RAID

ECE595
April 3

Y. Charlie Hu

[week12] Observations

- Getting first byte from disk read is slow
  - Access time = seek time + rotational delay + read time
- Peak disk bandwidth good, but rarely achieved
- Towards mitigate disk performance impact
  - Inside FS:
    - Move some disk data into main memory – file caching
  - Between FS and Disks:
    - Schedule requests to shorten seeks (disk scheduling)

Disk Scheduling

- Problem statement:
  - Given the mapping of 1-D array of logical blocks to the sectors of disk, and disk requests keep arriving, schedules disk requests currently in the queue to maximize the disk I/O throughput
  - *Simplification:* the scheduler knows above which cylinder the disk head is, but not which sector

→ Minimize seek time only (ignore rotation delay)
→ FIFO, SSTF, SCAN, C-SCAN

Practice questions (from textbook)

- Does SSTF scheduling tend to favor middle cylinders or the innermost/outmost cylinders?
- Why is rotational delay often not considered in disk scheduling?
  - Most disks do not expose this info
  - Even if they did, the info is changing – moving target

Practice questions

- Requests are not usually uniformly distributed. For example, a cylinder containing the file system FAT or inodes can be expected to be accessed more frequently than a cylinder that contains only files.

  Suppose you know that 50 percent of the requests are for a small, fixed number of cylinders, would any of the scheduling algorithms we discussed be particularly good for this case? Explain your answer.

[week12] Observations

- Getting first byte from disk read is slow
  - high ***latency***
- Peak disk bandwidth good, but rarely achieved
- Towards mitigate disk performance impact
  - Inside FS:
    - Move some disk data into main memory – file caching
  - Between FS and Disks:
    - Schedule requests to shorten seeks
  - *Inside disks?*

A startup company idea?

- *Random block access time = seek time + rotational delay + reading time*
- If you have double degrees in EE and ME, can you think of a revolutionary idea?
- Do you think you will get investors excited?

[week12] Observations

- Getting first byte from disk read is slow
  - high ***latency***
- Peak disk bandwidth good, but rarely achieved
- Towards mitigate disk performance impact
  - Inside FS:
    - Move some disk data into main memory – file caching
  - Between FS and Disks:
    - Schedule requests to shorten seeks
  - *Inside disks?*
    - *Adding multiple disk arms to the disk?*

Exploiting parallelism?

- Observation: disk performance matters more during heavy load
- Goal: improve performance under heavy load
- Suitability
  - do we have parallelism semantically?
- Feasibility
  - how to realize parallelism?

RAID

- Two motivations
  - (in the past) Operating in parallel can increase disk throughput
    - RAID = Redundant Array of Inexpensive Disks
  - (today) Redundancy can increase reliability
    - RAID = Redundant Array of Independent Disks

RAID -- Two main ideas (1)

- Parallel reading (for performance)
  - Splitting bits of a byte across multiple disks
  - 8 disks (bit-level striping)
    - Logically acts like single disk with sector size X 8 and access time / 8
    - Reduce the response time of large access (e.g. 1 4K disk block)
  - Alternatively, block-level striping
    - Increases the throughput of multiple small accesses (e.g. 8 512-byte disk blocks)

RAID -- Two main ideas (2)

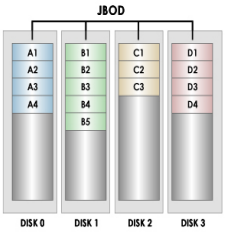
- Mirroring or shadowing (for reliability)
  - Local disk consists of 2 physical disks in parallel
  - Every write performed on both disks
    - Can read from either disk
    - Probability of both fail at the same time?

RAID – combing the two ideas!

- Mirroring gives reliability, but expensive
- Striping gives high data-transfer rate, but not reliability
- Challenge: can you provide redundancy at low cost?

JBOD (non-RAID architecture)

- Definition: concatenation of disks
  - E.g. combine 3GB, 15GB, 5.5GB drives into 23.5GB logical drive
  - Opposite of disk partitioning
  - Also uses an array of independent disks
- No performance advantage
- Failure of one disk?
- Use:
  - convenience

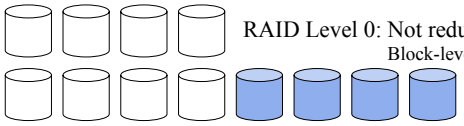


Synopsis of RAID Levels

RAID Level 0: Not redundant
Block-level striping

- Performance?
  - Read/write?
- If one disk fails?
  - FS MTTF?
- Use
  - Large read-only NFS servers (quick mounting)
  - Create a small number of large logical disks
    - Microsoft windows: # of drive letters <= 24

Synopsis of RAID Levels

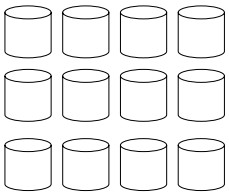


RAID Level 0: Not redundant
Block-level striping

RAID Level 1:
Mirroring

- Performance? (Independent controllers)
  - Read?
  - Seek time?
- Reliability (failure probability) ?
- Interesting Use
  - Online snapshot (split the mirror and backup)

Synopsis of RAID Levels



RAID Level 0: Not redundant
Block-level striping

RAID Level 1:
Mirroring

RAID Level 2:
Bit-interleaved, ECC

ECC can detect/correct which bit went wrong

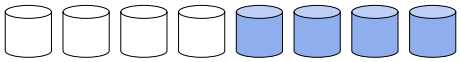
Low-level format of modern drives already contains ECC → controller can catch errors within a sector

Level 2 is never used.

