11-2009

DiskReduce: RAID for Data-Intensive Scalable Computing (CMU-PDL-09-112)

Bin Fan
Carnegie Mellon University

Wittawat Tantisiriroj
Carnegie Mellon University

Lin Xiao
Carnegie Mellon University

Garth Gibson
Carnegie Mellon University, garth@cmu.edu

DiskReduce: RAID for Data-Intensive Scalable Computing

Bin Fan
Carnegie Mellon University
binfan@cs.cmu.edu

Wittawat Tantisiriroj
Carnegie Mellon University
wtantisi@cs.cmu.edu

Lin Xiao
Carnegie Mellon University
lxiao@cs.cmu.edu

Garth Gibson
Carnegie Mellon University
garth@cs.cmu.edu

ABSTRACT

Data-intensive file systems, developed for Internet services and popular in cloud computing, provide high reliability and availability by replicating data, typically three copies of everything. Alternatively high performance computing, which has comparable scale, and smaller scale enterprise storage systems get similar tolerance for multiple failures from lower overhead erasure encoding, or RAID, organizations. DiskReduce is a modification of the Hadoop distributed file system (HDFS) enabling asynchronous compression of initially triplicated data down to RAID-class redundancy overheads. In addition to increasing a cluster's storage capacity as seen by its users by up to a factor of three, DiskReduce can delay encoding long enough to deliver the performance benefits of multiple data copies.

1. INTRODUCTION

The Google file system (GFS)[11] and Hadoop distributed file system (HDFS)[5], defined data-intensive file systems. They provide reliable storage and access to large scale data by parallel applications, typically through the Map/Reduce programming framework [10]. To tolerate frequent failures, each data block is triplicated and therefore capable of recovering from two simultaneous node failures. Though simple, a triplication policy comes with a high overhead cost in terms of disk space: 200%. The goal of this work is *to reduce the storage overhead significantly while retaining double node failure tolerance and multiple copy performance advantage.*

We present *DiskReduce*, an application of RAID in HDFS to save storage capacity. In this paper, we will elaborate and investigate the following key ideas:

- **A framework** is proposed and prototyped for HDFS to accommodate different double failure tolerant encoding schemes, including a simple "RAID 5 and mirroring" encoding combination and a "RAID 6" encoding. The framework is extensible by replacing an encoding/decoding module with other double failure tolerant codes, and could easily be extended to higher failure tolerance.

- **Asynchronous and delayed encoding**, based on a trace of the Yahoo! M45 cluster [2], enables most applications to attain the performance benefit of multiple copies with minimal storage overhead. With M45 usage as an example, delaying encoding by as little as an hour can allow almost all accesses to choose among three copies of the blocks being read.

The balance of the paper is as follows: In Section 2, we discuss work related to DiskReduce. We present its design and prototype in Section 3 and Section 4. Section 5 discusses deferred encoding for read performance. We conclude in Section 6.

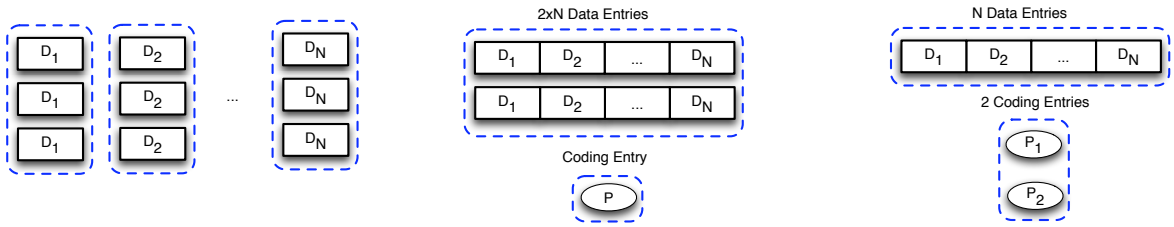
2. RELATED WORK

Almost all enterprise and high performance computing storage systems protect data against disk failures using a variant of the erasure protecting scheme known as Redundant Arrays of Inexpensive Disks [16]. Presented originally as a single disk failure tolerant scheme, RAID was soon enhanced by various double disk failure tolerance encodings, collectively known as RAID 6, including two-dimensional parity [12], P+Q Reed Solomon codes [20, 8], XOR-based EvenOdd [3], and NetApp's variant Row-Diagonal Parity [9]. Lately research as turned to greater reliability through codes that protect more, but not all, sets of larger than two disk failures [13], and the careful evaluation of the tradeoffs between codes and their implementations [17].

