# A discrete data dividing approach for erasure-code based storage applications

Weidong Sun, Yijie Wang, Yongquan Fu, Xiaoqiang Pei
*Science and Technology on Parallel and Distributed Processing Laboratory,*
*National University of Defense Technology,*
*Changsha, Hunan, P. R. China, 410073*
*Email: weidongsun, wangyijie, yongquanf, xiaoqiangpei@nudt.edu.cn*

***Abstract*—Erasure codes are promising for improving the reliability of the storage system due to its space efficiency compared to the replication methods. Traditional erasure codes split data into equal-sized data blocks and encode strips in different data blocks. This brings heavy repairing traffic when clients read parts of the data, since most strips read for repairing are not in the expected blocks. This paper proposes a novel discrete data dividing method to completely avoid this problem. The key idea is to encode strips from the same data block. We could see that for repairing failed blocks, the strips to be read are either in the same data block with corrupted strips or from the encoded strips. Therefore, no data is wasted. We design and implement this data layout into a HDFS-like storage system. Experiments over a small-scale testbed shows that the proposed discrete data divided method avoids downloading data blocks that are not needed for clients during the repairing operations.**

***Keywords*-Erasure Codes; Distributed Storage; Data Dividing; Partial Reading; HDFS**

## I. Introduction

We have entered the **big data** era. The past decades have witnessed ever-growing amounts of data volumes from all aspects of the society, including high energy physics, astronomy, bio-informatics, social networks [1], [2], etc. The large data volumes bring great opportunities for the storage industry, but also introduce new stringent challenges on storing and managing the data efficiently with good performance.

In modern distributed storage systems, the large scales of nodes and data volumes lead to many challenges. One is the failure-handling operation, as the failures have been routines rather than exceptions for large-scale infrastructures [3].

Consequently, redundancy technologies are indispensable to ensure the data reliability. Replication and erasure codes are two approaches for improving the reliability. Though the replication technique has been vastly employed in traditional distributed storage systems [4], [5] for its simplicity and high reading/writing efficiency, it is too expensive due to the low storage efficiency [6], [7], [8]. In contrast, erasure codes can significantly decrease the storage space, yielding greener energy consumption, higher reliability, and lower management cost. Therefore, the erasure-code technique is promising for large-scale distributed storage systems.

**Partial-reading** operations arise when consecutively requesting only a successive parts of a data object. The partial-reading is a common file-access pattern in storage system, taking over 70% of read operations [9]. If a client requests existing data blocks, the storage system directly responds the corresponding blocks.

Unfortunately, in a large scale storage applications, failures occur frequently. Many read accesses are performed under failure circumstances. The failed blocks can be repaired with lazy strategies. However, in order to repair the requested part, as much as $k$ times of data volumes are downloaded in order to reconstruct the requested part. Since the encoded strips are constructed by strips in different blocks in traditional erasure codes. This procedure not only decreases the speeds of the read operations, but also wastes a lot of the infrequent bandwidth resources.

In this paper, we propose a discrete data dividing approach in order to mitigate the above issue. $k$ successive strips in the original data block are uploaded into $k$ different blocks in the storage system, rather than in the same block. Therefore, these successive $k$ data strips are not only used for repairing failed strips, but also useful for clients.

Further, the discrete data dividing helps improve the system throughput. Firstly, the successive data reading/writing operations are more efficient in the discrete approach than those in prior approaches, as the magnetic head movement and rotation of disks are avoided due to reading adjacent regions. Secondly, when performing the partial-reading operations, the $k$ successive data strips for repairing are also requested by clients in the partial-reading. Therefore, the downloaded data volumes and the decoding computation are both decreased.

The remainder of this paper is organized as follows. In Sect. II, we briefly describe the problem of the data dividing, and analyze the drawbacks of traditional approaches. In Sect. III, we present the discrete data dividing approach, together with a detailed analysis of its performance. In Sect. IV, we present the implementation of a prototyping distributed erasure-code based storage and report our experiments over a 20-node testbed. We outline our conclusions and point out some directions for future work in Sect. V.

## II. Problem Description

In this section, we provide the basic concepts of erasure codes at first, and then describe the dividing problem of data dividing in the erasure-code based distributed storage applications.

