15-440 Distributed Systems

12 – Errors, Fault Tolerance and RAID
Outline

- Errors/error recovery
- Using multiple disks
  - Why have multiple disks?
  - problem and approaches
- RAID levels and performance
- Estimating availability
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.)
Some fun #s

- Both Microsoft and Google have recently started to identify DRAM errors as an increasing contributor to failures... Google in their datacenters, Microsoft on your desktops.

- We’ve known hard drives fail
  - Especially when students need to hand in HW/projects :)

_E.g._, See “DRAM Errors in the Wild: A Large-Scale Field Study”
# Replacement Rates

<table>
<thead>
<tr>
<th>Component</th>
<th>HPC1 %</th>
<th>COM1 Component</th>
<th>COM1 %</th>
<th>COM2 Component</th>
<th>COM2 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard drive</td>
<td>30.6</td>
<td>Power supply</td>
<td>34.8</td>
<td>Hard drive</td>
<td>49.1</td>
</tr>
<tr>
<td>Memory</td>
<td>28.5</td>
<td>Memory</td>
<td>20.1</td>
<td>Motherboard</td>
<td>23.4</td>
</tr>
<tr>
<td>Misc/Unk</td>
<td>14.4</td>
<td>Hard drive</td>
<td>18.1</td>
<td>Power supply</td>
<td>10.1</td>
</tr>
<tr>
<td>CPU</td>
<td>12.4</td>
<td>Case</td>
<td>11.4</td>
<td>RAID card</td>
<td>4.1</td>
</tr>
<tr>
<td>motherboard</td>
<td>4.9</td>
<td>Fan</td>
<td>8</td>
<td>Memory</td>
<td>3.4</td>
</tr>
<tr>
<td>Controller</td>
<td>2.9</td>
<td>CPU</td>
<td>2</td>
<td>SCSI cable</td>
<td>2.2</td>
</tr>
<tr>
<td>QSW</td>
<td>1.7</td>
<td>SCSI Board</td>
<td>0.6</td>
<td>Fan</td>
<td>2.2</td>
</tr>
<tr>
<td>Power supply</td>
<td>1.6</td>
<td>NIC Card</td>
<td>1.2</td>
<td>CPU</td>
<td>2.2</td>
</tr>
<tr>
<td>MLB</td>
<td>1</td>
<td>LV Pwr Board</td>
<td>0.6</td>
<td>CD-ROM</td>
<td>0.6</td>
</tr>
<tr>
<td>SCSI BP</td>
<td>0.3</td>
<td>CPU heatsink</td>
<td>0.6</td>
<td>Raid Controller</td>
<td>0.6</td>
</tr>
</tbody>
</table>

From “Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?”
Measuring Availability

- Fraction of time that server is able to handle requests
- Computed from MTBF and MTTR (Mean Time To Repair)

\[
\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}
\]

Available during these 3 periods of time.
Measuring Availability

- Mean time to failure (MTTF) - “uptime”
- Mean time to repair (MTTR)
- Mean time between failures (MTBF)
- $MTBF = MTTF + MTTR$

- Availability = $MTTF / (MTTF + MTTR)$
  - Suppose OS crashes once per month, takes 10min to reboot.
  - MTTF = 720 hours = 43,200 minutes
  - MTTR = 10 minutes
  - Availability = $43200 / 43210 = 0.997$ (~“3 nines”)
## 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>
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%
Real Devices

PRODUCT OVERVIEW
Cheetah 15K.4
Mainstream enterprise disc drive

Simply the best price/performance, lowest cost of ownership disc drive ever

KEY FEATURES AND BENEFITS
- The Cheetah™ 15K.4 is the highest-performance drive ever offered by Seagate®, delivering maximum IOPS with lower drive by yield lower TCO.
- The Cheetah 15K.4 price/performance value is tied with the breakthrough benefits of some enterprise SDDs, SASS, and SAS 3.5-inch drive for rock solid enterprise storage.
- Revisive, self-initiated, read/write management features improve media integrity, increase drive efficiency, reduce incidence of integrations failures and improve field warranty.
- The Cheetah 15K.4 shares its technology architecture and firmware base with Cheetah 10K.7 and Savvio™ to ensure better factory consistency and reduced time to market.

