Towards High-Performance and Cost-Effective Distributed Storage Systems with Information Dispersal Algorithms

Dongfang Zhao[1], Kent Burlingame[1,2], Corentin Debains[1], Pedro Alvarez-Tabio[1], Ioan Raicu[1,3]
[1]Illinois Institute of Technology, Chicago, IL, USA
[2]Microsoft Corporation, Redmond, WA, USA
[3]Argonne National Laboratory, Argonne, IL, USA

***Abstract*—Reliability is one of the most fundamental challenges for high performance computing (HPC) and cloud computing. Data replication is the de facto mechanism to achieve high reliability, even though it has been criticized for its high cost and low efficiency. Recent research showed promising results by switching the traditional data replication to a software-based RAID. In order to systematically study the effectiveness of this new method, we built two storage systems from the ground up: a POSIX-compliant distributed file system (FusionFS) and a distributed key-value store (IStore), both supporting information dispersal algorithms (IDA) for data redundancy. FusionFS is crafted to have excellent throughput and scalability for HPC, whereas IStore is architected mainly as a light-weight key-value storage in cloud computing. We evaluated both systems with a large number of parameter combinations. Results show that, for both HPC and cloud computing communities, IDA-based methods with current commodity hardware could outperform data replication in some cases, and would completely surpass data replication with the growing computational capacity through multi/many-core processors (e.g. Intel Xeon Phi, NVIDIA GPU).**

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 extreme-scale distributed systems [2] (e.g. applications generated by Swift [3, 4]), 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) [5] and cloud computing [6, 7], 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) [8] makes 3 replicas as the default. The Hadoop distributed file system [9] also uses replication to achieve high reliability. This technique is often sufficient: it is easy to implement and has excellent performance (assuming data are replicated asynchronously), at the cost of space efficiency. For example, with the original data and 3 replicas, the storage utilization rate is only $\frac{1}{1+3} = 25\%$. In this case the cost of storage is quadrupled when building a distributed system, which might not be economically acceptable in many applications. 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 of data redundancy 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. Literature [10, 11] shows 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, thus could not saturate the network bandwidth. With the state-of-the-art GPU/many-core technology, this computing bottleneck could potentially be alleviated. For example, a GPU-based software RAID solution [12] was recently proposed, particularly optimized for Lustre [13].

In this paper we systematically study the effectiveness of information dispersal algorithms (IDA) that operate using both CPUs and GPUs in general. Unlike recently published work that focused on adding customized layers to existing fully-fledged systems, we built two storage systems from the ground up for HPC (FusionFS) and cloud computing (IStore) with built-in IDA support. Experimental results show that IDA outperforms data replication in some cases even with current commodity CPU/GPU, and suggest that IDA would eventually replace data replication due to IDA's higher storage efficiency and higher I/O throughput. Thus the major contribution of this paper is to showcase how to (better) integrate IDA into the next generation of distributed storage systems for both the HPC and cloud computing (e.g. MapReduce [14]) communities. This work is also applicable to the new computing paradigm of Many-Task Computing (MTC) [15–17].

II. Related work

Recently both academia and industry show an interest in erasure coding in storage systems. A GPU-accelerated software RAID was proposed in [12]. This system was particularly crafted for the Lustre parallel file system, and evaluated on three storage servers. It is so tightly coupled to Lustre that it might be difficult to be ported to other parallel or distributed file systems. Shredder [18] is a framework to leverage GPUs to efficiently chunk files in an incremental fashion. It has been integrated into HDFS [9] and evaluated

978-1-4799-0898-1/13/$31.00 ©2013 IEEE

on 15 nodes. Similarly, it does not discuss its portability to other systems. Panasas [19] has an object-based software RAID as the backbone of the upper-level file system. However it remains unknown at the file level how IDA would affect the performance. The next generation of Google File System, called Google Colossus [20], plans to implement erasure coding as part of the fault tolerance mechanism.