### A. Preliminaries

Erasure codes stem from the data communication community, then were introduced to the storage systems for their high data reliability and storage efficiency properties. In the erasure-code based storage applications, each data object is firstly divided into $k$ equal sized **data blocks**, labeled as $D_0, D_1, \cdots, D_{k-1}$. Then an encoding process is performed on these $k$ data blocks to generate $m$ **parity blocks**, labeled as $P_0, P_1, \cdots, P_{m-1}$. Eventually, all the $n$ blocks are stored separately in $n$ different storage nodes, where $n = k + m$. When reading the original data object, one needs to download at least $k$ blocks[1] from the $n$ blocks at first, where there are $s$ data blocks and $k - s$ parity blocks. And then these $k$ downloaded blocks are decoded to generate $k - s$ data blocks which do not appear in the downloaded blocks. Eventually, the downloaded $k$ data blocks and the generated $k - s$ data blocks are merged to reconstruct the original data object. Fig. 1 and Fig. 2 illustrate the writing and reading process of the erasure-code based storage applications, respectively.

Practically, the encoding process is executed on some fine grained pieces of data strips rather than on the whole blocks [11]. Each time, $k$ ***data strips***, noted as $DS_0, DS_1, \cdots, DS_{k-1}$, are read from $k$ different data blocks into the memory. An encoding process is performed on these $k$ data strips to generate $m$ ***parity strips***, noted as $PS_0, PS_1, PS_{m-1}$. Then these $k$ data strips and $m$ generated parity strips are upload to $k+m$ different storage nodes respectively. And the process repeats until all the data object are encoded. For description simplicity, the $k$ data strips and $m$ generated parity strips in a single encoding process are defined as a **stripe**.

### B. Traditional Continuous Data Dividing

Traditional dividing approach is an intuitive and simple approach, in which the data object is first divided continuously into $k$ equal sized successive data blocks, as shown in Fig.3. When writing the data object to the distributed storage system, $k$ data strips are read from $k$ different data blocks into the memory each time. An encoding process is performed to generate $m$ parity strips. After that, all these $n$, where $n = k + m$, strips are uploaded to $n$ different nodes in the distributed storage system. The process repeated until the whole data object have been encoded and written to the system.[2] If some of the data blocks are lost, at least $k$ data blocks must be downloaded to perform the decoding process and reconstruct the original data object. There exist two significant drawbacks in this continuous dividing approach: disobey with the principle of locality, and low efficiency of partial-reading.

*1) Data locality:* The original locality principle means that when a program is executed, memory references are localized temporally and spatially. Because of locality, the cache and I/O prefetch successive data blocks. Consequently, successive data access gains higher performance than random data access. Continuous data dividing disobey with the locality principles in both the data writing process and reading process. When writing data to the system, each time, $k$ strips are read from local disk with non-successive positions to perform the encoding, which results in magnetic head movements and rotations. When reading data from the system to local disk, $k$ data strips from one decoding calculation must be written to non-successive positions on local disk. This also incurs magnetic head movement and brings in much I/O overhead and decrease the system throughput. If the downloaded data are processed without I/O writing, locality principles also tell us that non-successive strips will increase the cache miss rate.

*2) Partial Reading:* Partial-reading is an important and common access pattern in storage applications. Unfortunately, partial-reading is resource-and time-consuming in the continuous data dividing approach. When there is no data block failed, partial-reading just read the required data from corresponding blocks. If there are data block failures, partial-reading of the lost data part have to download as much as $k$ times of the requested data volumes, as shown in Fig. 4. This not only lowers the partial reading throughput, but also leads to heavy network traffic.

## III. Discrete Data Dividing

To address the above problems discussed above in continuous data dividing approach, we propose a discrete data dividing approach in this section.

### A. Working process

We observed that the data strips in a decoding stripe of continuous approach are non-successive, therefore, not all of the data strips in each decoding stripe are used to reconstruct the requested part. In the discrete data dividing approach, the data object is not divided into continuous data blocks at first. In contrast, the data object is first divided into logical data strips. These data strips are divided into groups, each of which consists of $k$ successive strips. $0s$ are padded when necessary to make the data object to be divided by both $k$ and strip size. Then an encoding process is executed on each

[1]Though our approach adapts to all erasure codes, here we only consider the MDS erasure codes for description simplicity, more information about erasure codes can be referred to [10].

[2]If the data object size can not be divided by $k$, the last block is padded with $0s$ so that all the blocks are equal sized. Similar operation is performed to make sure that all the strips are equal sized.

