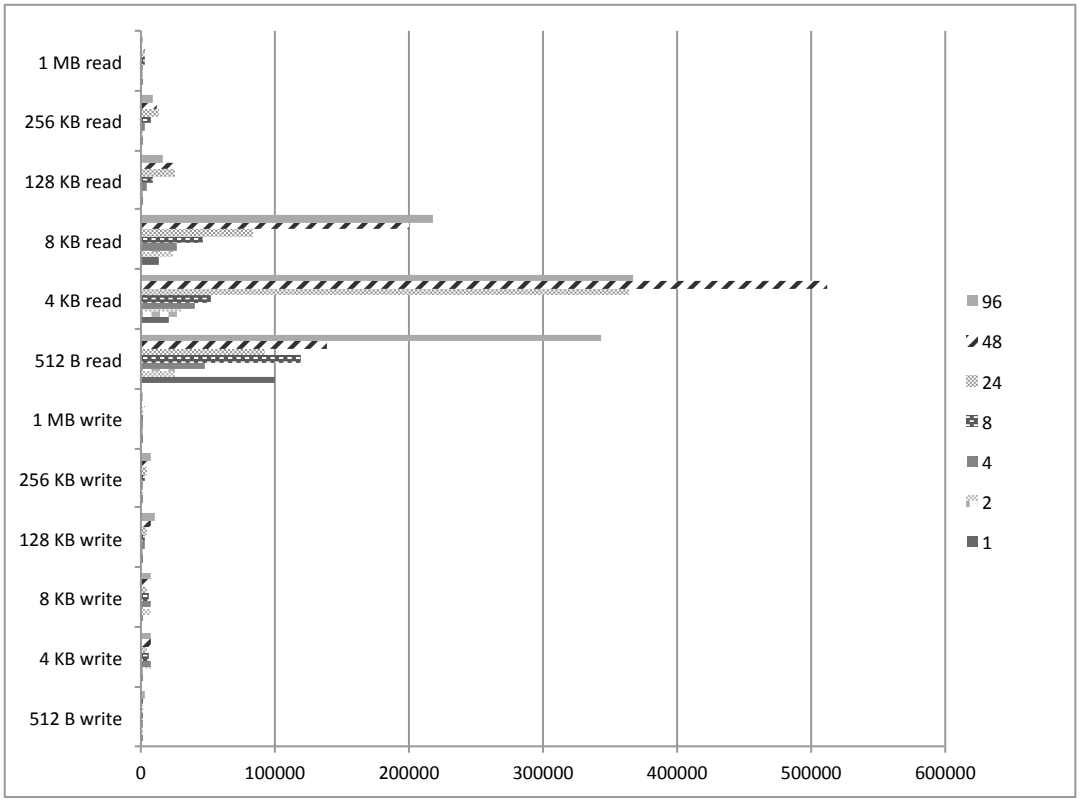

FIGURE 3.5: Disk subsystem IOPS measured with *fio*

TABLE 3.6: Disk IOPS — 1 - 48 measured with one Dell C8000 chassis, 96 with two

IOPs Disks							
BS	1	2	4	8	24	48	96
Write							
512 B	59	119	237	470	745	1,430	2,831
4 KB	1,744	8,013	6,875	6,157	5,028	7,031	7,745
8 KB	2,125	7,906	7,054	6,237	5,022	6,916	7,422
128 KB	1,064	1,781	2,590	2,274	3,818	7,939	**10,077**
256 KB	678	1,035	1,531	2,197	3,652	6,370	7,985
1 MB	189	357	652	1,140	2,104	3,377	2,004
Read							
512 B	100,310	25,975	47,142	119,959	92,582	138,526	342,881
4 KB	21,019	29,361	40,411	51,970	364,459	**512,106**	367,003
8 KB	13,947	23,240	26,701	45,960	83,943	199,932	218,503
128 KB	1,698	2,256	4,103	8,178	25,207	25,179	16,597
256 KB	735	1,157	3,088	7,156	13,686	11,996	8,196
1 MB	235	370	725	3,121	3,402	3,403	2,083

TABLE 3.7: Design maximum data path bandwidth

Bandwidth Data Path		
Component	BW	Capability
PCIe 3.0 8x	7.8 GB/s	8 lanes @ 985 MB/s per lane
LSI SAS 3 HBA	9.6 GB/s	8 lanes @ 12 Gb/s per lane
SAS 2 Expanders	4.8 GB/s	8 lanes per sled @ 6 Gb/s per lane

3.6 Bandwidth of Data Path

The bandwidth for data transfer to and from the disk drives and memory is limited by the components it must pass through. The path consists of the PCI 3.0 bus, the two LSI 9300-8e SAS 3 controllers, the Dell C8000XD SAS 2 Expanders and the SAS interface on each disk drive. The theoretical bandwidth for each of these components is showing in Table 3.7.

We have two configurations under test. The first configuration in **??** provides 1 PB disk storage and connects each Dell R730 with one C8000 chassis having 48 6 TB SAS 3 disk drives. This configuration provides 16 SAS lanes from the two controllers for data flow to and from the 48 disk drives. In this configuration the data flow to each of the four C8000XD disk sleds having 12 disk drives is provided by four SAS channels to

each sled giving a maximum bandwidth of 2.4 GB/s. The configuration provides four SAS lanes to each sled. The maximum bandwidth provided by a single disk is 280 MB/s so the 12 disks on a single sled can sink or source 3.360 GB/s. We conclude that the limiting component in the data path is the SAS Expander giving a maximum bandwidth per sled of 2.4 GB/s and a combined bandwidth of 4.8 GB/s for the two sleds connected to each controller. This is well under the PCIe bandwidth so the maximum performance of the disk subsystem for the single C8000 chassis is 9.6 GB/s. Averaging this over 48 disks indicates that the system can support a concurrent bandwidth of 204 MB/s read or write per drive.

In the second configuration in **??** we have two C8000 subsystems giving a total of 96 disk drives. But, we are still limited by the 16 SAS lanes from the two LSI HBAs that are capped at 6 Gb/s giving 9.6 GB/s for the disk subsystem. Each of the HBAs is now connected to one of the C8000 chassis providing two SAS lanes to each C8000XD sled and serving 24 disk drives. Each C8000 chassis will have 4.8 GB/s of bandwidth available through these eight lanes which now becomes the limiting factor. This configuration has a 9.6 GB/s maximum theoretical bandwidth, too. Because we have doubled the number of disks to 96 and use the same data path which has a bandwidth of 9.6 GB/s we reduce our maximum concurrent read or write bandwidth per disk to 102 MB/s.

3.7 Comparison with other systems

To validate the measurements that we have made on the Everest cluster with Dell systems using the C8000XD storage solutions we compared them with other systems that serve a similar purpose that we have on hand. The comparisons were only made using one disk drive using the *dd* and *fio* tools of the same version. The configuration of the systems that we have used for comparison are in our Red Mountain cluster. The systems have two Intel Xeon X5650 @ 2.67 GHz, 24 GB RAM, LSI SAS2008, WD2003FYYS-23W0B disk drives. The operating system on one of the Red Mountain nodes is Ubuntu

LTS 14.04 with 3.16 kernel and the other has the same distribution of CentOS 7.1 as the Everest nodes. The Red Mountain CentOS node has the kernel included with the distribution, 3.10, while we have upgraded the Everest nodes to the 3.18 kernel. The write performance of the Everest nodes for 512 byte and 4 KB operations are significantly low compared to the Red Mountain performance by an order of 100. We have also compared the performance with the recent report from CERN for their 30 PB Ceph configuration which they reported 110 MB/s write with 4 KB operations and 135 MB/s reads with the same data size using *dd* [82]. Note that our measured *dd* performance for write was less than our *fio* measurements by an order of 10 Our measurements are shown in Table 3.8.

3.8 Architecture

From our measurements we observed that our system architecture was capable of performing above 1 GB/s if we used larger sized reads and writes however, most studies were only showing performance in the tens to hundreds of MB/s range (see section 2.6 in chapter 2). We also observed that HPC was using disk storage between PFS and tape archive as a holding area which was beginning to see growing capacity and data lifetime. For example, at Lawrence Livermore National Laboratory they increased their "grace period" for data in their HPSS disk cache from 20 days to 300 days with a recent upgrade [164] and at Los Alamos National Lab they introduced Campaign Storage with Trinity [3, 165]. Since the requirements for this tier of storage, a buffer between the high performant storage layer and the long term tape archive layer constrained the usage to larger file sizes, immutable files, and moderate data lifetime on the order of a year, we found this to be an interesting problem for our research. Erasure coding on the GPU with Gibraltar performed much better when we were able to do more compute on the data. This property provided a good fit with the larger file size constraints for the nearline archive storage, we investigated this more closely and decided to design an architecture for NLDA that exploited the performance of Gibraltar with the large stripe sizes. With larger stripe sizes, having the same storage efficiency as with smaller stripe sizes resulted

TABLE 3.8: Comparison between Red Mountain nodes and Everest nodes, single disk

	Red Mountain		R730
fio IOPS	Ubuntu LTS 14.0.4 3.16	Centos 7 3.10	Centos 7 3.18
512 write	24556	24692	59
4k write	21743	21843	1744
8k write	15700	15969	2125
128k write	1578	1667	1064
512 read	26483	26519	100310
4k read	29107	26915	21019
8k read	19789	20041	13947
128k read	1992	2015	1698
fio BW (KB/s)	Ubuntu 3.16	Centos 7 3.10	Centos 7 3.18
512 write	12278	12346	30
4k write	86972	87376	7000
8k write	12560	127758	17000
128k write	202031	213397	136000
512 read	13242	13260	50000
4k read	116431	107663	84000
8k read	158315	160335	112000
128k read	255045	258015	217000
dd IOPS	Ubuntu 3.16	Centos 7 3.10	Centos 7 3.18
512 write	14928	13528	51
4k write	11500	10860	119
8k write	9400	9152	119
128k write	1147	1179	112
512 read	194881	226370	122732
4k read	38924	37777	36632
8k read	18006	18681	20701
128k read	1119	1149	1682
dd BW (KB/s)	Ubuntu 3.16	Centos 7 3.10	Centos 7 3.18
512 write	4900	6900	26
4k write	47100	44500	490
8k write	77000	75000	981
128k write	150000	155000	14800
512 read	99800	116000	62800
4k read	159000	155000	150000
8k read	148000	153000	170000
128k read	147000	151000	221000

in a greater amount of parity shards. Also, since the cost of repairing erasures has the cost of data transfer and compute, we were interested in learning how long we could

delay erasure repairs using the greater number of parity shards.

By reducing the urgency to repair erasures in nearline disk archive storage systems, we can eliminate most of the additional work required by waiting until the user reads the data to perform the repairs. When the user reads data from the nearline disk storage system, *K* error free shards are read and repairs are performed to return the data stripe to the full reliability configuration, i.e., $RS(10, 2)$. Several studies [128, 138] have shown that erasure repair at the $RS(10, 2)$ and $RS(20, 4)$ level can only be delayed for a short amount of time before the risk of data loss becomes significant. We have shown that erasure coding on GPU using the Gibraltar library [23, 24] can produce large stripe at high bandwidth and low latency which does not affect the user experience for nearline archive storage, e.g., the user expectations for bandwidth and latency for archive storage is based on writing and reading magnetic tapes [23, 24]. When the stripe size is increased, the number of parity shards can also be increased according to the ratio of K to M that the system owners have established. For example, setting $K = 120$ and $M = 24$ provides the same storage efficiency as $RS(10, 2)$. With the larger stripe sizes, we have increased the number of redundant shards by a factor of 12.

HPC data center owners provide File Transfer Appliances (FTAs) as job resources to users to move data between archive storage and the HPC compute storage i.e., PFS. Since data would be moved by these appliances to and from the PFS to the NLDS, there is an opportunity to perform the erasure coding and repair on the FTAs; this is a type of computing on data in transit. Since there are an order of magnitude fewer FTAs than storage servers, configuring the FTAs with GPU devices is an economical design decision. The current market price for the NVIDIA P4 GPU device is around $2,000 which can be paired with the Intel E5-1630 v4 costing $450 for a total cost of $2,450 while the Intel E5-2966A v4 processor which would be required to meet the erasure coding performance of the K40 costs around $4,000, these device configurations perform about the same for erasure coding. This provides a capital savings of about 40% per FTA. In addition, by removing the requirement to compute erasure coding on the OSSs,

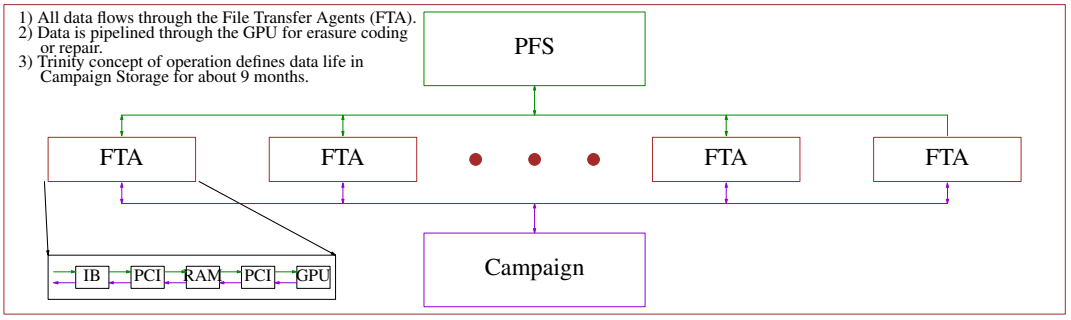

FIGURE 3.6: HSM Data Flow — Data flow to/from the PFS to the NLDA.

they can be configured with less expensive CPUs that are sufficient to handle the storage workload.

Figure 3.6 shows the data flow to or from the PFS via the FTAs. The data moves over the network fabric from the PFS to the FTA into memory, it is off-loaded to the GPU for erasure coding, and moved back to memory. This erasure coded stripe of data is then stored on the NLDA by using the Ceph client API to write to storage. The Ceph storage system places the data according to the CRUSH data placement system [166]. The data moves to the OSSs over the storage network fabric and is written to the disk where it was assigned by the system. When reading data from the NLDA, the data flow is in the reverse direction. When data is read from the disk it is stored in memory on the OSS. It is then sent over the network fabric to the FTA. The data is moved to the FTA memory, then to the GPU memory. Erasure repair is performed on the data. The repaired shards are copied back to the FTA memory. The K data shards are sent to the user out the Infiniband interface. The repaired shards are sent back to the object storage system. In Ceph, there is a process that manages each disk drive in the system, called an Object Storage Device (OSD), which handles the IO and bookkeeping required to store or retrieve the data from the object. The object is the set of storage elements that are used for keeping the data on the media. The OSD provides the object as an abstraction over these elements. The applications only need to keep up with the name of the object where the data are stored in order to retrieve them.

Some important properties about NLDA usage in HPC also make it possible to have a more simple architecture. Data will usually be stored in objects that are on the order

of 1 GiB. Since the data are for archive of facts that do not change, the immutability property removes the requirement to perform random updates on the data. Once it is written, it will only be read when needed. Since the lifetime of some data may be quite long and the number of times that it will be accessed are low, it is more economical to store the data on magnetic tape. Magnetic tape does not require any energy for data at rest, energy is only used when the tape is being written or read[3]. At DKRZ in 2015, they expected to consume 250KW for 52 PB of disk storage while they only expected to consume 25KW for 500 PB of tape storage [167]. For their case, disk storage consumes about 50KW per PB while tape archive consumes 50W per PB! For this reason, the lifetime of data on the NLDA is expected to be less than a year.

We have studied the use of object storage for nearline disk archive (NLDA) for exascale computing [23, 24, 168]. By exploiting the HPC architecture which uses file transfer appliances (FTAs) to move data between high performance file systems and archive, we have shown that using GPUs can enable the use of larger stripe sizes. This provides a means to increase the effectiveness of the erasure coding redundancy by allowing the number of parity shards to increase while maintaining the storage efficiency ratio. Because erasure coding is an embarrassingly parallel problem, GPUs are able to excel in this application compared to typical typical CPU architecture. In Figure 3.7 we compare erasure encoding between Gibraltar [22, 28] and Intel® ISA-L [169] for various stripe sizes. The CPU erasure coding performs as well up to about 20 shards but drops off quickly afterwards. Gibraltar performance drops off much more gradually as the number of shards increase. Erasure repair performance is similar as shown in Figure 3.8 and Figure 3.9. In the first repair example with a single erasure, the Intel ISA-L does a fair job but with four erasures, the Intel ISA-L performance drops off significantly while Gibraltar maintains the same performance.

[3]We are ignoring the energy costs to maintain the tapes in the archive in a controlled environment but this is similar to the environment where other storage media might be used.

