IStore: Towards High Efficiency, Performance, and Reliability in Distributed Data Storage with Information Dispersal Algorithms

Corentin Debains[1], Pedro Alvarez-Tabio[1], Dongfang Zhao[1], Kent Burlingame[1], Ioan Raicu[1,2]
[1]Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
[2]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA
corentin@debains.net, pedro.tabio@navteq.com, dzhao8@hawk.iit.edu, kburling@hawk.iit.edu, iraicu@cs.iit.edu

***Abstract*—Reliability is one of the major challenges for high performance computing and cloud computing. Data replication is a commonly used mechanism to achieve high reliability. Unfortunately, it has a low storage efficiency among other shortcomings. As an alternative to data replication, information dispersal algorithms offer higher storage efficiency, but at the cost of being too computing-intensive for today's modern processors. This paper explores the possibility of utilizing erasure coding (a form of information dispersal algorithms) for data redundancy while accelerating its operation with GPUs. We evaluate the performance improvements of the erasure coding from the CPU to the GPU, showing a 10X higher throughput for the GPU. With this promising result, we design and implement a distributed data store with erasure coding, called IStore, to demonstrate that erasure coding could serve as a better alternative to traditional data replication in distributed file systems. A performance evaluation is performed on a 32-node cluster to evaluate the scalability and parameter sensitivity of IStore.**

I. Introduction

The reliability of a computer system refers to the property that a system can run continuously without failure [1]. Here "without failure" by no means indicates that failures do not happen. Rather, with data intensive applications deployed on a cloud which is usually built with commodity hardware, failures are the norm instead of the exception. The reliability a computer system could achieve becomes a problem of how well failures can be handled. Ideally, these failures should be completely transparent to the users, with a relatively low or even negligible cost. Keeping high reliability is one of the most important metrics for high performance computing (HPC) and cloud computing systems, and is often listed as mean-time-to-failure (MTTF) in the service-level agreement (SLA).

One of the most commonly used techniques to make data highly reliable is replication. For example, Google File System (GFS) [2] makes 3 replicas as the default. The Hadoop distributed file system [3] also uses replication. This technique is often sufficient; it is easy to implement and has excellent performance, at the cost of space efficiency. For example, with the original data and 3 replicas, the storage utilization rate is only $\frac{1}{1+3} = 25\%$. This simple math indicates that the cost of storage is quadrupled when building a distributed system, which might not be economically acceptable in many application cases. Another drawback of replication is that it consumes network bandwidth to migrate data across different nodes to maintain the consistency and reliability of replicas. Moreover, replicating the intact and non-encrypted data can potentially expose more security holes.

Other than replication, another important technique in the data replication family is erasure coding which is well known for its storage efficacy. Erasure coding partitions a file into multiple fragments which are encoded and stored on different nodes. Literatures [4, 5] show that erasure coding delivers a better space efficiency but, unfortunately, cannot meet the bandwidth requirement for a large-scale distributed file system because the encoding/decoding computation hits the bottleneck of CPU capacity. With the state-of-art GPU many-core technology, this computing bottleneck could potentially be alleviated. This paper is an early study to showcase the potential of a GPU-based erasure coding mechanism, to be used as a building block for ongoing work to develop a distributed file system for exascale computing.

The first part of this paper evaluates several information dispersal algorithms (IDAs) that operate using both CPUs and GPUs. Experiments show a 10X performance improvement on the GPU-based libraries compared to the conventional CPU-based libraries. Based on this promising result, we argue that GPU-based IDAs would unlikely be the bottleneck of distributed file systems, and would enable erasure coding to be a competitive alternative to data replication. To demonstrate this, the second part of this paper discusses the design and implementation of a prototype of an IDA-enabled distributed storage system, called IStore. We perform various micro benchmarks which show that IStore is scalable with good performance while offering better storage efficiency.

In summary, this paper has the following contributions:

1) *Extensive performance evaluation of various IDA libraries between CPU and GPU implementations*
2) *Show evidence that information dispersal algorithms can be performance-competitive with replication*
3) *Design and implement an IDA-enabled prototype of a distributed storage system*

The remainder of this paper is structured as follows: Section II introduces the background of erasure coding and GPU computing, as well as some related work in GPU-accelerated applications and recent findings in data redundancy. We present the design and implementation of IStore

in Section III. Section IV discusses the experimental results. Section V concludes this paper, and introduces how IStore will accommodate new computing devices as well as how we plan to incorporate it into the next generation of distributed file systems for exascale computing.

II. Background and Related Work

This section gives a brief background of erasure coding and GPU computing, which are two enabling technologies for IStore.