Some recent works have been focused on improving the the performance of replications with new replication algorithms/mechanisms, even though the storage efficiency of replications would unlikely change due to its nature. A dynamic replication was proposed for service-oriented systems in [21]. iFlow [22] is a replication-based system that can achieve both fast and reliable processing of high volume data streams on the Internet scale. DARE [23] is an adaptive data replication algorithm to improve data locality. CRDM [24] is a model to capture the relationship between the availability and the replica number.

III. Building Blocks

A. ZHT: a light-weight distributed hash table

ZHT [25] (Zero-hop distributed Hash Table) was originally designed as a general-purpose distributed hash table for HPC. We extended it in two directions: 1) it serves as the underlying storage for metadata of FusionFS [26] filesystem, and 2) it becomes the underlying data structure of the built-in support for distributed data provenance [27]. It would also replace the current metadata management in the HyCache [28] heterogeneous store system. ZHT has been tuned to scale up to 32K cores on IBM Blue Gene/P [29] supercomputer and 96 Amazon EC2 [30] instances.

B. FDT: an efficient and reliable file transfer service

We have developed our own data transfer service called FDT (Fast Data Transfer) with APIs provided by UDP-based Data Transfer (UDT) [31], 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 potentially higher speed than TCP under certain conditions.

C. Erasure coding libraries

A good review of erasure coding libraries was presented in [32]. In this paper, we have integrated two libraries Jerasure [33] and Gibraltar [34] as the default CPU and GPU libraries, respectively.

IV. Adding IDA support to FusionFS

A. FusionFS overview

FusionFS [26] is a POSIX-compliant distributed file system mounted on user space with FUSE [35]. Unlike most parallel/distributed file systems (e.g. [13, 36, 37]) for HPC, there is no distinction between compute and storage nodes. Each compute node actively participates in serving as one of the storage nodes.

B. IDA data redundancy in FusionFS

Different designs have been under our consideration on when to backup the primary copy. The first solution is to incrementally update replicas, simultaneously as the primary copy is modified. That is, whenever a file is opened, replicas are created and kept synchronous to whatever changes made to the primary copy. This method provides the strongest consistency at the block level. However making synchronous block-level replication introduces huge overhead to the I/O throughput.

To avoid the block-level synchronization, we could employ a "lazy" backup mechanism. That is, only when the file is modified and fully flushed to the disk, then the file is split up and sent to different nodes. This looks to be the most straightforward mechanism and easiest to implement, but brings an obvious concern: waiting for the primary copy to be written to the hard disk could take a long time, especially for data-intensive applications. Consequently, the end users would likely feel lags when saving files.

We finally decided to mix the above two methods: IDA is applied to the primary copy as soon as the file is closed. This method avoids the block-level synchronization, and operates before the potential I/O bottleneck of the underlying persistent storage. In the FusionFS implementation, IDA logic is implemented in the $fusion\_release()$ interface, which is exactly the point right after a file is closed but before it is flushed to the disk. As long as the file is closed (and still in memory), this file is considered "complete", and is ready to be split into n chunks by the libraries mentioned in Section III-C. These n chunks are then transferred to n different physical nodes by FDT as discussed in Section III-B.

V. Design and Implementation of IStore

A high-level architecture of IStore is shown in Figure 1. We assume the system has n nodes, or instances in the context of cloud computing. IStore installs two services on each node: 1) a distributed metadata management and, 2) a high-efficiency and reliable data transfer service. 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 logins on node $\#i$, then the IDA module is to encode or decode those files involved in the applications.

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, FDT 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. 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.

