Errors and RAID
February 15, 2022
Types of Errors

- **Hard errors**: The component is dead.

- **Soft errors**: A signal or bit is wrong, but it doesn’t mean the component must be faulty.

- **Note**: You can have recurring soft errors due to faulty, but not dead, hardware.
Examples

• DRAM errors

  • Hard errors: Often caused by motherboard - faulty traces, bad solder, etc.

  • Soft errors: Often caused by cosmic radiation or alpha particles (from the chip material itself) hitting memory cell, changing value. (Remember that DRAM is just little capacitors to store charge... if you hit it with radiation, you can add charge to it.)
Measuring Availability

- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- \( \text{MTBF} = \text{MTTF} + \text{MTTR} \) (mean time between failure)

Availability = \( \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \)

- = (time sys operational) / (time sys should have been operational)
- Suppose OS crashes once per month, takes 10min to reboot.
  - MTTF = 720 hours = 43,200 minutes
  - MTTR = 10 minutes
- Availability = \( \frac{43200}{43210} = 0.997 \) (~“3 nines”)

\( \text{Availability} = \frac{\text{time system was running}}{\text{time system should have been running}} \)

Down time = \( 1 - \text{Availability} \)
## Availability

<table>
<thead>
<tr>
<th>Availability %</th>
<th>Downtime per year</th>
<th>Downtime per month*</th>
<th>Downtime per week</th>
</tr>
</thead>
<tbody>
<tr>
<td>90% (&quot;one nine&quot;)</td>
<td>36.5 days</td>
<td>72 hours</td>
<td>16.8 hours</td>
</tr>
<tr>
<td>95%</td>
<td>18.25 days</td>
<td>36 hours</td>
<td>8.4 hours</td>
</tr>
<tr>
<td>97%</td>
<td>10.96 days</td>
<td>21.6 hours</td>
<td>5.04 hours</td>
</tr>
<tr>
<td>98%</td>
<td>7.30 days</td>
<td>14.4 hours</td>
<td>3.36 hours</td>
</tr>
<tr>
<td>99% (&quot;two nines&quot;)</td>
<td>3.65 days</td>
<td>7.20 hours</td>
<td>1.68 hours</td>
</tr>
<tr>
<td>99.50%</td>
<td>1.83 days</td>
<td>3.60 hours</td>
<td>50.4 minutes</td>
</tr>
<tr>
<td>99.80%</td>
<td>17.52 hours</td>
<td>86.23 minutes</td>
<td>20.16 minutes</td>
</tr>
<tr>
<td>99.9% (&quot;three nines&quot;)</td>
<td>8.76 hours</td>
<td>43.8 minutes</td>
<td>10.1 minutes</td>
</tr>
<tr>
<td>99.95%</td>
<td>4.38 hours</td>
<td>21.56 minutes</td>
<td>5.04 minutes</td>
</tr>
<tr>
<td>99.99% (&quot;four nines&quot;)</td>
<td>52.56 minutes</td>
<td>4.32 minutes</td>
<td>1.01 minutes</td>
</tr>
<tr>
<td>99.999% (&quot;five nines&quot;)</td>
<td>5.26 minutes</td>
<td>25.9 seconds</td>
<td>6.05 seconds</td>
</tr>
<tr>
<td>99.9999% (&quot;six nines&quot;)</td>
<td>31.5 seconds</td>
<td>2.59 seconds</td>
<td>0.605 seconds</td>
</tr>
<tr>
<td>99.99999% (&quot;seven nines&quot;)</td>
<td>3.15 seconds</td>
<td>0.259 seconds</td>
<td>0.0605 seconds</td>
</tr>
</tbody>
</table>

For a reliable component, may have to wait a long time to determine its availability/downtime!
Availability in practice

• Carrier airlines (2002 FAA fact book)
  • 41 accidents, 6.7M departures
  • 99.9993% availability

• 911 Phone service (1993 NRIC report)
  • 29 minutes per line per year
  • 99.994%

• Standard phone service (various sources)
  • 53+ minutes per line per year
  • 99.99+%

• End-to-end Internet Availability
  • 95% - 99.6%
Coping with failures...