Networked RAID has also been explored, initially as a block storage scheme [15], then later for symmetric multi-server logs [14], Redundant Arrays of Independent Nodes [4], peer-to-peer file systems [22] and is in use today in the PanFS supercomputer storage clusters [23]. This paper explores similar techniques, specialized to the characteristics of large-scale data-intensive distributed file systems.

Deferred encoding for compression, a technique we use to recover capacity without loss of the benefits of multiple copies for read bandwidth, is similar to two-level caching-and-compression in file systems [7], delayed parity updates in RAID systems [21], and alternative mirror or RAID 5 representation schemes [24].

Supercomputing PDSW '09, Nov. 15, 2009. Portland, OR, USA.



(a) Triplication (b) RAID 5 and mirror (c) RAID 6

Figure 1: Codewords providing protection against double node failures

Finally, our basic approach of adding erasure coding to data-intensive distributed file systems has been introduced into the Google File System [19] and, as a result of an early version of this work, into the Hadoop Distributed File System [6]. This paper studies the advantages of deferring the act of encoding.

3. DESIGN

In this section we introduce DiskReduce, a modification of the HDFS[5].

3.1 Hadoop Distributed File System

HDFS[5] is the native file system in Hadoop[1], an open source Map/Reduce parallel programming environment, and is highly similar to GFS[11]. HDFS supports write-once-read-many semantics on files. Each HDFS cluster consists of a single metadata node and a usually large number of data nodes. The metadata node manages the namespace, file layout information and permissions. To handle failures, HDFS replicates files three times.

In HDFS, all files are immutable once closed. Files are divided into blocks, typically 64MB, each stored on a data node. Each data node manages all file data stored on its persistent storage. It handles read and write requests from clients and performs "make a replica" requests from the metadata node. There is a background process in HDFS that periodically checks for missing blocks and, if found, assigns a data node to replicate the block having too few copies.

3.2 DiskReduce Basics

One principle of the DiskReduce design is to minimize the change to original HDFS logic. Specifically, DiskReduce takes advantage of following two important features of HDFS: (1) files are immutable after they are written to the system and (2) all blocks in a file are triplicated initially. DiskReduce makes no change to HDFS when files are committed and triplicated. Then DiskReduce exploits the background re-replication in HDFS, but in a different way: in HDFS the background process looks for insufficient number of copies, while in DiskReduce it looks for blocks with high overhead (i.e. blocks triplicated) that can be turned into blocks with lower overhead (i.e. RAID encoding). Redundant blocks will not be deleted before the encoding is done to ensure data reliability during encoding phase. Since this process is inherently asynchronous, DiskReduce can further delay encoding, when space allows, to facilitate temporally local accesses to choose among multiple copies.

3.3 Encoding

Files are written initially as three copies on three different data nodes. We later *compress* the capacity used by encoding redundancy and deleting the excess copies.

In our prototype, we have implemented two codes:

- **RAID 5 and Mirror** As shown in Figure 1(b), we both maintain a mirror of all data and a RAID 5 encoding. The RAID 5 encoding is only needed if both copies of one block are lost. In this way, the storage overhead is reduced to $1+1/N$ where N is the number of blocks in the parity's RAID set.
- **RAID 6** DiskReduce also implements the leading scheme for double disk failure protection as shown in Figure 1(c). The storage overhead is $2/N$ where N is the number of data blocks in a RAID set.

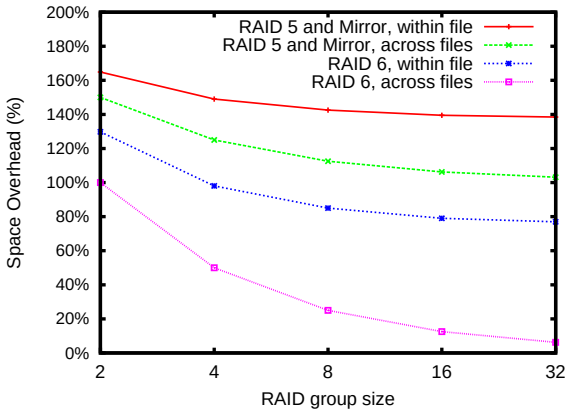


Figure 2: Space overhead by different grouping strategies according to the file size distribution on the Yahoo! M45 cluster. The overhead of triplication is 200%