A. Erasure Coding

Erasure coding, together with data replication, are the two major mechanisms to achieve data redundancy. Erasure coding has been studied by the computer communication community since the 1990's [6, 7], as well as in storage and file systems [8–10]. The idea is simple: a file is split into k chunks and encoded into $n > k$ chunks, where any k chunks out of these n chunks can reconstruct the original file. We denote $m = n - k$ as the number of redundant or parity chunks. We assume each chunk is stored on a distinct storage disk. That assumption is supported by Weatherspoon et al. in [11], where they showed that for N, the total number of machines, and M, the number of unavailable machines, the availability of a chunk (or replica) A can be calculated as

$$A = \sum_{i=0}^{n-k} \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}.$$

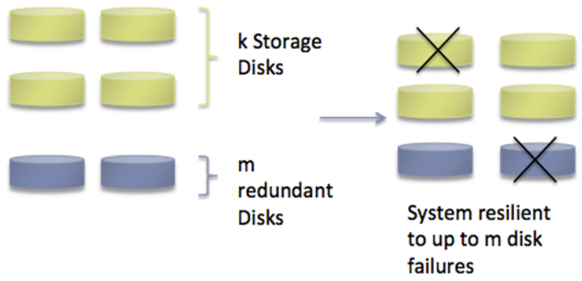


Fig. 1. Encoding k chunks into $n = k + m$ chunks so that the system is resilient to m failures.

Figure 1 describes what the encoding process looks like. At first glance, the scheme looks similar to data replication; allocate some extra disks as backups and the more redundant disks available, the more reliable the system is. However, the underlying rationale of erasure coding is quite different from the idea of simple replication, which relies on complex mathematical properties. For example, Reed-Solomon coding [12] uses a generator matrix built from a Vandermonde matrix to multiply the k data to get the encoded $k + m$ codewords, as shown in figure 2.

Compared to data replication, erasure coding has 3 important features. First, it offers a higher storage efficiency, denoted as $E_{storage}$, which is defined as $\frac{k}{n}$. Andrew et al [1] showed that erasure coding outperforms data replication by 40% - 200% in terms of storage efficiency. The second advantage of erasure coding is an immediate consequence of the first one. Because erasure coding offers a higher efficiency of storage, fewer copies of data exist in the system, which in turn saves network bandwidth for data migration. This feature is critical to the success of applications in a context of limited network resources, e.g. geographically dispersed Internet-connected cloud computing systems built with commodity hardware, such as in TAHOE-LAFS [13]. The last and often underrated advantage is security. Rather than copying the intact and non-encrypted data from one node to another, erasure coding chops the data and encodes the chunks to disperse them to multiple nodes. This process is hard to reverse by wisely choosing the encoding matrix. Erasure coding based data redundancy can guarantee the security of data with up to $k - 1$ nodes compromised because the minimal number of chunks to restore the original file is k, as explained. Note that a simple replication cannot tolerate any compromised nodes; if one node (with the replica) is compromised, the entire file is immediately compromised. Therefore, for those applications with sensitive data, erasure coding is the preferred mechanism over replication. In recent research, for example, AONT-RS [14] is a system that blends an All-or-Nothing Transform to achieve high security.

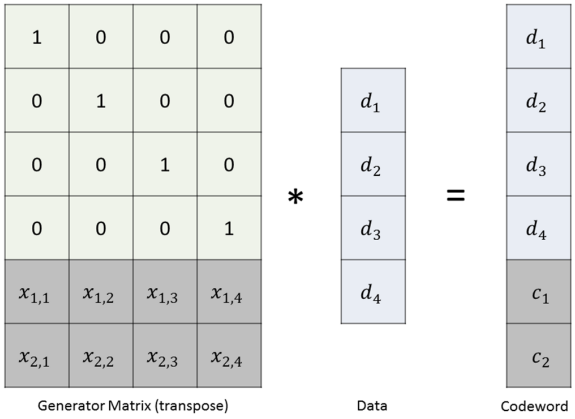


Fig. 2. Encoding 4 data into 6 codewords with Reed-Solomon coding.

The drawback of erasure coding comes from its compute intensive nature. It places an extensive burden on the CPU, which makes it impractical in today's production distributed file systems that require high performance. This is one of the reasons why prevailing distributed file systems [2, 3] prefer data replication to erasure codings.

B. GPU Computing

The graphics processing unit (GPU) was originally designed to rapidly process images for the display. The nature of image manipulations on displays is quite different than the regular tasks performed by the CPU. Most image operations are done with single instruction and multiple data (SIMD), where a general-purpose application on a CPU takes multiple instructions and multiple data (MIMD). To meet the requirement

TABLE I
COMPARISONS OF TWO MAINSTREAM GPU AND CPU

Device	Number of Cores	Frequency (GHz)	Power Consumption per Core (W)
Nvidia GTX460 (GPU)	336	1.56	160 / 336 = 0.48
AMD Phenom X6 1100T (CPU)	6	3.3	125 / 6 = 20.83