- A failure
  - Let’s say one bit in your DRAM fails.

- Propagates
  - Assume it flips a bit in a memory address the kernel is writing to. That causes a big memory error elsewhere, or a kernel panic.
  - Your program is running one of a dozen storage servers for your distributed filesystem.
  - A client can’t read from NFS, so it hangs.
Recovery Techniques

• We’ve already seen some: e.g., retransmissions in TCP and in your RPC system

• **Modularity** can help in failure isolation: preventing an error in one component from spreading.
  • Analogy: The **firewall** in your car keeps an engine fire from affecting passengers

• **Redundancy** and **Retries**
  • Network: retransmit an RPC
  • Specific techniques used in file systems, disks (RAID)
What are our options?

1. Silently return the wrong answer.
2. Detect failure.
3. Correct / mask the failure
Options in dealing with failure

1. Silently return the wrong answer.

2. Detect failure.

3. Correct / mask the failure
Block error detection/correction

- **EDC** = Error Detection and Correction bits (redundancy)
- **D** = Data protected by error checking, may include header fields
- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection and correction

Diagram:

```
+----------------+      +----------------+
| datagram       |      | datagram       |
|                |      |                |
| ← d data bits  |      | Y               |
|                |      | all bits in D'?|
| D | EDC      |      | OK             |
|                |      | ?               |
|                |      | N detected error|
|                |      | bit-error prone link |
|                |      | D' | EDC'            |
```

Parity Checking

Single Bit Parity:
Detect single bit errors

Calculated using XOR over data bits:
- 0 bit: even number of 0s
- 1 bit: odd number of 0s
Error Detection - Checksum

- Used by TCP, UDP, IPv4 (not IPv6)
- *Ones complement sum* of all 16-bits in packet
- Simple to implement
  - Break up packet into 16-bits strings
  - Sum all the 16-bit strings
  - Take complement of sum = checksum; add to header
  - On receiver, compute same sum, add sum and checksum, check that the result is 0 (no error)
- Relatively weak detection
  - Easily tricked by typical loss patterns (bursty errors)
Example: Internet Checksum

- Goal: detect “errors” (e.g., flipped bits) in transmitted segment

Sender
- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1’s complement sum) of segment contents
- Sender puts checksum value into checksum field in header

Receiver
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless?
Options in dealing with failure

1. Silently return the wrong answer.

2. Detect failure.

3. Correct / mask the failure
Error Recovery

- Two forms of error recovery
  - Redundancy
    - Error Correcting Codes (ECC)
    - Replication/Voting
  - Retry

- ECC
  - Keep encoded redundant data to help repair losses
  - Forward Error Correction (FEC) – send bits in advance
    - Reduces latency of recovery at the cost of bandwidth
Error Recovery – Error Correcting Codes (ECC)

Two Dimensional Bit Parity:
Detect and correct single bit errors

\[
\begin{array}{cccccccc}
\text{column parity} & \text{row parity} \\
\hline
\text{d}_{1,1} & \ldots & \text{d}_{1,j} & \text{d}_{1,j+1} \\
\text{d}_{2,1} & \ldots & \text{d}_{2,j} & \text{d}_{2,j+1} \\
\ldots & \ldots & \ldots & \ldots \\
\text{d}_{i,1} & \ldots & \text{d}_{i,j} & \text{d}_{i,j+1} \\
\text{d}_{i+1,1} & \ldots & \text{d}_{i+1,j} & \text{d}_{i+1,j+1} \\
\hline
\end{array}
\]

Examples:

- **No errors**
  
  \[
  \begin{array}{c}
  101011 \\
  111110 \\
  011101 \\
  001010 \\
  \hline
  \end{array}
  \]

- **Correctable single bit error**
  
  \[
  \begin{array}{c}
  101011 \\
  101100 \\
  101101 \\
  001010 \\
  \hline
  \end{array}
  \]