Based on a talk about our previous DiskReduce work [6], a userspace RAID 5 and mirror encoding scheme has been implemented on top of HDFS and may appear in the next HDFS release. In that implementation, only blocks from the same file will be grouped together. Alternatively, Figure 2 shows the capacity overhead derived from a file size distribution from the Yahoo! M45 cluster for two encoding schemes: blocks grouped for encoding within a file or grouped across

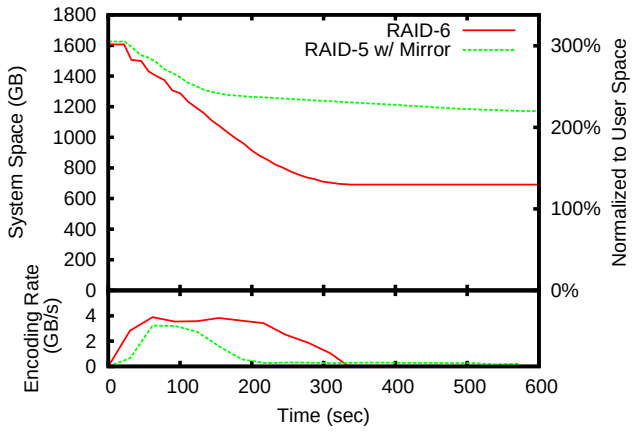


Figure 3: Storage utilized and rate of capacity recovered by encoding

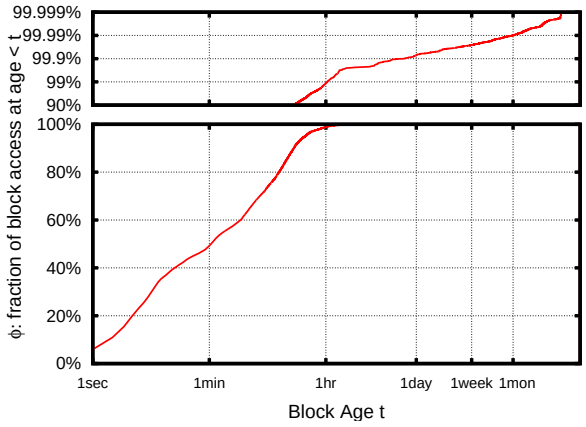


Figure 4: CDF of block age at time of access

files. M45 is a cluster with approximately 4,000 processors, three terabytes of memory, and 1.5 petabytes of disk. We can see that grouping across files may result in 40% reduction in capacity overhead for RAID 5 and mirror and 70% for RAID 6 because files on M45 are typically small relative to 64 MB blocks, and users often split data sets into many files. Our prototype explores this difference by grouping consecutively created blocks regardless of file boundaries on each node.

4. DISKREDUCE PROTOTYPE

We have prototyped DiskReduce as a modification to the Hadoop Distributed File System (HDFS) version 0.20.0. Currently, the DiskReduce prototype supports only two encoding schemes: RAID 6 and RAID 5 and mirror. The RAID 6 encoding scheme uses the open-source Blaum-Roth RAID 6 code released in the Jerasure erasure coding library developed by the University of Tennessee [18]. The RAID 5 and mirror encoding scheme uses a simple block XOR operation written in Java to replace one of the three data copies of N blocks with a single-block RAID parity. Without comments, our prototype implementation consists of less than 2,000 lines of Java code along with the Jerasure erasure coding library which is itself about 7,000 lines of C code.

Our prototype runs in a cluster of 63 nodes, each containing two quad-core 2.83GHz Xeon processors, 16 GB of memory, and four 7200 rpm SATA 1 TB Seagate Barracuda ES.2 disks. Nodes are interconnected by 10 Gigabit Ethernet. All nodes run the Linux 2.6.28.10 kernel and use the ext3 file system to store HDFS blocks.

While our prototype is not fully functioned, for example some reconstruction cases are still a work-in-progress, it functions enough for preliminary testing. To get a feel for its basic encoding function, we set up a 32 node partition and had each node write a file of 16 GB into a DiskReduce modified HDFS, spread over the same 32 nodes using RAID groups of 8 data blocks. Figure 3 shows the storage capacity recoverd for this 512GB of user data after it has been written three times.

While this experiment is simple, it shows the encoding process recovering $400GB$ and $900GB$ for the RAID 5 and mirror and RAID 6 schemes, respectively, bringing overhead down from 200% to 113% and 25%, respectively.