KEY SPECIFICATIONS
- 144, 72, and 36 Gbps speeds
- 3.5 max average read and 3.3 max average write seek times
- Up to 22 Mbyte/sec sustained transfer rate
- 1.4 million hour full load safe (MTBF
- Serial Attached SCSI (SAS), Ultra320 SCSI and 2 Gbit/sec Fibre Channel interfaces
- 5 years warranty

For more information on why 15K is the industry's best price/performance disc drive for use in mainstream storage applications, visit https://www.seagate.com/15K
Real Devices – the small print

- The Cheetah 15K.4 is the highest-performance drive ever offered by Seagate, delivering maximum IOPS with fewer drives to yield lower TCO.
- The Cheetah 15K.4 price-per-performance value united with the breakthrough benefits of serial attached SCSI (SAS) make it the optimal 3.5-inch drive for rock solid enterprise storage.
- Proactive, self-initiated background management functions improve media integrity, increase drive efficiency, reduce incidence of integration failures and improve field reliability.
- The Cheetah 15K.4 shares its electronics architecture and firmware base with Cheetah 10K.7 and Savvio™ to ensure greater factory consistency and reduced time to market.

KEY SPECIFICATIONS

- 146-, 73- and 36-Gbyte capacities
- 3.3-msec average read and 3.8-msec average write seek times
- Up to 96 Mbytes/sec sustained transfer rate
- **1.4 million hours full duty cycle MTBF**
- Serial Attached SCSI (SAS), Ultra320 SCSI and 2 Gbits/sec Fibre Channel interfaces
- 5-year warranty

For more information on why 15K is the industry’s best price/performance disc drive for use in mainstream storage applications, visit [http:// specials.seagate.com/15k](http://specials.seagate.com/15k)
Real Devices – the small print

- The Cheetah 15K.4 is the highest-performance drive ever offered by Seagate, delivering maximum IOPS with fewer drives to yield lower TCO.
- The Cheetah 15K.4 price-per-performance value united with the breakthrough benefits of hot-pluggable SAS (SAS) make it the optimal 3.5-inch drive for rock solid enterprise storage.
- Improved background management functions improve media integrity, data integrity, consistency, and reduced write seek times.
- 3Gb/s Fibre Channel interfaces for the best price/performance disc drive for
  a
  b
  c

170 years….??!

Last Will and Testament

One hard drive for my great-grandkids to use
Disk failure conditional probability distribution - Bathtub curve

- **Infant mortality**
- **Burn out**
- **Stable failure period**
- **Expected operating lifetime**

1 / (reported MTTF)
Other (more morbid) Bathtub Curves

Data from http://www.mortality.org

Human Mortality Rates (US, 1999)

Other Bathtub Curves

Figure 2. Age-specific death rates: United States, preliminary 2007

SOURCE: National Vital Statistics System, Mortality

Data from http://www.mortality.org
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.
  - This program is running one of a dozen storage servers for your distributed filesystem.
  - A client can’t read from the DFS, so it hangs.
  - A professor can’t check out a copy of your 15-440 assignment, so he gives you an F.
What are our options?

1. Silently return the wrong answer.

2. Detect failure.

3. Correct / mask the failure
Parity Checking

**Single Bit Parity:**
Detect single bit errors

```
  0111000110101011 0
```

- d data bits
- parity bit
Block Error Detection

- **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

```
<table>
<thead>
<tr>
<th>datagram</th>
<th>Y</th>
<th>all bits in D' OK ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td>N detected error</td>
</tr>
</tbody>
</table>

bit-error prone link
```
Error Detection - Checksum

- Used by TCP, UDP, IP, etc..
- Ones complement sum of all words/shorts/bytes in packet
- Simple to implement
- Relatively weak detection
  - Easily tricked by typical error patterns – e.g. bit flips
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?
Error Detection – Cyclic Redundancy Check (CRC)

- Polynomial code
  - Treat packet bits as coefficients of an n-bit polynomial
  - Choose a r+1 bit generator polynomial (well known – chosen in advance)
  - Add r bits to packet such that message is divisible by generator polynomial

- Better loss detection properties than checksums
  - Cyclic codes have favorable properties in that they are well suited for detecting burst errors
  - Therefore, used on networks/hard drives
Error Detection – CRC