- **Correctable single bit error**
  
  \[
  \begin{array}{c}
  101011 \\
  101100 \\
  011101 \\
  001010 \\
  \hline
  \end{array}
  \]
Replication/Voting

• If you take this to the extreme, three software versions: [r1] [r2] [r3]

• Send requests to all three versions of the software: Triple modular redundancy
  • Compare the answers, take the majority
  • Assumes no error detection

• In practice - used mostly in space applications; some extreme high availability apps (stocks & banking? maybe. But usually there are cheaper alternatives if you don’t need real-time)
Retry – Network Example

- Sometimes errors are transient / need to mask
- Need to have error detection mechanism
  - E.g., timeout, parity, checksum
  - No need for majority vote
One key question

• How correlated are failures?
• Can you assume independence?
  • If the failure probability of a computer in a rack is \( p \),
  • What is \( \Pr(\text{computer 2 failing} \mid \text{computer 1 failed}) \)?
    • Maybe it’s \( p \)... or maybe they’re both plugged into the same UPS...

• Why is this important?
On disk errors
What are our options?

1. Silently return the wrong answer.
2. Detect failure.
   • Every sector has a header with a checksum. Every read fetches both, computes the checksum on the data, and compares it to the version in the header. Returns error if mismatch.
3. Correct / mask the failure
   • Re-read if the firmware signals error (may help if transient error, may not)
   • Use an error correcting code (what kinds of errors do they help?)
     • Bit flips? Yes. Block damaged? No
   • Have the data stored in multiple places (RAID)
Motivation: Why use multiple disks?

- **Capacity**
  - More disks allows us to store more data
- **Performance**
  - Access multiple disks in parallel
  - Each disk can be working on independent read or write
  - Overlap seek and rotational positioning time for all
- **Reliability**
  - Recover from disk (or single sector) failures
  - Will need to store multiple copies of data to recover

**So, what is the simplest arrangement?**
Just a bunch of disks (JBOD)

- Yes, it’s a goofy name
  - industry really does sell “JBOD enclosures”
Disk Striping

- Interleave data across multiple disks
  - Large file streaming can enjoy parallel transfers
  - High throughput requests can enjoy good load balancing
    - If blocks of hot files equally likely on all disks (really?)

File Foo:

stripe unit or block

Stripe
Disk striping details

• How disk striping works
  • Break up total space into fixed-size stripe units
  • Distribute the stripe units among disks in round-robin
  • Compute location of block #B as follows
    • disk# = B\%N (\% = modulo, N = # of disks)
Now, What If A Disk Fails?

- In a JBOD (independent disk) system
  - one or more file systems lost
- In a striped system
  - a part of each file system lost

- Backups can help, but
  - backing up takes time and effort
  - backup doesn’t help recover data lost during that day
    - Any data loss is a big deal to a bank or stock exchange
Redundancy via replicas

- Two (or more) copies
  - mirroring, shadowing, duplexing, etc.
- Write both, read either

![Diagram showing redundancy via replicas](image)
Lower Cost Data Redundancy

- Single failure protecting codes
  - general single-error-correcting code is overkill
  - General code finds error and fixes it
- Disk failures are self-identifying (a.k.a. erasures)
  - Don’t have to find the error
- Parity is single-disk-failure-correcting code
  - recall that parity is computed via XOR
  - it’s like the low bit of the sum
Simplest approach: Parity Disk

- One extra disk
- All writes update parity disk
  - Potential bottleneck
  - (different data in different As, Bs, Cs, Ds)
  - (Ap contains parity for all As)
Updating and using the parity

**Fault-Free Read**

- D
- D
- D
- P

**Fault-Free Write**

1. D → D → D → P

**Degraded Read**

- X
- D
- D
- P

**Degraded Write**

- X
- D
- D
- P
The parity disk bottleneck

- Reads go only to the data disks
  - But, hopefully load balanced across the disks

- All writes go to the parity disk
  - And, worse, usually result in Read-Modify-Write sequence
  - So, parity disk can easily be a bottleneck
Solution: Striping the Parity