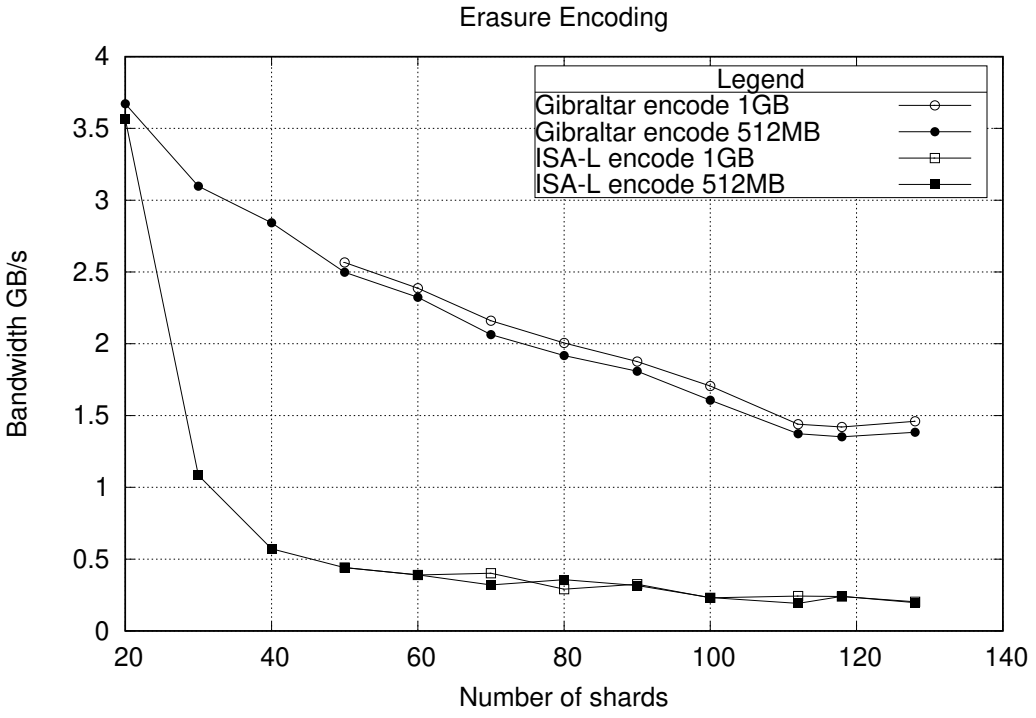

FIGURE 3.7: Effect on erasure coding by shard count — Erasure coding bandwidth results with increasing number of shards. Coding shards are held to a ratio of one coding shard to five data shards.

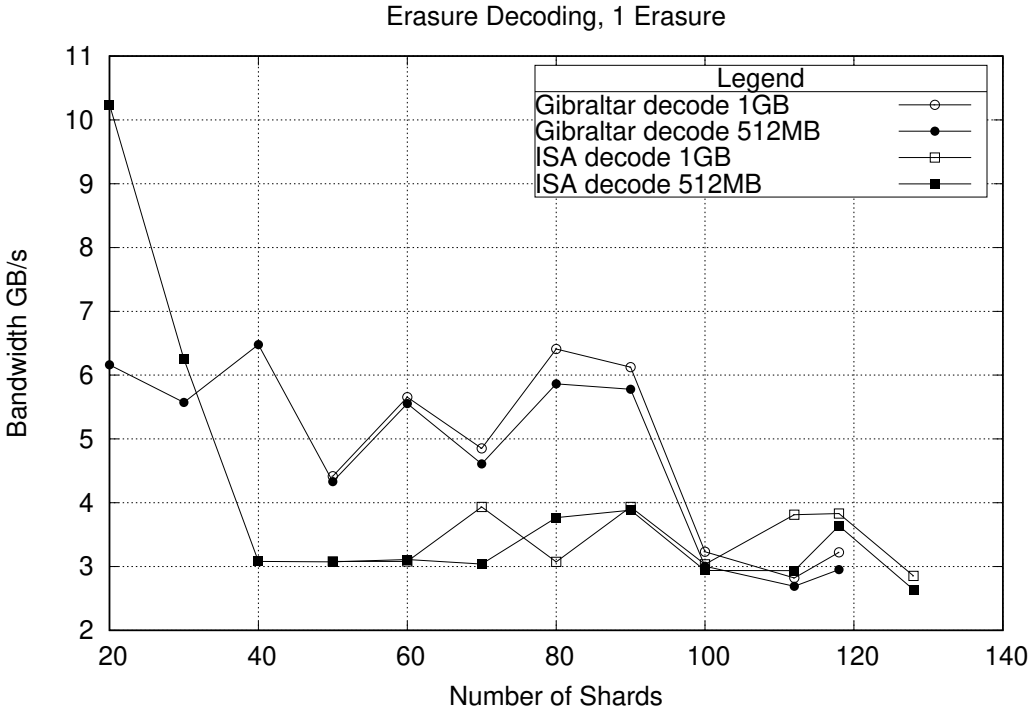

FIGURE 3.8: Erasure repair of one erasure — Erasure recovery bandwidth results with increasing number of shards and one erasure. Coding shards are held to a ratio of one coding shard to five data shards.

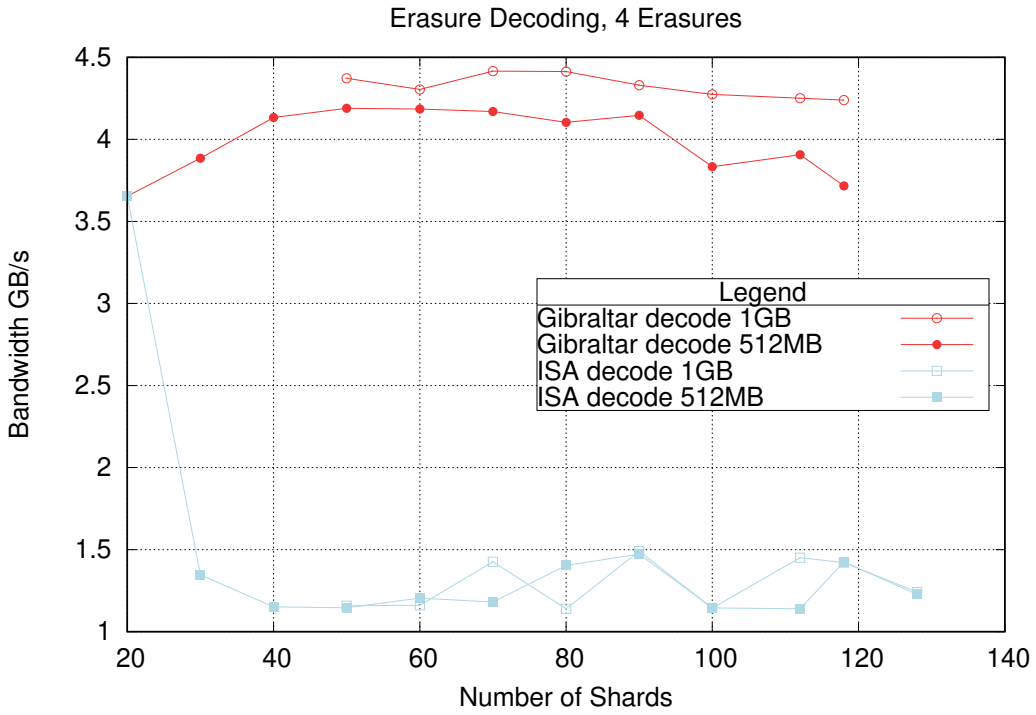

FIGURE 3.9: Erasure repair of four erasures — Erasure recovery bandwidth results with increasing number of shards and four erasures. Coding shards are held to a ratio of one coding shard to five data shards.

3.9 Cost Analysis

Acquisition costs of storage systems is difficult to find in the literature. Pricing for storage is usually included as a feature in proposals having a bottom line price for the entire system that is the subject of the acquisition. Since we have personal familiarity of two recent Lustre parallel file systems by the U. S. Air Force, we will perform a rough analysis to compare the cost of the high performance file system with the lower performance object file systems. The cost for the object file system comes directly from the system under test, see Table 4.1 in chapter 4. We have the following samples for our analysis with the average cost of each type given:

- **$500K** Cost of 700 TB Dell Lustre system with two OSS nodes, two MDS nodes, SAS-3 SAN, FDR Infiniband.
- **$450K** Cost of 500 TB DDN Lustre system with four OSS nodes, two MDS nodes, SAS-3 SAN, EDR Infiniband.
- **$250K** Cost of 2 PB Dell system with four OSS nodes, three client nodes, SAS-2 SAN, FDR Infiniband, 10 GbE, NVIDIA K40 GPUs.
- **$0.79 per GB** Average cost of Lustre storage.
- **$0.12 per GB** Average cost of NLDA storage.

The cost of parallel file systems is six times greater than the cost of object storage systems. It makes economic sense to move data from the parallel file systems to the object storage systems when the data is not actively being used for computation or analysis.

3.10 Encryption

In this research our goal is to show the practicality of performing encryption of data while it is being moved to the storage destination. We also want to encrypt the data

while it is within a security boundary that can be established, guarded and monitored with appropriate controls. By providing encryption while the data is resident on the FTA and before it is stored in the object storage system, we can achieve this objective. Also, we can benefit from additional computation on data that is already present in the GPU memory providing a greater amortization of the cost to move the data onto and off of the GPU. There is one difference that will increase the data communications cost when we perform encryption on the GPU, the data will now be changed (encrypted) and will have to be copied back to the host for transmission to the storage system. When we are not encrypting, the data is not changed so only the parity shards need to be copied back to the host. However, when we study other research about using GPUs for encryption, they all have to contend with the same requirement. We used the AES CUDA encryption functions that were developed by Berlanga in [150] in the CTR [170] mode where the encryption for each 128 bit block begins with adding the initialization vector (IV) to the CUDA block index, an approved optimization which enables the parallel encryption of multiple 128 bit blocks of data. We used a 256 bit key for AES-256 encryption. We only used the transformation tables from the Berlanga source code and the code for performing the 14 rounds of encryption that are required with a 256 bit key. These source code items were originally provided in the NIST documentation but coded as data structures and operations in CUDA by Berlanga. In our design, we move a full stripe of data to the GPU memory from the host, encrypt the data, perform the erasure coding, and copy the data back to the host for storage in the object storage system. When reading data back, data is copied to the GPU memory, erasures are repaired (if required, very likely), decrypted, and copied back to the host.

Since the NVIDIA® K40 GPU that we are using is capable of concurrently copying data from the host to the GPU, executing a CUDA kernel, and copying data from the GPU to the host using the streams feature, we decided to implement the feature in our Gibraltar code. We expected this to provide a speedup of about 2X. We did not meet all of the requirements to make our implementation of AES FIPS compliant but the

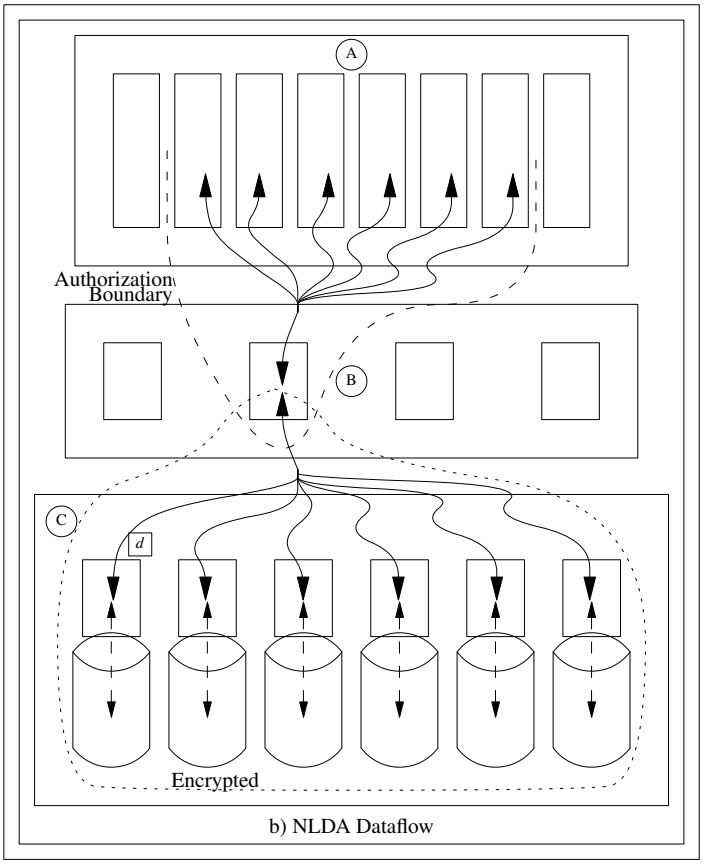

FIGURE 3.10: Security Authorization Boundary — By encrypting data on the FTA, the scope of the authorization boundary can be reduced so that the object storage system is not included. In the illustration, (A) is the PFS, (B) is the FTA, and (C) is the NLDA. The data shards of a stripe are represented by [d].

performance and feasibility demonstrated by our prototype should give a good example of the performance and practicality of this approach. For example, since we use the CUDA index number of the block to increment the IV, each stripe of data that is executed would have the same IV, a property that is not allowed for encryption [170]. In order to comply with this requirement, an initialization vector (IV) would need to be created and stored for each stripe of data. We are leaving the details of this to future research in order to focus on the main goal of the prototype implementation.

By performing encryption on the FTAs the scope of the authorization boundary for the system information security plan is reduced. The object storage system can be part of a shared information system having lower security requirements since the more sensitive data has been encrypted. This is illustrated in Figure 3.10.

3.11 Lazy Repair

In previous research the authors demonstrated that recovery could be delayed without reducing the MTTDL while reducing the repair work and associated costs [128]. They did this through the lens of small stripe sizes with few redundant shards for the stripe. Other research has shown that Reed-Solomon [77] erasure coding could be combined with replication to increase the MTTDL by a factor of 4,000 [136]. Both authors used a Monte Carlo model to compute the MTTDL while another study used a combinatorial model to show that the likelihood of failure was much less as the stripe sizes were increased [136]. The idea of delaying erasure repair can be extended with larger stripe sizes and with a larger number of redundant shards to allow the deferral of repair much longer, possibly beyond the lifetime of the data in the storage system. Under the constraint that the NLDA is used for short term archive after which the data may no longer be required or the data may be moved on to more economical long term archive storage, the lifetime of the data on the NLDA will typically be on the order of six months. In addition, when data are read from the NLDA during the lifetime, stripes are repaired and written back to the NLDA to restore full redundancy, this occurs at little additional cost. More aggressive repair strategies would require the complete read of data for each single erasure that was detected. The likelihood of a data loss for traditional stripe sizes, e.g., $K = 10, M = 4$, as shown in Silberstein *et al.*, by delaying repair until there were two erasures was still 100 times lower than three replicas while reducing repair bandwidth by a factor of 12. Adding one more redundant shard reduced the repair bandwidth by a factor of 20 with slightly less likelihood of data loss. However, using immediate erasure repair lowered the likelihood of data loss by another 1,000 times.

We envision a use case of the NLDA where the system owners determine the policy for data reliability based on their SLA with their users. For example, the policy could be that users must delete or archive their data within three months and assuming that having $RS(120, 12)$ would provide protection with lazy repair for up to three months.

Extending the example to an SLA that leads to a six month tenancy limitation which we assume could be provided with a configuration of $RS(120, 18)$. And the last illustration of this use case would be to meet an SLA that provides nine months of tenancy with an assumed $RS(120, 24)$ configuration. There would still be a risk that outliers would occur due to the many threats on storage media so mitigations could be provided to repair the data automatically when a preset threshold has been reached. Thus, this model for lazy repair provides ways for system owners to determine the best configuration for their SLAs and assurances that they have detective means to avert data loss for extreme conditions.

The architecture for Lazy Repair would operate in two modes. In the normal mode, data are repaired when the user reads a stored item from the object storage system. In this case, the system would select K good shards from each stripe in the data item, repair each stripe, send the K elements of the data stripe on to the user application, write the repaired shards from the $K + M$ in the stripe back to the object storage system. This would restore the stripe to full protection with $K + M$ good shards. In the protection mode, the system would be notified when the number of defective shards reaches a limit set by the system administrators. When the limit value is reached, a read of the stripe would be scheduled on one of the FTA nodes which would read K good shards, repair the failed shards, and write the repaired shards back to the object store. The protection mode would only be invoked once the number of failures reached the limit which would reduce the number of repair jobs for a stripe. For example, if the stripe configuration was $RS(120, 24)$ and the protection limit was set to 18, then the repair of the previous 17 shard failures would have been avoided saving the need to read the full stripe of data that 17 times.

As an example, we assume a storage system that starts up with 1,440 10 TB disk drives and we store our data in 256 MB objects. No two objects in a single stripe are stored on the same disk drive and the objects in the stripes are uniformly distributed across the 1,440 disk drives in the system. In our model, we do not replace a disk that

fails and we assume that all of the data was stored in the storage system at startup. This gives us 240,000 stripes in the system with $RS(20, 4)$ and 400,000 stripes in the system with $RS(120, 24)$.