5. ASYNCHRONOUS ENCODING

The background re-replication process in HDFS and GFS makes it easy to shift in time when data is encoded.

It is generally believed that having multiple copies can improve read performance. In this section we bound the performance degradation that might result from decreasing copies with encoding.

If we reduce the number of data copies in HDFS, there are several reasons that Map/Reduce applications might suffer a performance penalty:

- **Backup tasks:** In Map/Reduce a backup task is the redundant execution of a task if it fails or runs slowly. Backup tasks run in a different node, preferably in a node with a local copy of the data the original node was reading, otherwise more network traffic is needed to support the backup task. More data copies give scheduling more choices for assigning backup tasks on a node with a local copy.
- **Disk bandwidth:** Popular, small datasets may be read by many jobs at the same time, making the total number of disk spindles holding the data a bottleneck. Copies of data may increase the number of spindles with desired data, increasing total bandwidth. GFS may make more than three copies for such "hot spots" [11].
- **Load Balance:** When N blocks are spread at random over M nodes, they will not be perfectly evenly distributed, and the total work to be done by the most loaded node may determine the completion time. With more data copies, the job tracker has more flexibility to assign tasks to nodes with a local data copy, leading to better load balance.

The impact of these three factors is dependent on the encoding, frequency of failures, slow nodes, hot small files, and the ratio of disk to network bandwidth. For this study we are looking to bound the impact of a simple delaying scheme, so we will model the penalty as a factor of r, corresponding to a $100(1-r)\%$ degradation. Our strategy is to exploit locality in the accesses of data, delaying encoding until there is a small impact on overall performance regardless of r.

We obtained a data access trace from Yahoo's M45 cluster, recording the HDFS activity from December 2007 to July 2009. We count all block access requests, and calculate the "block age" when blocks are read. The cumulative distribution function (CDF) of the age of block accesses distribution is shown in Figure 4.

From this figure, one can observe that most data access happens a short time after its creation. For instance, more than 99% of data accesses. happen within the first hour of a data block's life.

If we delay the encoding for t seconds, i.e. the background process will not encode data blocks until they have lived for at least t seconds, we can obtain the full performance of having three copies from a block's creation until it is t seconds old. The expected performance achieved by delaying encoding t seconds can be bounded as:

$$1 \times \Phi(t) + r \times (1 - \Phi(t))$$

Where Φ is the CDF of block access with regard to block age, derived from the trace and shown in Figure 4. Different r gives different expected performance bounds, as shown in Figure 5.

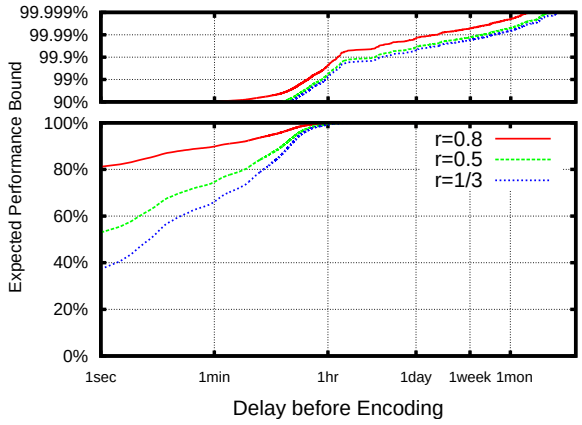


Figure 5: Bound in performance degradation with delayed encoding as function of degradation when insufficient copies are available

As we can see in Figure 5, even if one copy achieves only 1/3 of the performance of three copies, by delaying the encoding for one hour, there is very little system performance penalty.

Delaying encoding delays recovering the disk capacity used by copies. Consider a bound for the disk capacity consumed by delaying by one hour the encoding. Each disk cannot write faster than about 100MB/s, and is unlikely to sustain more than 25MB/s through a file system today, because file systems cannot generally avoid fragmentation altogether. With disk capacities now 1-2TB, a workload of continuous writing at 25 MB/s per disk for one hour would consume $6-12\%$ of total capacity.

Combining these two bounds, regardless of the performance degradation a workload might suffer from not having extra copies, $M45$ usage suggests that at a capacity overhead of less than $6-12\%$, overall performance degradation will be negligible if all encoding is delayed for an hour.