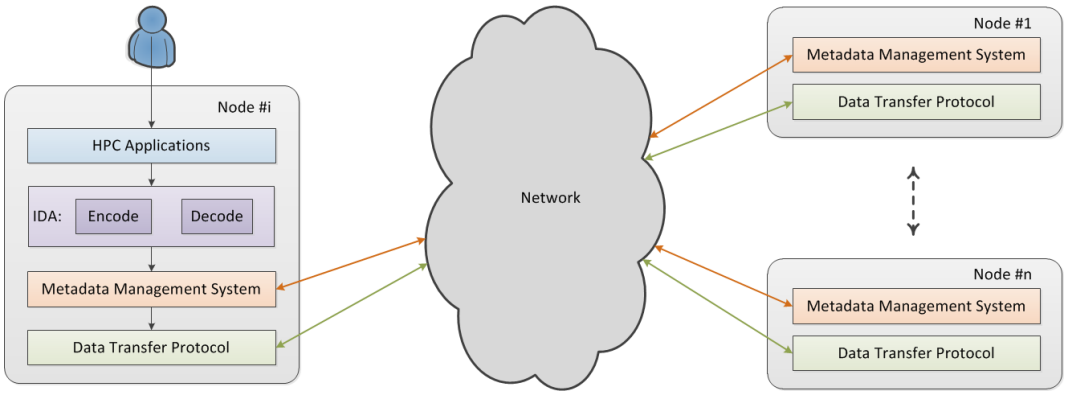


Fig. 3. An architectural overview of IStore deployed on an n-node HPC system. End users run the application on the i^{th} node where files are encoded and/or decoded by IDA. All nodes are installed with two daemon services for metadata management and data transfer, respectively.

of computer graphics, the GPU is designed to have many more cores on a single chip, all of which carry out the same instructions at the same time.

The attempt to leverage the GPU's massive number of computing units can be tracked back to as early as the 1970's in [15]. However, the GPU did not get popular for processing general applications because it lacked support in terms of programmability until GPU-specific programming languages and frameworks were introduced, e.g. OpenCL [16] and CUDA [17]. These tools greatly eased the development of general applications running on GPUs, and thus made a lot of opportunities to improve the performance by utilizing GPUs, which are usually called general-purpose computing on graphics processing units (GPGPU).

Leveraging GPGPU gains tremendous research interest in the HPC community because of the huge potential to improve the system performance by exploiting the parallelism of GPU's many-core architecture as well as GPU's relatively low power consumption, as shown in [18, 19]. Table I shows a comparison between two mainstream GPU and CPU devices, which will also serve as the two test beds in the evaluation section later. We note that, even though the frequency of GPUs are only about 50% of CPUs, the amount of cores outnumbers those in CPUs by $\frac{336}{6} = 66$X, which far overcomes the shortcoming of its low frequency. The largest GPUs come with 3072-cores while the largest CPUs come with 16-cores, making the gap even larger (192X). One of the major challenges of extreme-scale computing is power consumption. Current CPU-based architectures would not be viable to exascale computing because the power cost would be unacceptable. Table I suggests that GPUs consume 2 orders of magnitude less energy, which seems to be significantly more appropriate as we scale up supercomputers towards exascale. At the time of this writing, the fastest supercomputer in the world is equipped with Nvidia GPUs [20].

C. Related Work

Recent research shows quite a lot of interest in improving applications by GPU acceleration. A GPU accelerated storage prototype was proposed in [21]. Ravi et al [22] shows how to improve the throughput of job scheduling on a cluster of CPU-GPU nodes. A GPU-based pattern classification method was presented in [23]. Rocki and Suda [24] leverage GPU to accelerate the Traveling Salesman Problem (TSP). Parallelizing cryo-EM 3D reconstruction on a CPU-GPU heterogeneous system was introduced in [25].

Data redundancy is one of the key research areas in HPC and Cloud Computing. A dynamic replication was proposed for service-oriented systems in [26]. iFlow [27] is a replication-based system that can achieve both fast and reliable processing of high volume data streams on the Internet scale. Power-aware replica strategies in tree networks were presented in [28]. Ramabhadran and Pasquale [29] investigated the roles of replication vs. repair to achieve durability in large-scale distributed storage systems. DARE [30] is an adaptive data replication algorithm to improve data locality. CRDM [31] is a model to capture the relationship between availability and replica number.

III. DESIGN AND IMPLEMENTATION

We have designed and implemented a prototype of a distributed storage system incorporating GPU-based erasure coding for data redundancy called IStore. IStore is implemented in C/C++ for both performance and for portability across a large number of the largest supercomputers.

A. Overview

A high-level architecture of IStore is shown in figure 3. We assume the HPC system has n nodes. IStore installs two services on each node: 1) a distributed metadata management (the orange box) and, 2) a high-efficiency and reliable data transfer protocol (the green box). Each instance of these two services on a particular node would communicate to other peers over the network if necessary (i.e. if the needed metadata and/or data cannot be found locally). Assuming the end user logs in on node $\#i$, then an extra IDA layer is needed to encode and/or decode those files involved in the applications.

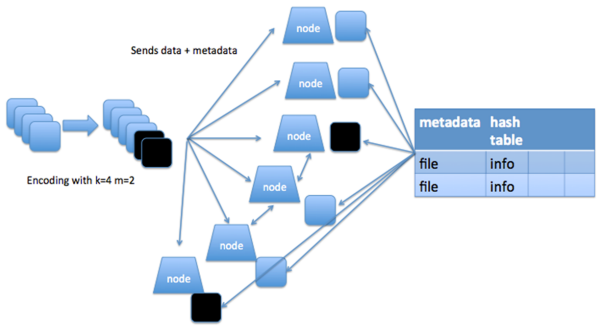


Fig. 4. An example of file writing in IStore.