Using the combinatorial models from other research [136], we compare the likelihood of data loss of the $RS(20, 4)$ configuration with the $RS(120, 24)$ configuration to show there is a significant difference in the likelihood of failure between the two configurations. We assume that there are 40,000 256 MB objects stored on each disk so $p(k) \times 240,000 \approx 10^{-4}$ in Equation 3.7 and the likelihood of data loss is $1 - p(k) \approx \frac{1}{10^6}$. In Equation 3.8 we compute the probability of a data loss for a single stripe with the $RS(120, 24)$ configuration is 10^{-26} so, with 40,000 stripes in the system, we have the likelihood of data loss of $1 - p(k) \times 40,000 \approx \frac{1}{10^{22}}$.

$$p(k) = \frac{\binom{24}{5}}{\binom{1440}{5}} \approx \frac{10^4}{10^{13}} \approx 10^{-9} \tag{3.7}$$

$$p(k) = \frac{\binom{144}{25}}{\binom{1440}{25}} \approx \frac{10^{27}}{10^{53}} \approx 10^{-26} \tag{3.8}$$

CHAPTER 4

A NEARLINE DISK ARCHIVE STORAGE FOR HPC

4.1 Introduction

In this chapter we describe our configuration and testing methodology for our nearline disk archive storage system for high performance computing. We describe the software features that allow us to select from available erasure coding libraries in Ceph, our platform and the results of our measurements. When we compare the results with the research reported in chapter 2 we see that the performance of the NLDA using the Gibraltar erasure coding on the FTAs is the best solution.

4.2 Test Environment

We have implemented our features in Ceph which provides a convenient Application Programming Interface (API) that we have used to implement our erasure coding on the FTAs. The API also allowed us to use a plugin feature to integrate the Gibraltar library into our environment as shown in Figure 4.1. This plugin is also used for several erasure coding libraries that are already implemented in Ceph including the Intel® ISA-L. This feature provided a very convenient way to compare the performance of several erasure coding libraries. Since the Intel® ISA-L performed the best, we have only included these results in our comparisons. Data are stored in a Ceph data structure called a bufferlist which allows the Ceph product to manage the system memory efficiently. Data being stored by the application is copied into K bufferlist data structures which are then referenced by the API call for the erasure coding to be performed. When the library has completed the erasure coding, it returns after creating M additional bufferlist objects that

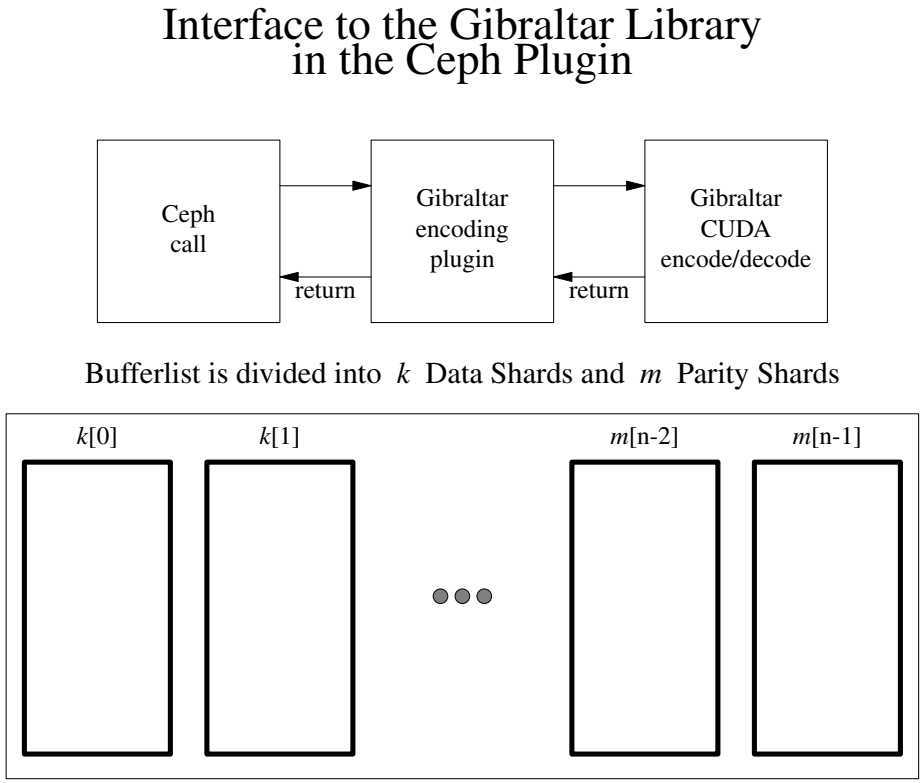

FIGURE 4.1: Ceph Erasure Code Plugin — Ceph calls the erasure code plugin to configure the erasure coding library that is chosen by the application. Subsequent calls are made via a handle to generate erasure coding shards or to repair data stripes.

TABLE 4.1: Dell R730 Servers.

CPUs	2x Intel® Xeon® E5-2650 v3 @ 2.3 GHZ (Hyperthread-enabled: 40 threads)
RAM	128 GB 2133 MT/s RDIMM
Network	Intel® X520 DP 10Gb DA/SFP+, I350 DP 1Gb Ethernet
System Drives	2x 300 GB 10K SAS 2

store the redundant erasure coding. The erasure repair process is similar except that the library reconstructs erasures to the application.

We have built a test bed in order to measure the performance of the NLDA on a small scale based on the Ceph Object storage system product. The test bed consists of four OSSs and three FTAs. These servers are connected via a 10 GbE network fabric and we have bonded two ports together on each machine to get 20 GbE of network bandwidth. All servers have two Intel E5-2660 v3 processors and 128 GB of memory. The FTAs have NVIDIA® K40 GPUs installed. The OSSs each have Serial Attached SCSI (SAS) Host Bus Adapters (HBAs). SAS enclosures house 98 six TB archive class disk drives for each OSS. The general configuration of all seven servers is shown in Table 4.1.

4.3 Measurements

4.3.1 Baseline Performance of Ceph Erasure Coding

We first measured the performance of erasure coding between ISA-L, Jerasure, and Gibraltar using a 1 MB data block with $RS(10,2)$, $RS(10,3)$, and $RS(10,4)$. This benchmark matched the performance that we were able to get from the Ceph community for the ISA-L and Jerasure algorithms [171]. The results of our experiments are shown in Figure 4.2. The results did not provide any motivation to use the Gibraltar library over the Jerasure or ISA-L implementations. After reviewing work done at LANL [147], we repeated the experiments with 1 GB data blocks and found that Gibraltar performed slightly better than Jerasure and ISA-L. We also observed that Gibraltar out performed Jerasure and ISA-L as the number of shards were increased. With this insight, we focused our attention on the use of a larger number of shards.

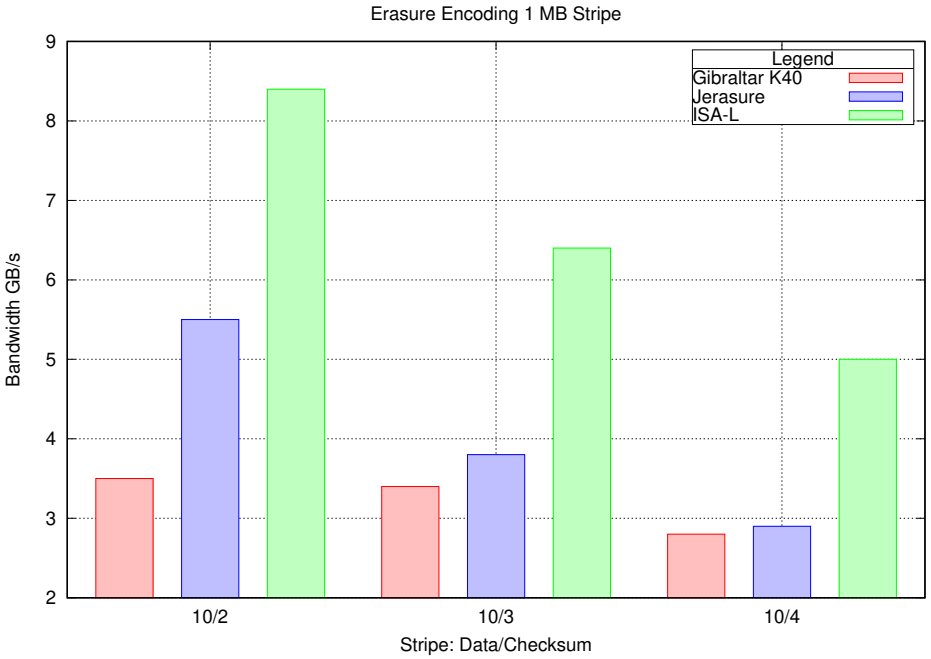

FIGURE 4.2: Initial Erasure coding bandwidth — Erasure coding with 1 MB stripes, $K = 10, M = 2$, $K = 10, M = 3$, and $K = 10, M = 4$.

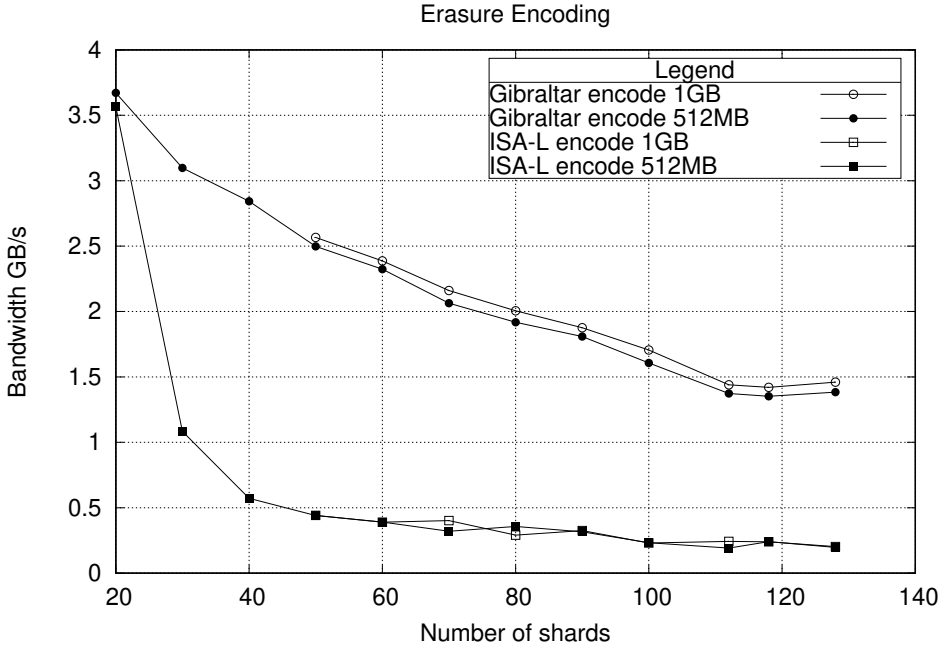

FIGURE 4.3: Erasure coding bandwidth results with increasing number of shards. Coding shards are held to a ratio of one coding shard to five data shards.

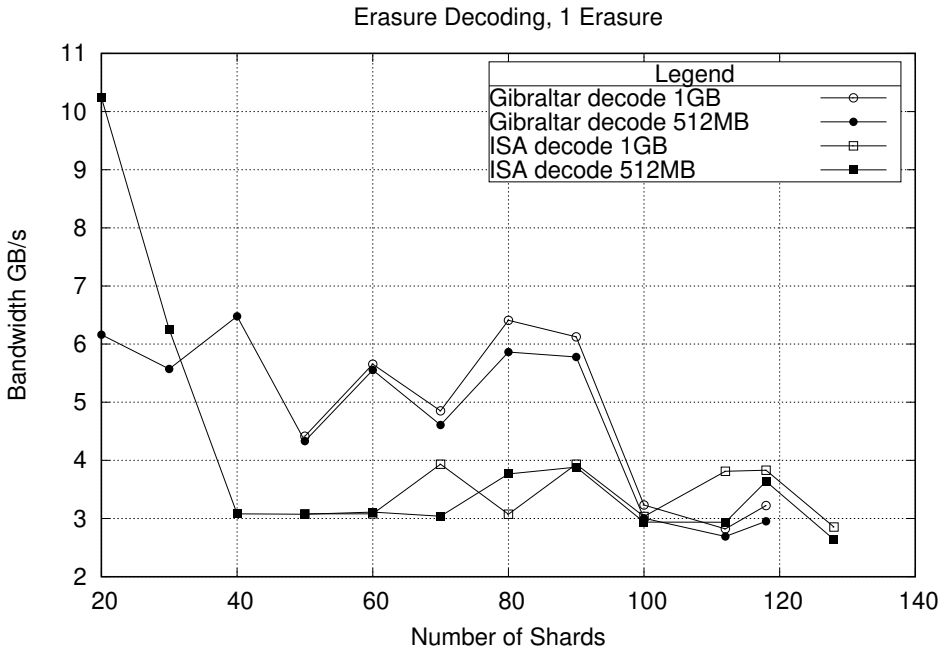

FIGURE 4.4: Erasure recovery bandwidth results with increasing number of shards and one erasure. Coding shards are held to a ratio of one coding shard to five data shards.

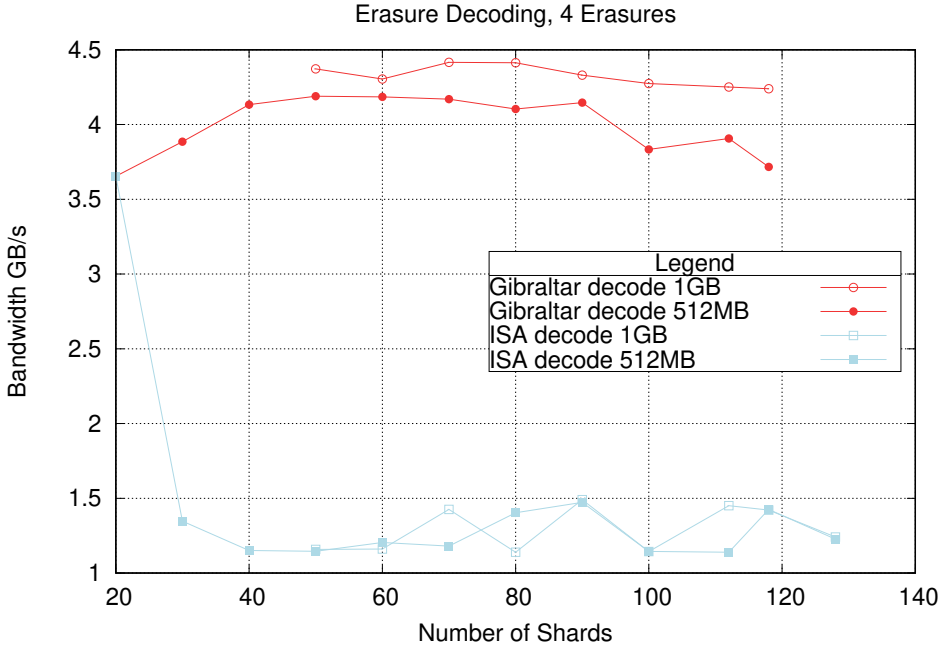

FIGURE 4.5: Erasure recovery bandwidth with 4 erasures — Erasure recovery bandwidth results with increasing number of shards and four erasures. Coding shards are held to a ratio of one coding shard to five data shards.

We measured erasure coding and decoding performance to compare results between the Gibraltar library [22] and the Intel® ISA-L [169] using the NVIDIA® GPU and Intel® Xeon® CPU to determine the best configuration for providing a high bandwidth method for our NLDA. In Figure 4.3 we show the encoding performance for stripe sizes of 512 MiB and 1 GiB. With RS(20,4), the two implementations perform about the same. As the stripe size increases, the ISA-L implementation performance drops quickly performing below 500 MiB/s after a stripe size of RS(40,8). The Gibraltar implementation is able to perform above 1.5 GiB/s until the stripe size exceeds RS(100,20) and performs a nearly 1.5 GiB/s through the last measured stripe size of RS(128,25). We measured the decoding performance to determine how the implementation would be able to recover erasures when the user reads from the NLDA showing the results for repairing a single erasure in Figure 4.4. A second measurement was made showing the results when recovering four erasures in Figure 4.5. The ISA-L repaired the single erasure with RS(20,4) at over 10 GiB/s while the Gibraltar implementation performed this operation in over 6 GiB/s. The performance for larger stripe sizes shows that the ISA-L implementation drops to between 3 and 4 GiB/s throughout the remainder of the test while the Gibraltar implementation performs between 4.5 and 6 GiB/s until RS(100,20) where it drops to between 3 and 4 GiB/s. Both the 512 MiB and 1 GiB Gibraltar stripes would provide more than 3.5 GiB/s performance for repairing 4 erasures using stripe sizes up to RS(128,5).

