Redundancy, Fault Tolerance & RAID

Rishabh Iyer

12-11-2020

Faults, Errors and Failures

- Fault
  - Underlying defect
  - May or may not cause problems
- Error
  - Active fault
- Failure
  - Error at an interface between modules

Standard Jargon (Recap)

- MTTF
- MTTR
- MTBF
- Availability
- Bathtub curve

Reacting to errors within a module

- Do nothing
- Fail-fast
- Fail-safe
- Fail soft

RAID

- Motivation:
  - Fault Tolerance
  - Optionally throughput
- Idea:
  - Use cheap commodity disks
  - Split the array in **reliability groups**
  - Use extra **check disks**

Why do we need redundancy?

- A single disk has an acceptable MTTF

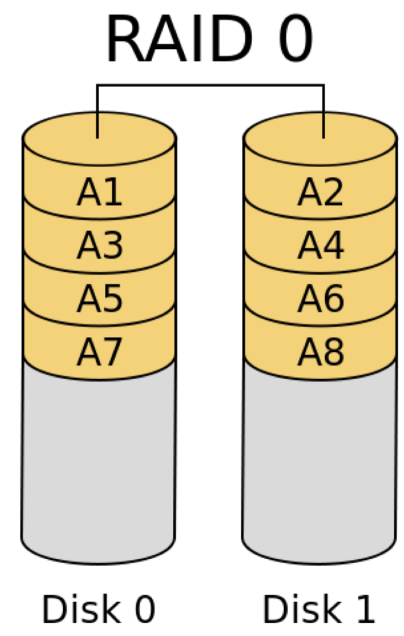
$$MTTF\ of\ a\ Disk\ Array = \frac{MTTF\ of\ a\ Single\ Disk}{Number\ of\ Disks\ in\ the\ Array}$$

MTTF of RAID?

- $MTTF_{Group} = \frac{MTTF_{Disk}}{G+C} * \frac{1}{Probability\ of\ another\ failure\ in\ the\ group\ before\ repairing\ the\ first\ one}$
- $P(another\ failure) = \frac{MTTR}{MTTF_{Disk}/_{(G+C-1)}}$
- $MTTF_{RAID} = \frac{MTTF_{Group}}{n_G}$