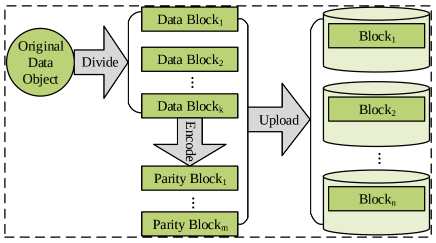

Figure 1. A general writing process in the erasure-code based storage system.

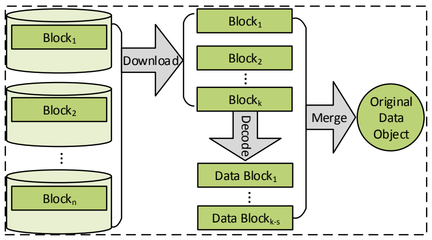

Figure 2. A general reading process in the erasure-code based storage system.

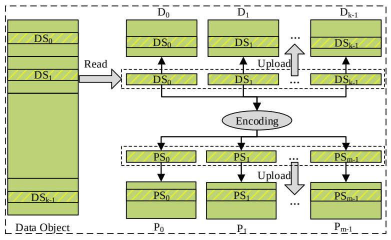

Figure 3. The writing process of continuous data dividing approach.

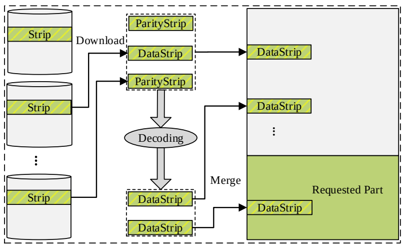

Figure 4. The partial-reading process in the continuous data divide approach.

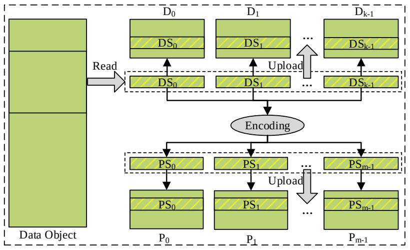

Figure 5. The writing process of a discrete data dividing approach.

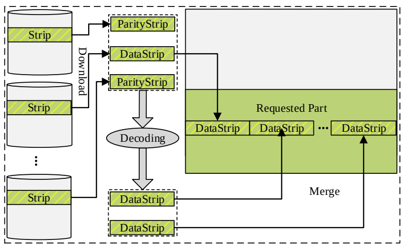

Figure 6. The partial-reading process of the discrete data divide approach.

group to generate $m$ parity strips, and then all these $k+m$ strips are distributed to construct $k+m$ different blocks, as shown in Fig. 5 and Algorithm 1.

**Algorithm 1:** Algorithm of the discrete data dividing approach

```
Input  : filesize: size of the data object.
         stripsize: size of the strip.
         k: number of data blocks in the erasure codes.
Output: $block_i, 0 \leq i < k$
1 begin
2     $stripNum \leftarrow ceil[\frac{filesize}{stripsize}]$ ;
3     $stripeNum \leftarrow ceil[fracstripNumk]$ ;
4     for $i \leftarrow 0$ to $stripeNum - 1$ do
5         for $j \leftarrow 0$ to $k - 1$ do
6             assign $strip_{i*k+j}$ to $block_j$ ;
7         end
8     end
9 end
```

Based on the discrete data dividing approach, the partial-reading can acquire $k$ successive strips in a single decoding process, as shown in Algorithm 2.

### B. Analysis

In the discrete dividing approach, the stripe consists of successive strips. When some data blocks are lost, this successive-strip based stripe can get $k$ successive strips in one decoding operation. Consequently, both the number of decoding operations and the downloaded data volumes are decreased. And the partial-reading performance of the system can be upgraded greatly. The discrete dividing approach gains some advantages as following:

- When reading data from local disk for encoding, the successive data strips are read consecutively. Thus magnetic head movement and rotation are avoided, I/O throughput are improved, as shown in Fig. 5.
- Each time, $k$ successive data strips are acquired in a single decoding calculation. All these $k$ strips are utilized to compose the request part, except for the start or the end, as shown in Fig. 6. Therefore, the downloaded data volumes nearly equal to that of the request data volumes, rather than $k$ times of that in the continuous approach. This decreases both the downloaded data volumes and the decoding calculations.
- When storing data into local disk, $k$ successive strips are generated in a single decoding calculation in the discrete data dividing approach, thus the magnetic head movement and rotation of local disk are also avoided.