4.3.2 Performance of Storing Data in Object Storage

We measured the native Ceph erasure coded pool performance to provide a baseline against to compare our FTA erasure coding architecture. We only measure the performance of the ISA-L for this baseline because we do not implement the Gibraltar library on the Ceph storage cluster. This is a measurement of the native Ceph capability. In Figure 4.6 we show the network utilization and in Figure 4.7 we show the CPU utilization. The network utilization and CPU performance were also measured using the FTAs to

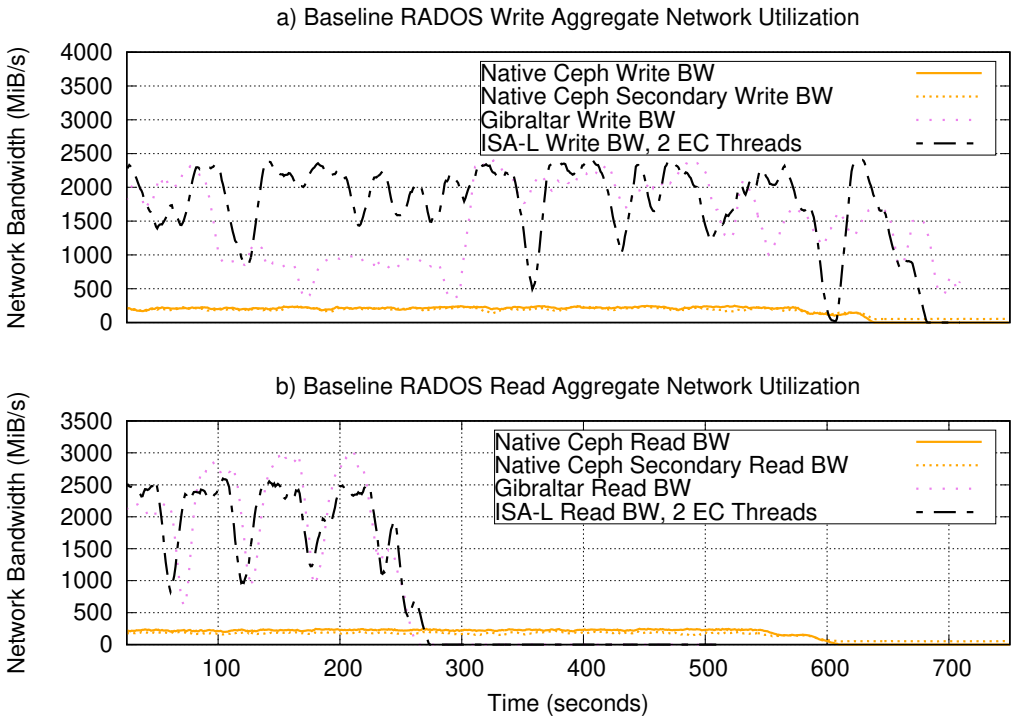

FIGURE 4.6: Baseline comparison of Ceph network performance — Comparison of Ceph network write (graph a) and read (graph b) performance between native Ceph erasure Coded pool with erasure coding on the FTAs. K=20, M=4, one thread of Gibraltar encoding, two threads of ISA-L encoding.

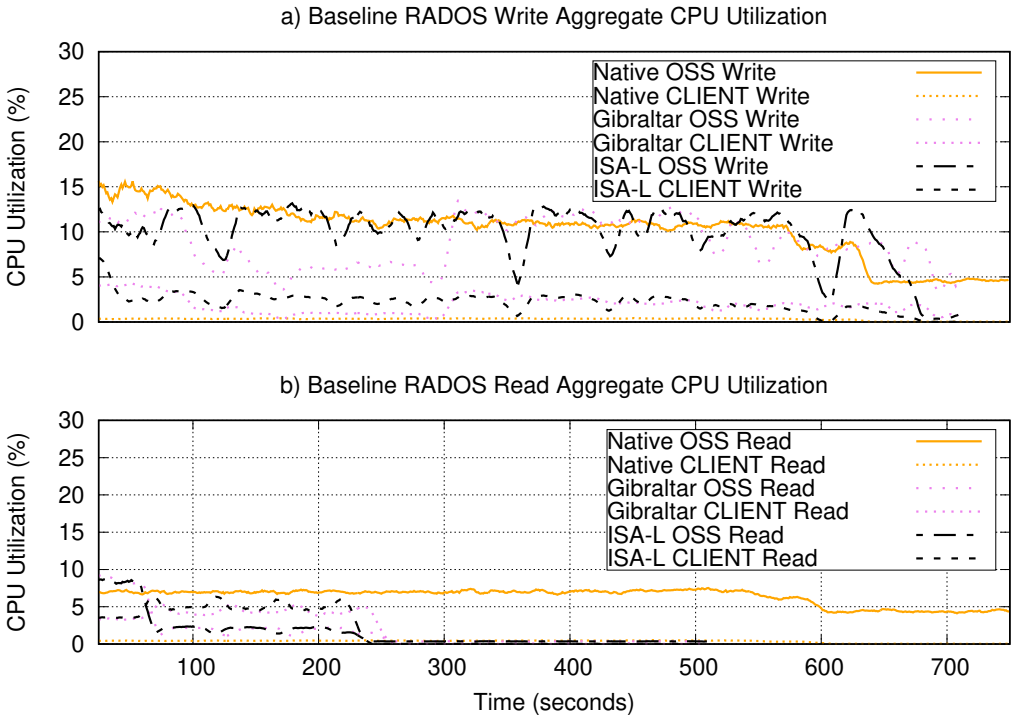

FIGURE 4.7: Baseline comparison of Ceph CPU performance — Comparison of Ceph CPU write (graph a) and read (graph b) performance between native Ceph erasure coded pool with erasure coding on the FTAs, K=20, M=4, one thread of Gibraltar encoding, two threads of ISA-L encoding.

perform the erasure coding. All of these were performed using RS(20,4) which was the largest stripe that we could use on our Ceph object storage system. We ran this test using three FTAs concurrently and have aggregated the total network bandwidth and CPU utilization. The maximum network bandwidth as inferred from the network utilization for the native Ceph erasure coded RS(20,4) pool using ISA-L was only around 250 MiB/s while using the same configuration and performing the erasure coding on the FTAs with the ISA-L implementation, we see peaking at 2.5 GiB/s on read and write. The Gibraltar implementation performed nearly as well as the ISA-L in this test when run on the FTAs where earlier, we showed that both perform erasure coding at nearly the same rate with RS(20,4). Both the Gibraltar implementation and the ISA-L implementation performed reads of the data at near the same rate except Gibraltar peaked around 3 GiB/s during the reads while the native ceph implementation peaked around 250 MiB/s during the read operations. The reads for this test did not measure any repair activity, e.g., these were only the K data shards being read. The CPU utilization shows that erasure coding load on the FTAs for the Gibraltar implementation and the ISA-L implementation are 3 to 4 % higher than the native Ceph erasure coding, the latter performs only data transfer operations on the FTA, so the compute load for RS(20,4) is very light on this platform. However, the CPU load on the OSSs shows that this light erasure coding load is being performed on the OSS where most of the load comes from the Ceph processes to store the data on the OSDs. The FTA implementations load the OSSs slightly less than the CPU load measured for the native configuration. The erasure coding load in the native Ceph implementation is spread over four OSS nodes while the erasure coding load that was measured for the FTA implementations was measured over three FTAs. The CPU load on the OSSs for erasure coding is expected to be about 8% less than the load on the FTAs.

After showing that the FTA implementation out performs the native Ceph erasure coding implementation, we ran tests to measure the performance with very large stripe sizes choosing RS(120,24) as a meaningful configuration for NLDA which can provide

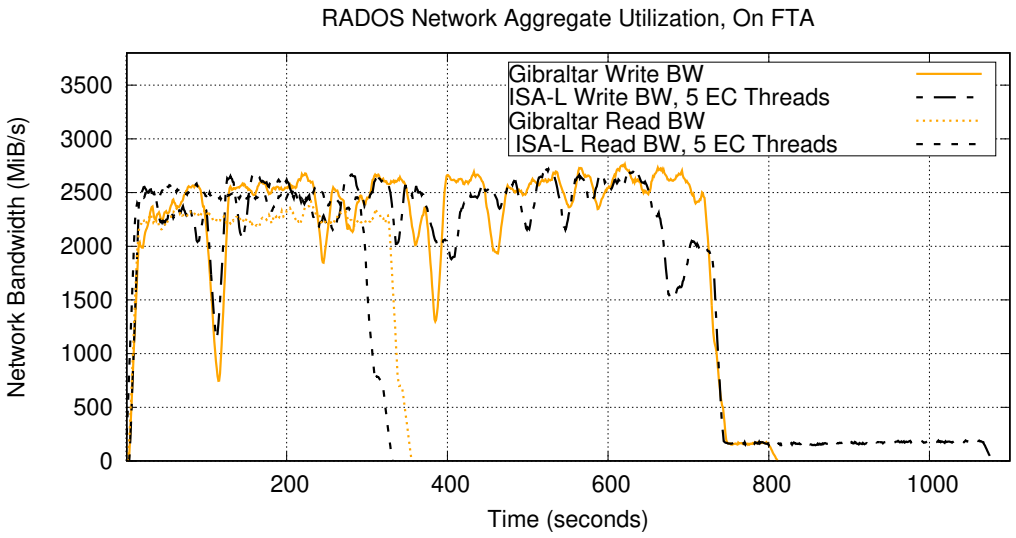

FIGURE 4.8: RADOS network performance with write and read of 600 stripes, K=120, M=24, shard size 8388608 bytes, one thread of Gibraltar encoding compared with five threads of ISA-L encoding.

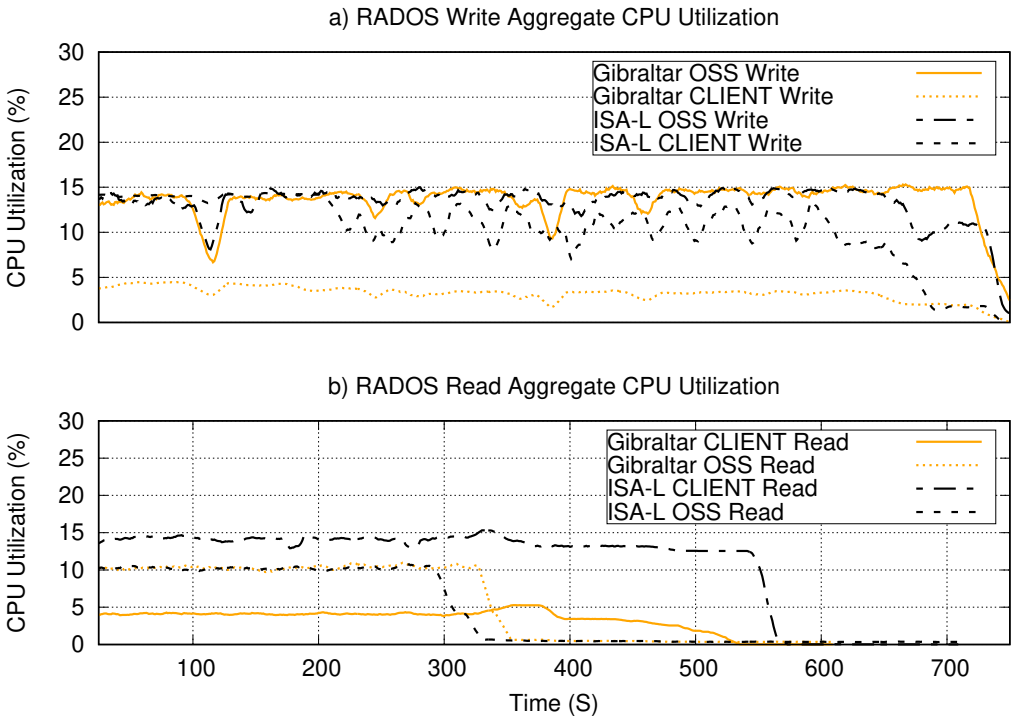

FIGURE 4.9: RADOS CPU performance with write and read of 600 stripes, K=120, M=24, shard size 8388608 bytes, one thread of Gibraltar encoding compared with five threads of ISA-L encoding. Write performance shown in graph a and read performance shown in graph b.

TABLE 4.2: Comparison of Erasure Coding Performance.

Property	Ceph Pool	FTA ISA-L	FTA Gibraltar
Bandwidth Write[a]	.22	1.9	1.7
Bandwidth Read[a]	.23	.79	.84
FTA Write CPU	1%	2.5%	1.7%
FTA Read CPU	1%	1.7%	1.7%
OSS Write CPU	11%	10.9%	8.8%
OSS Read CPU	7%	.9%	.9%

[a]Bandwidth is in GB/S.

stripe data storage sizes of 512 MiB using 4 MiB shards and 1 GiB using 8 MiB shards. In Figure 4.8 we show the network utilization for these tests while writing 1.8 TiB of data from our three FTAs and we show the CPU utilization in Figure 4.9. In these tests, we configured the ISA-L implementation to run five concurrent erasure coding threads in order to produce the same throughput as the single Gibraltar implementation which is only running one stream on the NVIDIA® GPU. In the read test, we simulated the repair work load by having the application rebuild $M = 24$ data shards on each stripe read. The CPU workloads for the OSS servers are similar for both reads and writes between the Gibraltar implementation and the ISA-L implementation as we expect because the workload consists only of performing Ceph processes. On the FTAs, we run five threads of the ISA-L implementation and see that the CPU load is about three times that of the Gibraltar CPU workload. We assume that if we subtract the baseline workload from these measurements we would find that the CPU load for erasure coding on the ISA-L implementation scales linearly with the number of threads and should be about five times the Gibraltar utilization for this configuration. As expected, both implementations produced about the same network utilization during the test writing the 1.8 TiB to the storage system at peaks around 2.5 GiB/s.

In summary, Table 4.2 shows the average performance differences between all of the configurations that we tested. The CPU comparisons with the native Ceph should

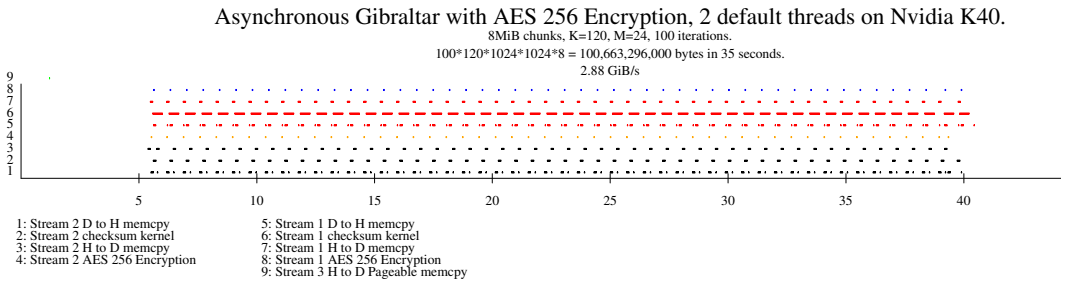

FIGURE 4.10: AES Encryption with Erasure Coding on NVIDIA® K40 Performance — We performed AES 256 encryption and erasure coding on 100 stripes of RS(120,24) with the NVIDIA® K40.

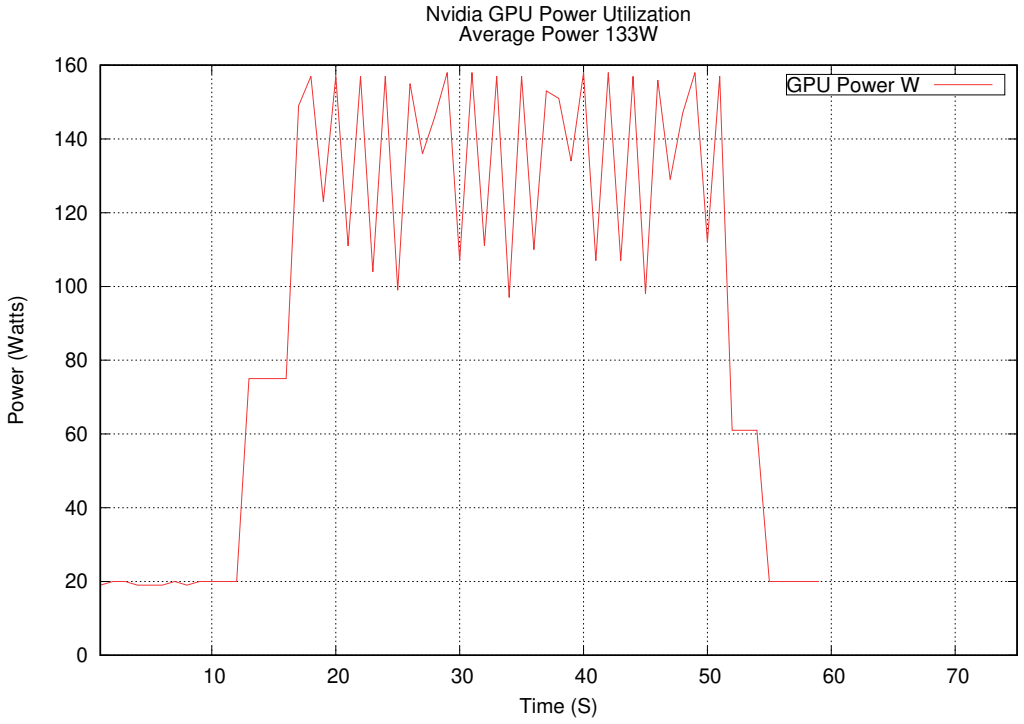

FIGURE 4.11: AES Encryption with Erasure Coding on NVIDIA® K40 Power — The power consumption on the NVIDIA® K40 averaged 133 watts during the full compute load.

be adjusted to account for the difference in the amount of work performed. The FTA implementations performed erasure coding at nearly 10 times the rate of the native Ceph implementation so the native implementation took about 10 times longer to write or read the same amount of data, hence, the CPU utilization is about $1/10$ the utilization that the FTA implementations consumed.