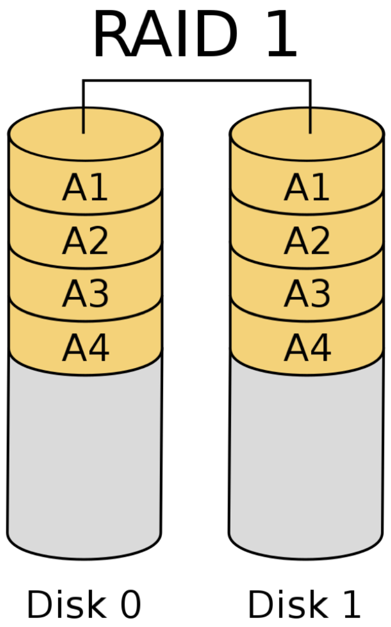
RAID 0

- More read/write throughput
- No redundancy
- No fault tolerance

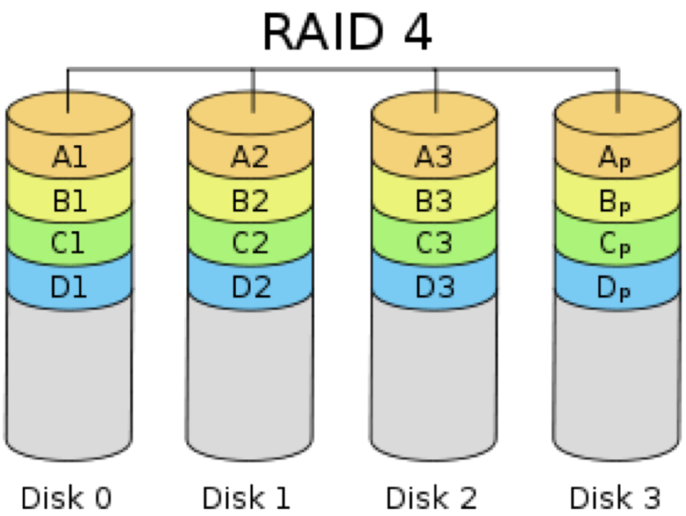


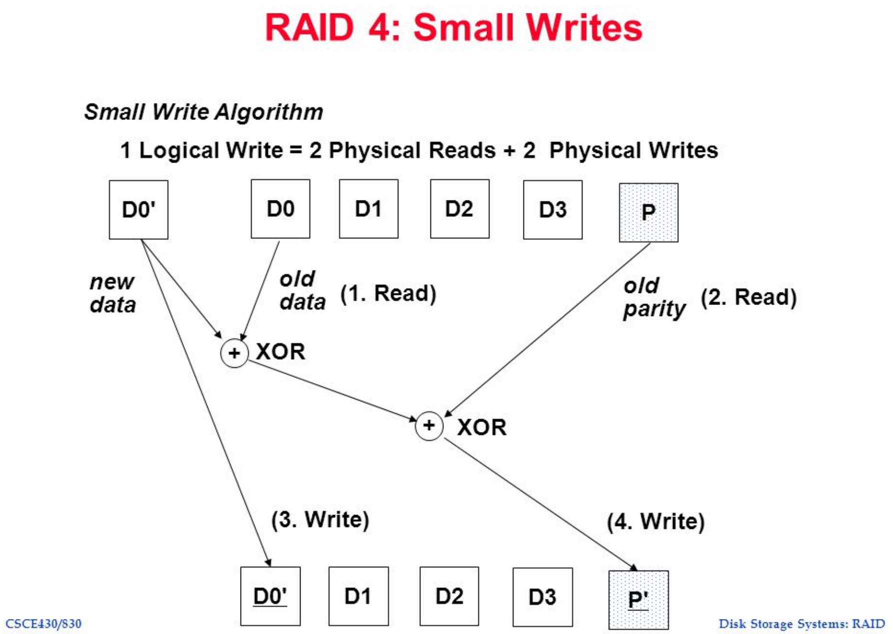
RAID 1

- Increased read throughput
- Almost the same write throughput
- Half of available storage is used



RAID 4

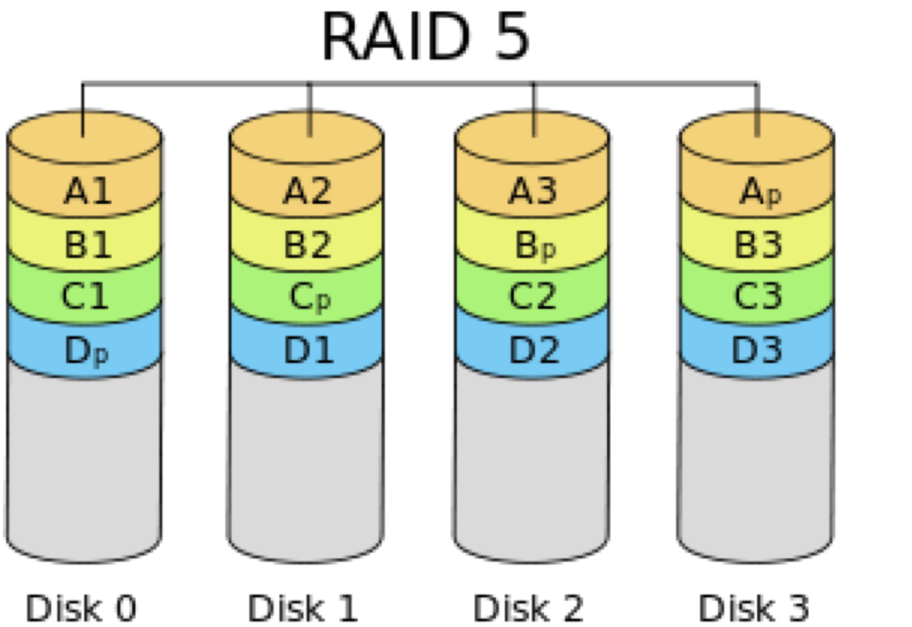




- Parallel Reads
- Check disk is a write bottleneck

RAID 5

- Solves the RAID 4 write bottleneck
- Widely used solution
- Let's see that in action



