44. Data Integrity and Protection

Operating System: Three Easy Pieces

Disk Failure Modes

- Old days: fail-stop, Now: **Latent-sector Errors** and **block Corruption**

- LSEs
  - Arise when Head crash, cosmic rays, material wear out, … etc
  - **Detectable** and correctable (sometimes) via ECC

- Corruption
  - Firmware bugs, faulty bus or connections, …
  - **Not detectable** by the disk itselft
  - Silent faults

Disk Failure Modes (Cont.)

- Common and worthy of failures are **frequency of latent-sector errors(LSEs)** and **block corruption**. (3 years, 1.5 million disks, cheap disks 10x more prone to failures)

	Cheap	**Costly**
LSEs	9.40%	1.40%
Corruption	0.50%	0.05%

Frequency of LSEs and Block Corruption

Disk Failure Modes (Cont.)

- Frequency of latent-sector errors(LSEs)
  - Costly drives with more than one LSE are as likely to develop additional.
  - For most drives, annual error rate increases in year two
  - LSEs increase with disk size
  - Most disks with LSEs have less than 50
  - Disks with LSEs are more likely to develop additional LSEs
  - There exists a significant amount of spatial and temporal locality
  - Disk scrubbing is useful (most LSEs were found this way)

Disk Failure Modes (Cont.)

- Block corruption:
  - Chance of corruption varies greatly across different drive models
  - Within the same drive class
  - Age affects are different across models
  - Workload and disk size have little impact on corruption
  - Most disks with corruption only have a few corruptions
  - Corruption is not independent with a disk or across disks in RAID
  - There exists spatial locality, and some temporal locality
  - There is a weak correlation with LSEs

- A **reliable storage** system, **requires** machinery to detect and recover from LSE and block corruption

Handling Latent Sector Errors

- Latent sector errors are easily detected and handled.
- Using **redundancy mechanisms**:
  - In a mirrored RAID or RAID-4 and RAID-5 system based on parity, the system should reconstruct the block from the other blocks in the parity group.

Detecting Corruption: The Checksum

- How can a client tell that a block has gone bad?
- Using **Checksum mechanisms:**
  - This is simple the result of a function that takes a chunk of data as input and computes a function over said data, producing a small summary of the contents of the data.

Common Checksum Functions (Cont.)

- Different functions are used to compute checksums and vary in strength.
  - One simple checksum function that some use is based on **exclusive or(XOR).**

```
365e c4cd ba14 8a92 ecef 2c3a 40be f666
```

If we view them in binary, we get the following:

```
0011 0110 0101 1110      1100 0100 1100 1101
1011 1010 0001 0100      1000 1010 1001 0010
1110 1100 1110 1111      0010 1100 0011 1010
0100 0000 1011 1110      1111 0110 0110 0110
```

It is easy to see what the resulting checksum will be:

```
0010 0000 0001 1011      1001 0100 0000 0011
```

The result, in hex, is 0x201b9403.

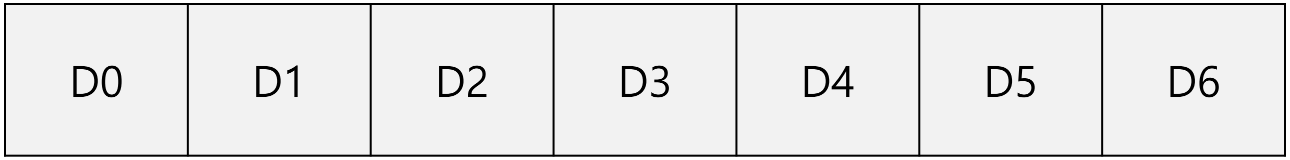
  - XOR is a reasonable checksum but has its limitations.
    - Two bits in the same position within each checksumed unit changed the checksum will not detect the corruption.

Common Checksum Functions (Cont.)

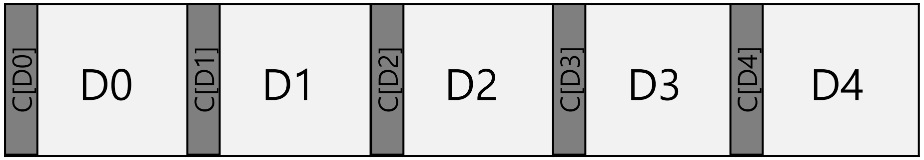
- Addition Checksum
  - This approach has the advantage of being fast.
  - Compute 2's complement addition over each chunk of the data
    - ignoring overflow
- Fletcher Checksum
  - Compute two check bytes, s1 and s2.
    - Assuming a block D consists of bytes d1…dn; s1 is simply in turn is
      - s1 = (s1 + di) mod 255 (compute over all di);
      - s2 = (s2 + s1) mod 255 (again over all di);
- Cyclic redundancy check(CRC)
  - Treating D as if it is a large binary number and divide it by an agreed upon value k.
    - The remainder of this division is the value of the CRC.