- Removes parity disk bottleneck
RAID Taxonomy

- Redundant Array of Inexpensive Independent Disks
  - Constructed by UC-Berkeley researchers in late 80s (Garth)
- RAID 0 – Coarse-grained Striping with no redundancy
- RAID 1 – Mirroring of independent disks
- RAID 2 – Fine-grained data striping plus Hamming code disks
  - Uses Hamming codes to detect and correct multiple errors
  - Originally implemented when drives didn’t always detect errors
  - Not used in real systems
- RAID 3 – Fine-grained data striping plus parity disk
- RAID 4 – Coarse-grained data striping plus parity disk
- RAID 5 – Coarse-grained data striping plus striped parity
- RAID 6 – Coarse-grained data striping plus 2 striped codes
RAID-0: Striping

- Stripe blocks across disks in a “chunk” size
  - How to pick a reasonable chunk size?

How to calculate where chunk # lives?

Disk #:
Offset within disk:
RAID-0: Striping

- Evaluate for D disks
- Performance: How much faster than 1 disk? (best case)
- Reliability: More or less reliable than 1 disk?
RAID-1: Mirroring

- Motivation: Handle disk failures
- Put copy (mirror or replica) of each chunk on another disk

- Capacity
- Reliability
- Performance
RAID-4: Parity

- **Motivation:** *Improve capacity*
- **Idea:** Allocate parity block to encode info about blocks
  - Parity checks all other blocks in stripe across other disks
- **Parity block = XOR over others (gives “even” parity)**
  - Example: $0 \ 1 \ 0 \ \rightarrow \ Parity \ value$?
- **How do you recover from a failed disk?**
  - Example: $x \ 0 \ 0 \ and \ parity \ of \ 1$
  - What is the failed value?

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
RAID-4: Parity

- Capacity:
- Reliability:
- Performance:
  - Reads
  - Writes: How to update parity block?
    - Two ways:
      - Use parity disk
      - Re-compute parity from non-parity disks
    - (Parity disk is the bottleneck)
Updating and using the parity

**Fault-Free Read**

**Fault-Free Write**

**Degraded Read**

**Degraded Write**
RAID-5: Rotated/Striped Parity

Rotate location of parity across all disks

- Capacity:
- Reliability:
- Performance:
  - Reads:
  - Writes:
  - Still requires 4 I/Os per write, but not always to same parity disk
Comparison

<table>
<thead>
<tr>
<th>RAID Level</th>
<th>RAID-0</th>
<th>RAID-1</th>
<th>RAID-4</th>
<th>RAID-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>$N$</td>
<td>$N/2$</td>
<td>$N - 1$</td>
<td>$N - 1$</td>
</tr>
<tr>
<td>Reliability</td>
<td>0</td>
<td>1 (for sure)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Read</td>
<td>$N \cdot S$</td>
<td>$(N/2) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>Sequential Write</td>
<td>$N \cdot S$</td>
<td>$(N/2) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>Random Read</td>
<td>$N \cdot R$</td>
<td>$N \cdot R$</td>
<td>$(N - 1) \cdot R$</td>
<td>$N \cdot R$</td>
</tr>
<tr>
<td>Random Write</td>
<td>$N \cdot R$</td>
<td>$(N/2) \cdot R$</td>
<td>$\frac{1}{2} \cdot R$</td>
<td>$\frac{N}{4} \cdot R$</td>
</tr>
<tr>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>$D$</td>
<td>$D$</td>
<td>$D$</td>
<td>$D$</td>
</tr>
<tr>
<td>Write</td>
<td>$D$</td>
<td>$D$</td>
<td>$2D$</td>
<td>$2D$</td>
</tr>
</tbody>
</table>

Table 38.7: RAID Capacity, Reliability, and Performance

N: number of disks
S: throughput of 1 disk sequential read/write
R: throughput of 1 disk random read/write
D: delay to read/write from 1 disk
### Comparison

N: number of disks  
S: throughput of 1 disk sequential read/write  
R: throughput of 1 disk random read/write  
D: delay to read/write from 1 disk