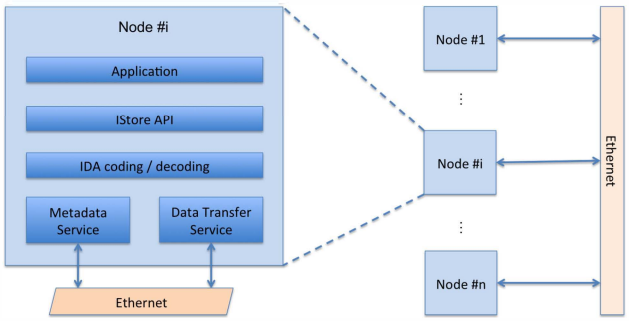


Fig. 1. An architectural overview of IStore deployed on n nodes

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).

As an example, Figure 2 illustrates the scenario when writing/reading a file with IStore for $k = 4$ and $m = 2$. On the left hand side when the original file (i.e. $orig.file$) is written, 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 table, which is also physically distributed across these 6 nodes. A file read request (on the right hand side) is essentially the reversed procedure of a file write: retrieving the metadata, transferring the data, and decoding the chunks.

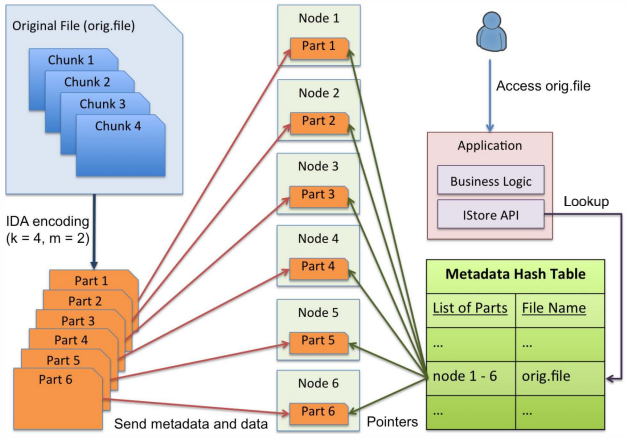


Fig. 2. An example of file encoding/decoding in IStore

VI. Evaluation

A. FusionFS

FusionFS is deployed on an 8-node cluster, each of which has an 8-core 3.1GHz AMD FX-8120 CPU, and an 384-core 900MHz NVIDIA GeForce GT640 GPU. Each node has 16GB RAM, and is interconnected via 1Gbps Eithernet. The operating system is OpenSUSE with Linux kernel 3.4.11 with CUDA version 5.0 installed.

In this experiment we keep the total number of nodes fixed (i.e. $n = k + m = 8$). The tolerated number of failures (i.e. m) ranges from 2 to 6. It is equivalent to having 3 to 7 replications (including the primary copy) in the naive data replication algorithm (REP). Figure 3 shows the utilization rates of IDA and REP algorithms. A larger m makes IDA closer to REP, and in the extreme case when $m = n - 1$, IDA would be degraded to REP where $E_{storage} = \frac{1}{n}$.

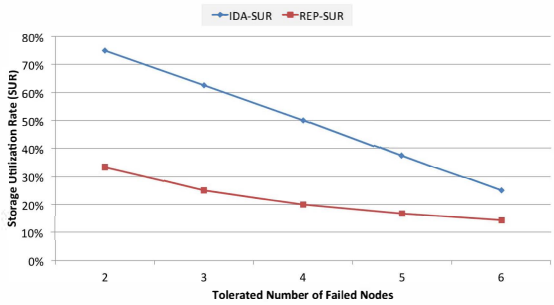


Fig. 3. Storage utilization rate of FusionFS with REP and IDA

Figure 4 shows the throughput of IDA- and REP-based data redundancy in FusionFS. Only when $m = 2$, REP slightly outperforms IDA, and the difference is almost negligible. Starting from $m = 3$, IDA clearly shows its advantage over REP, and the speedup increases for the larger m. Particularly, when $m = 6$, i.e. to keep the system's integrity allowing 6 failed nodes, IDA throughput is 1.82 higher than the traditional REP method.