Checksum Layout

- The disk layout without checksum:



- The disk layout **with checksum:**



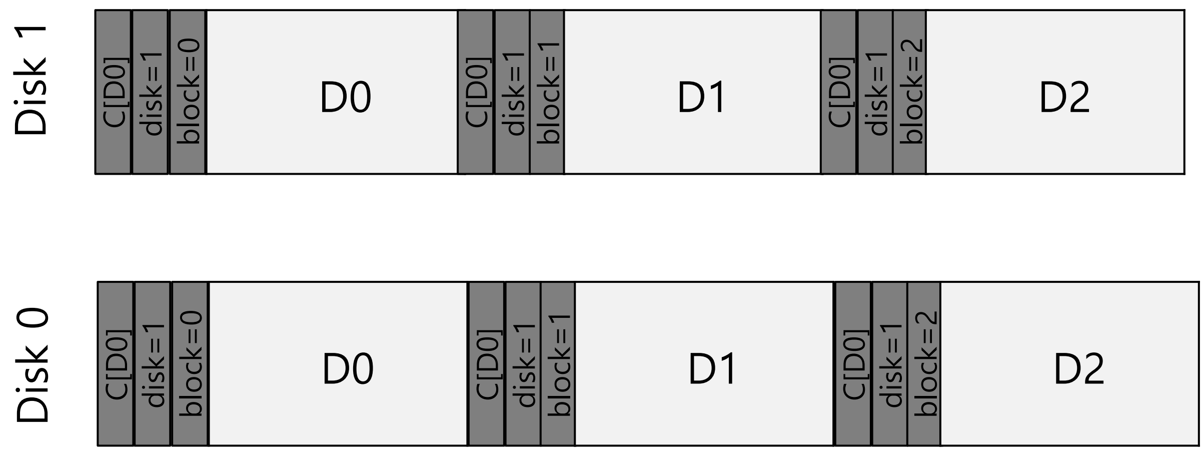
  - Store the checksums packed into 512-byte blocks.

Using Checksums

- When reading a block D, the client reads its checksum from disk Cs(D), **stored checksum**
- Computes the checksum over the retrieved block D, **computed checksum** Cc(D).
- Compares the stored and computed checksums;
  - If they are equal (Cs(D) == Cc(D)), the data is in safe.
  - If they do not match (Cs(D) != Cc(D)), the data has changed since the time it was stored (since the stored checksum reflects the value of the data at that time).

A New Problem: Misdirected Writes

- Modern disks have a couple of unusual failure-modes that require different solutions.
  - Misdirected write arises in disk and RAID controllers which the data to disk correctly, except in the *wrong* location (correct data in the wrong location)



One Last Problem: Lost Writes

- Lost Writes, occurs when the device informs the upper layer that a write has completed but in fact it never is persisted

- Solutions
  1. Write verify or read-after-write
  2. Adds checksums elsewhere. Zettabyte File System (ZFS) includes checksum for each file system inode and indirect block for every block included within a file
     - Every data block write lost will not match with the corresponding inode checksum
     - Still doesn’t not prevent misses in inode an data!

Scrubbing

- When do these checksums actually get checked?
  - Some amount when the files are accessed by the apps
  - Most data is rarely accessed, and thus remain unchecked.

- To remedy this problem, many systems utilize **disk scrubbing**.
  - By **periodically reading** through every block of the system
  - Checking whether checksum are still valid
  - Reduce the chances that all copies of certain data become corrupted

Overhead of Checksumming

- Two distinct kinds of overheads : **space** and **time**
- Space overheads
  - **Disk itself**: A typical ratio might be an 8byte checksum per 4KB data block, for a 0.19% on-disk space overhead.
  - **Memory of the system**: This overhead is short-lived and not much of a concern.
- Time overheads
  - CPU must compute the checksum over each block
    - To reducing CPU overheads is to combine data copying and checksumming into one streamlined activity.

- This lecture slide set was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea at University of Wisconsin.