<table>
<thead>
<tr>
<th></th>
<th>RAID-0</th>
<th>RAID-1</th>
<th>RAID-4</th>
<th>RAID-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>N</td>
<td>N/2</td>
<td>N−1</td>
<td>N−1</td>
</tr>
<tr>
<td>Reliability</td>
<td>0</td>
<td>1 (for sure)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>N/2</td>
<td>(if lucky)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Read</td>
<td>N · S</td>
<td>(N/2) · S</td>
<td>(N−1) · S</td>
<td>(N−1) · S</td>
</tr>
<tr>
<td>Sequential Write</td>
<td>N · S</td>
<td>(N/2) · S</td>
<td>(N−1) · S</td>
<td>(N−1) · S</td>
</tr>
<tr>
<td>Random Read</td>
<td>N · R</td>
<td>N · R</td>
<td>(N−1) · R</td>
<td>N · R</td>
</tr>
<tr>
<td>Random Write</td>
<td>N · R</td>
<td>(N/2) · R</td>
<td>1/2 · R</td>
<td>N/4 R</td>
</tr>
<tr>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Write</td>
<td>D</td>
<td>D</td>
<td>2D</td>
<td>2D</td>
</tr>
</tbody>
</table>

Table 38.7: RAID Capacity, Reliability, and Performance
Comparison

<table>
<thead>
<tr>
<th></th>
<th>RAID-0</th>
<th>RAID-1</th>
<th>RAID-4</th>
<th>RAID-5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capacity</strong></td>
<td>$N$</td>
<td>$N/2$</td>
<td>$N - 1$</td>
<td>$N - 1$</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>0</td>
<td>1 (for sure)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{N}{2}$ (if lucky)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Read</td>
<td>$N \cdot S$</td>
<td>$(N/2) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>Sequential Write</td>
<td>$N \cdot S$</td>
<td>$(N/2) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>Random Read</td>
<td>$N \cdot R$</td>
<td>$N \cdot R$</td>
<td>$(N - 1) \cdot R$</td>
<td>$N \cdot R$</td>
</tr>
<tr>
<td>Random Write</td>
<td>$N \cdot R$</td>
<td>$(N/2) \cdot R$</td>
<td>$\frac{1}{2} \cdot R$</td>
<td>$\frac{N}{4} R$</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>$D$</td>
<td>$D$</td>
<td>$D$</td>
<td>$D$</td>
</tr>
<tr>
<td>Write</td>
<td>$D$</td>
<td>$D$</td>
<td>$2D$</td>
<td>$2D$</td>
</tr>
</tbody>
</table>

Table 38.7: RAID Capacity, Reliability, and Performance

N: number of disks
S: throughput of 1 disk sequential read/write
R: throughput of 1 disk random read/write
D: delay to read/write from 1 disk

Because RAID-5 is basically identical to RAID-4 except in the cases where it is better, it has almost completely replaced RAID-4 in the marketplace. The only place where it has not is in systems that know they will never perform anything other than a large write, thus avoiding the small-write problem altogether [HLM94]; in those cases, RAID-4 is sometimes used as it is slightly simpler to build.

38.8 RAID Comparison: A Summary

We now summarize our simplified comparison of RAID levels in Table 38.7. Note that we have omitted a number of details to simplify our analysis. For example, when writing in a mirrored system, the average seek time is a little higher than when writing to just a single disk, because the seek time is the max of two seeks (one on each disk). Thus, random write performance to two disks will generally be a little less than random write performance of a single disk. Also, when updating the parity disk in RAID-4/5, the first read of the old parity will likely cause a full seek and rotation, but the second write of the parity will only result in rotation.

However, our comparison does capture the essential differences, and is useful for understanding tradeoffs across RAID levels. We represent the summary in the table below; for the latency analysis, we simply use $D$ to represent the time that a request to a single disk would take.

