Persistence: RAID

CS 537: Introduction to Operating Systems

Louis Oliphant & Tej Chajed

University of Wisconsin - Madison

Spring 2024

Administrivia

- Project 5 due today
- Project 6 out today, due Apr 16th @ 11:59pm
- Exam 2 grades later this week
- Project 4 grades later this week

Review: I/O devices

- Protocol: polling vs interrupts
- Communication: I/O instructions vs memory-mapped I/O

Review: Hard disk seek, rotation, transfer

- Seek: 4–10ms, avg seek is 1/3 of max
- Rotation: typical speeds are 5400 RPM, 7200 RPM, avg is 1/2
- Transfer: depends on RPM and sector density, 100MB/s typical

Review: Scheduling

Given stream of I/O requests for different sectors, what order to process them?

Different from CPU scheduling: can predict time based on sector position

Makes a big performance difference

Disks summary

Disks: seek between tracks, rotate within a track

I/O time: rotation + seek + transfer

Sequential vs random throughput

Scheduling: SSTF, SCAN, C-SCAN
Benefits of violating work conservation

Quiz 15: Disks Transfer Rates

https://tinyurl.com/cs537-sp24-q15

Agenda for today

- What is RAID?
- Understand Levels 0 (striping), 1 (mirroring), 4 (parity), and 5 (rotating parity)
- Measuring Capacity, Performance, and Reliability compared to a single disk

Redundant Arrays of Inexpensive Disks

- Externally, a **RAID** looks like a disk (it is transparent to the OS)
- Internally, there are lots of configurations (this lecture: level 0, 1, 4, 5)
- RAID aims to:
  - Be larger than a single disk (**Capacity**)
  - Work faster (**Performance**)
    - IO is often a bottleneck to performance
    - Highly dependent on workload type (**random** and **sequential**)
  - Provide **Reliability**
    - Functioning with failure of one or more disks

RAID Level 0 - Striping

- No redundancy, blocks are striped across the array of disks
- Blocks in the same row are called a stripe.
- Chunk size can vary between RAID arrays (1 block, 2 block, etc.)
  - Small chunk size means files will be striped across many disks, increasing parallelism
  - Reduces intra-file parallelism, relies on multiple concurrent requests

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Disk 0	Disk 1	Disk 2	Disk 3
0	2	4	6
1	3	5	7
8	10	12	14
9	11	13	15

RAID 0 Analysis

RAID 0 of N disks

- Capacity: perfect – the same as N individual disks
- Reliability: perfectly horrible – any disk failure and data is lost
- Performance:
  - Single Read latency
  - Steady-state bandwidth
    - Sequential
    - Random
- Assume single disk performance:
  - Holds B blocks
  - S MB/s for sequential workload
  - R MB/s for random workload

RAID 0 Analysis (Performance)

Can use all disks at once (Maximize Parallelism):

- single read latency – nearly identical to that of a single disk
- Sequential Rate – $N \cdot S$ MB/s
- Random Rate – $N \cdot R$ MB/s

RAID Level 1 - Mirroring

- Make more than one copy of each block in the system; each copy should be placed on a separate disk
- When reading a block there is a choice (can read from either)
- When writing, need to write both copies (can be done in parallel)

RAID 1 + 0
Mirrored pairs and then stripes

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

RAID 0 + 1
Stripes and then mirrors

Disk 0	Disk 1	Disk 2	Disk 3
0	1	0	1
2	3	2	3
4	5	4	5
6	7	6	7

RAID 1 Analysis

- Capacity: $(N \cdot B)/2$ blocks
- Reliability: Tolerate 1 failure (if lucky up to N/2 failures)
- Performance:
  - Latency: Same as a single disk
  - Sequential Write: 2 physical writes for each logical write
    $(N/2) \cdot S$ MB/s
  - Sequential Read: Each disk skips every other block
    $(N/2) \cdot S$ MB/s
  - Random Read: $N \cdot R$ MB/s (can parallelize requests)
  - Random Write: $\frac{N}{2} \cdot R$ MB/s

RAID Level 4 - Saving Space with Parity

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

Use **XOR Parity**, xor-ing the blocks

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1)=0
0	1	0	0	XOR(0,1,0,0)=1

RAID 4 Analysis

- Capacity: $(N-1) \cdot B$ blocks
- Reliability: Tolerate 1 disk failure
- Performance:
  - Latency: same as single disk for read, twice as long for write (why?)
  - Sequential Read: $(N-1) \cdot S$ MB/s
  - Sequential Write: $(N-1) \cdot S$ MB/s
    - Utilize **full-stripe** write
  - Random Read: $(N-1) \cdot R$ MB/s
  - **Random Write:** $(R/2)$ **MB/s**
    - **Parity Disk is a bottleneck**
    - **subtractive parity**: $P_{new} = (C_{old} \oplus C_{new}) \oplus P_{old}$

RAID 5 - Rotating Parity

- Rotate the parity block across drives
- Now the parity disk is not the bottleneck
  - Performance on Random Writes goes to $\frac{N}{4} \cdot R$

Disk 0	Disk 1	Disk 2	Disk 3	Disk 3
0	1	2	3	**P0**
4	5	6	**P1**	7
8	9	**P2**	10	11
12	**P3**	13	14	15
P4	16	17	18	19

Comparing RAID Levels

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$	$(N-1) \cdot B$
Reliability	0	1 (maybe more)	1	1
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} \cdot R$
Latency Read	T	T	T	T
Latency Write	T	T	2T	2T

Consistent updates

In RAID 1–5, what happens if there is a failure while updating the mirror or parity?

RAID hardware can buffer writes in non-volatile storage to solve this

Implementing RAID

- Hardware RAID: sophisticated device with controller (CPU), memory, non-volatile storage, and several disks
- Software RAID (but how to do consistent updates?)

Takeaways and Beyond RAID

- RAID is useful for large, cheap storage (e.g., raw video)
- RAID and SSDs are both transparent block devices with sophisticated internals
- Striping is common (e.g., SSDs, DRAM)
- Cloud storage uses replication and error correction

Summary

RAID is a transparent technique for improving capacity and reliability of drives.

- RAID-0: striping
- RAID-1: mirroring
- RAID-4, RAID-5: striping + parity

File Systems

Disks alone would be hard to use

A **file system** is an abstraction for persistent storage

Main concepts: files and directories

Why care about the file system?

Common to many, many systems: Window, macOS, Linux, Android, iOS

Essentially all storage goes through a file system

You will likely use this API

What's cool about file systems?

User management: you can interact with file system directly

Allocation: file system helps you dynamically allocate storage without thinking about it too much

Implementation: you'll be able to understand how the API is implemented