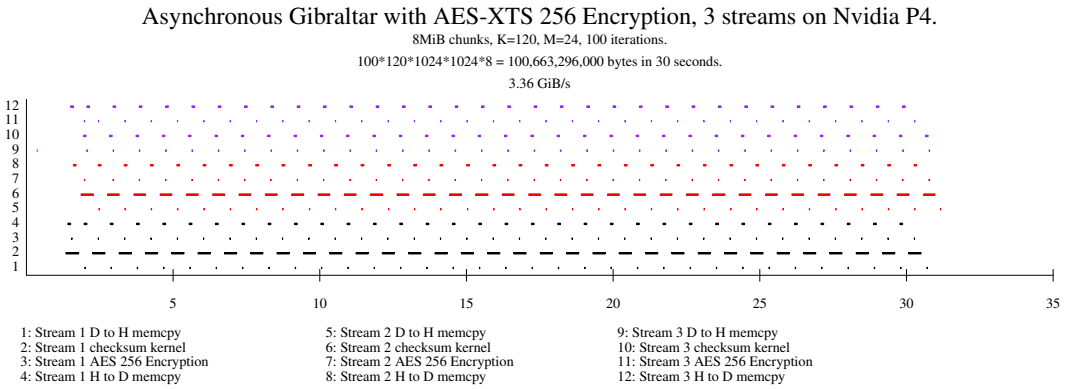

FIGURE 4.12: AES Encryption with Erasure Coding on NVIDIA® P4 Performance — We performed AES 256 encryption and erasure coding on 100 stripes of RS(120,24) with the NVIDIA® P4.

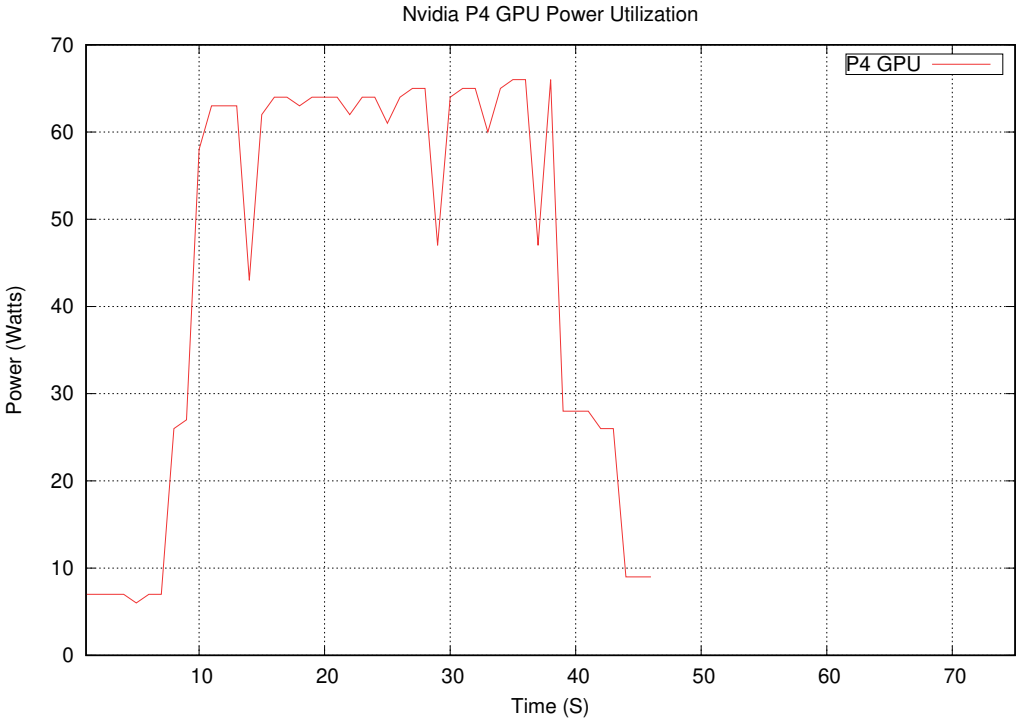

FIGURE 4.13: AES Encryption with Erasure Coding on NVIDIA® P4 Power — The power consumption on the NVIDIA® P4 averaged 55 watts during the full compute load.

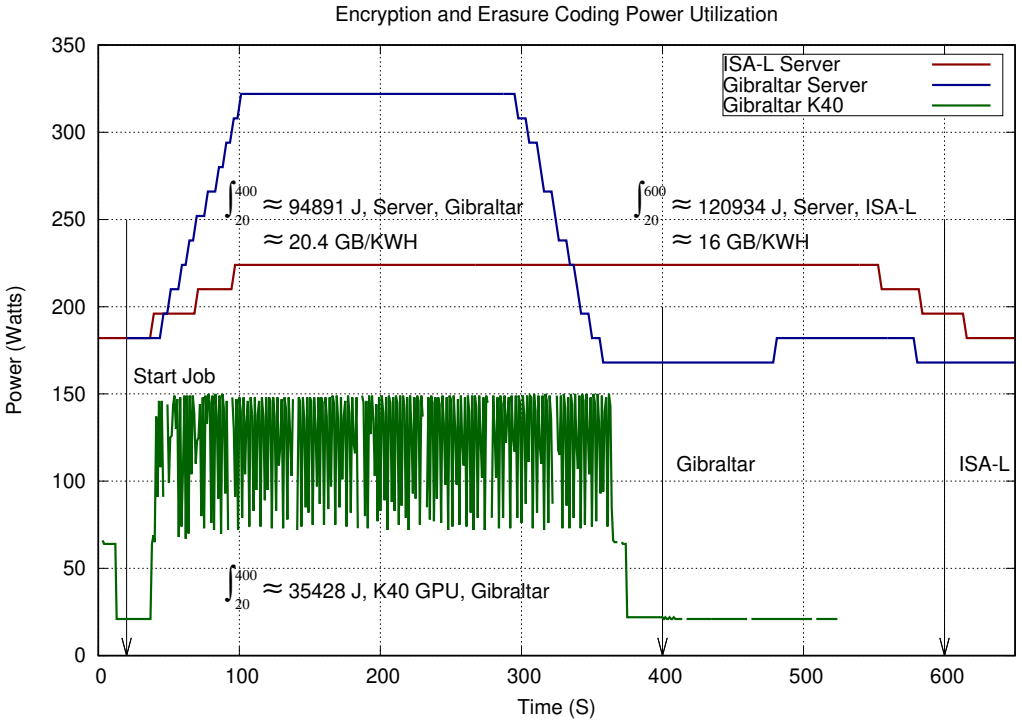

FIGURE 4.14: AES Encryption with Erasure Coding comparing Gibraltar with ISA-L — We performed AES 256 encryption and erasure coding on 500 1 GB stripes of data using $RS(120, 24)$. The Gibraltar K40 encrypted and erasure coded 20.4 GB per KWH while ISA-L performed the same work yielding only 16 GB per KWH.

4.4 AES Encryption with Erasure Coding

We measured the performance of computing AES encryption in addition to erasure coding in Figure 4.10 at 2.88 GiB/s. In Figure 4.11 we show that the power costs of computing encryption and erasure coding on 100 GB averages about 133W for 35 seconds, about 13 mw per GB. With the NVIDIA® P4 GPU in Figure 4.12 we measured 3.36 GiB/s on 100 stripes of RS(120,24) of 1 GiB with a power consumption for the encryption and erasure coding on 100 GB about 55W for 30 seconds, about 4.6 mw per GB shown in Figure 4.13.

In another experiment, we measured the power consumption to perform AES 256 encryption and erasure coding at $RS(120, 24)$ on two Dell servers as specified in Table 4.1. One server was chosen with the NVIDIA® K40 where we encrypted and erasure coded 500 1 GB stripes using Gibraltar. The other server did not have an NVIDIA® K40

installed where we ran the ISA-L using OpenSSL AES 256 encryption and ISA-L for erasure coding. The encryption using OpenSSL was performed using a single thread as it is not thread-safe. We used five threads of the ISA-L to perform the erasure encoding. As shown in Figure 4.14, we also measured the power used by the NVIDIA® K40 using the nvidia-smi tool which shows that 37% of the power is consumed by the NVIDIA® K40 GPU. In this test we show that Gibraltar is 27.4% more efficient than the ISA-L implementation. Based on the comparison between the NVIDIA® K40 Figure 4.11 and P4 shown in Figure 4.13, the P4 should provide even greater power efficiency.

4.5 Lazy Repair

We simulated the operation of a cluster of 1,000 disks using 10 TB disk drives. We used the High-Fidelity Reliability (HFR) Simulator [138, 139] to provide an estimated data loss for various erasure code configurations. We configured the simulator to delay the repair of a failure for 1 year. All of the erasure coding configurations were specified in the simulator's model database. Our parameters for the tests are given in Table 4.3. The results of our experiment are shown in Figure 4.15. Compared to the $RS(20, 4)$ configuration that is commonly in use for object storage systems, which shows an average data loss of 424 bytes, our NLDA with a stripe size of $RS(120, 24)$ shows zero data loss.

TABLE 4.3: Comparison of Erasure Coding Data Loss.

Parameter	Value
Duration	1 year
Iterations	100
Disks	1,000
Disk Size	10 TB
Failure Model	(1.1, 230693)
Repair Model	(1.0, 21474836480.0)

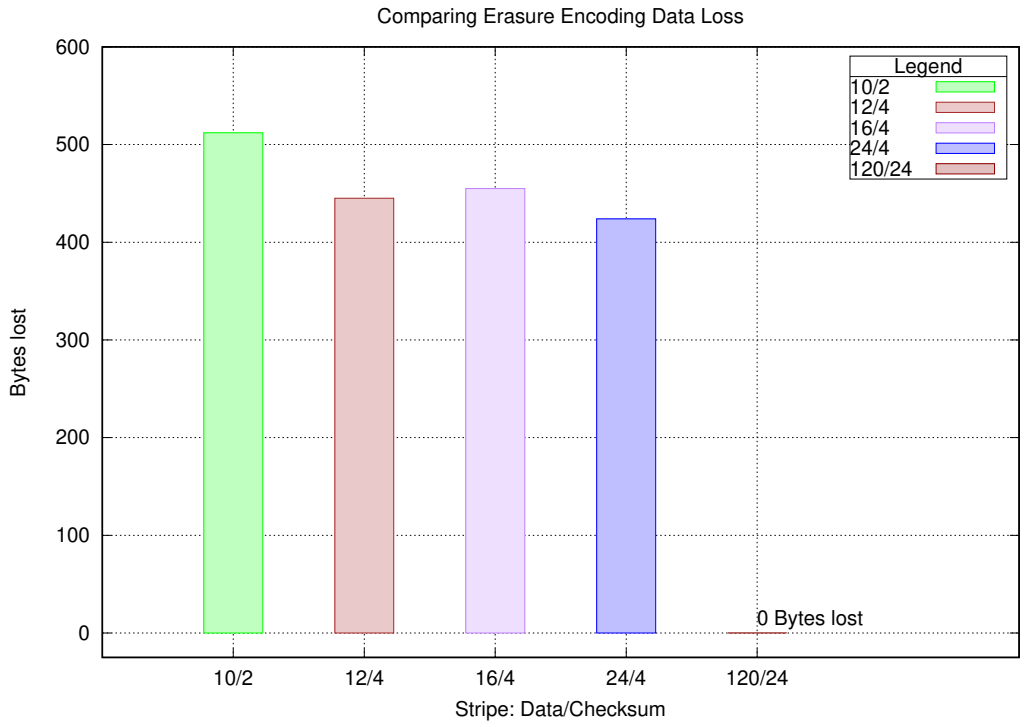

FIGURE 4.15: Erasure Coding Data Loss Simulation — Results from simulation of data loss with repair delayed for 1 year.

CHAPTER 5

SUMMARY

5.1 Dissertation Statement

We design an architecture for nearline disk archive storage that is:

- more performant,
- of lower cost,
- more energy saving,
- more secure,
- and more reliable than existing designs.

5.2 Contributions

In this dissertation we have made the following contributions:

- The FTA erasure coding implementation was shown to be faster than the native Ceph erasure coded pool implementation.
- The Gibraltar implementation was shown to be five times faster than the ISA-L performance.
- We showed that performing erasure coding on the FTAs reduces capital and energy costs.
- We have shown that performing encryption on the FTAs can reduce the scope of the security boundary.

- We have shown that delaying erasure repair with large stripe sizes can be a strategy for reducing the overall compute, storage and network capacity requirements.

5.3 Broader Impacts

The work here has greater impact than we could address in this dissertation. Listed here are three of the major impacts:

- **Feature detection** — Our architecture can be extended to insert feature detection into the erasure coding data flow. The feature detection would provide patterns that describe the features of interest. The feature module would emit properties of the detected features including the location, size, range, etc., which could be provided to the metadata service.

- **HDF5** — HDF5 is a data model that is well supported with tools. The HDF5 file format allows for storing metadata with the data as well as providing multiple streams for parallel IO. HDF5 can be implemented on our architecture to provide a low cost, high performance object storage platform for large datasets. For example, the USAF awarded a Phase I SBIR to the HDF group to provide a prototype of storing the Chapter 10 IRIG 106 data sets to HDF5.

- **Accumulo** — Accumulo is a NOSQL database that is an open source Apache project. Some implementations are using over 200 PB of HDFS storage. Our architecture could be used to move colder "tablets" to object storage until needed, reducing the amount of online Hadoop storage that is required. Another possibility would be to classify the Hadoop nodes as FTAs and include the function of moving data to the object storage system.

5.4 Future Work

There are opportunities to extend the research of this dissertation further. We give these four areas which should have high value:

- **Models for sizing HPC storage** — With the continuing changes in the price, capacity, and performance in the storage media markets, we need to have tools to help in planning the allocation of storage system resources to meet the data center requirements in a cost effective way. The model needs to consider the performance time, the costs, the physical dimensions, physical weight, power consumption and cooling requirements. Because some data life times exceed that of the media service life, the data migration process must be included in the model.

- **Encryption library for GPU** — We have shown the practicality of including encryption with erasure coding on the GPU with the Gibraltar erasure coding library using the AES-CBC 256. However, there are several other encryption algorithms and key sizes that might be needed by the users of the systems. The development of other encryption options would make this capability more useful.

- **Data Compression** — Data compression of the data stream on the FTA is a hard problem and would require domain specific knowledge to provide requirements for specific use cases. However, data compression on GPUs has been done in research. Also, studies have been done on the cost effectiveness of compressing data for HPC storage at the DKRZ in Germany where they compared the use cases of the user compressing the data while on the scratch workspace before moving it to archive vs performing the compression enroute to the near line disk archive storage. Furthermore, tape drives used for long term archive storage provide compression as a common feature, for example, an LTO-8 tape stores 12 TB raw but up to 30 TB compressed. These tape drives also provide encryption of the data as well.

- **Hadoop storage in NLDA** — Since Hadoop and related applications are generally based on pushing compute down to the data storage, there role is very similar to the FTA. Providing an HDFS abstraction over the NLDA architecture that we have provided should be able to perform well in many use cases. For example, where there are HDFS clusters greater than 10 or 20 PB, these might be implemented using our architecture and provide nearly equivalent performance at lower cost.