6. CONCLUSION

Data-intensive file systems are part of the core of data-instensive computing paradigms like Map/Reduce. We envision a large increase in the use of large scale parallel programming tools for science analysis applications applied to massive data sets such as astronomy surveys, protein folding, public information data mining, machine translation, etc. But current data-intensive file systems protect data against disk and node failure with high overhead triplication schemes, undesirable when data sets are massive and resources are shared over many users, each with their own massive datasets.

DiskReduce is a modification of the Hadoop distributed file system (HDFS) to asynchronously replace multiple copies of data blocks with RAID 5 and RAID 6 encodings. Because this replacement is asynchronous, it can be delayed wherever spare capacity is available. If encoding is delayed long enough, most read accesses will occur while multiple copies are available, protecting all potential performance benefits achievable with multiple copies. By examining a trace of block creation and use times on the Yahoo! M45 Hadoop cluster, we find that 99% of accesses are made to blocks younger than one hour old, and that far less than 12% of disk capacity is needed to delay encoding for an hour. We conclude that delaying encoding by about one hour is likely to be a good rule of thumb for balancing capacity overhead and performance benefits of multiple copies.

We are completing our implementation of DiskReduce as well as exploring the costs of "cleaning" partially deleted RAID sets, the reliability differences between our different encodings, and explorations of different encoding schemes than were presented in this paper.

7. ACKNOWLEDGMENT

The work in this paper is based on research supported in part by the Department of Energy, under award number DE- FC02-06ER25767, by the National Science Foundation under awards OCI-0852543, CNS-0546551 and SCI-0430781, and by a Google research award. We also thank the member companies of the PDL Consortium (including APC, DataDomain, EMC, Facebook, Google, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Microsoft, NEC, NetApp, Oracle, Seagate, Sun, Symantec, and VMware) for their interest, insights, feedback, and support.

8. REFERENCES

[1] Hadoop. http://hadoop.apache.org/.

[2] Yahoo! reaches for the stars with m45 supercomputing project. http://research.yahoo.com/node/1879.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Trans. Comput.*, 44(2):192–202, 1995.

[4] V. Bohossian, C. C. Fan, P. S. LeMahieu, M. D. Riedel, L. Xu, and J. Bruck. Computing in the rain: A reliable array of independent nodes. *IEEE Trans. Parallel and Distributed Systems*, (2), 2001.

[5] D. Borthakur. The hadoop distributed file system: Architecture and design, 2009. http://hadoop.apache.org/common/docs/current/hdfs-design.html.

[6] D. Borthakur. Hdfs and erasure codes, aug. 2009. http://hadoopblog.blogspot.com/2009/08/hdfs-and-erasure-codes-hdfs-raid.html.

[7] V. Cate and T. Gross. Combining the concepts of compression and caching for a two-level filesystem. In *ACM ASPLOS-IV*, pages 200–211, April 1991.

[8] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. Raid: High-performance, reliable secondary storage. In *ACM Computing Surveys, 1994*.

[9] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *USENIX FAST*, pages 1–14, 2004.

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *USENIX OSDI'04*, Berkeley, CA, USA, 2004.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *ACM SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[12] G. A. Gibson, L. Hellerstein, R. M. Karp, R. H. Katz, and D. A. Patterson. Failure correction techniques for large disk arrays. *ACM ASPLOS*, pages 123–132, 1989.

[13] J. L. Hafner. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *USENIX FAST 2005*.

[14] J. Hartman and J. Ousterhout. The zebra striped network file system. In *Proc. 14th ACM SOSP, 1994*.

[15] D. D. E. Long, B. R. Montague, and L.-F. Cabrera. Swift/raid: A distributed raid system. *ACM Computing Systems*, (3):333–â359, 1994.

[16] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). *ACM SIGMOD Rec.*, 17(3):109–116, 1988.

[17] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *USENIX FAST*, February 2009.

[18] J. S. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A library in c/c++ facilitating erasure coding for storage applications - version 1.2. Technical Report UT-CS-08-627, University of Tennessee Department of Computer Science, August 2008.

[19] D. Presotto. Private communications, 2008.

[20] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. In *Journal of the Society for Industrial and Applied Mathematics, 8:300âĂŞ304, 1960*.

[21] S. Savage and J. Wilkes. Afraid: A frequently redundant array of independent disks. In *Proc. of annual conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 1996.

[22] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS*, Mar. 2002.

[23] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In *USENIX FAST*, 2008.

[24] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The hp autoraid hierarchical storage system. *ACM Trans. Comput. Syst.*, 14(1):108–136, 1996.