Figure 4 illustrates the scenario when writing a file with IStore. Firstly, the file is chopped into $k = 4$ chunks and encoded into $n = k+m = 6$ chunks. These 6 chunks are then dispersed into 6 different nodes after which their metadata are sent to the metadata hash tables that are also distributed across these 6 nodes. A file read request is essentially the reversed procedure of a file write: retrieving the metadata, transferring the data, and decoding the chunks. We will explain more implementation details and discuss our design decisions in the following subsections.

B. Daemon Services

A daemon service is a background process that is managed by the operating system or another application rather than under the direct control of the end users. For example, the daemon service of File Transfer Protocol (FTP) protocol is called $ftpd$ in UNIX-like machines. The daemon service needs to be started before any client could make the request. To make the service efficient and responsive to concurrent users in a timely fashion, a new thread (or a cached one from before) is dedicated to accomplish the work submitted by the end user. IStore introduces two daemon services that are used for distributed metadata management and light-weighted data transfer, respectively.

1) Distributed Metadata Management: The traditional way of handling metadata for distributed file systems is to allocate the metadata information into one or a few nodes with the assumption that metadata only contains very high level information, so that a central repository would suffice. The majority of production distributed and parallel file systems all employ centralized metadata management. For example, the Google File System keeps all the metadata information in the "GFS master". This architecture is easy to implement and maintain, but exposes a performance bottleneck and even a point of failure if the metadata itself gets too large or receives too many concurrent requests. Other file systems also have a similar architecture with centralized metadata, e.g. HDFS [3]. As a more concrete example, we have run a micro benchmark on GPFS [32] which has a centralized metadata management scheme. To test the performance of metadata we create a large number of empty files on a variable number of nodes. Figure 5 shows that creating files in multiple directories cannot scale well after 16 nodes, and it is even worse for the case of a single directory; GPFS does not scale at all in terms of metadata performance. This fact suggests that the centralized metadata management would hinder the overall performance of distributed file systems. Therefore, just like what has been done for data, metadata needs to be distributed to multiple nodes as well.

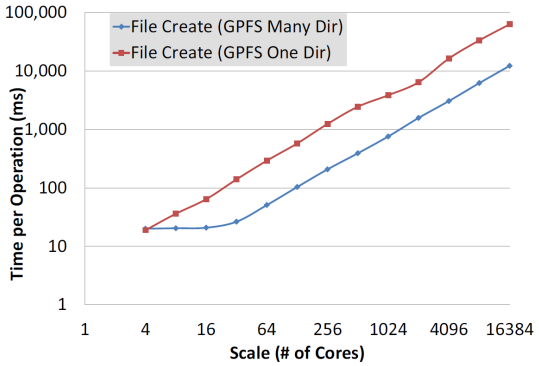


Fig. 5. Metadata performance of GPFS on a variable number of processors on IBM Bluegene/P. On many directories, i.e. each node only creates files on its own exclusive directory, metadata can only scale to 16 nodes. On a single directory, metadata performance does not scale at all.

The authors of this paper have been working on another project called Zero-Hop Distributed Hash Table (ZHT) that can serve as a distributed metadata management system. An early version of ZHT was published in [33]. There are multiple choices of distributed hash table (DHT) available. For example, Memcached [34] and Dynamo [35] . We chose to incorporate ZHT into IStore because ZHT has some features that are critical to the success of serving as a metadata manager. As summarized in Table II, ZHT has many advantages, such as being implemented in C/C++, having the lowest routing time, and supporting both persistent hashing and dynamic membership. ZHT is installed as a daemon service on each node of IStore. A more detailed description of ZHT is currently under review at another venue.

We give a very high level overview of ZHT. As shown in Figure 6, ZHT has a similar ring-shaped look as the traditional DHT [36]. The node IDs in ZHT can be randomly distributed across the network. The correlation between different nodes is computed with some logistic information like IP address, for example. The hash function maps a string to an ID that can be retrieved by a $lookup(k)$ operation at a later point.

TABLE II
COMPARISONS BETWEEN DIFFERENT DHT IMPLEMENTATIONS

	ZHT	Memcached	Dynamo
Impl. Language	C/C++	C	Java
Routing Time	0 - 2	2	0 - $\log N$
Persistence	Yes	No	Yes
Dynamic Member	Yes	No	Yes

Besides common key-value store operations like $insert(k, v)$, $lookup(k)$ and $remove(k)$, ZHT also supports a unique operation, $append(k, v)$, which we have found quite useful in implementing lock-free concurrent write operations.