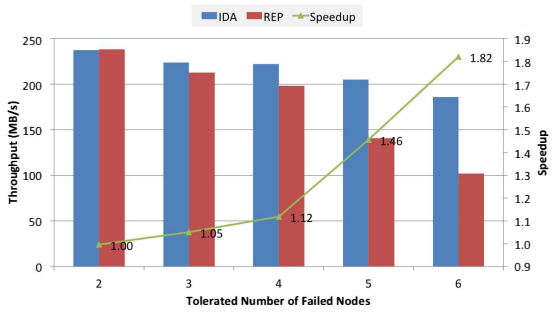


Fig. 4. Throughput of IDA and REP on FusionFS (block size 1MB)

We just showed that in FusionFS installed with a commodity GPU, IDA outperforms REP in terms of both performance and storage utilization (except for the edge case $m = 2$, where IDA and REP are comparable). We believe a high-end GPU would cause a larger gap, and make IDA the top candidate for data redundancy.

B. IStore

IStore has been deployed on a 32-node Linux cluster each node of which has 8GB RAM and dual AMD Opteron quad-

core processors (8-cores at 2GHz). The features of different algorithms to be evaluated are summarized in Table I.

TABLE I
COMPARISON ON REPLICATIONS, EFFICIENCY AND $(k:m)$ RATIOS

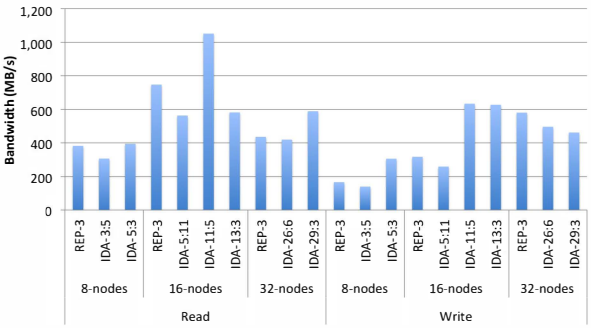
Algorithm	# Replications	$E_{storage}$	$(k:m)$
REP-3	3	33.33%	1:2
IDA-3:5	6	37.5%	3:5
IDA-5:3	4	62.5%	5:3
IDA-5:11	12	31.25%	5:11
IDA-11:5	6	68.75%	11:5
IDA-13:3	4	81.25%	13:3
IDA-26:6	7	81.25%	26:6
IDA-29:3	4	90.63%	29:3

We compare the read and write throughput of 8 different algorithms for four different file sizes: 1GB, 100MB, 10MB and 1MB, in Figure 5. The buffer size is set to 1MB. We observe that, besides the number of nodes, $k:m$ ratio also plays a critical role when 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. 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 higher aggregate throughput, in general.

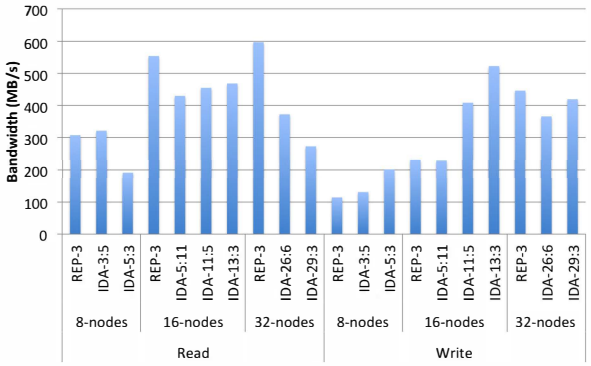
Even though it has been well accepted that traditional replication is faster than IDA approach, we found that by carefully tuning the parameters, IDA is able to outperform replications. For example, when reading a 1GB file, the following IDA algorithms deliver higher throughput than replication: IDA-5:3 on 8 nodes, IDA-11:5 on 16 nodes, and IDA-29:3 on 32 nodes. Similarly, when writing a 1GB file, the following IDA algorithms are faster than replication: IDA-5:3 on 8 nodes, IDA-11:5 and IDA-13:3 on 16 nodes. We believe that part of the explanation lies in the fact that replication uses more network bandwidth, e.g. making 3 replicas involves more than doubled amount of data to be transferred than an approach based on IDA. Not surprisingly, some other IDA algorithms cannot catch up with the REP-3 throughput. But note that we only use commodity CPUs for the encoding/decoding. We expect that in IStore with high-end CPUs or GPUs, IDA would deliver a (much) higher throughput than REP. Moreover, with a higher replication number, the REP algorithm would be slowed down more significantly than IDA, which we have seen in the FusionFS case in Figure 4.