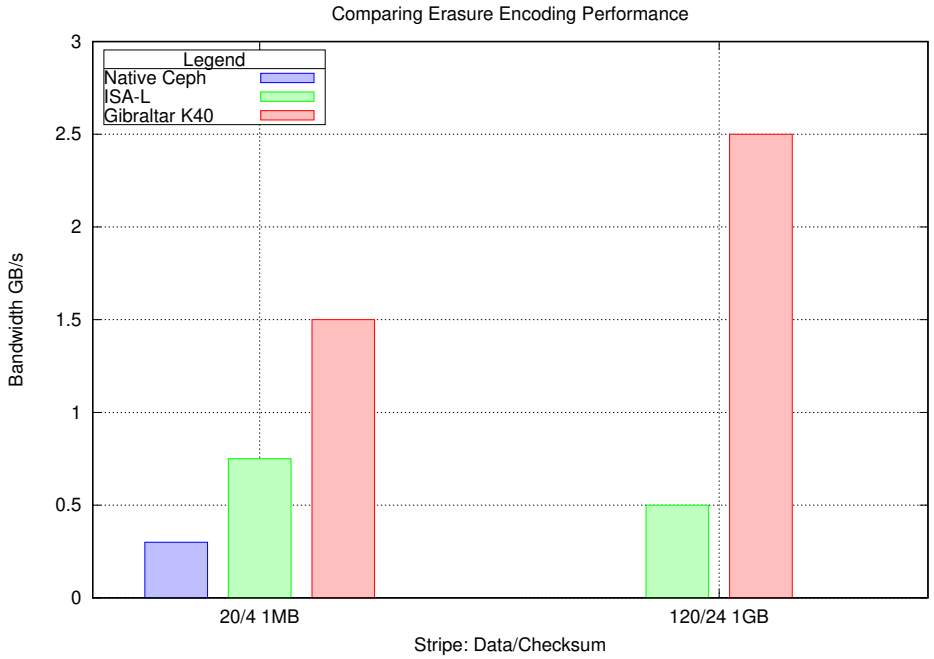

FIGURE 5.1: Erasure coding bandwidth comparison — Erasure coding with native Ceph, ISA-L, and Gibraltar. $K = 20, M = 4$ and $K = 120, M = 24$.

Because we had assumed an immutable constraint for the NLDA architecture, this may need to be relaxed for the HDFS file system, especially if the application, like Accumulo, would be storing data in HDFS. However, if the application was read only, then the constraint would not need to be relaxed.

5.5 Conclusion

We have designed an architecture for nearline disk archive storage that is more performant, of lower cost, more energy saving, more secure, and more reliable than existing designs. As shown in Figure 5.1 we have provided an architecture that outperforms the native Ceph object storage system performance. In summary, we have satisfied our objectives in the following ways:

- We demonstrated that the FTA erasure coding implementation was faster than Ceph's erasure coded pool implementation.

- We demonstrated that the Gibraltar implementation was faster than the ISA-L performance.
- We described how performing erasure coding on the FTAs reduces capital and energy costs.
- We gave an example of how performing encryption on the FTAs can reduce the scope of the security boundary.
- We have explained how delaying erasure repair with large stripe sizes can be a strategy for reducing the overall compute, storage and network capacity requirements.

LIST OF REFERENCES

[1] M. Mesnier, G.R. Ganger, and E. Riedel. Storage area networking - Object-based storage. *IEEE Communications Magazine*, 41(8):84–90, August 2003. ISSN 0163-6804. doi: 10.1109/MCOM.2003.1222722. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1222722.

[2] Zuse Institute Berlin (ZIB). Fairness and Load in Distributed File Systems, 2016. URL http://www.zib.de/features/fairness-and-load-distributed-file-systems.

[3] David Morton. Trinity: DataManagement Scheme and Performance. American Institute of Aeronautics and Astronautics, January 2017. ISBN 978-1-62410-447-3. doi: 10.2514/6.2017-0812. URL http://arc.aiaa.org/doi/10.2514/6.2017-0812.

[4] Gary Grider. Storage Lessons from HPC: A Multi-Decadal Struggle, September 2018. URL https://www.snia.org/sites/default/files/SDC/2018/presentations/General_Session/Grider_Gary_Storage_Lessons_from_HPC_A_Multi-Decadal_Struggle.pdf.

[5] LANL. Crossroads 2021 Technical Requirements Document, July 2018. URL https://www.lanl.gov/projects/crossroads/Exhibit%20D%20Technical%20Specifications%20Document%201-16.docx.

[6] Gary Grider. MarFS, May 2015. URL http://storageconference.us/2015/Presentations/Grider.pdf.

[7] ACES Team. Trinity Platform Introduction and Usage Model, August 2015. URL https://www.lanl.gov/projects/trinity/_assets/docs/trinity-usage-model-presentation.pdf.

[8] www.cray.com. WP-Cray-Sonexion-3000-Storage-0616www.cray.comCray® Sonexion® 3000 Storage System, June 2016. URL https://www.cray.com/sites/default/files/resources/WP-Cray-Sonexion-3000-Storage-Systems.pdf.

[9] Tom Coughlin. The Costs Of Storage, June 2016. URL https://www.forbes.com/sites/tomcoughlin/2016/07/24/the-costs-of-storage/#2630382f3239.

[10] James Byron, Darrell D. E. Long, and Ethan L. Miller. Using Simulation to Design Scalable and Cost-Efficient Archival Storage Systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 25–39, Milwaukee, WI, September 2018. IEEE. ISBN 978-1-5386-6886-3. doi: 10.1109/MASCOTS.2018.00011. URL https://ieeexplore.ieee.org/document/8526869/.

[11] Nicole Hemsoth. The Slow Death of the Parallel File System. *The Next Platform*, January 2016. URL https://www.nextplatform.com/2016/01/12/the-slow-death-of-the-parallel-file-system/.

[12] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[13] John Bent. The Correct Number of Tiers is Two, July 2017. URL http://lustre.ornl.gov/ecosystem-2017/documents/Day-2_Keynote-2_Bent.pdf.

[14] John Bent, B Settlemeyer, and G Grider. Serving Data to the Lunatic Fringe. *;login:*, 41(2):34–38, 2016. URL https://www.usenix.org/system/files/login/articles/login_summer16_08_bent.pdf.

[15] Center for Medicare and Medicaid Services. HIPAA Security Series - Security Standards: Technical Safeguards. 2(4), March 2007. URL https://www.hhs.gov/sites/default/files/ocr/privacy/hipaa/administrative/securityrule/techsafeguards.pdf.

[16] CGS Team. CGS Data Protection Capability, July 2012. URL https://iase.disa.mil/cgs/Documents/Data_Protection_v1.1.1.pdf.

[17] National Security Agency. Information Assurance Capabilities - Data at rest capability package, January 2018. URL https://www.nsa.gov/Portals/70/documents/resources/everyone/csfc/capability-packages/dar-cp.pdf.

[18] National Institute of Standards and Technology. Minimum security requirements for federal information and information systems. Technical Report NIST FIPS 200, National Institute of Standards and Technology, Gaithersburg, MD, March 2006. URL https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.200.pdf.

[19] Elaine B. Barker, William C. Barker, William E. Burr, William T. Polk, and Miles E. Smid. Recommendation for Key Management - Part 1: General (Revision 3), July 2012. URL http://www.nist.gov/customcf/get_pdf.cfm?pub_id=910342.

[20] William C. Barker, Elaine B. Barker, William E. Burr, William T. Polk, and Miles E Smid. Recommendation for Key Management Part 2: Best Practices for Key Management Organization, August 2005. URL http://www.nist.gov/customcf/get_pdf.cfm?pub_id=151315.

[21] Elaine B. Barker, William E. Burr, Alicia Clay Jones, William T. Polk, Scott W. Rose, Miles E Smid, and Quynh H. Dang. NIST Special Publication 800-57 Recommendation for Key Management Part 3: Application-Specific Key Management Guidance, December 2009. URL http://www.nist.gov/customcf/get_pdf.cfm?pub_id=903633.

[22] Matthew L. Curry, Anthony Skjellum, H. L. Ward, and Ron Brightwell. Gibraltar: A Reed-Solomon coding library for storage applications on programmable graphics processors. *Concurrency and Computation: Practice and Experience*, 23(18):2477–2495, December 2011. ISSN 15320626. doi: 10.1002/cpe.1810. URL http://doi.wiley.com/10.1002/cpe.1810.

[23] Walker Haddock, Matthew L Curry, Purushotham V Bangalore, and Anthony Skjellum. GPU Erasure Coding for Campaign Storage. In *International Conference on High Performance Computing*, pages 145–159, Frankfurt, Germany, June 2017. Springer.

[24] Walker Haddock, Purushotham V. Bangalore, Matthew L. Curry, and Anthony Skjellum. High Performance Erasure Coding for Very Large Stripe Sizes. In *2019 Spring Simulation Conference (SpringSim)*, pages 1–12, Tucson, AZ, USA, April 2019. IEEE. ISBN 978-1-5108-8388-8. doi: 10.23919/SpringSim.2019.8732912. URL https://ieeexplore.ieee.org/document/8732912/.

[25] Elwyn R. Berlekamp. *Algebraic coding theory*. Aegean Park Press, Laguna Hills, Calif, 2. rev., ed edition, 1984. ISBN 978-0-89412-063-3. OCLC: 10787423.

[26] Matthew L. Curry, Anthony Skjellum, H. L. Ward, and Ron Brightwell. Accelerating Reed-Solomon coding in RAID systems with GPUs. In *Proceedings of the 2008 IEEE international parallel & distributed processing symposium*, pages 1–6, Miami, FL, USA, April 2008. IEEE. ISBN 978-1-4244-1693-6. doi: 10.1109/IPDPS.2008.4536322. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4536322.

[27] Matthew L. Curry, H. L. Ward, Anthony Skjellum, and Ron Brightwell. A Lightweight, GPU-Based Software RAID System. In *2010 39th International Conference on Parallel Processing*, pages 565–572, San Diego, CA, USA, September 2010. IEEE. ISBN 978-1-4244-7913-9. doi: 10.1109/ICPP.2010.64. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5599249.

[28] Curry, Matthew L. *A highly reliable GPU-based RAID system*. PhD thesis, University of Alabama at Birmingham, 2010. URL http://contentdm.mhsl.uab.edu/cdm/ref/collection/etd/id/854.

[29] A.D. Hospodor and A.S. Hoagland. The changing nature of disk controllers. *Proceedings of the IEEE*, 81(4):586–594, April 1993. ISSN 00189219. doi:

10.1109/5.219343. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=219343.

[30] S.S. Coleman and R.W. Watson. The emerging paradigm shift in storage system architectures. *Proceedings of the IEEE*, 81(4):607–620, April 1993. ISSN 00189219. doi: 10.1109/5.219345. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=219345.

[31] Li-Pin Chang. A Hybrid Approach to NAND-Flash-Based Solid-State Disks. *IEEE Transactions on Computers*, 59(10):1337–1349, October 2010. ISSN 0018-9340. doi: 10.1109/TC.2010.14. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5383350.

[32] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun network filesystem. In *Proceedings of the Summer USENIX conference*, pages 119–130, 1985.

[33] B. Phillips. Have storage area networks come of age? *Computer*, 31(7):10–12, July 1998. ISSN 0018-9162. doi: 10.1109/2.689672.

[34] Garth A. Gibson and Rodney Van Meter. Network Attached Storage Architecture. *Commun. ACM*, 43(11):37–45, November 2000. ISSN 0001-0782. doi: 10.1145/353360.353362. URL http://doi.acm.org/10.1145/353360.353362.

[35] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988. ISSN 07342071. doi: 10.1145/35037.35059. URL http://portal.acm.org/citation.cfm?doid=35037.35059.

[36] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. Rosenthal, and F. Donelson Smith. Andrew: a distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986. ISSN 00010782. doi: 10.1145/5666.5671. URL http://portal.acm.org/citation.cfm?doid=5666.5671.

[37] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda: a highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990. ISSN 00189340. doi: 10.1109/12.54838. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=54838.

[38] Luis-Felipe Cabrera and Darrell D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computing Systems*, 4(4):405–436, 1991.

[39] Howard Gobioff, Garth Gibson, and Doug Tygar. Security for network attached storage devices. Technical report, DTIC Document, 1997.

[40] Garth A. Gibson, Jim Zelenka, David F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, and David Rochberg. A cost-effective, high-bandwidth storage architecture. *ACM SIGPLAN Notices*, 33(11):92–103, November 1998. ISSN 03621340. doi: 10.1145/291006.291029. URL http://portal.acm.org/citation.cfm?doid=291006.291029.

[41] Zvi Dubitsky, Israel Gold, Ealan Henis, Julian Satran, and Dafna Sheinwald. DFS - Data Sharing Facility. IBM Research Report H-0141, IBM Research Division Haifa Research Laboratory, Haifa 31905, Israel, October 2002.

[42] O. Rodeh and A. Teperman. zFS - a scalable distributed file system using object disks. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings.*, pages 207–218, San Diego, CA, USA, 2003. IEEE Comput. Soc. ISBN 978-0-7695-1914-2. doi: 10.1109/MASS.2003.1194858. URL http://ieeexplore.ieee.org/document/1194858/.

[43] Dean Hildebrand and Peter Honeyman. Exporting storage systems in a scalable manner with pNFS. In *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on*, pages 18–27. IEEE, 2005.

[44] P. F. Corbett, D. G. Feitelson, J.-P. Prost, G. S. Almasi, S. J. Baylor, A. S. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. D. Herr, J. Kavaky, T. R. Morgan, and A. Zlotek. Parallel file systems for the IBM SP computers. *IBM Systems Journal*, 34(2):222–248, 1995. ISSN 0018-8670. doi: 10.1147/sj.342.0222. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5387272.

[45] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thaku. PVFS: A parallel file system for Linux clusters. In *Proc. of 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, October 2000. USENIX Assoc.

[46] David Goodell, Seong Jo Kim, Robert Latham, Mahmut Kandemir, and Robert Ross. An Evolutionary Path to Object Storage Access. pages 36–41. IEEE, November 2012. ISBN 978-0-7695-4956-9 978-1-4673-6218-4. doi: 10.1109/SC.Companion.2012.17. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6495799.

[47] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. pages 1–10. IEEE, May 2010. ISBN 978-1-4244-7152-2. doi: 10.1109/MSST.2010.5496972. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5496972.

[48] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. page 29. ACM Press, 2003. ISBN 978-1-58113-757-6. doi: 10.1145/945445.945450. URL http://portal.acm.org/citation.cfm?doid=945445.945450.

[49] Frank B Schmuck and Roger L Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *FAST*, volume 2, page 19, 2002.

[50] Brent Welch, Marc Unangst, Zainul Abbasi, Garth A Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable Performance of the Panasas Parallel File System. In *FAST*, volume 8, pages 1–17, 2008.

[51] Brent Welch and Garth A Gibson. Managing Scalability in Object Storage Systems for HPC Linux Clusters. In *MSST*, pages 433–445. Citeseer, 2004.

[52] Peter J Braam and Philip Schwan. Lustre: The intergalactic file system. In *Ottawa Linux Symposium*, page 50, 2002.

[53] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[54] Julian Martin Kunkel, Michael Kuhn, and Thomas Ludwig. Exascale Storage Systems — An Analytical Study of Expenses. *Supercomputing Frontiers and Innovations*, 1(1), September 2014. ISSN 23138734. doi: 10.14529/jsfi140106. URL http://superfri.org/superfri/article/view/20.

[55] Hal Jespersen. POSIX Retrospective. *StandardView*, 3(1):2–10, March 1995. ISSN 1067-9936. doi: 10.1145/210308.210313. URL http://doi.acm.org/10.1145/210308.210313.

[56] IEEE Computer Society, Portable Applications Standards Committee, England) Open Group (Reading, and Institute of Electrical and Electronics Engineers. *Standard for information technology: portable operating system interface (POSIX) : base specifications, issue 7*. 2013. ISBN 978-0-7381-8331-2 978-1-937218-28-7. URL http://ieeexplore.ieee.org/servlet/opac?punumber=6506089.

[57] S.A. Weil, K.T. Pollack, S.A. Brandt, and E.L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. pages 4–4. IEEE, 2004. ISBN 0-7695-2153-3. doi: 10.1109/SC.2004.22. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1392934.

[58] Michael A. Sevilla, Noah Watkins, Carlos Maltzahn, Ike Nassi, Scott A. Brandt, Sage A. Weil, Greg Farnum, and Sam Fineberg. Mantle: a programmable metadata load balancer for the ceph file system. pages 1–12. ACM Press, 2015.

ISBN 978-1-4503-3723-6. doi: 10.1145/2807591.2807607. URL http://dl.acm.org/citation.cfm?doid=2807591.2807607.

[59] Jeff Inman, Gary Grider, and Hsing Bung Chen. Cost of Tape versus Disk for Archival Storage. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 208–215, Anchorage, AK, USA, June 2014. IEEE. ISBN 978-1-4799-5063-8 978-1-4799-5062-1. doi: 10.1109/CLOUD.2014.37. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6973743.