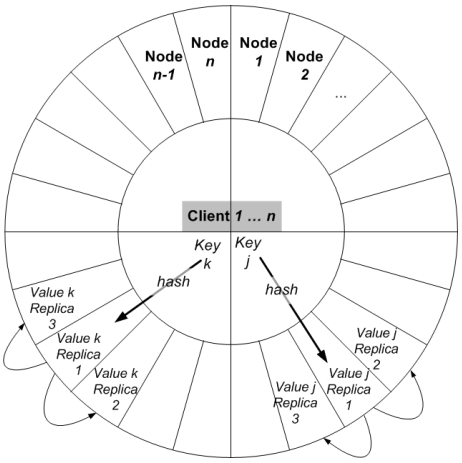


Fig. 6. ZHT architecture: namespace, hash function and replications

2) Data Transfer Protocol: As a distributed system, IStore needs some data transfer mechanism to migrate data back and forth across the network. Ideally, this mechanism should be efficient, reliable and light-weight, because otherwise IStore would introduce too much overhead on data transfer. User Datagram Protocol (UDP) is efficient in transferring data, but is an unreliable protocol. Transmission Control Protocol (TCP), on the other hand, is reliable but has a relatively lower efficiency. Ideally, a hybrid UDP/TCP protocol might be best; essentially a protocol that is both reliable and efficient.

We have developed our own data transfer service called $fUDT$ with APIs provided by UDP-based Data Transfer (UDT) [37], which is a reliable UDP-based application level data transport protocol for distributed data-intensive applications. UDT adds its own reliability and congestion control on top of UDP which thus offers a higher speed than TCP. UDT provides an abundant set of APIs to develop other application-level tools for fast and reliable data transport. Similarly to ZHT, $fUDT$ is installed as a daemon service on each node of IStore.

C. Erasure Coding Libraries

Besides the daemon services running at the back end, some encoding/decoding mechanisms are needed on the fly. In our case, performance is the top priority when choosing from available libraries, since that is our motivation for why we want to leverage GPUs for data redundancy. Plank et al [9] presents a good review of these libraries. In this initial release of IStore, we support two built-in libraries Jerasure [38] and Gibraltar [39] as the default CPU and GPU libraries, respectively. IStore is flexible enough to allow other libraries to be installed.

1) Jerasure: Jerasure is a C/C++ library that supports a wide range of erasure codes: RS coding, Minimal Density RAID-6 coding, CRS coding and most generator matrix coding. One of the most popular codes is the Reed-Solomon encoding method, which has been used for the RAID-6 disk array model. This coding can either use Vandermonde or Cauchy matrices to create generator matrices.

2) Gibraltar: Gibraltar is a Reed-Solomon coding library for storage applications. It has been demonstrated to be highly efficient when tested in a prototype RAID system. This library is known to be more flexible than other RAID standards; it is scalable with parity's size of an array. Gibraltar has been created in C using Nvidia's CUDA framework.

D. Client APIs

IStore provides a completely customizable set of parameters for the applications to tune how IStore will behave. In particular, users can specify which coding library to use, the number of chunks to split the file into (i.e. k), the number of parity chunks (i.e. $m = n - k$) and the buffer size (default is 1MB).

When an application writes a file, IStore splits the file into k chunks. Depending on which coding library the user chooses to use, these k chunks are encoded into $n = k + m$ chunks. Meanwhile, because the encoded data is buffered, $fUDT$ can disperse these n encoded chunks onto n different nodes. This pipeline with the two levels encoding and sending allows for combining the two costs instead of summing them, as described in Figure 7. At this point the data migration is complete, and we will need to update the metadata information. To do so, ZHT on these n nodes is pinged to update the entries of these n data chunks. This procedure of metadata update can be completed with low overhead, as its backed by an in-memory hash-map, which is asynchronously persisted to disk.

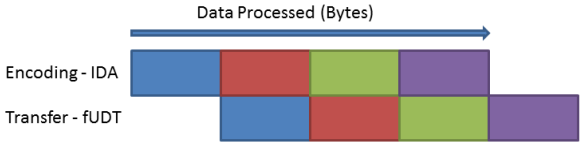


Fig. 7. Pipelining of Encoding and Transferring for a write operation in IStore

Reading a file is just the reversed procedure of writing. IStore retrieves the metadata from ZHT and uses $fUDT$ to transfer the k chunks of data to the node where the user logs in from. These k chunks are then decoded by the user-specified library and transformed into the original file.

IV. Evaluation

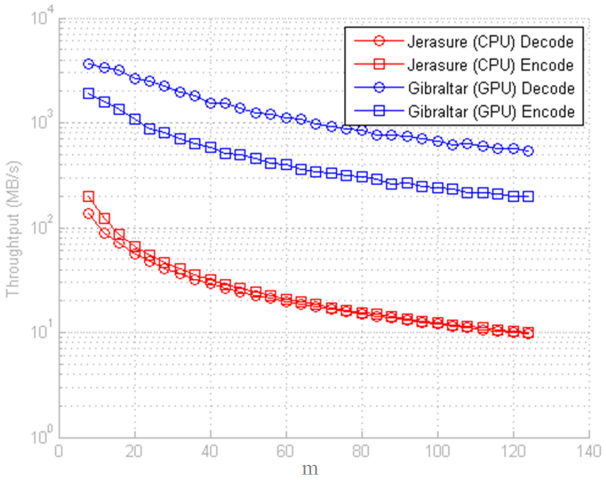
We evaluate IStore with both GPU and CPU libraries to show the potential of GPU-acceleration. The results give us hope that GPU-based erasure coding would likely be feasible to accelerate distributed file systems. To demonstrate its scalability, IStore is deployed on a Linux cluster. We also analyze the parameters' sensitivity to IStore's throughput.

