RAID6-based Distributed Storage System Implementation

Kai Liu
School of Computer Engineering
Nanyang Technological University
Singapore 639798
kliu006@e.ntu.edu.sg

Bingshui Da
School of Computer Engineering
Nanyang Technological University
Singapore 639798
da0002ui@e.ntu.edu.sg

Jianglei Han
School of Computer Engineering
Nanyang Technological University
Singapore 639798
jhan011@e.ntu.edu.sg

I. Introduction

In distributed systems, data is partitioned and stored in storage devices (nodes) that are separated from each others in location. For example, in a small scale distributed system, a file could be stored in multiple hard disks. In a large system, data segments could be stored in different data centers located in separate geographical locations. When a particular data is being accessed by an user, the system resembles data from partitions and response to the request. From the user's perspective, read and write access to a distributed file should be no different from accessing a local file.

Two of the most important considerations when designing distributed storage system is performance and redundancy. The former refers to I/O speed and latency user may experience. It is subjected to system architecture design and quality of hardware and communication channels. Data redundancy provides backup to data to lower the probability of error during data access due to system faults. Noted that there exists another type of error where data being tempered in storage or transmission. Detecting and correcting such error is another complicated problem in the domain digital communication, where is not in the scope of this paper. Individual disk failure is a norm instead of an exception in distributed systems. Except hardware difficulties, a storage node goes off-line during upgrading and being excluded when the workload is exceptionally high. In either cases, the system should be able to serve the data request without accessing the unavailable nodes.

In this project, we implemented a mini-actual distributed storage system in local operating and file system by implementing low-level coding algorithms. The performance is evaluated and compared across different implementations. The main features of the system include:

1) Storing and access a single file across simulated storage nodes using RAID 6 for fault-tolerance
2) Ability to determine the failure of storage nodes at execution time
3) Ability to rebuild lost redundancy at a replacement storage node
4) Generalized encoding algorithm in extension to the original 6+2 erasure coding

The rest of this paper is organized as follows. We briefly introduce background concepts and technologies that are relevant to the system design and development in Section 2. System architecture and algorithms are covered in Section 3. Experiment setup and results analysis are found in Section 4, followed by conclusion in Section 5.

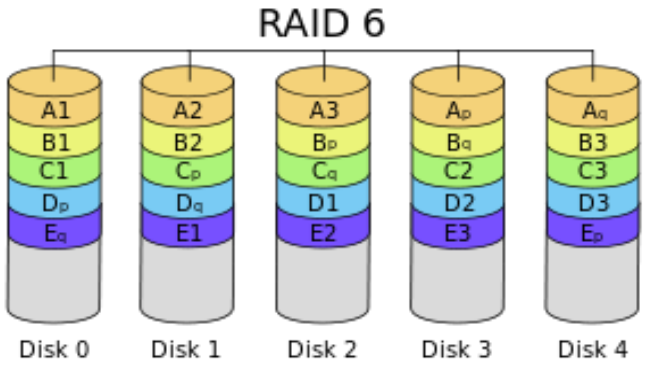

Fig. 1. Diagram of a RAID 6 storage system with 3 block level data strippings and 2 parities [2]

II. Background

A. RAID

Redundant array of independent disks (RAID) is designed as a visualization tool that construct a single storage space with multiple disks. In a RAID system, a file is stripped and stored in different physical disks. A number of variations are evolved to provide different level of data redundancy and I/O performance enhancement [1]. These variations are *levels*, known as 'RAID' followed by a level number. For example, the most common types of RAID are RAID-0, RAID-1 to RAID-6. Different levels of RAID are differ in their strategies in file striping, mirroring and parity computation. We focus on RAID 6 in this paper.

As a significant advantage compared to the lower levels, RAID 6 is resilient to two arbitrarily disk failure. To be more specific, it is capable of carrying on read and write request to any logic disk despite any two of them are not accessible. Figure 1 shows an example of a RAID 6 storage system with data blocks A, B, C, D and E, each block is stripped into three parts (1-3) and two parities p and q.