To conclude, if you strictly want performance and do not care about reliability, striping is obviously best. If, however, you want random I/O performance and reliability, mirroring is the best; the cost of updating the parity disk in RAID-4/5 is lost capacity. If capacity and reliability are your main goals, then RAID-5 is the winner; the cost you pay is in small-write performance. Finally, if you are always doing sequential I/O and want to maximize capacity, RAID-5 also makes the most sense.
Comparison

<table>
<thead>
<tr>
<th>RAID-0</th>
<th>RAID-1</th>
<th>RAID-4</th>
<th>RAID-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>$N$</td>
<td>$N/2$</td>
<td>$N - 1$</td>
</tr>
<tr>
<td>Reliability</td>
<td>0</td>
<td>1 (for sure)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$N$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N$</td>
<td></td>
<td>$N$</td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential Read</td>
<td>$N \cdot S$</td>
<td>$(N/2) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>Sequential Write</td>
<td>$N \cdot S$</td>
<td>$(N/2) \cdot S$</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>Random Read</td>
<td>$N \cdot R$</td>
<td>$N \cdot R$</td>
<td>$(N - 1) \cdot R$</td>
</tr>
<tr>
<td>Random Write</td>
<td>$N \cdot R$</td>
<td>$(N/2) \cdot R$</td>
<td>$1/2 \cdot R$</td>
</tr>
<tr>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>$D$</td>
<td>$D$</td>
<td>$D$</td>
</tr>
<tr>
<td>Write</td>
<td>$D$</td>
<td>$D$</td>
<td>$2D$</td>
</tr>
</tbody>
</table>

*Table 38.7: RAID Capacity, Reliability, and Performance*

N: number of disks
S: throughput of 1 disk sequential read/write
R: throughput of 1 disk random read/write
D: delay to read/write from 1 disk
Comparison

<table>
<thead>
<tr>
<th>RAID Type</th>
<th>Capacity</th>
<th>Reliability</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAID-0</td>
<td>$N$</td>
<td>0</td>
<td>$N \cdot S$</td>
</tr>
<tr>
<td>RAID-1</td>
<td>$N/2$</td>
<td>1 (for sure)</td>
<td>$(N/2) \cdot S$</td>
</tr>
<tr>
<td>RAID-4</td>
<td>$N - 1$</td>
<td>1</td>
<td>$(N - 1) \cdot S$</td>
</tr>
<tr>
<td>RAID-5</td>
<td>$N - 1$</td>
<td>1</td>
<td>$(N - 1) \cdot S$</td>
</tr>
</tbody>
</table>

- **N**: number of disks
- **S**: throughput of 1 disk sequential read/write
- **R**: throughput of 1 disk random read/write
- **D**: delay to read/write from 1 disk

**Table 38.7: RAID Capacity, Reliability, and Performance**
Advanced Issues

• What happens if more than one fault?
  • Example: One disk fails plus “latent sector error” on another
  • RAID-5 cannot handle two faults
  • Solution: RAID-6: add multiple parity blocks

• Why is NVRAM useful?
  • Example: What if update 2, don’t update P0 before power failure (or crash), and then disk 1 fails?
  • NVRAM solution: Use to store blocks updated in same stripe
    • If power failure, can replay all writes in NVRAM
  • Software RAID solution: Perform parity scrub over entire disk

![Diagram showing RAID arrays and parity blocks]
Conclusions

- RAID turns multiple disks into a larger, faster, more reliable disk
- RAID-0: Striping
  Good when performance and capacity really matter, but reliability doesn’t
- RAID-1: Mirroring
  Good when reliability and write performance matter, but capacity (cost) doesn’t
- RAID-4: Parity disk
- RAID-5: Rotating parity
  Good when capacity and cost matter or workload is read-mostly
  • Good compromise choice
Summary

- Definition of MTTF/MTBF/MTTR: Understanding availability in systems.
- Failure detection and fault masking techniques
- Engineering tradeoff: Cost of failures vs. cost of failure masking.
  - At what level of system to mask failures?
- Replication as a general strategy for fault tolerance: RAID today, next: replicated services
- Thought to leave you with:
  - What if you have to survive the failure of entire machine? Of a rack of machines? Of a datacenter?