A. Experiment Setup

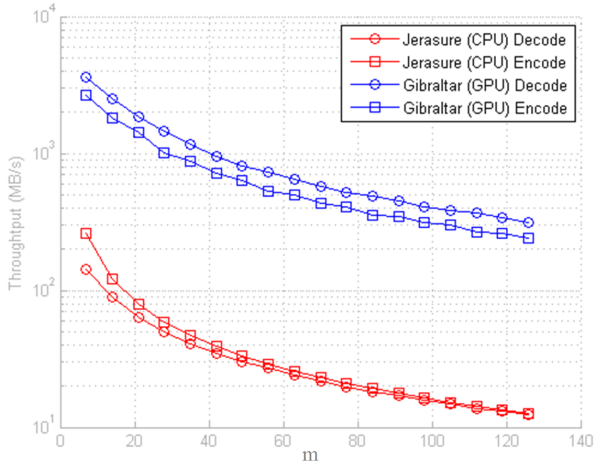
The CPU and GPU used in our experiment are AMD Phenom X6 1100T (6-cores at 3.1GHz) and Nvidia GTX460 (336-cores at 1.56GHz). The operating system is Ubuntu 10.04 with Linux kernel version 2.6.32 and CUDA version 4.1. IStore is deployed on a 32-node Linux cluster each of which has 8GB memory and dual AMD Opteron quad-core processors (8-cores at 2GHz). The features of different algorithms to be evaluated are summarized in Table III.

TABLE III
Comparison on replication number, storage efficiency and $(k:m)$ ratios

Algorithm	# Replications	$E_{storage}$	$(k:m)$
REP-3	3	33.33%	1:2
IDA-3:5	6	37.5%	3:5
IDA-5:3	4	62.5%	5:3
IDA-5:11	12	31.25%	5:11
IDA-11:5	6	68.75%	11:5
IDA-13:3	4	81.25%	13:3
IDA-26:6	7	81.25%	26:6
IDA-29:3	4	90.63%	29:3



(a) $E_{storage} = 33\%$



(b) $E_{storage} = 75\%$

Fig. 8. Encoding and decoding throughput with 1MB buffer size.

B. GPU vs. CPU on a single node

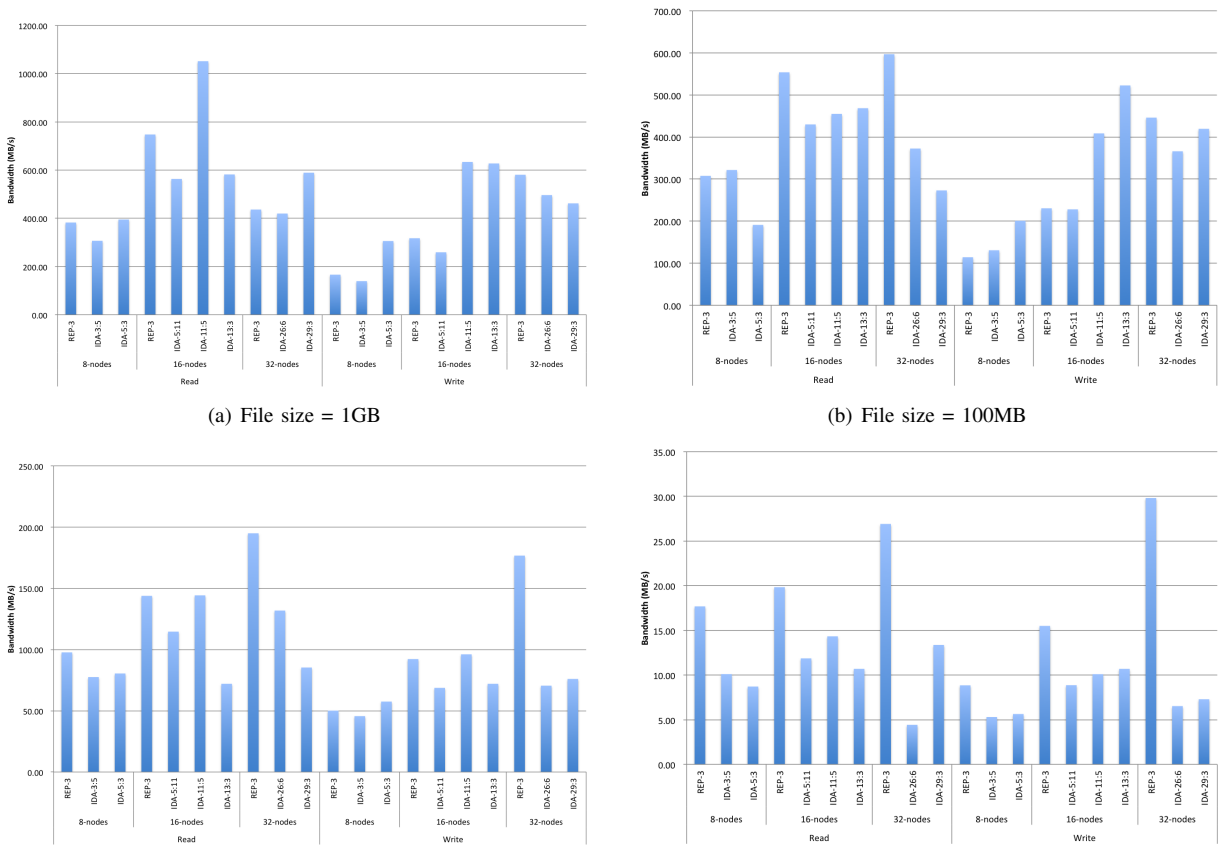
The encode and decode throughput of Jerasure and Gibraltar are plotted in figure 8(a) with m increasing from 2 to 128 while keeping $E_{storage} = 33\%$. The buffer size is set to 1MB. In all cases, with larger m values, the throughput decreases exponentially. This is because when the number of parity chunks increases, encoding and decoding take more time which reduces the throughput. A more interesting observation is the gap between Gibraltar and Jerasure for both encode and decode. There is more than 10X speedup with Gibraltar which suggests that GPU-based erasure coding would likely break through the CPU bottleneck in distributed file systems.