[60] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, and L. Yerushalmi. Towards an object store. pages 165–176. IEEE Comput. Soc, 2003. ISBN 0-7695-1914-8. doi: 10.1109/MASS.2003.1194853. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1194853.

[61] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[62] Brent Welch. Object Storage Technology, 2013. URL http://www.snia.org/sites/default/education/tutorials/2013/spring/file/BrentWelch_Object_Storage_Technology.pdf.

[63] M. Zhou, R. Zhang, D. Zeng, and W. Qian. Services in the Cloud Computing era: A survey. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 40–46, October 2010. doi: 10.1109/IUCS.2010.5666772.

[64] Openstack. OpenStack SWIFT, 2014. URL http://www.openstack.org/software/releases/liberty/components/swift.

[65] Salman Toor, Rainer Toebbicke, Maitane Zotes Resines, and Sverker Holmgren. Investigating an Open Source Cloud Storage Infrastructure for CERN-specific Data Analysis. pages 84–88. IEEE, June 2012. ISBN 978-0-7695-4722-0 978-1-4673-1889-1. doi: 10.1109/NAS.2012.14. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6310879.

[66] Yong Zhao, Yanzhe Zhang, Wenhong Tian, Ruini Xue, and Cui Lin. Designing and Deploying a Scientific Computing Cloud Platform. pages 104–113. IEEE, September 2012. ISBN 978-1-4673-2901-9. doi: 10.1109/Grid.2012.12. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6319160.

[67] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. MapReduce in the Clouds for Science. pages 565–572. IEEE, November 2010. ISBN 978-1-4244-9405-7. doi: 10.1109/CloudCom.2010.107. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708501.

[68] P. F. Corbett, D. G. Feitelson, J. P. Prost, and S. J. Baylor. Parallel access to files in the Vesta file system. In *Supercomputing '93. Proceedings*, pages 472–481, November 1993. doi: 10.1109/SUPERC.1993.1263495.

[69] Peter F. Corbett and Dror G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, August 1996. ISSN 07342071. doi: 10.1145/233557.233558. URL http://portal.acm.org/citation.cfm?doid=233557.233558.

[70] Gary Grider. Preparing Applicaions for Next Generaion IO/Storage. Technical report, 2015.

[71] Jay Lofstead, Ivo Jimenez, and Carlos Maltzahn. Consistency and Fault Tolerance Considerations for the Next Iteration of the DOE Fast Forward Storage and IO Project. pages 61–69. IEEE, September 2014. ISBN 978-1-4799-5615-9. doi: 10.1109/ICPPW.2014.21. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7103439.

[72] M. Satyanarayanan and E.H. Siegel. Parallel communication in a large distributed environment. *IEEE Transactions on Computers*, 39(3):328–348, March 1990. ISSN 00189340. doi: 10.1109/12.48864. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=48864.

[73] Darrell DE Long, Bruce R Montague, and Luis-Felipe Cabrera. Swift/RAID: a distributed RAID system. In *Computing Systems*. Citeseer, 1994.

[74] D. Black, S. Fridella, and J. Glasgow. Parallel NFS (pNFS) Block/Volume Layout. RFC 5663, January 2010. URL http://tools.ietf.org/pdf/rfc5663.pdf.

[75] D. Black, Ed., J. Glasgow, and S. Faibish. Parallel NFS (pNFS) Block Disk Protection. RFC 6688, January 2012. URL http://tools.ietf.org/pdf/rfc6688.pdf.

[76] B. Halevy, B. Welch, and J. Zelenka. Object-Based Parallel NFS (pNFS) Operations. RFC 5664, January 2010. URL http://tools.ietf.org/pdf/rfc5664.pdf.

[77] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[78] Lustre. Lustre* Software Release 2.x Operations Manual, September 2013. URL https://build.hpdd.intel.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.pdf.

[79] IBM. An introduction to IBM Spectrum Scale A fast, simple, scalable and complete storage solution for today's data-intensive enterprise, February 2015. URL http://public.dhe.ibm.com/common/ssi/ecm/dc/en/dcw03057usen/DCW03057USEN.PDF?

[80] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 31–31, Tampa, FL, USA, November 2006. IEEE. ISBN 0-7695-2700-0. doi: 10.1109/SC.2006.19. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4090205.

[81] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. RADOS: a scalable, reliable storage service for petabyte-scale storage clusters. In *In Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing '07 (PDSW '07)*, page 35, Reno, Nevada, 2007. ACM Press. ISBN 978-1-59593-899-2. doi: 10.1145/1374596.1374606. URL http://portal.acm.org/citation.cfm?doid=1374596.1374606.

[82] Dan van der Ster and Herve Rousseau. Ceph ~30pb Test Report. Technical report, 2015. URL http://cds.cern.ch/record/2015206/files/CephScaleTestMarch2015.pdf.

[83] Feiyi Wang, Mark Nelson, Sarp Oral, Scott Atchley, Sage Weil, Bradley W. Settlemyer, Blake Caldwell, and Jason Hill. Performance and scalability evaluation of the Ceph parallel file system. pages 14–19. ACM Press, 2013. ISBN 978-1-4503-2505-9. doi: 10.1145/2538542.2538562. URL http://dl.acm.org/citation.cfm?doid=2538542.2538562.

[84] Richard Bradshaw and Carl Schroeder. Fifty years of IBM innovation with information storage on magnetic tape. *IBM Journal of Research and Development*, 47 (4):373–383, 2003.

[85] Walter Hinton, Philip Sciuto, Lynn Orlando, Alain Dufaux, and Caryl Jones. Preservation and Archive: The next 100 Years. pages 1–22. IEEE, October 2015. ISBN 978-1-61482-956-0. doi: 10.5594/M001645. URL http://ieeexplore.ieee.org/document/7399633/.

[86] Jason Buffington and Adam DeMattia. Analyzing the Economic Value of LTO Tape for Long-term Data Retention, February 2016. URL https://www.lto.org/wp-content/uploads/2014/06/ESG-WP-LTO-EVV-Feb_2016.pdf.

[87] Nick Balthaser. Tape's Not Dead At LBNL/NERSC, May 2019. URL http://storageconference.us/2019/Invited/Balthaser.slides.pdf.

[88] Dan Feng, Lingfang Zeng, Fang Wang, and Peng Xia. TLFS: High performance tape library file system for data backup and archive. In *Proceedings of 7th International Meeting on High Performance Computing for Computational Science. Rio de Janeiro, Brazil: Springer*, 2006.

[89] Lingfang Zeng, Dan Feng, Fang Wang, Ke Zhou, and Peng Xia. Hybrid RAID-tape-library storage system for backup. In *Second International Conference on Embedded Software and Systems (ICESS'05)*, pages 6–pp. IEEE, 2005.

[90] Hisashi Kobayashi. *Modeling and analysis*. The Systems Programming Series. 1978.

[91] Jane WS Liu. *Real-time systems*. Prentice Hall, 2000.

[92] W.W. Chu. Optimal File Allocation in a Multiple Computer System. *IEEE Transactions on Computers*, C-18(10):885–889, October 1969. ISSN 0018-9340. doi: 10.1109/T-C.1969.222542. URL http://ieeexplore.ieee.org/document/1671135/.

[93] Michael O'Sullivan, Cameron Walker, and DongJin Lee. Designing data storage tier using integer programing. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 426–433. ACM, 2012.

[94] DongJin Lee, Michael O'Sullivan, and Cameron Walker. Benchmarking and modeling disk-based storage tiers for practical storage design. *ACM SIGMETRICS Performance Evaluation Review*, 40(2):113, October 2012. ISSN 01635999. doi: 10.1145/2381056.2381080. URL http://dl.acm.org/citation.cfm?doid=2381056.2381080.

[95] D Reinsel, J. Gantz, and J. Rydning. Data age 2025: The Digitization of the WorldFrom Edge to Core, November 2018. URL https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[96] Energy Information Administration. Electric power monthly, February 2019. URL https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_6_a.

[97] Ian F. Adams, Mark W. Storer, and Ethan L. Miller. Analysis of Workload Behavior in Scientific and Historical Long-Term Data Repositories. *ACM Transactions on Storage*, 8(2):1–27, May 2012. ISSN 15533077. doi: 10.1145/2180905.2180907. URL http://dl.acm.org/citation.cfm?doid=2180905.2180907.

[98] Preeti Gupta, Avani Wildani, Ethan L. Miller, Daniel Rosenthal, Ian F. Adams, Christina Strong, and Andy Hospodor. An Economic Perspective of Disk vs. Flash Media in Archival Storage. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 249–254, Paris, France, September 2014. IEEE. ISBN 978-1-4799-5610-4. doi: 10.1109/MASCOTS.2014.39. URL http://ieeexplore.ieee.org/document/7033661/.

[99] David SH Rosenthal, Daniel C Rosenthal, Ethan L Miller, Ian F Adams, Mark W Storer, and Erez Zadok. The economics of long-term digital storage. 2012.

[100] Dong-Oh Kim, Hong-Yeon Kim, Young-Kyun Kim, and Jeong-Joon Kim. Cost analysis of erasure coding for exa-scale storage. *The Journal of Supercomputing*, October 2018. ISSN 0920-8542, 1573-0484. doi: 10.1007/s11227-018-2663-4. URL http://link.springer.com/10.1007/s11227-018-2663-4.

[101] Xuefeng Wu, Jie LI, and Hisao KAMEDA. Reliability modeling of declustered-parity RAID considering uncorrectable bit errors. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(8): 1508–1515, 1997.

[102] Julian Kunkel and Thomas Ludwig. IOPm – Modeling the I/O Path with a Functional Representation of Parallel File System and Hardware Architecture.

In Rainer Stotzka, Michael Schiffers, and Yiannis Cotronis, editors, *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 554–561, Garching, Germany, 2012. IEEE Computer Society. ISBN 978-0-7695-4633-9.

[103] Dan Pilone and Neil Pitman. *UML 2.0 in a Nutshell*. " O'Reilly Media, Inc.", 2005.

[104] Anneke G Kleppe, Jos Warmer, Jos B Warmer, and Wim Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[105] Alan Dennis, Barabara Haley Wixom, and David Tegarden. *Systems Analysis and Design UML Version 2.0*. Wiley, 2009.

[106] Jon Holt and Simon Perry. *SysML for systems engineering*, volume 7. IET, 2008.

[107] F. Piedad, M. Hawkins, and M.W. Hawkins. *High Availability: Design, Techniques, and Processes*. Enterprise computing series. Prentice Hall PTR, 2001. ISBN 978-0-13-096288-1. URL https://books.google.com/books?id=kHB0HdQ98qYC.

[108] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 109–116, New York, NY, USA, 1988. ACM. ISBN 0-89791-268-3. doi: 10.1145/50202.50214. URL http://doi.acm.org/10.1145/50202.50214.

[109] Hakim Weatherspoon and John D Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Peer-to-Peer Systems*, pages 328–337. Springer, 2002. ISBN 978-3-540-44179-3. 10.1007/3-540-45748-8_31.

[110] Rodrigo Rodrigues and Barbara Liskov. High availability in DHTs: Erasure coding vs. replication. In *Peer-to-Peer Systems IV*, pages 226–239. Springer, 2005.

[111] J. S. Plank. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[112] J. L. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications. Technical Report Technical Report CS-08-627, University of Tennessee, Knoxville, TN 37996, 2008. URL http://www.cs.utk.edu/~plank/plank/papers/CS-08-627.html.

[113] J. S. Plank, M Blaum, and J. L. Hafner. SD codes: erasure codes designed for how storage systems really fail. In *FAST*, pages 95–104, San Jose, CA, USA, February 2013.

[114] J. S. Plank and M. G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *Dependable Systems and Networks, 2004 International Conference on*, pages 115–124. IEEE, 2004. doi: 10.1109/DSN.2004.1311882.

[115] Deshmukh, Parth, Maginnis, Sean, and Chandler, Josh. *Jerasure 2.0*. PhD thesis, University of Tennessee Honors Thesis Projects, University of Tennessee – Knoxville, 2011. URL http://trace.tennessee.edu/utkchanhonoproj/1362/.

[116] KV Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster. *Proc. USENIX HotStorage*, 2013.

[117] Yasushi Saito, Svend Frølund, Alistair Veitch, Arif Merchant, and Susan Spence. FAB: building distributed enterprise disk arrays from commodity components. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS

XI, pages 48–58, Boston, MA, USA, 2004. ACM Press. ISBN 978-1-58113-804-7. doi: 10.1145/1024393.1024400. URL http://portal.acm.org/citation.cfm?doid=1024393.1024400.

[118] K.V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, volume 44, pages 331–342, Chicago, Illinois, USA, 2014. ACM Press. ISBN 978-1-4503-2836-4. doi: 10.1145/2619239.2626325. URL http://dl.acm.org/citation.cfm?doid=2619239.2626325.

[119] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure Coding in Windows Azure Storage. In *Usenix annual technical conference*, pages 15–26. Boston, MA, 2012.

[120] James Lee Hafner, Veera Deenadhayalan, Tapas Kanungo, and KK Rao. Performance metrics for erasure codes in storage systems. *IBM Res. Rep. RJ*, 10321, 2004.

[121] Kevin Greenan. Reliability and Power-Efficiency in Erasure-Coded Storage Systems. Technical Report UCSC-SSRC-09-08, University of California, Santa Cruz, December 2009.

[122] Mingqiang Li and Jiwu Shu. DACO: A high-performance disk architecture designed specially for large-scale erasure-coded storage systems. *Computers, IEEE Transactions on*, 59(10):1350–1362, 2010. doi: 10.1109/TC.2010.22.

[123] Marcos Kawazoe Aguilera, Ramaprabhu Janakiraman, and Lihao Xu. Using erasure codes efficiently for storage in a distributed system. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 336–345. IEEE, 2005.

[124] Huaxia Xia and Andrew A. Chien. RobuSTore: A Distributed Storage Architecture with Robust and High Performance. In *Proceedings of the 2007 ACM/IEEE*

Conference on Supercomputing, SC '07, pages 44:1–44:11, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-764-3. doi: 10.1145/1362622.1362682. URL http://doi.acm.org/10.1145/1362622.1362682.

[125] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network Coding for Distributed Storage Systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, September 2010. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2010.2054295. URL http://ieeexplore.ieee.org/document/5550492/.

[126] James S Plank, Jianqiang Luo, Catherine D Schuman, Lihao Xu, Zooko Wilcox-O'Hearn, and others. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage. In *FAST*, volume 9, pages 253–265, 2009.

[127] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in a partitioned network: a survey. *ACM Computing Surveys*, 17(3):341–370, September 1985. ISSN 03600300. doi: 10.1145/5505.5508. URL http://portal.acm.org/citation.cfm?doid=5505.5508.

[128] Mark Silberstein, Lakshmi Ganesh, Yang Wang, Lorenzo Alvisi, and Mike Dahlin. Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage. In *Proceedings of International Conference on Systems and Storage*, pages 1–7. ACM, 2014.

[129] Qin Xin, E.L. Miller, T. Schwarz, D.D.E. Long, S.A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings.*, pages 146–156, San Diego, CA, USA, 2003. IEEE Comput. Soc. ISBN 978-0-7695-1914-2. doi: 10.1109/MASS.2003.1194851. URL http://ieeexplore.ieee.org/document/1194851/.

[130] T.J.E. Schwarz, Qin Xin, E.L. Miller, D.D.E. Long, A. Hospodor, and Spencer Ng. Disk scrubbing in large archival storage systems. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and*

Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings., pages 409–418, Volendam, The Netherlands, EU, 2004. IEEE. ISBN 978-0-7695-2251-7. doi: 10.1109/MASCOT.2004.1348296. URL http://ieeexplore.ieee.org/document/1348296/.

[131] Jon G Elerath and Michael Pecht. Enhanced reliability modeling of raid storage systems. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 175–184. IEEE, 2007.

[132] Osama Khan, Randal C Burns, James S Plank, William Pierce, and Cheng Huang. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In *FAST*, page 20, 2012.

[133] Guoyang Chen, Huiyang Zhou, Xipeng Shen, Josh Gahm, Narayan Venkat, Skip Booth, and John Marshall. OpenCL-based erasure coding on heterogeneous architectures. pages 33–40. IEEE, July 2016. ISBN 978-1-5090-1503-0. doi: 10.1109/ASAP.2016.7760770. URL http://ieeexplore.ieee.org/document/7760770/.

[134] Lavanya Mandava and Liudong Xing. Reliability analysis of cloud-RAID 6 with imperfect fault coverage. *International Journal of Performability Engineering*, 130(3), 2017.

[135] Albert Myers. *Complex System Reliability*, volume 0 of *Springer Series in Reliability Engineering*. Springer London, London, 2010. ISBN 978-1-84996-413-5 978-1-84996-414-2. doi: 10.1007/978-1-84996-414-2. URL http://link.springer.com/10.1007/978-1-84996-414-2.