- View data bits, $D$, as a binary number
- Choose $r+1$ bit pattern (generator), $G$
- Goal: choose $r$ CRC bits, $R$, such that
  - $<D,R>$ exactly divisible by $G$ (modulo 2)
  - Receiver knows $G$, divides $<D,R>$ by $G$. If non-zero remainder: error detected!
  - Can detect all burst errors less than $r+1$ bits
- Widely used in practice: Ethernet, disks

$D \cdot 2^r \text{ XOR } R$

$D$: data bits to be sent
$R$: CRC bits

bit pattern

mathematical formula
CRC Example

Want:

\[ D \cdot 2^r \text{ XOR } R = nG \]

equivalently:

\[ D \cdot 2^r = nG \text{ XOR } R \]

equivalently:

if we divide \( D \cdot 2^r \) by \( G \),
want reminder \( R \)

\[ R = \text{remainder}[\frac{D \cdot 2^r}{G}] \]
What are our options?

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

<table>
<thead>
<tr>
<th></th>
<th>d&lt;sub&gt;1,1&lt;/sub&gt;</th>
<th>...</th>
<th>d&lt;sub&gt;1,j&lt;/sub&gt;</th>
<th>d&lt;sub&gt;1,j+1&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>d&lt;sub&gt;2,1&lt;/sub&gt;</td>
<td>...</td>
<td>d&lt;sub&gt;2,j&lt;/sub&gt;</td>
<td>d&lt;sub&gt;2,j+1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>d&lt;sub&gt;i,1&lt;/sub&gt;</td>
<td>...</td>
<td>d&lt;sub&gt;i,j&lt;/sub&gt;</td>
<td>d&lt;sub&gt;i,j+1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>d&lt;sub&gt;i+1,1&lt;/sub&gt;</td>
<td>...</td>
<td>d&lt;sub&gt;i+1,j&lt;/sub&gt;</td>
<td>d&lt;sub&gt;i+1,j+1&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

- 1010111
- 111100
- 011101
- 001010

---

- **no errors**
- **correctable single bit error**

**Example:**

- 101011  
- 101011  
- 10100  
- 011101  
- 001010  

**Error:**

- 101011  
- 101100  
- 011101  

**Corrected:**

- 101011  
- 101100  
- 011101  
- 001010
Replication/Voting

• If you take this to the extreme
  \([r1] \ [r2] \ [r3]\)

• 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)
  • Stuff we cover later: surviving malicious failures through voting (byzantine fault tolerance)
Retry – Network Example

- Sometimes errors are transient
- Need to have error detection mechanism
  - E.g., timeout, parity, chksum
  - 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 \( p(\text{computer 2 failing} | \text{computer 1 failed}) \)?
    - Maybe it’s \( p \)… or maybe they’re both plugged into the same UPS…

- Why is this important?
  - Correlation reduces value of redundancy
Fault Tolerant Design

• Quantify probability of failure of each component
• Quantify the costs of the failure
• Quantify the costs of implementing fault tolerance

• This is all probabilities...
Outline

• Errors/error recovery

• Using multiple disks
  • Why have multiple disks?
  • problem and approaches

• RAID levels and performance

• Estimating availability
## Replacement Rates

<table>
<thead>
<tr>
<th>Component</th>
<th>HPC1 %</th>
<th>COM1 Component</th>
<th>COM1 %</th>
<th>COM2 Component</th>
<th>COM2 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard drive</td>
<td>30.6</td>
<td>Power supply</td>
<td>34.8</td>
<td>Hard drive</td>
<td>49.1</td>
</tr>
<tr>
<td>Memory</td>
<td>28.5</td>
<td>Memory</td>
<td>20.1</td>
<td>Motherboard</td>
<td>23.4</td>
</tr>
<tr>
<td>Misc/Unk</td>
<td>14.4</td>
<td>Hard drive</td>
<td>18.1</td>
<td>Power supply</td>
<td>10.1</td>
</tr>
<tr>
<td>CPU</td>
<td>12.4</td>
<td>Case</td>
<td>11.4</td>
<td>RAID card</td>
<td>4.1</td>
</tr>
<tr>
<td>motherboard</td>
<td>4.9</td>
<td>Fan</td>
<td>8</td>
<td>Memory</td>
<td>3.4</td>
</tr>
<tr>
<td>Controller</td>
<td>2.9</td>
<td>CPU</td>
<td>2</td>
<td>SCSI cable</td>
<td>2.2</td>
</tr>
<tr>
<td>QSW</td>
<td>1.7</td>
<td>SCSI Board</td>
<td>0.6</td>
<td>Fan</td>
<td>2.2</td>
</tr>
<tr>
<td>Power supply</td>
<td>1.6</td>
<td>NIC Card</td>
<td>1.2</td>
<td>CPU</td>
<td>2.2</td>
</tr>
<tr>
<td>MLB</td>
<td>1</td>
<td>LV Pwr Board</td>
<td>0.6</td>
<td>CD-ROM</td>
<td>0.6</td>
</tr>
<tr>
<td>SCSI BP</td>
<td>0.3</td>
<td>CPU heatsink</td>
<td>0.6</td>
<td>Raid Controller</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Back to Disks…
What are our options?