RAID 5 Example

	DISK0	DISK1	DISK2	DISK3
STRIPE0	0100	0101	0010	?
STRIPE1	0010	0000	?	0100
STRIPE2	0011	?	1010	1000
STRIPE3	?	0001	1101	1010

- Complete the parity entries

RAID 5 Example

	DISK0	DISK1	DISK2	DISK3
STRIPE0	0100	0101	0010	0011
STRIPE1	0010	0000	0110	0100
STRIPE2	0011	0001	1010	1000
STRIPE3	0110	0001	1101	1010

- STRIPE0,DISK3 = 0100 XOR 0101 XOR 0010 = 0011
- STRIPE1,DISK2 = 0010 XOR 0000 XOR 0100 = 0110
- STRIPE2,DISK1 = 0011 XOR 1010 XOR 1000 = 0001
- STRIPE3,DISK0 = 0001 XOR 1101 XOR 1010 = 0110

RAID 5 Example - Writes

	DISK0	DISK1	DISK2	DISK3
STRIPE0	0100	0101	0010	0011
STRIPE1	0010	0000	0110	0100
STRIPE2	0011	0001	1010	1000
STRIPE3	0110	0001	1101	1010

Modifying STRIPE0 in DISK2 to 1101. Outline the steps.

How many reads and writes will you need?

RAID 5 Example - Writes

	DISK0	DISK1	DISK2	DISK3
STRIPE0	0100	0101	~~0010~~ 1101	~~0011~~ 1100
STRIPE1	0010	0000	0110	0100
STRIPE2	0011	0001	1010	1000
STRIPE3	0110	0001	1101	1010

STRIPE0, DISK3 = 0010 XOR 1101 XOR 0011 = 1100

RAID 5 Example – Disk Failure

	DISK0	DISK1	DISK2	DISK3
STRIPE0	0100	0101	110[illegible]	1100
STRIPE1	0010	0000	0110	0100
STRIPE2	0011	0001	101[illegible]	1000
STRIPE3	0110	0001	1[illegible]1	1010

Disk 2 died. How can the RAID controller serve a read requests for STRIPE0 for DISK2?

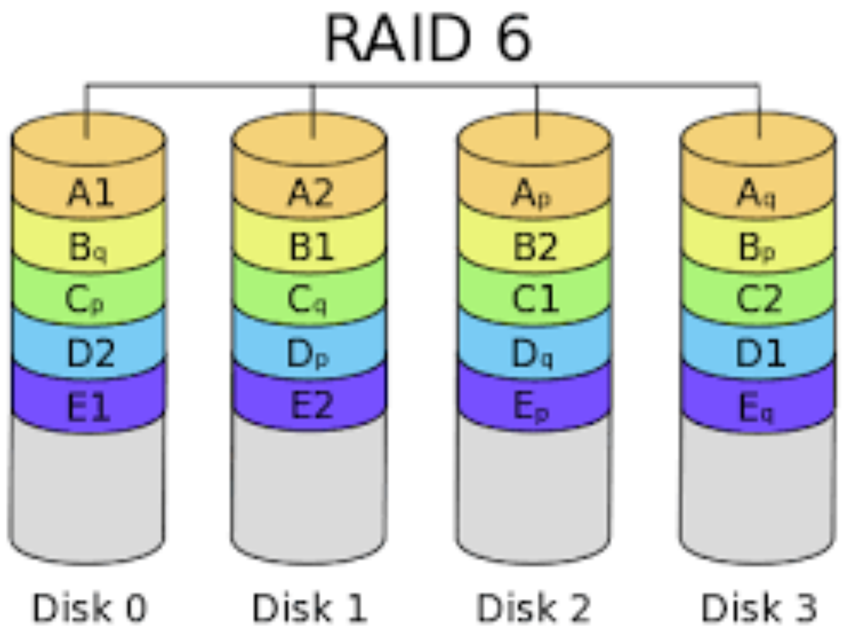
RAID 5 Example – Disk Failure

	DISK0	DISK1	DISK2	DISK3
STRIPE0	0100	0101	110[illegible]	1100
STRIPE1	0010	0000	0110	0100
STRIPE2	0011	0001	101[illegible]	1000
STRIPE3	0110	0001	1[illegible]1	1010

