Data Integrity

CS333
Williams College
This Video

• Media Errors
  • Types of failures
  • Causes/effects

• Techniques for Identifying Errors
  • Error correcting codes
  • Checksums
Revisiting Failure Modes

• In RAID, we assumed errors were fail-stop

• In reality, other types of errors not only exist, but are unfortunately more common than we’d like
Additional Types of HDD Errors

• **Latent-sector errors (LSEs)**
  - A sector or group of sectors becomes damaged
    - For example, if a disk head scrapes against the platter, damaging the surface

• **Sector corruption**
  - A disk block’s contents store incorrect data
    - For example, if the bus that transfers data from the host to the drive has an error: the disk may write what it was told to write but that data is not correct
    - For example, if the firmware has a bug and writes to the wrong sector: the data is valid, but the location is wrong

• LSEs are detectable by the disk
• Sector corruptions go unnoticed
What Should We Do?

Two main challenges:

1. Detection
   • We want to know when a failure happens so that we don’t propagate the problem

2. Correction
   • Ideally we wouldn’t just learn that we have an error, we would fix it
The Anatomy of a Sector

- In addition to the data, an HDD sector actually stores some additional information
  - **Header**: information used by the drive controller (firmware), possibly including:
    - Address (so the R/W head can identify its position)
    - Flags (e.g., to note that the sector is broken)
    - Alternate address (e.g., to use if the sector failed)
  - **Data**:
    - User’s bytes
    - Error correcting codes (ECC)
Note about ECCs

• Error Correcting/Correction Codes contain redundant information that can both identify and fix a subset of possible errors
  • Commonly used in modern HDDs and SSDs
  • Handled entirely within the device’s firmware
    • Low-overhead and transparent to users

• We won’t discuss specific ECCs; we’ll focus on software strategies built on top of disks.
Detecting Errors

- Although ECCs can fix some errors, we still want to detect **LSEs** and **block corruptions**
  - Once we identify an error, we can utilize other techniques, like RAID or replication, to recover
- Challenge: want a **low-overhead** mechanism to detect what our data **should be**, so we can compare it to what our data **actually is**
  - CPU cost
  - Storage cost
  - Memory cost
Detecting Errors

• Checksums are a relatively cheap and effective approach to identify data discrepancies
  • Store output of a deterministic function over our data
    • XOR
    • CRC
    • Fletcher checksum

• If we recompute the checksum function on our data, we should get the same result
  • Store a checksum *somewhere* after each write I/O
  • On each read, ERROR if:
    stored checksum != recomputed checksum
Storing Checksums

- Disk sector is our fixed-size transfer unit
  - If the drive internally uses checksums (as was the case before ECC), can be stored alongside data
  - Otherwise, we need a place that is separate from our sector data. But where?

```
Data (sector-aligned)
```
```
Data + chksums (shifted)
```
```
Disk
```
Storing Checksums

- Idea: store checksums as data in a dedicated checksum block, separate (but near) data
  - But now we need two write I/Os + one read I/O to update a data block
    - Read checksum block, update checksum, write updated checksum block
Detecting Errors

• Can compute & compare checksums *reactively*
  • When we read a block, we recompute its checksum and compare
  • Lazy/on-demand approach

• Can compute & compare checksums *actively*
  • **Disk scrubbing**: intentionally traversing on-disk data and looking for checksum mismatches
    • **Pros**: detect errors sooner, hopefully fix before needed
    • **Cons**: expensive, scales with size of disk
Limitations/Issues

- Block checksums let us answer the question:
  - Does a physical data block match a physical value?
- They do not answer:
  - Was a data block written to the logically correct location? (misdirected write)
  - Was a data block written at all? (lost write)
  - Are my data structures logically consistent?

Checksums (or similar ideas) can be used as tools to answer these questions. We just need careful design: which layer do we perform the checksumming? What objects are we verifying? Do we need additional information beyond a checksum well?
Brief Case Study

- Consider the following (simplified) inode:

  ```c
  struct inode {
    size_t size;
    int blocks;
    lba_t block_ptrs[10];
  }
  ```

  If a block pointer refers to logically incorrect data, (misdirected or lost write) the block checksum might still match.

- Compare it to:

  ```c
  struct inode {
    size_t size;
    int blocks;
    lba_t block_ptrs[10];
    long block_checksums[10];
  }
  ```

  By adding a checksum inside the inode, we can verify that the data refers to the correct logical value, not just that the physical sector’s contents match the most recent write.
Summary

- Checksums give some confidence that a value has not changed or been corrupted
  - Only detect certain classes of changes/errors
  - Only detect *physical* changes/errors
- Checksums can be a useful building block for safeguarding our systems
  - After identify errors, can be fixed by other means
- Checksums can be deployed at different layers with different advantages/costs
  - Disk, block layer, FS layer, application layer, …