1. Silently return the wrong answer.

2. Detect failure.
   - Every sector has a header with a CRC. Every read fetches both, computes the CRC 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 with?)
     - Bit flips? Yes. Block damaged? No
   - Have the data stored in multiple places (RAID)
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
  - Small requests benefit from load balancing
    - If blocks of hot files equally likely on all disks (really?)

File Foo:

stripe unit or block

Stripe
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
Tolerating and masking disk failures

• If a disk fails, it’s data is gone
  • may be recoverable, but may not be

• To keep operating in face of failure
  • must have some kind of data redundancy

• Common forms of data redundancy
  • replication
  • erasure-correcting codes
  • error-correcting codes
Redundancy via replicas

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

![Diagram showing four replicas with values 0, 0, 2, 2, 1, 1, 3, 3]
Mirroring & Striping

- Mirror to 2 virtual drives, where each virtual drive is really a set of striped drives
  - Provides reliability of mirroring
  - Provides striping for performance (with write update costs)
A Better Approach?: Parity Disk

- Capacity: one extra disk needed per stripe
- Disk failures are self-identifying (a.k.a. erasures)
  - Don’t have to find the error
Lower Cost Data Redundancy

- Disk failures are self-identifying (a.k.a. erasures)
  - Don’t have to find the error

- Fact: N-error-detecting code is also N-erasure-correcting
  - Error-detecting codes can’t find an error, just know its there
  - But if you independently know where error is, allows repair

- Parity is single-disk-failure-correcting code
  - recall that parity is computed via XOR
  - it’s like the low bit of the sum
Updating and using the parity

**Fault-Free Read**

- D
- D
- D
- P

**Fault-Free Write**

1. D
2. D
3. D
4. P

**Degraded Read**

- D
- D
- P

**Degraded Write**

- D
- D
- P
Performance

• Suppose 1 drive gives bandwidth $B$
• Fault-Free Read = $3B$
• Degraded Read = $1B$
• Fault-Free Write = $0.5 \text{ B}$
  • But can do $2B$ Fault-Free Read at the same time
• Degraded Write = $1 \text{ B}$
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
Outline

- Using multiple disks
  - Why have multiple disks?
  - problem and approaches

- RAID levels and performance

- Estimating availability
RAID Taxonomy

• Redundant Array of Inexpensive Independent Disks
  • Constructed by UC-Berkeley researchers in late 80s (Garth Gibson)
• 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 – Extends RAID 5 by adding another parity block
• RAID 10 – RAID 1 (mirroring) + RAID 0 (striping)
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
- Capacity: How much space is wasted?
- Performance: How much faster than 1 disk?
- 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 → Parity value?
• **How do you recover from a failed disk?**
  • Example: x 0 0 and parity of 1
  • What is the failed value?
RAID-4: Parity

- Capacity:
- Reliability:
- Performance:
  - Reads
  - Writes: How to update parity block?
    - Two different approaches
      - Small number of disks (or large write):
      - Large number of disks (or small write):
    - Parity disk is the bottleneck
RAID-5: Rotated 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></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 · S</td>
<td>N x R</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>

Key takeaways: writes are expensive, small writes are really expensive! File systems may help (see LFS)
Outline

• Using multiple disks
  • Why have multiple disks?
  • problem and approaches

• RAID levels and performance

• Estimating availability
Availability / Reliability Metrics

- We want to calculate MTTDL
  - Mean Time To First Data Loss (MTTDL)
  - Given that we know the MTTF
  - Under simplifying assumptions

![Diagram showing availability and reliability metrics]

Available during these 3 periods of time.
Estimating Availability / Reliability

• Back-of-the-envelope calculation
  • Assuming failures across devices are independent
  • Assuming failures over time are independent
  • Assuming failure rate from stable failure period

\[
\frac{1}{\text{reported MTTF}}
\]
How often are failures?