Instead,

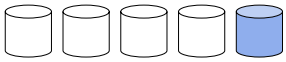
Synopsis of RAID Levels

RAID Level 0: Not redundant



RAID Level 1:
Mirroring

RAID Level 2:
Bit-interleaved, ECC

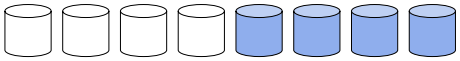


RAID Level 3:
Byte-interleaved, parity

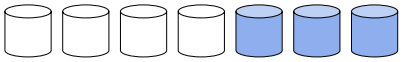
Compared to RAID Level 1:
High transfer rate for each access
A req involves all drives + seek/rotational delay →
cannot service multiple requests simultaneously
Computing/writing parity expensive →
Dedicated parity hardware (offload CPU)
Rarely used

Synopsis of RAID Levels

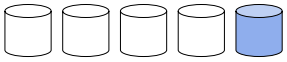
RAID Level 0: Not redundant
Block-level striping



RAID Level 1:
Mirroring



RAID Level 2:
Bit-interleaved, ECC



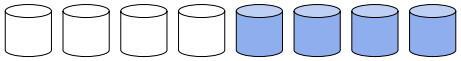
RAID Level 3:
Byte-interleaved, parity

RAID Level 4:
Block-interleaved, parity

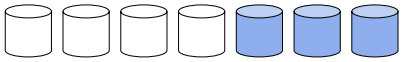
Lower transfer rate for each block (by single disk)
Higher overall rate (many small files, or a large file)
Large writes → parity bits can be written in parallel
Small writes → 2 reads / 2 writes !

Synopsis of RAID Levels

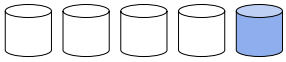
RAID Level 0: Not redundant
Block-level striping



RAID Level 1:
Mirroring



RAID Level 2:
Bit-interleaved, ECC



RAID Level 3:
Bit-interleaved, parity

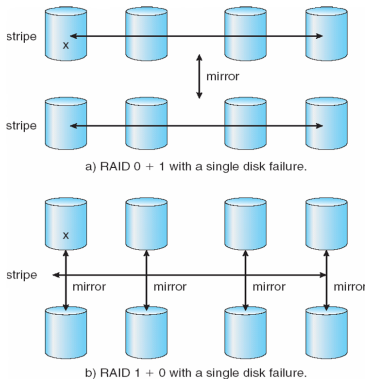
RAID Level 4:
Block-interleaved, parity

RAID Level 5:
Block-interleaved, distributed parity

Later additions

- RAID 6: RAID 5 + additional parity block
- Nested RAID levels
  - RAID 01 (0+1): mirrored RAID 0 (stripe)
  - RAID 10 (1+0): stripe of RAID 1 (mirror)
  - RAID 50 (5+0): stripe of RAID 5
  - RAID 100 (10+0): stripe of RAID 10
- Proprietary RAID levels
  - Double parity
  - RAID 1.5
  - RAID 7
  - …

RAID (0 + 1) and (1 + 0)



RAID Implementation

- Typically in hardware
  - Special-purpose RAID controller (PCI card)
  - Manages disks
  - Performs parity calculation
- Can be in software (by OS)
  - Can be fast
  - At the cost of CPU time

Practice question

- Consider a RAID Level 5 organization comprising five disks, with the parity for sets of four blocks on four disks stored on the fifth disk. How many blocks are accessed in order to perform the following?
  - A write of one block of data
  - A write of seven continuous blocks of data

Practice question

- True-or-False:
  It is faster to write 1 block in a RAID Level 5 organization (with 5 disks) than in a RAID Level 1 organization (with 2 disks)

Practice question

- Assume that
  - you have a mixed configuration comprising disks organized as RAID Level 1 and as RAID Level 5 disks;
  - the system has flexibility in deciding which disk organization to use for storing a particular file.
  - you have a mixed workload of *frequently-read* and *frequently-written* files
- Which files should be stored in the RAID Level 1 disks and which in the RAID Level 5 disks in order to optimize performance?

Reading

- Chapter 12

RAID

- Main idea
  - Store the error correcting codes on other disks
  - General error correcting codes are too powerful
  - Use XORs for single parity
  - Upon any failure, one can recover the entire block from the spare disk (or any disk) using XORs
- Pros
  - Reliability
  - High bandwidth
- Cons
  - The controller is complex