## IV. Experiments and Evaluation

### A. Implementation

To evaluate the performance of our proposed discrete dividing approach, we design and implement an prototype system of erasure-code based distributed storage. The prototype owns a similar architecture with GFS [4] and HDFS [5], [12], and employs the **Ice 3.5.1** [13] middle-ware as the communication service.

The prototype system has a single **namenode**, which stores all the meta information of the distributed storage system, including the nodes, the encoding parameters, the stored

**Algorithm 2:** The partial-reading process in the discrete data dividing approach

```
Input : $offset$: the offset of the request content in the data object.
        $length$: the length of the request content.
        $liveBlocks$: the live block list to read data from.
        $stripsize$: the size of the strip.
        $k$: the number of data blocks in the erasure codes.
1  begin
2      $decoding \leftarrow false$;
3      $stripesSet \leftarrow \emptyset$;
4      $beginStripeIdx \leftarrow offset/(stripsize * k)$;
5      $endStripeIdx \leftarrow (offset + length - 1)/(stripsize * k)$;
6      if $k \geq numberofdatablocksinliveBlocks$ then
7          $decoding = true$;
8      end
9      if $decoding = true$ then
10         for $index \in [beginStripeIdx, endStripeIdx]$ do
11             Download $strip_{index}$ from each block in $liveBlocks$ to compose $stripe_{index}$;
12             Decode $stripe_{index}$ to get lost data strips;
13             for each downloaded or decoded data strip $strip_{index}$ do
14                 Write $strip_{index} \cap requestPart$ to the result file;
15             end
16         end
17     else
18         for $index \in [beginStripeIdx, endStripeIdx]$ do
19             Download $strip_{index}$ from each data block in $liveBlocks$;
20             Write $strip_{index} \cap requestPart$ to the result file;
21         end
22     end
23 end
```

data objects, the blocks, and their mapping relationships, etc. We can get all kinds of information we need from the namenode, except for the data content. The prototype system has a large number of **blocknodes**, which stores data blocks and parity blocks. Each blocknode make a registration to the namenode at the beginning, then they communicate through heartbeat information. The data object is divide, encoded, and decoded on the client, thus the dividing approach is also implemented in the client.

## B. Experiments setup

The prototype is deployed on a test bed consists of 20 server nodes, among which there are one namenode and 19 blocknodes. The hardware and software environments are listed in Table I.

Table I
PARAMETER SETTINGS.

| Parameter | Value |
|---|---|
| CPU Type | Intel Xeon E-2640 |
| Number of CPUs | 4 |
| Number of cores per CPU | 6 |
| Memory Size | 64GB |
| Network Card | Intel(R) PRO/1000 |
| Operating System | Red Hat Enterprise Linux Server 6.1 |
| Kernel version | Linux 2.6.32-131.0.15.el6.x86_64 |
| Compiler | Java Development Kit 6 64bit |
| Communication Middle-ware | Ice 3.5.1 |
| Number of Server Nodes | 20 |

Without special notation, the encoding parameters are arbitrarily selected, with $k = 6$, $m = 5$, and the **stripsize=**$8K$. In the discrete dividing approach, the decoding speed can reach up to $32MB/s$, while in the continuous dividing approach, the decoding speed is limited within $28MB/s$. The reason we analyzed is that in the continuous dividing approach, the cache misses are larger than that in discrete dividing approach.

We perform five experiments both with continuous dividing approach and discrete dividing approach, so as to evaluate different metrics, described as follows.

- Upload experiment, noted as **Upload**. In this experiment, 10 files sized from $100MB$ to $2000MB$ are uploaded to the distributed storage system, so as to evaluate the upload speed.
- Download without decoding experiment, noted as **Download-D**. In this experiment, 10 file that have been uploaded are downloaded from the system, so as to evaluate the download speed. And all the data blocks of the downloaded file are available, thus the original data object is merged without decoding.
- Download with decoding experiment, noted as **Download-P**. In this experiment, 10 files that have been uploaded are downloaded from the system, so as to evaluate the download speed with decoding. Different number of data blocks, varied from 1 to $m$, are lost, thus decoding is necessary to regenerate the original data object.
- Partial-download without decoding experiment, noted as **Partial-D**. In this experiment, 1000 **partial-reading** operations for each of the 10 uploaded file are generated according to a uniform distribution. Each partial-reading consists of a two random variables, **offset** and **length**. The **offset** is uniformly distributed in range **[0, filesize**), and **length** is uniformly distributed in range **[0, filesize-offset]**. These $10*1000$ corresponding

downloads are performed to evaluate the partial-reading speed. All the data blocks are available and no decoding is needed.

