DiskReduce: RAIDing the Cloud

Bin Fan, Wittawat Tantisiriroj, Lin Xiao, Garth Gibson (*Carnegie Mellon University*)

Data-Intensive Scalable Computing (DISC) file systems such as HDFS employs replication for reliability, typically delivering users with only about a third of the storage capacity of the raw disks. In this project, we investigate DiskReduce, a framework for integrating RAID into these replicated storage systems to lower storage capacity overhead, for example, from 200% to 25% when triplicated data is dynamically replaced with 8+2 RAID 6 encoding.

We gathered usage data from large HDFS DISC systems and find that DISC files are huge relative to traditional and HPC file systems, but because DISC blocks are also huge, per-file RAID wastes significant capacity. We chose to encode blocks across files. We also studied the implication of reading RAIDed data to MapReduce job performance. We measured read performance benefits from replication that will be lost with erasure encoding. We find that triplicated files can be read at higher bandwidth than single-copy files as expected, but this advantage is perhaps smaller than expected, and is absent in many cases.

1 Integrating RAID into HDFS

To integrate RAID to HDFS, we group k different blocks into one RAID set and calculate m encoding blocks. The system keeps k data blocks and m encoding blocks on different datanodes, and thus ensures that the data is still available from m node failures. When $m = 2$ (i.e. RAID 6 encoding), data can survive any 2 node failures which matches triplication. Though the idea is simple, to use RAIDed HDFS efficiently is in fact complicated and has at least following challenges:

Storage Overhead Since each HDFS block is large (64 MB), if we follow the common rule only to group blocks from the same file into the same RAID set, it may not be trivial to find enough number of blocks for one group. Insufficient blocks in the RAID set leads to higher capacity overhead.

Performance Degradation After data is encoded, the number of copies of data is reduced from three copies to only one. We explore the performance degradation for reading data that might result from having fewer sources in Section 3.

2 RAID Grouping Strategy

With RAID encoding, storage overhead is usually determined by the code itself. However we show that in HDFS the grouping strategy can also significantly affect the storage overhead achieved. In this section, we evaluate two basic grouping strategies to select different blocks into RAID sets.

RAID per file is a strategy that always groups blocks *from the same file* into one RAID set. This strategy is adopted in "HDFS RAID"[1] and PanFS [4] due to simplicity. Per-file RAID ensures that whenever a block is deleted, all the other blocks in this RAID set are also deleted since they are all from the same file. Consequently there is no need to update the check blocks on delete. However, due to the large block size in HDFS, making RAID per file suffers from high capacity overhead for files with a small number of blocks.

RAID across files can potentially achieve lower capacity overhead than per-file RAID, by allowing blocks from different files to be encoded into one RAID set. On the other hand, *small write problem* will be introduced and require extra maintenance work on RAID mutation.

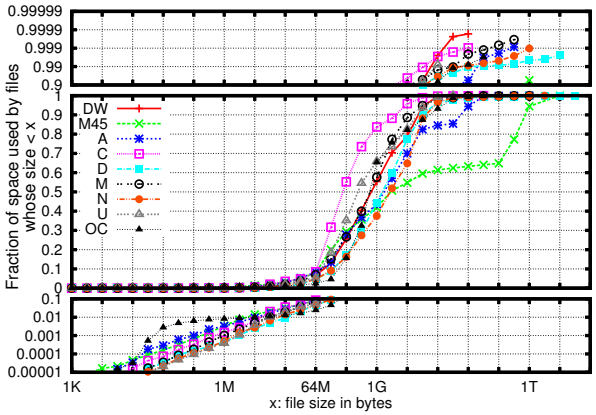


Figure 1: The CDF of total capacity used in the files of given size collected from cloud computing clusters (each line represents each cluster). X axis is in log scale of base 4. The Y axis is split into 3 sections. Section from 0 to 100 percentile is linear; section from 90 to 99.999 percentile and 0.001 to 10 percentile is log scale.

To evaluate the two grouping strategies, we gathered file usage data from HDFS clusters in CMU, Facebook and Yahoo! and show the statistics in Figure 1 where the CDF of the files of given size is plotted. Across all file systems, the largest single file observed is about 5 TB in size. The median size ranges widely from 16 KB to 120 MB and the average ranges from 22 MB to 577 MB. Even though, in all clusters except U more than 80% of files are still smaller than 64 MB (the default block size in HDFS).