VII. CONCLUSION

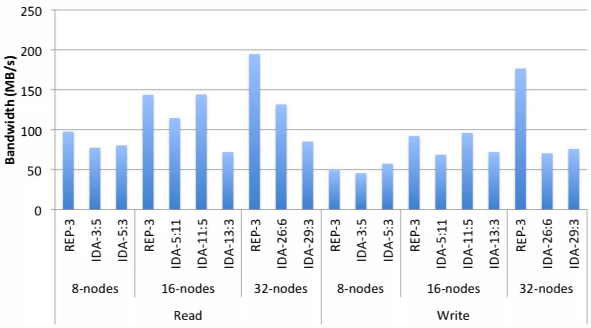
In this paper we designed and implemented two systems with built-in IDA support in HPC (FusionFS) and cloud computing (IStore). We found that even with current commodity CPU/GPU computing capacity, IDA could outperform the naive data replication in some cases. Based on our findings, we



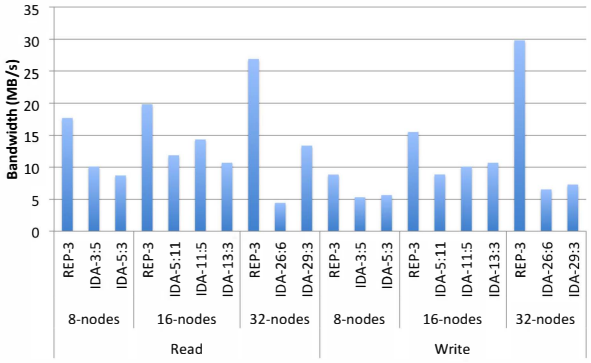
(a) File size = 1GB



(b) File size = 100MB



(c) File size = 10MB



(d) File size = 1MB

Fig. 5. IStore aggregate throughput of naive replication (REP-3) and IDA

predict that IDA would replace data replication soon with the emerging many-core chips e.g. Intel Xeon Phi [38] in future distributed systems [39].

ACKNOWLEDGEMENT

This work was partially supported by the National Science Foundation (NSF) under grant OCI-1054974.

REFERENCES

[1] A. S. Tanenbaum *et al.*, *Distributed Systems: Principles and Paradigms*, pp. 531–532. Prentice Hall; 2nd edition, 2006.

[2] I. Raicu, *et al.*, "The quest for scalable support of data-intensive workloads in distributed systems," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, 2009.

[3] Y. Zhao, *et al.*, "Swift: Fast, reliable, loosely coupled parallel computation," in *Services, 2007 IEEE Congress on*, 2007.

[4] M. Wilde, *et al.*, "Extreme-scale scripting: Opportunities for large task parallel applications on petascale computers," in *SCIDAC, Journal of Physics: Conference Series 180. DOI*, pp. 10–1088, 2009.

[5] D. Zhao, *et al.*, "Exploring reliability of exascale systems through simulations," in *Proceedings of the High Performance Computing Symposium*, HPC '13, 2013.

[6] Y. Zhao, *et al.*, "Opportunities and challenges in running scientific workflows on the cloud," in *Proceedings of the 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, CYBERC '11, 2011.

[7] I. Foster, *et al.*, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008.

[8] S. Ghemawat, *et al.*, "The Google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, 2003.