- Partial-download with decoding experiment, noted as **Partial-P**. In this experiment, $10,000$ partial-reading operations same to the above are performed, except that some of data blocks are lost, thus decoding may needed to regenerate the required data part.

In our experiments, all the evaluations are performed more than $10$ times to get average results.

### *C. Evaluation Results and Analysis*

We first calculate the number of average strips downloaded in the reading operations, as shown in Fig. 7. The results is evaluated with a file sized of $1GB$. We plot three bars at each X-axis scale, which represent the number of failed data blocks. The first bar denotes the number of strips that the partial-reading requests, the bottom part represents the number of failed strips, and the top part represents the active strips. The total number of strips that are downloaded in the continuous approach is denoted by the second bar, the bottom part of which represents the number of strips that are used for decoding, and the top part represents the number of strips that do not participate the decoding operation. The third bar denotes the number of strips that are downloaded in the discrete approach. Fig. 7 indicates that the number of downloaded strips in the continuous approach is far larger than the number of requested strips. Moreover the number strips in the continuous grows fast as the number of failed data blocks increases, and most of the strips are downloaded for decoding. In the discrete approach, the number of downloaded strips almost equals to the number of requested strips, and all of the strips are utilized for decoding.

Fig. 8 shows the speed distribution in all the repeated experiments. The bottom and top of each box are the first and third quartiles, the band inside the box is the median, and the whiskers denote the minimum and maximum value. The left gray box represents the continuous data dividing approach, and the right green box represents the discrete data dividing approach. From the figure, we can see that the discrete data dividing approach outperforms the continuous approach in all the five experiments. Fig. 9 shows that when there are some data blocks failed, the discrete approach can improve the partial-download speed about $80\%$. And in other experiments, the speeds are also improved a little.

In Fig. 10, we depict the average download speeds of the continuous and discrete data dividing approaches when the number of failed data blocks varies from $0$ to $5$, respectively. The speedup is in the right axis. In Fig. 11, we draw the average partial-download speeds of continuous and discrete data dividing approach when the number of failed data blocks varies from $0$ to $5$. The speedup is depicted in the right axis. These two figures demonstrate that the speedup increases as the number of failed data blocks grows.

The transferred data volumes are demonstrated in Fig.12 as the number of failed data blocks varies from $0$ to $5$. The figure shows that in the continuous data dividing approach, the transferred data volumes are larger than the required data volumes, and grows dramatically as the number of failed data blocks grows. In the discrete approach, the size of transferred data volumes almost equal to the required data volumes, and is far less than the continuous approach.

## V. Conclusions and Future Directions

To improve the efficiency of the erasure-code based distributed storage, we propose a discrete data dividing approach to avoid wasted reads in repairing failed blocks during the partial-read periods. We implement our method in a prototype system and deploy the system in a 20-nodes testbed. Compared to the continuous approach, our proposed discrete approach improves the upload speed and the download speed. Data volumes downloaded in the partial-reading operation are decreased significantly. The discrete data dividing approach decreases the transferred data volumes in the partial-reading process. However, it brings little improvements for the data regeneration. Further efforts are required to explore the trade offs.

## Acknowledgment

This work was supported by the National Grand Fundamental Research 973 Program of China (Grant No.2011CB302601), the National Natural Science Foundation of China (Grant No.61379052), the National High Technology Research and Development 863 Program of China (Grant No.2013AA01A213), the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No.S2010J5050), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No.20124307110015).

## References

[1] Y. Wang, W. Sun, S. Zhou, X. Pei, and X. Li, "Key technologies of distributed storage for cloud computing," *Journal of Software*, vol. 23, no. 4, pp. 962–986, 2012.

[2] X. Lu, H. Wang, J. Wang, J. Xu, and D. Li, "Internet-based virtual computing environment: Beyond the data center as a computer," *Future Generation Computer Systems (FGCS)*, vol. 29, pp. 309–322, 2011.