As a result, the simple per-file RAID encoding suffers from a large fraction of "small" files observed in real HDFS and the capacity saving will be much smaller than idea RAID encoding.

3 Performance Degradation

In this section we explore the performance degradation that might result from decreasing the number of copies with encoding. It is generally believed that having replications can improve read performance due to (1) When small datasets are read by many different jobs or tasks in a job at the same time, the total number of disk spindles serving the data becomes the key bottleneck. (2) MapReduce exploits *shipping code to data* and thus try to assign computation tasks to storage nodes with data co-located. With more data copies the job tracker has more flexibility of the task assignment, leading to better load balancing.

Experiment Setup We employ a HDFS cluster with one node serving as a master while 60 nodes serve as clients and slaves for MapReduce and HDFS. Each node contains two quad-core 2.83GHz Xeon processors, 16 GB of memory, and four 7200 rpm SATA 1 TB disks. Nodes are interconnected by 10 Gigabit Ethernet.

Benchmarking MapReduce Read For datasets of 10, 300, 1500 and 6000 64MB blocks, we construct different settings and measure the time needed to read all data blocks, one map task per block. We compare three replicas, one replica in which data is uniformly distributed across data nodes and one replica with a skewed distribution. The skewed distribution is obtained by reducing the replication level from three to one.

Figure 2 shows little performance difference between one and three replicas for small datasets (10 or 300 blocks). However, as the datasets become bigger, there is an increasing gap between the speed of reading triplicated and single copy data. With large data, this gap is as large as 50% and 10% in the skewed distribution and the uniformly distribution cases, respectively. The experiment shows (1) that RAID 6 encoded data can be read often as fast as triplicated data, (2) that in some cases triplicated data is notably faster to read especially if data distribution is uncontrolled.

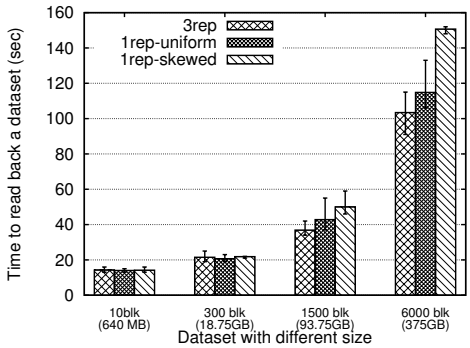
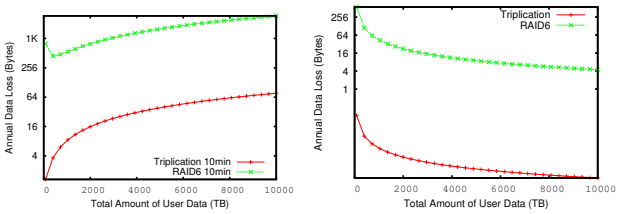


Figure 2: Time to read back a dataset with different settings

4 Reliability

We focus on catastrophic failure of magnetic disks and assume disk failures, disk repairs and data loss events as Poisson events, with exponential inter-arrival times [3, 2]. We employ a Markov model to compare the reliability of triplication and RAID in large storage systems with respect to their annual data loss rate.

We coded the model into Mathematica and run through a few interesting cases. Consider HDFS (64MB blocks) storage systems with 80TB to 10PB user data built with 1 TB disks where each disk is 80% utilized, and assume disks typically sustain 25 MB/s each for repair. Disk failure rates are assumed to be 2% per year. There is a detection time before repair starts. Once the repair starts, the repair rate is determined by the amount of reconstruction work divided by the number of disks, each providing 25 MB/s. RAID 6 groups contain eight data blocks and two check blocks.



(a) 10 minute detection time (b) Zero detection time

Figure 3: Annual data loss rate with different detection time

First, triplication is shown to be more reliable than RAID 6. For example in Figure 3(a), when detection of a failure takes 10 minutes, the expected annual amount of data loss in the largest systems is about an order of magnitude larger with RAID 6. Note however that the absolute number of lost bytes is quite small: less than 4KB per year.