In Red Hat Linux operating system's kernel implements RAID-6 for fault-tolerance, where the parities are computed as shown in Equation 1:

$$p = d_0 \oplus d_1 \oplus ... \oplus d_{k-1}$$
$$q = d_0 \oplus 2(d_1) \oplus ... \oplus 2^{k-1}(d_{k-1}) \quad (1)$$

The arithmetic operates in Galois field $GF(2^8)$, where addition $\oplus$ is equivalent to XOR and multiplication is more complicated but all results are found in a closed set from 0 to 2^8. An interesting property about parity q is that its computation can be transformed using only addition and multiplication by two, i.e. $q = 2(2(...2(2d_{k-1} \oplus d_{k-2})...) \oplus d_1) \oplus d_0$, benefiting from efficient vector computation tricks in hardware.

B. Erasure Coding

Erasure coding is the technique used to compute the parities in RAID. It encodes data into data plus codings, so the original contents are recoverable when disks with data fail. More formally, for a given k disks space of data, erasure coding creates $k+m$ disks of data, where m disks are codings or parities. In the best case, the contents are recoverable by decoding the erasure code when up to m disks failed. This optimality is called *maximum distance separable* (MDS). In the example from Figure 1, a specific erasure coding algorithm can be used to calculate from three disks of data to produce $3 + 2 = 5$ disks of data and stored in five disks, where parities p and q are codings information that is essential for data recovery. If the encoding can recover data despite of an arbitrary two disks failure, the coding is MDS.

In the example, parities are being stored in all available disks with each stripe rotates the identities of the disks . A more common alternative is to store p segments and q segments in two disks separated from the data parts, this is called a systematic coding.

C. Reed-Solomen Coding

A popular erasure coding scheme is Reed–Solomon coding. It was used for Linux RAID-6, Windows Azure Cloud storage and many other production systems. An *encoding matrix*, or *Generator Matrix* is used to achieve the encoding. Figure 2 shows an encoding matrix used in Linux RAID-6 for (6+2) erasure coding. The encoding is dot product of the encoding matrix and data vector. Noted that the top square matrix is an identity matrix, ensuring the first k blocks are identical to the data blocks, here $k = 6$. The last two rows of the matrix are implementation of Equation 1 to compute parities p and q.

In this paper, we implement a RAID-6 based erasure coding by rewriting Reed–Solomen coding algorithm.

III. Implementations

A. System Architecture

B. Pseudo Code

IV. Experiments

Compared 2+2, 3+2, 4+2, 5+2, 6+2

A. Storage

B. Recoverability

Raid-6 recoverability theory

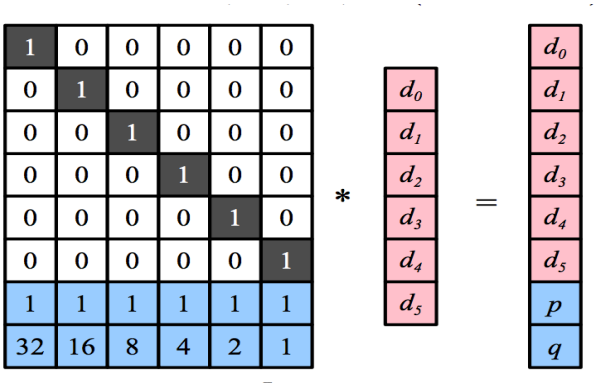

Fig. 2. An encoding matrix used in Linux RAID-6 with 6 data disks and 2 parities (6+2) [3]

C. Speed

V. Conclusion

to conclude

References

[1] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating systems: Three easy pieces*, 2012.

[2] Wikipedia, "Standard raid levels," 2015, [Online; accessed 13-Nov-2015]. [Online]. Available: https://en.wikipedia.org/wiki/Standard_RAID_levels#RAID_6

[3] J. Plank and C. Huang, "Tutorial: Erasure coding for storage applications," in *Slides presented at FAST-2013: 11th Usenix Conference on File and Storage Technologies*, 2013.