We then change the storage efficiency $E_{storage} = 75\%$ and measure the throughput with different m values in figure 8(b). Similar observations and trends are found just like the case for $E_{storage} = 33\%$. Therefore IStore achieves a stable throughput and does not seem to be sensitive to $E_{storage}$.

C. Throughput on a 32-node cluster

We compare the read and write bandwidth of 10 different algorithms for four different file sizes: 1GB, 100MB, 10MB and 1MB, in Figure 9. The buffer size is set to 1MB. We observe that, besides the number of nodes, $k : m$ ratio also plays a critical role for tuning the performance. Solely increasing the number of nodes does not necessarily imply a higher throughput. The reason is that it might "over" split the data into a more than enough number of chunks. In other words, the cost of splitting and encoding/decoding offsets the benefit from the concurrency. Of course, this is partially because we allocate one chunk on one node for a given job. Once this is not a restriction, a larger number of nodes would imply a non-lower aggregate throughput, in general.

We anticipated that traditional replication would outperform any IDA approach, due to the simplicity of replication. However, we found that replication with 3 replicas performed comparably to the information dispersal algorithm for many configurations. We believe that part of the explanation lies in the fact that replication uses more network bandwidth (e.g. often 3 replicas involves more than double the amount of data to be transferred than an approach based on information dispersal algorithms) but keeping processing simpler and more scalable. Based on the findings of this paper, we expect that clusters with GPUs will outperform traditional replication



(a) File size = 1GB

(b) File size = 100MB

(c) File size = 10MB

(d) File size = 1MB

Fig. 9. Aggregate throughput of replication and information dispersal algorithms from 8-nodes to 32-nodes, with a variety of file sizes (1MB - 1GB)

approaches.

V. Conclusion and Future Work

In this paper, we presented IStore, a distributed data store based on GPU-accelerated IDAs to achieve a more secure, faster and more cost-effective data redundancy. We started by implementing both GPU- and CPU-based erasure coding algorithms and conducting a comprehensive performance comparison. With the promising GPU results, we proposed to leverage GPU to improve the data redundancy of distributed file systems. We designed and implemented an IDA-enabled prototype of a distributed data store. Experiments show that IStore scales well and is feasible to be incorporated into a higher level distributed file system for better data redundancy and efficiency, while still maintaining high-performance. We plan to integrate IStore into FusionFS [40], a new distributed file system aimed at extreme scales.

IStore is agnostic about the underlying computing hardware, either CPU or GPU, as long as the interfaces are implemented. That said, there is nothing architecturally preventing us from leveraging new computing devices to further accelerate the encoding/decoding process. Among others, Intel Many Integrated Core (MIC) multiprocessors will be an interesting test bed. We plan to study how the MIC can help improve IStore once it is released later this year. We also plan on exploring larger scale experiments on GPU-based systems, such as the 3K-GPU Blue Waters supercomputer at UIUC.

IStore is a building block for the FusionFS distributed file system, whose preliminary results will be presented in [40]. The goal of FusionFS is to develop both theoretical and practical aspects of building the next generation of distributed files systems scalable to exascale by allocating local persistent storage to compute nodes. By integrating IStore and FusionFS, FusionFS will support both the traditional replication strategies and information dispersal algorithms for data reliability with high performance and high efficiency.

References

[1] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*, pages 531–532. Prentice Hall; 2nd edition, 2006.