Figure 3(a) also shows, especially for RAID 6, two competing factors: larger systems have more repair bandwidth, decreasing annual data loss, and larger systems have more disks failing, increasing annual loss. A RAID 6 configuration needs fewer disks and more read bandwidth, so it exposes an initially decreasing data loss rate as the system gets larger.

The failure detection time is a key parameter for system developers. If it could be reduced to 0, as shown in Figure 3(b) , larger systems could have better reliability, because repair would be getting faster proportionally to disk failure rates increasing. Unfortunately, instant detection is unrealistic due to network latency and other issues.

5 Conclusion

This paper proposes DiskReduce, a tool to integrate RAID technology into HDFS. We investigate the file statistics and show that even encoding simplicity encourages the contents of each RAID set be taken from the same file, but because DISC file systems have such large block size, this leads to excessive capacity overhead.

Our prototype of DiskReduce has been built for HDFS and released with the Apache project as MAPREDUCE-2036 for all users' benefit.

References

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

[2] 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.

[3] 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.

[4] 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.

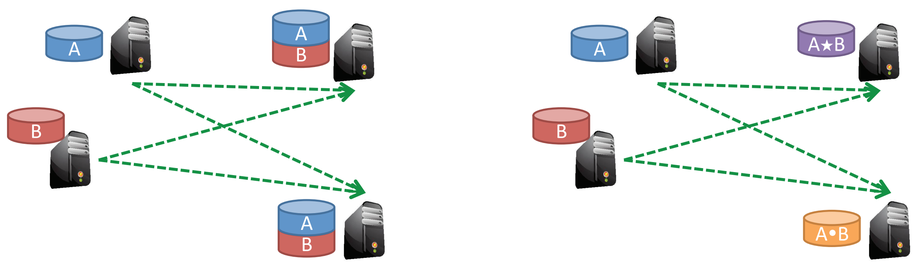
DiskReduce: RAIDing the Cloud

Bin Fan, Wittawat Tantisiriroj, Lin Xiao, Garth Gibson

Overview

Google FS/ HDFS on Data Intensive Scalable Computers

- **Triplication can recover from 2 failures but it trades 200% extra storage for this redundancy**
- **Parity saves storage and tolerates the loss of any two nodes**



GFS / HDFS DiskReduce

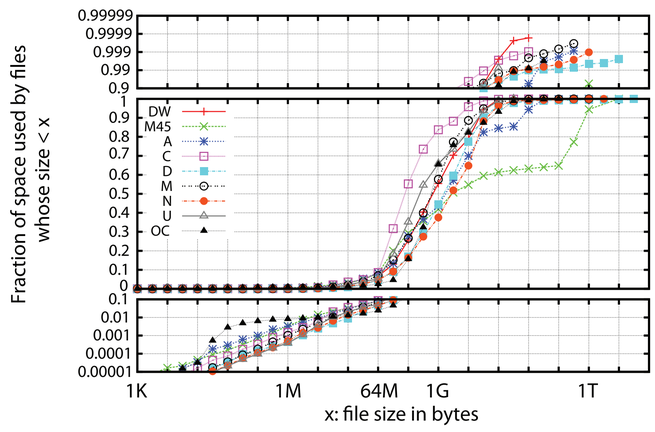
RAID Per-File vs. RAID Across-Files

RAID Per-file: blocks in a RAID set are from the same file

+ Simple

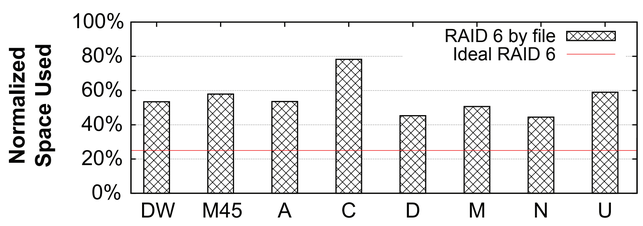
- Too much overhead

Cloud file size distribution



- **Across 9 file systems (1.5 - 21 PB), 30 - 80% of the storage used by files smaller than 1 GB (size of 16 blocks, 64 MB each)**
- **Since each block is large (64 MB by default), small files tend to form short RAID sets**

Space overhead



- **When group size w = 8, per-file RAID 6 requires about 50%overhead while ideal RAID 6 requires only 25% overhead**