STRIPE0,DISK2 = 0100 XOR 0101 XOR 1100 = 1101

RAID 6

- Double Parity
- Can tolerate 2 failures in a group



Changing times

- Is RAID still the go-to for fault tolerant storage?
  - Scale
  - Evolving speeds/bottlenecks

RAID for SSDs?

- SSDs have higher MTTF
  - Check Flash Translation Layer
- No performance argument (for RAID 1)
  - High throughput SSDs
  - Internal parallelism
- TRIM command for garbage collection

RAID vs E2E Argument

- The E2E argument argues against providing reliability in lower layers of the stack because ultimately it is the endpoints that know what they want. RAID provides a lot of reliability at the hardware device level. Is it a violation of the E2E argument? Why are both still considered good design guidelines/ well designed systems?

RAID vs E2E Argument

- RAID is a layer above the hardware device, software RAID implementations exist
- Sometimes errors are better masked at a lower layer, because it understands the error better
- All applications have the same interface to disk, so which layer you put the reliability in does not matter.

Summary

- Redundancy is the de-facto method used to achieve fault tolerance
- RAID:
  - ❖ Required increased throughput from an array of disks
  - ❖ Used redundancy to ensure improved reliability
  - ❖ Decreasing performance, storage overheads with increasing RAID levels

The RAMCloud Storage System

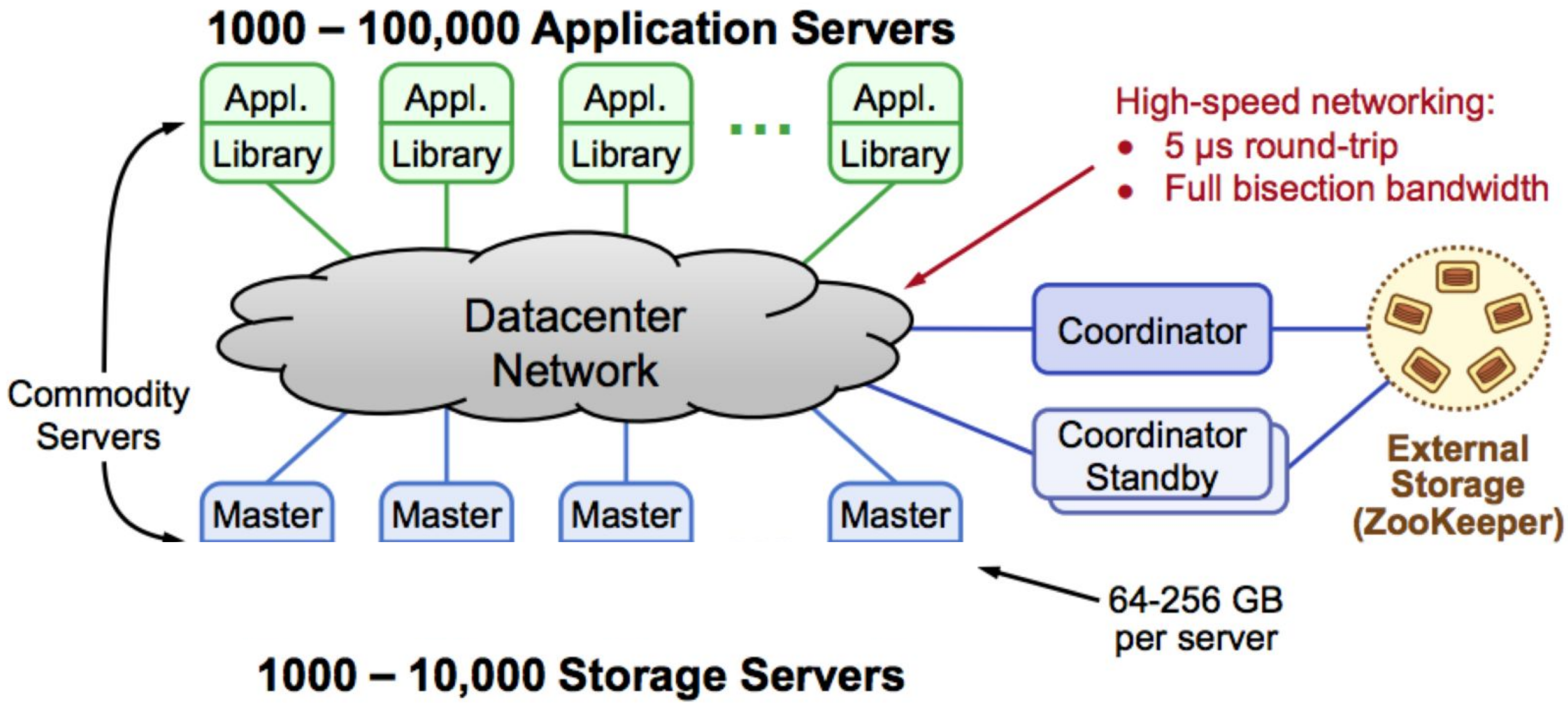
Lei Yan
(Slides partially adopted from Marios Kogias)