[9] K. Shvachko, *et al.*, "The Hadoop distributed filesystem: Balancing portability and performance," in *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[10] R. Rodrigues *et al.*, "High availability in DHTs: erasure coding vs. replication," in *Proceedings of the 4th international conference on Peer-to-Peer Systems*, IPTPS'05, 2005.

[11] H. Xia *et al.*, "RobuSTore: a distributed storage architecture with robust and high performance," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, 2007.

[12] A. Khasymski, *et al.*, "On the use of GPUs in realizing cost-effective distributed RAID," in *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '12, 2012.

[13] P. Schwan, "Lustre: Building a file system for 1,000-node clusters," in *Proceedings of the linux symposium*, p. 9, 2003.

[14] J. Dean *et al.*, "MapReduce: Simplified Data Processing on Large Clusters," in *Sixth Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[15] I. Raicu, *et al.*, "Middleware support for many-task computing," *Cluster Computing*, vol. 13, Sept. 2010.

[16] I. Raicu, "Many-task computing: Bridging the gap between high-throughput computing and high-performance computing," Doctoral dissertation, The University of Chicago, 2009.

[17] I. Raicu, *et al.*, "Many-task computing for grids and supercomputers," in *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, 2008.

[18] P. Bhatotia, *et al.*, "Shredder: GPU-accelerated incremental storage and computation," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, 2012.

[19] D. Nagle, *et al.*, "The Panasas ActiveScale storage cluster: Delivering scalable high bandwidth storage," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, 2004.

[20] A. Fikes, "Storage architecture and challenges," in *Google Faculty Summit*, 2010.

[21] M. Bjorkqvist, *et al.*, "Dynamic replication in service-oriented systems," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, 2012.

[22] C. McConnell, *et al.*, "Detouring and replication for fast and reliable internet-scale stream processing," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, 2010.

[23] C. L. Abad, *et al.*, "Dare: Adaptive data replication for efficient cluster scheduling," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, 2011.

[24] Q. Wei, *et al.*, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, 2010.

[25] T. Li, *et al.*, "ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table," in *IEEE International Parallel & Distributed Processing Symposium*, IEEE IPDPS '13, 2013.

[26] D. Zhao *et al.*, "Distributed File System for Exascale Computing," Doctoral Research, Supercomputing (SC '12), 2012.

[27] D. Zhao, *et al.*, "Distributed data provenance for large-scale data-intensive computing," in *IEEE International Conference on Cluster Computing*, IEEE CLUSTER '13, 2013.

[28] D. Zhao *et al.*, "HyCache: a user-level caching middleware for distributed file systems," International Workshop on High Performance Data Intsensive Computing, IEEE IPDPS '13, 2013.

[29] "Intrepid," https://www.alcf.anl.gov/resource-guides/intrepid-file-systems.

[30] Amazon Elastic Compute Cloud (Amazon EC2), " http://aws.amazon.com/ec2/."

[31] Y. Gu *et al.*, "Supporting configurable congestion control in data transport services," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, 2005.

[32] J. S. Plank, *et al.*, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *Proccedings of the 7th conference on File and storage technologies*, FAST '09, 2009.

[33] J. S. Plank, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications," tech. rep., University of Tennessee, 2007.

[34] M. L. Curry, *et al.*, "Gibraltar: A Reed-Solomon coding library for storage applications on programmable graphics processors," *Concurr. Comput. : Pract. Exper.*, vol. 23, pp. 2477–2495, Dec. 2011.

[35] FUSE Project, "http://fuse.sourceforge.net."

[36] F. Schmuck *et al.*, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, 2002.

[37] I. F. Haddad, "PVFS: A parallel virtual file system for linux clusters," *Linux J.*, vol. 2000, Nov. 2000.

[38] "Intel xeon phi coprocessor 5110p," http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html.

[39] I. Raicu, *et al.*, "Making a case for distributed file systems at exascale," in *Proceedings of the third international workshop on Large-scale system and application performance*, LSAP '11, 2011.