[2] Sanjay Ghemawat et al. The Google file system. SOSP, pages 29–43, 2003.

[3] Konstantin Shvachko et al. The Hadoop Distributed File System. MSST, pages 1–10, 2010.

[4] Rodrigo Rodrigues and Barbara Liskov. High Availability in DHTs: Erasure Coding vs. Replication. IPTPS, pages 226–239, 2005.

[5] Huaxia Xia and Andrew A. Chien. RobuSTore: a distributed storage architecture with robust and high

performance. Supercomputing (SC '07), pages 44:1–44:11, 2007.

[6] A. J. McAuley. Reliable broadband communication using a burst erasure correcting code. SIGCOMM, pages 297–306, 1990.

[7] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *SIGCOMM Comput. Commun. Rev.*, 27(2):24–36, April 1997.

[8] Osama Khan et al. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. FAST, 2012.

[9] James S. Plank et al. A performance evaluation and examination of open-source erasure coding libraries for storage. FAST, pages 253–265, 2009.

[10] James Lee Hafner et al. Matrix methods for lost data reconstruction in erasure codes. FAST, pages 183–196, 2005.

[11] H. Weatherspoon and J.D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *Peer-to-Peer Systems*, 2002.

[12] Irving Reed and Golomb Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8(2):300–304, 06/1960 1960.

[13] Z. Wilcox-O'Hearn and B. Warner. *Tahoe-The Least-Authority FileSystem*. Allmydata Inc.

[14] Jason K. Resch and James S. Plank. AONT-RS: blending security and performance in dispersed storage systems. FAST, pages 191–202, 2011.

[15] J. N. England. A system for interactive modeling of physical curved surface objects. SIGGRAPH, pages 336–340, 1978.

[16] J. E. Stone et al. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science Engineering*, 12(3):66 –73, may-june 2010.

[17] John Nickolls et al. Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53, March 2008.

[18] Weiguo Liu et al. Bio-sequence database scanning on a GPU. IPDPS, pages 8–17, 2006.

[19] Shuai Che et al. A performance study of general-purpose applications on graphics processors using CUDA. *J. Parallel Distrib. Comput.*, 68(10):1370–1380, October 2008.

[20] TOP 500 Supercomputers. [Online] Available: http://www.top500.org/.

[21] Abdullah Gharaibeh et al. A GPU accelerated storage system. HPDC, pages 167–178, 2010.

[22] Vignesh T. Ravi et al. Scheduling Concurrent Applications on a Cluster of CPU-GPU Nodes. CCGRID, pages 140–147, 2012.

[23] Mahdi Nabiyouni and Delasa Aghamirzaie. A Highly Parallel Multi-class Pattern Classification on GPU. CCGRID, pages 148–155, 2012.

[24] Kamil Rocki and Reiji Suda. Accelerating 2-opt and 3-opt Local Search Using GPU in the Travelling Salesman Problem. CCGRID, pages 705–706, 2012.

[25] Linchuan Li et al. Experience of parallelizing cryo-EM 3D reconstruction on a CPU-GPU heterogeneous system. HPDC, pages 195–204, 2011.

[26] Mathias Björkqvist et al. Dynamic Replication in Service-Oriented Systems. CCGRID, pages 531–538, 2012.

[27] Christopher McConnell et al. Detouring and replication for fast and reliable internet-scale stream processing. HPDC, pages 737–745, 2010.

[28] Anne Benoit et al. Power-Aware Replica Placement and Update Strategies in Tree Networks. IPDPS, pages 2–13, 2011.

[29] Sriram Ramabhadran and Joseph Pasquale. Analysis of durability in replicated distributed storage systems. IPDPS, pages 1–12, 2010.

[30] Cristina L. Abad et al. DARE: Adaptive Data Replication for Efficient Cluster Scheduling. CLUSTER, pages 159–168, 2011.

[31] Qingsong Wei et al. CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster. CLUSTER, pages 188–196, 2010.

[32] Frank B. Schmuck and Roger L. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. FAST, pages 231–244, 2002.

[33] Tonglin Li et al. Exploring distributed hash tables in HighEnd computing. *SIGMETRICS Perform. Eval. Rev.*, 39(3):128–130, December 2011.

[34] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[35] Giuseppe DeCandia et al. Dynamo: amazon's highly available key-value store. SOSP, pages 205–220, 2007.

[36] Ion Stoica et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, February 2003.

[37] Yunhong Gu and Robert L. Grossman. Supporting Configurable Congestion Control in Data Transport Services. Supercomputing (SC '05), pages 31–41, 2005.

[38] James S. Plank et al. Jerasure: A library in C/C++ facilitating erasure coding for storage applications. Technical report, University of Tennessee, 2007.

[39] Matthew L. Curry et al. Gibraltar: A Reed-Solomon coding library for storage applications on programmable graphics processors. *Concurr. Comput. : Pract. Exper.*, 23(18):2477–2495, December 2011.

[40] Dongfang Zhao and Ioan Raicu. Distributed File System for Exascale Computing. Doctoral Research, Supercomputing (SC '12), 2012.