RAMCloud

Serving large data (100s of TB) with low latency (5 - 10us)

- Store data in DRAM
- Distributed

RAMCloud Architecture



Durability and High Availability

- Durability

  No loss of committed updates

  At least as durable as disk-based systems (assuming every server's data is replicated in the disks of another two servers)

- High Availability

  < 5 sec down time

Failures

- Server Crashes
- Data center power Failures
- Malicious acts, e.g., hackers deleting data

Design Exercise

Goals:

- **Durability:** Probability of losing data at a given time point should be as low as disk-based systems
- **High availability:** < 5 sec down time
- **Low latency:** 5-10us
- **High throughput:** Highest possible while achieving the above goals

Assumptions:

- 100+ Servers
- 100MB/s disk bandwidth
- 1GB/s network bandwidth per server
- 50GB DRAM per server
- Probability of server crash = Probability of disk failure
- Probability that a disk and one replica fail < Probability of power failure

Durability

First try: Replicate data in DRAM

- Does it work?

Second try: Replicate data in Disk

- Does it work?

Fast Recovery

First try: Replicate entire data on one primary backup, restore entire data on a single new server

- What is the bottleneck?

  Disk accesses

- Recovery time?

  100MB/s for 50GB -> 500 sec

Fast Recovery

Second try: Partition replicas among 100 primary backup servers, restore data on a single new server

- What is the bottleneck?

  Network bandwidth

- Recovery time?

  1GB/s for 50GB -> 50 sec

Fast Recovery

Last try: Partition replicas among 100 primary backup servers, restore data on the backups

- Recovery time?

  100 * 100MB/s for 50GB -> 5 sec

High Throughput

First try: Store replicas on disk as hash table

- Random write access to disk, low write throughput

Second try: Store replicas as logs on disk, every update to the data is appended to the log

- Sequential write access to disk

Low Latency

When should storage servers reply to the client that the write succeeds?

How to handle other failures within the storage servers, e.g., memory corruption, after we detecting them?

How to handle these failures with the techniques available in the system we just designed after we detecting them?

- Hint: Recall George’s one-hammer-for-all approach of handling failures with rebooting

- Failure promotion: promote the failures to a crash, and reusing techniques used for handling crashes.

- Reduce system complexity: BL’s “make it simple”

- Assumption: The failures happens rare, so the cost of crash recovery is amortized.

Full RAMCloud paper:

https://dl.acm.org/doi/pdf/10.1145/2806887

RAMCloud full paper is a good read, gives a lot of details of the system, covers many aspects of system design, scalability, fault tolerance and so on... A lot of analysis of various trade-offs and how they affect the design of the system. Definitely have a look if you are interested in systems