- **MTTF** (Mean Time Between Failures)
  - \( MTTF_{\text{disk}} \approx 1,200,000 \text{ hours} \ (\sim 136 \text{ years, } <1\% \text{ per year}) \)
- **MTTDL** _multi-disk system_ = mean time to first disk failure
  - which is \( MTTF_{\text{disk}} \) / (number of disks)
  - For a striped array of 200 drives
  - \( MTTDL_{\text{array}} = 136 \text{ years} / 200 \text{ drives} = 0.65 \text{ years} \)
Reliability without rebuild

- 200 data drives with MTTF\textsubscript{drive}
  - MTTDL\textsubscript{array} = MTTF\textsubscript{drive} / 200

- Add 200 drives and do mirroring
  - MTTF\textsubscript{pair} = (MTTF\textsubscript{drive} / 2) + MTTF\textsubscript{drive} = 1.5 * MTTF\textsubscript{drive}
  - MTTDL\textsubscript{array} = MTTF\textsubscript{pair} / 200 = MTTF\textsubscript{drive} / 133

- Add 50 drives, each with parity across 4 data disks
  - MTTF\textsubscript{set} = (MTTF\textsubscript{drive} / 5) + (MTTF\textsubscript{drive} / 4) = 0.45 * MTTF\textsubscript{drive}
  - MTTDL\textsubscript{array} = MTTF\textsubscript{set} / 50 = MTTF\textsubscript{drive} / 111

- These are approximations
Rebuild: restoring redundancy after failure

- After a drive failure
  - data is still available for access
  - but, a second failure is BAD

- So, should reconstruct the data onto a new drive
  - online spares are common features of high-end disk arrays
    - reduce time to start rebuild
    - must balance rebuild rate with foreground performance impact
      - a performance vs. reliability trade-offs

- How data is reconstructed
  - Mirroring: just read good copy
  - Parity: read all remaining drives (including parity) and compute
Reliability consequences of adding rebuild

- No data loss, if fast enough
  - That is, if first failure fixed before second one happens

- New math is...
  - \( \text{MTTDL}_{\text{array}} = \text{MTTF}_{\text{firstdrive}} \times \left(1 / \text{prob of 2}\text{nd failure before repair}\right) \)
    ... where prob is \( \text{MTTR}_{\text{drive}} / \text{MTTF}_{\text{seconddrive}} \)

- For mirroring
  - \( \text{MTTDL}_{\text{pair}} = \left(\frac{\text{MTTF}_{\text{drive}}}{2}\right) \times \left(\frac{\text{MTTF}_{\text{drive}}}{\text{MTTR}_{\text{drive}}}\right) \)

- For 5-disk parity-protected arrays
  - \( \text{MTTDL}_{\text{set}} = \left(\frac{\text{MTTF}_{\text{drive}}}{5}\right) \times \left(\frac{\text{MTTF}_{\text{drive}}}{4}\right) / \text{MTTR}_{\text{drive}} \)
Three modes of operation

- **Normal mode**
  - everything working; maximum efficiency

- **Degraded mode**
  - some disk unavailable
  - must use degraded mode operations

- **Rebuild mode**
  - reconstructing lost disk’s contents onto spare
  - degraded mode operations plus competition with rebuild
Mechanics of rebuild

- Background process
  - use degraded mode read to reconstruct data
  - then, write it to replacement disk

- Implementation issues
  - Interference with foreground activity and controlling rate
    - Rebuild is important for reliability
    - Foreground activity is important for performance
  - Using the rebuilt disk
    - For rebuilt part, reads can use replacement disk
    - Must balance performance benefit with rebuild interference
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?
  • Leading into replication as a general strategy for fault tolerance
• Thought to leave you with:
  • What if you have to survive the failure of entire computers? Of a rack? Of a datacenter?
Summary

- 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-5**: Rotating Parity
  Good when capacity and cost matter or workload is read-mostly
  - Good compromise choice