[136] Kevin M Greenan, Ethan L Miller, Thomas JE Schwarz, and Darrell DE Long. Disaster Recovery Codes: Increasing Reliability with Large-Stripe Error Correction Codes. In *In StorageSS'07*. Citeseer, 2007.

[137] Kevin M Greenan, Ethan L Miller, and SJ Thomas JE Schwarz. Optimizing Galois Field arithmetic for diverse processor architectures and applications. In *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pages 1–10. IEEE, 2008.

[138] Kevin M Greenan, James S Plank, Jay J Wylie, and others. Mean Time to Meaningless: MTTDL, Markov Models, and Storage System Reliability. In *HotStorage*, pages 1–5, 2010.

[139] Kevin M. Greenan. *Reliability and Power-Efficiency in Erasure-Coded Storage Systems*. PhD thesis, University of California, Santa Cruz, December 2009. URL https://www.ssrc.ucsc.edu/pub/ssrctr-09-08.html.

[140] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994. ISSN 03600300. doi: 10.1145/176979.176981. URL http://portal.acm.org/citation.cfm?doid=176979.176981.

[141] John Kerl. Computation in finite fields. *Arizona State University and Lockheed Martin Corporation*, 2004.

[142] Rudolf Lidl and Harald Niederreiter. *Encyclopedia of Mathematics and its Applications*, volume 20 of *Algebra*. Addison-Wesley, Reading, Mass., 1983. ISBN 0-201-13519-1.

[143] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. Wiley, New York, 2nd ed edition, 1996. ISBN 978-0-471-12845-8 978-0-471-11709-4.

[144] James Plank, Kevin Greenan, and Ethan L. Miller. Screaming Fast Galois Field Arithmetic Using Intel SIMD Extensions. In *Proceedings of the 11th Conference on File and Storage Systems (FAST 2013)*, February 2013.

[145] H Peter Anvin. The mathematics of RAID-6. *online paper*, 2007. URL ftp://ftp2.de.debian.org/kernel/linux/kernel/people/hpa/raid6.pdf.

[146] Blair Crossman-New Mexico Tech. Functional and Performance Assessment of Erasure Coded Storage Systems. 2013.

[147] Hsing-bung Chen, Gary Grider, Jeff Inman, Parks Fields, and Jeff Alan Kuehn. An empirical study of performance, power consumption, and energy cost of erasure code computing for HPC cloud storage systems. pages 71–80. IEEE, August 2015. ISBN 978-1-4673-7891-8. doi: 10.1109/NAS.2015.7255220. URL http://ieeexplore.ieee.org/document/7255220/.

[148] National Institute of Standards and Technology. Advanced encryption standard (AES). Technical Report NIST FIPS 197, National Institute of Standards and Technology, Gaithersburg, MD, November 2001. URL https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[149] Morris J Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC| NIST. Technical Report NIST Special Publication 800-38D, NIST, 2007. URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.

[150] Jesús Martín Berlanga. GPU accelerated AES. June 2017. URL http://oa.upm.es/47172/.

[151] Binbing Hou, Feng Chen, Zhonghong Ou, Ren Wang, and Michael Mesnier. Understanding I/O Performance Behaviors of Cloud Storage from a Client's Perspective. *ACM Transactions on Storage*, 13(2):16, 2017.

[152] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. RACS: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, page 229, Indianapolis, Indiana, USA, 2010. ACM Press. ISBN 978-1-4503-0036-0. doi: 10.1145/1807128.1807165. URL http://portal.acm.org/citation.cfm?doid=1807128.1807165.

[153] Yih-Farn Robin Chen. The Growing Pains of Cloud Storage. *IEEE Internet Computing*, 19(1):4–7, January 2015. ISSN 1089-7801. doi: 10.1109/MIC.2015.14. URL http://ieeexplore.ieee.org/document/7031827/.

[154] David John Bonnie, Susan K Coulter, Jason Cody Hick, Brett Jason Hollander, Cory Lueninghoener, Jesse Edward Martinez, Michael A Mason, David Richard

Montoya, Andrew J Montoya, Timothy C Randles, and others. Next Generation Infrastructure Plan FY18-FY22. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2017.

[155] Matthew L. Curry, Anthony Skjellum, H. Lee Ward, and Ron Brightwell. Arbitrary dimension Reed-Solomon coding and decoding for extended RAID on GPUs. pages 1–3. IEEE, November 2008. ISBN 978-1-4244-4208-9. doi: 10.1109/PDSW.2008.4811887. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4811887.

[156] NVIDIA. CUDA Parallel Computing Platform, March 2017. URL http://www.nvidia.com/object/cuda_home_new.html.

[157] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, pages 190–201, New York, NY, USA, 2000. ACM. ISBN 1-58113-317-0. doi: 10.1145/378993.379239. URL http://doi.acm.org/10.1145/378993.379239.

[158] Garth R Goodson, Jay J Wylie, Gregory R Ganger, and Michael K Reiter. Efficient Byzantine-tolerant erasure-coded storage. In *Dependable Systems and Networks, 2004 International Conference on*, pages 135–144. IEEE, 2004. doi: 10.1109/DSN.2004.1311884.

[159] Klein Andy. Backblaze Hard Drive Stats for 2018, January 2019. URL https://www.backblaze.com/blog/hard-drive-stats-for-2018/.

[160] Judith L. Gersting. *Mathematical structures for computer science*. W.H. Freeman, New York, 5th ed edition, 2003. ISBN 978-0-7167-4358-3.

[161] Michael Brown. SanDisk announces Extreme Pro enthusiast SSDs with crazy long 10-year warranties, 2014. URL http://www.pcworld.

com/article/2357767/sandisk-announces-extreme-pro-\
ssd-series-backs-drives-with-10-year-warranties.html#
tk.fbpc.

[162] Jon L. Jacobi. Seagate is the first to hit 16tb capacity with its IronWolf, IronWolf Pro, and Exos x16 hard drives, 2019. URL https://www.pcworld.com/article/3399920/seagate-is-the-first-to-hit-16tb-\capacity-with-its-ironwolf-ironwolf-pro-and-exos-\x16-hard-drives.html.

[163] Ole Tange. Gnu parallel-the command-line power tool. *The USENIX Magazine*, 36(1):42–47, 2011.

[164] Lothlórien Watkins. HPSS: Data Archiving in a Supercomputing Environment, January 2019. URL https://computation.llnl.gov/projects/hpss-data-archiving-in-a-supercomputing-environment.

[165] K. Lamb. Trinity Campaign Storage and Usage Model, August 2015. URL https://www.lanl.gov/projects/trinity/_assets/docs/trinity-usage-model-presentation.pdf.

[166] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. Grid resource management—CRUSH: controlled, scalable, decentralized placement of replicated data. page 122. ACM Press, 2006. ISBN 0-7695-2700-0. doi: 10.1145/1188455.1188582. URL http://portal.acm.org/citation.cfm?doid=1188455.1188582.

[167] Jakob Luttgau, Jens Jensen, Julian Kunkel, and Bryan Lawrence. D4.1 Business Model with Alternative Scenarios. Technical Report WP4 Exploitability, DKRZ, February 2017.

[168] Walker Haddock, Matthew L. Curry, Purushotham V Bangalore, and Antho Skjellum. Campaign Storage: Eraure Coding with GPUs. In *SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, 2017. ACM. ISBN 978-1-4503-5114-0. URL http://sc17.supercomputing.org/.

[169] Greg Tucker. ISA-L open source v2.14 API doc, April 2014. URL https://01.org/sites/default/files/documentation/isa-l_open_src_2.10.pdf.

[170] Morris Dworkin. *NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. NIST, December 2001.

[171] Loic Dachary. https://ceph.com/geen-categorie/ceph-erasure-code-jerasure-plugin-benchmarks/, May 2014. URL https://ceph.com/geen-categorie/ceph-erasure-code-jerasure-plugin-benchmarks/.

Appendices

Appendix A

A.1 Application design and implementation

In order to measure the performance and effectiveness of the NLDA architecture, we designed and implemented a benchmark application that created the processes that are needed by the architecture and a front end to setup various test cases which could be executed and measured. We began with an erasure code benchmark that is included with the Ceph source code, ceph_erasure_code_benchmark. This tool allowed for the selection and configuration of the erasure code library and the specification of the benchmark to be executed. We used this tool initially to measure the erasure coding performance of the Jerasure, the ISA-L, and the Gibraltar libraries. This tool did not implement any feature to interact with the object storage system. Another tool that is included in Ceph is the radosbench.py which writes and reads data to the object store to measure the IO performance. Ceph provides and application programming interface (API) to interact with the object storage system along with some examples about how to use this API. We decided to use the ceph_erasure_code_benchmark tool and extend it with features that allowed writing to the object storage system.

A.1.1 Challenges

In our original approach we expected that the Ceph erasure code plugin feature would allow us to perform erasure coding on the OSSs just as it does with the CPU based libraries. However, we found that this was not possible. Ceph assumes that all compute nodes in the object storage systems, including the metadata servers, etc. will have the same hardware capabilities. When we tried to create erasure coding pools on the object storage system using Gibraltar, the system would panic. After much effort

debugging, we found that the metadata server was failing when the Gibraltar pool was created because it did not have the GPU nor the NVIDIA® CUDA® libraries installed. Even when we added these capabilities to the metadata server, the system continued to fail. However, we did not have any problems running the Gibraltar libraries as a client on the FTAs and we quickly realized that this was the superior configuration. We did not continue to diagnose the problem with running the GPUs on the OSSs.

The other great challenge was to keep the Ceph API running at full utilization. This required that a queue of objects needed to be provided for the modules that wrote objects to the object storage system, performed erasure coding or encryption. When reading from the object storage system, we needed to keep a queue of objects for the erasure repair module and decryption modules to stay fully utilized. Another issue with using the Ceph API concerned the memory buffer management. Ceph developed a "bufferlist" data structure which efficiently manages the memory on the heap. The API depends on a new bufferlist to be allocated by the API as an input to the write or read API calls. These bufferlist objects are then managed by Ceph throughout their lifetime. After Ceph is finished with an operation, write or read, it releases the memory resources and cleans up the references to the data structure. Since a bufferlist is required for each object written or read and we use an object to store a shard in, each stripe required $K + M$ bufferlist objects. The allocation of these bufferlists took a small amount of time to create and would introduce delays in the pipeline. By creating a module to produce bufferlist objects and push them to a queue, we were able to keep a ready supply for the other modules in the pipeline.

We used the asynchronous API calls to write and read data to the object storage system. Neither the API documentation nor the examples provided sufficient guidance for using these API calls. We learned that several intuitive approaches did not work. Neither did we find any discussion in the developer nor user community about using the Ceph API asynchronous features. The trivial examples provided worked, but following this approach would quickly result in failure when writing hundreds of objects concurrently

which was required to maximize performance. We spent a good amount of time studying the tools provided in the Ceph source code where these API calls were used and learned how to use the callback mechanism. This required a good bit of record keeping for each shard that was written or read from the object storage system but worked very well after we implemented and tested.

In the end, our extended version of the ceph_erasure_code_benchmark has over 2,600 lines of code. We also had to extend the Gibraltar library API in order to interface with the Ceph plugin and several header files were implemented to support the new data structures and functions that were required to implement our highly concurrent application. Next, we discuss the modules of the system.

A.1.2 Command Line Parameters

There are two main modes of operation for the benchmark application. The first is to write data to the object storage system and the second is to read data from the object storage system. The application has several configuration parameters that are supplied on the command line to perform the test:

testname — Each test run is given a name. This name is used as the root for the object names that store the data shards. Each shard is stored in an object that is named based on the testname, the stripe number and the position of the shard in the stripe.

iterations — The number of stripes to be written or read during the experiment.

queuesize — The maximum number of shards that can be stored in any of the queues at any time. As the data moves through the pipeline, each module puts a data shard or stripe into an inter-module queue. The size of the queue is based on this parameter which provides the maximum number of shards for the queues. In the case of queues that hold stripes, each stripe has $K + M$ shards.

mode — The modes are for writing or reading to the object store which are denoted as encode and decode respectively.

pool — The pool that has been configured in the object storage system to store the objects in.

plugin — The erasure code library to be used for the test. We used the ISA-L, Jerasure, and the Gibraltar plugins.

technique — The erasure code libraries may provide multiple implementations of the erasure coding algorithm. We used reed-solomon for Gibraltar and Jerasure and cauchy for ISA-L.

shard-size — The data size is specified by the shard size. For our testing we used 4 MB and 8 MB. With $K = 120$, this gave us stripe sizes of about 512 MB and 1 GB.

threads — This is the maximum number of Ceph API asynchronous writes or reads that are allowed to be active at any time. This is intended to prevent overloading the object storage system.

ecthreads — This is the number of instances of the erasure coding or erasure decoding module that is running concurrently. In order to produce enough shards to keep the object storage server busy when using the ISA-L or Jerasure libraries, it was necessary to run multiple instances of these modules. For Gibraltar, we only run a single thread.

A.1.3 Software Modules

There are seven major modules in our application with a few other utility modules that facilitate moving data objects between the main modules. The main utility provided is to assemble shards back into stripes for erasure coding or erasure repair. Some of the modules operate on a stripe of data and others operate on individual shards.

Bufferlist Creator — This thread creates bufferlist objects and pushes them into a bufferlist queue. A configuration parameter provides the upper limit for the size of this queue. When the limit is reached, the module, running as a thread, waits until

the supply falls below the limit to run again. This keeps the queue filled so there is a supply for the downstream operations to consume. We have implemented a counter which checks the number of bufferlists that have been created against the number of stripes that were configured for the benchmark. When the number of required bufferlist objects for the test have been produced, the module exits. The behavior of this module results in a rapid increase in the memory used by the application at startup, a leveling off of the memory resources during the test, followed by a rapid decrease in memory utilization after all of the buffers have been produced, consumed and completed.

Erasure Encoding — The erasure encoding module also runs as a thread when the test is configured to write data to the object storage system. The module pulls a stripe of K data shards from a stripe queue and sends them to the erasure code library. The stripe is returned with an additional M checksum shards that were computed. After erasure coding a stripe, the shards are marshaled into queue that supplies the write module.

Erasure Decoding — The erasure decoding module runs as a thread when the mode is set to decode from the object storage system. A stripe of data is pulled from the read data stripe queue and sent to the erasure code plugin for repair. The erasure repair queue will also have to handle the configuration of the call to the plugin to perform the repairs. K good shards must be provided and the decoding call will need to be configured with the information describing the order of the shards in the stripe. Some of the shards may be checksum shards that will replace defective data shards. Also, when performing a repair, it may be necessary to recompute some of the checksum shards. After the data are repaired, the benchmark application discards the data and completes the read of the stripe unless we are decrypting. If we are decrypting, the stripe is put into the decryption queue.

Object Write — The object write module runs as a thread when the mode is set to encode. Shards are pulled from the input queue and written to the object storage

system using the asynchronous write call. One of the inputs to the call is a callback object. The callback object provides a pointer to a function that is called with a value after the operation completes. The write module creates a record of the callback and stores it in a map. When the callback is executed by the Ceph API, the index into the map is provided by the message. The function called looks up the record from the map and deletes it. Once all of the entries in the map are removed and the map has no more elements, all of the writes to the object storage system have completed.

Object Read — The object read module runs as a thread when the mode is set to decode. Based on the testname, K, M, shardsize, and iterations given on the command line, stripes of empty bufferlist objects are created for reading. Since the name of the objects that were stored in the object storage system were constructed from these values, the read module begins to make asynchronous read calls to the object storage system for each shard in each stripe. A callback object is created and included in the API call. These callback objects are inserted into a map. When the call completes, the function called by the callback module is executed. The callback function looks the record up in the map, retrieves the data returned by the call and writes it to the bufferlist for that shard. A utility thread watches the map for completed shards in stripe order then shard order. As the shards are retrieved from the object storage system, they are assembled into stripes and the callback record is deleted. The completed stripes are pushed into a queue for downstream operations.

Encryption Module — The encryption module runs as a thread when encryption is enabled and the mode is encode. If we are encrypting data, the data is encrypted prior to erasure coding. When the encryption module is running, a helper thread pulls a stripe of data from the input queue and marshals the shards into a queue for the encryption modules. There may be multiple encryption threads executing for parallelism. Each encryption thread pulls a shard from the input shard queue

and performs encryption on it. The encrypted shards are pushed on to an output queue. Another helper thread assembles the encrypted shards back into stripes in the correct order and inserts the stripes into the input queue for the erasure coding module.

Decryption Module — The decryption module runs as a thread when encryption is enabled and the mode is decode. The decryption thread pulls shards from the input queue and performs decryption. There may be multiple decryption threads executing. The current application deletes the data after the decryption is performed and completes the read operation for the stripe.