RAID Across-files: blocks in a RAID set can be from different files

+ Per-directory RAID 6 can archieve much lower overhead

- Small write problem - potential read-modify-write to update parity blocks on single file deletion

- **To reduce extra work**
  1. group blocks by directory (likely to be deleted together)
  2. defer deletion
  3. after awhile, replace deleted blocks with new blocks

Immediate vs. Background Encoding

Immediate encoding:

+ Efficient

- Complex: Handling failures on critical path

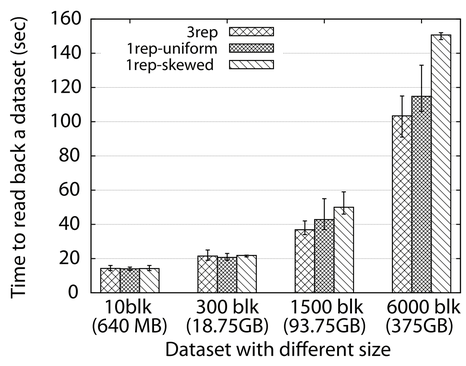
Background encoding:

+ Simple & no change in client code

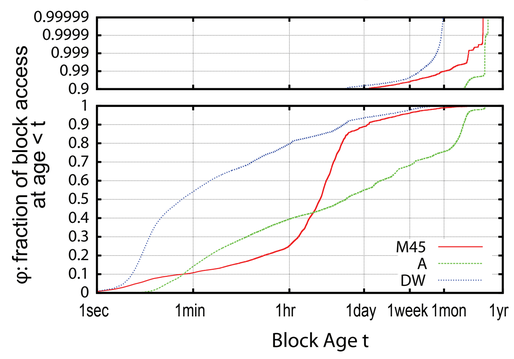
+ Cache young data for higher read bandwidth

- Less efficient

Reading performance



Cloud data access pattern



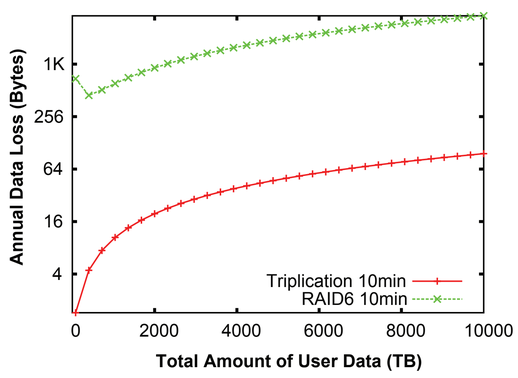
- **Read performance:**
  - **RAID encoded data can be read as fast as if triplicated**
  - **If data access very skewed, more copies helps performance**
- **Treat triplicated data as in cache:**
  - **Locality: over 90% of data blocks accessed within 1st day after creation in M45 & DW and 50% in cluster A**

Reliability Modeling

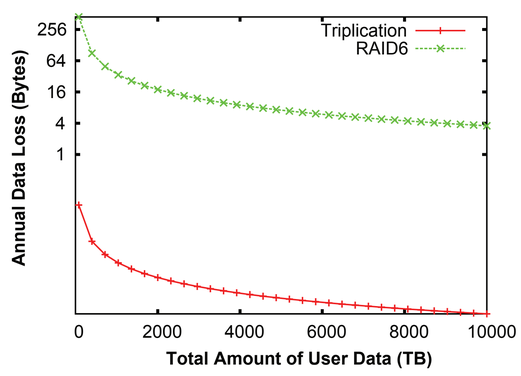
We expect Triplication to be more reliable than RAID6

- **Time to fail/repair model is exponentially distributed, 2% annual failure rate**
- **1TB/disk 80% full, 64MB/chunk, 8+2, 25MB/s/disk repair**
- **Compare bytes lost per year as a function of total sizes**
- **Orders of magnitude difference is still only a few bytes per year**

10 minute failure detection



instant failure detection



Interaction of parallel repair and detection delay

- **With instant detection, scalable repair improves reliability with scale**
- **With more common delayed detection, scalable repair less effective**

Released as Mapreduce-2036 patch for HDFS 0.22.0
@ http://issues.apache.org/jira/browse/MAPREDUCE-2036

Carnegie Mellon YAHOO! pdsi facebook Parallel Data Laboratory PDLR10