[3] W. SUN, Y. WANG, and X. PEI, "Tree-structured parallel regeneration for multiple data losses in distributed storage systems based on erasure codes," *China Communications*, vol. 10, no. 4, pp. 113–125, 2013. [Online]. Available: http://www.chinacommunications.cn/EN/abstract/article_8119.shtml

[4] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proceedings of ACM SIGOPS Operating Systems Review*, vol. 37. ACM, 2003, pp. 29–43.

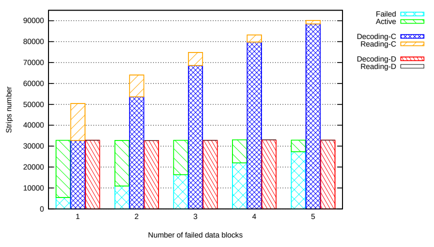

Figure 7. The number of downloaded strips in the continuous and discrete approaches.

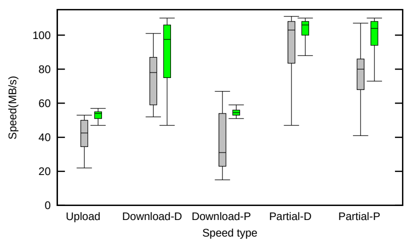

Figure 8. The speed distribution of the continuous and discrete approaches.

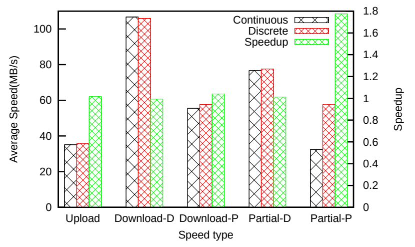

Figure 9. The average speed comparison between the continuous and discrete approaches.

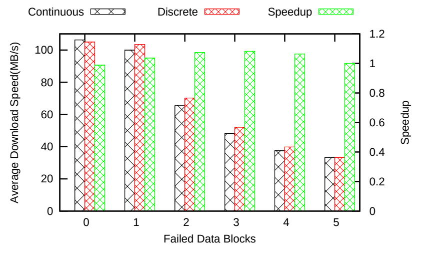

Figure 10. The whole-file reading speed as a function of the number of failed data blocks.

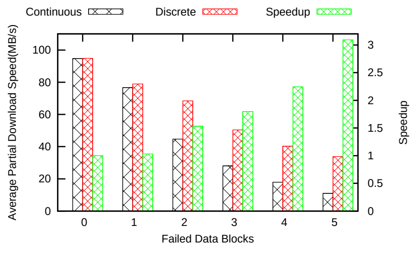

Figure 11. The partial-reading speed as a function of the number of failed data blocks.

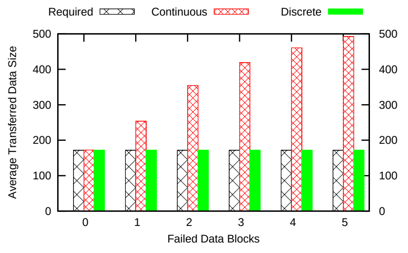

Figure 12. The data size as a function of the number of failed data blocks.

[5] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, p. 21, 2007.

[6] Y. Wang and S. Li, "Research and performance evaluation of data replication technology in distributed storage systems," *International Journal of Computers and Mathematics with Applications*, vol. 51, no. 11, pp. 1625–1632, 2006.

[7] Z. Zhang, A. Deshpande, X. Ma, E. Thereska, and D. Narayanan, "Does erasure coding have a role to play in my data center?" Microsoft Research, Tech. Rep. MSR-TR-2010-52, 2010.

[8] *DiskReduce: RAID for Data-Intensive Scalable Computing*. Portland: ACM, 2009.

[9] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "Measurements of a distributed file system," in *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '91. New York, NY, USA: ACM, 1991, pp. 198–212. [Online]. Available: http://doi.acm.org/10.1145/121132.121164

[10] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Amsterdam. The Netherlands: North Holland, 1977.

[11] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications - Version 1.2," http://www.cs.utk.edu/~plank/plank/papers/CS-08-627.html, University of Tennessee, Tech. Rep. CS-08-627, August 2008.

[12] *The Hadoop Distributed File System*. Lake Tahoe, Nevada, USA: IEEE Computer Society, 2010.

[13] H. of Ice, "Ice project," http://www.zeroc.com/, August 2013.