ECE5990 Lecture 10

Reliability

Christina Delimitrou

https://sites.google.com/site/topicsondatacentercomputing/
Announcements

- Project meetings
  - Starting this week

- Midterm 3/26 (in class)
  - All lectures & papers until midterm
Reminder: DC Metrics

- **Performance**
  - Throughput and tail latency

- **Power and energy consumption**
  - PUE = \( \frac{\text{Total Datacenter Power}}{\text{IT Equipment Power}} \)

- **Availability**
  - 99.99% \( \rightarrow \) 1h of downtime per year

- **Total cost of ownership (TCO)**
Availability & Reliability
Availability & Reliability

- What is availability and reliability?
- What are the principles of high-availability?
- What are some specific design optimizations?

Readings

- Barroso & Hoelzle textbook, chapter 7
- Hennessy & Patterson, chapter 1.7
Tail Latency

- **Cross-thread interference:** avg low, 95\(^{th}\) percentile very high \(\rightarrow\) in a single server
Larger clusters \(\Rightarrow\) more prone to high tail latency

\(^1\)The Tail at Scale. Jeffrey Dean, Luiz André Barroso. CACM, Vol. 56 No. 2, Pages 74-80, 2013
Reliability & Availability

- Common goal for services: 99.99% availability
  - 1 hour of down-time per year

- But with thousands of nodes, things will crash
  - Example: with 10K servers rated at 30 years of MTBF, you should expect to have 1 failure per day
Reliability

Reliability: measure of continuous “service”

- **MTTF**: mean time to failure
  - Time to produce first incorrect output

- **MTTR**: mean time to repair
  - Time to detect and repair a failure

- **MTBF = mean time between failures = MTTF + MTTR**

- **Failure in time (FIT) = failures per billion hours of operation = $10^9/\text{MTTF}$**
  - E.g., MTTF = 1,000,000 hours $\rightarrow$ 1,000 FIT
(Service) Availability

Steady state availability

\[ = \frac{MTTF}{MTTF + MTTR} \]
(Service) Availability

- Availability often quoted in “9s”

- E.g., the telephone system has five 9s availability

- 99.999% availability or 5 minutes of downtime per year

<table>
<thead>
<tr>
<th>Uptime</th>
<th>Downtime in one year</th>
</tr>
</thead>
<tbody>
<tr>
<td>99% (two 9’s)</td>
<td>87.6 hours</td>
</tr>
<tr>
<td>99.9% (three 9’s)</td>
<td>8.76 hours</td>
</tr>
<tr>
<td>99.99% (four 9’s)</td>
<td>53 min</td>
</tr>
<tr>
<td>99.999% (five 9’s)</td>
<td>5 min</td>
</tr>
<tr>
<td>99.9999% (six 9’s)</td>
<td>32 sec</td>
</tr>
<tr>
<td>99.99999% (seven 9’s)</td>
<td>3 sec</td>
</tr>
</tbody>
</table>
## Why is availability important?

<table>
<thead>
<tr>
<th>Application</th>
<th>Cost of downtime per hour</th>
<th>Annual losses with downtime of</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1% (87.6 hrs/yr)</td>
</tr>
<tr>
<td>Brokerage operations</td>
<td>$6,450,000</td>
<td>$565,000,000</td>
</tr>
<tr>
<td>Credit card authorization</td>
<td>$2,600,000</td>
<td>$228,000,000</td>
</tr>
<tr>
<td>Package shipping services</td>
<td>$150,000</td>
<td>$13,000,000</td>
</tr>
<tr>
<td>Home shopping channel</td>
<td>$113,000</td>
<td>$9,900,000</td>
</tr>
<tr>
<td>Catalog sales center</td>
<td>$90,000</td>
<td>$7,900,000</td>
</tr>
<tr>
<td>Airline reservation center</td>
<td>$89,000</td>
<td>$7,900,000</td>
</tr>
<tr>
<td>Cellular service activation</td>
<td>$41,000</td>
<td>$3,600,000</td>
</tr>
<tr>
<td>Online network fees</td>
<td>$25,000</td>
<td>$2,200,000</td>
</tr>
<tr>
<td>ATM service fees</td>
<td>$14,000</td>
<td>$1,200,000</td>
</tr>
</tbody>
</table>

Figure 1.3 Costs rounded to nearest $100,000 of an unavailable system is shown by analyzing the cost of downtime (in terms of immediately lost revenue), assuming three different levels of availability, and that downtime is distributed uniformly. These data are from Kembel [2000] and were collected and analyzed by Contingency Planning Research.

**Mission-critical (100% uptime), business-critical (minimal interruptions)**
Types of Faults

- **Hardware faults**
  - Radiation, noise, thermal issues, variation, wear-out, faulty equipment

- **Software faults**
  - OS, applications, drivers: bugs, security attacks

- **Operator errors**
  - Incorrect configurations, shutting down wrong server, incorrect operations

- **Environmental factors**
  - Natural disasters, air conditioning, and power grids
  - Wild dogs, sharks, dead horses, thieves, blasphemy, drunk hunters (Barroso’10)

- **Security breaches**
  - Unauthorized users, malicious behavior (data loss, system down)

- **Planned service events**
  - Upgrading HW (add memory) or upgrading SW (patch)
Types of Faults

- **Permanent**
  - Defects, bugs, out-of-range parameters, wear-out, ...

- **Transient (temporary)**
  - Radiation issues, power supply noise, EMI, ...

- **Intermittent (temporary)**
  - Oscillate between faulty & non-faulty operation
  - Operation margins, weak parts, activity, ...
  - Sometimes called Bohrbugs and Heisenbugs
Data Center Example

- Typical first year for a new cluster:
  - ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
  - ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
  - ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
  - ~1 network rewiring (rolling ~5% of machines down over 2-day span)
  - ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
  - ~5 racks go wonky (40-80 machines see 50% packetloss)
  - ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
  - ~12 router reloads (takes out DNS and external vips for a couple minutes)
  - ~3 router failures (have to immediately pull traffic for an hour)
  - ~dozens of minor 30-second blips for dns
  - ~1000 individual machine failures
  - ~thousands of hard drive failures

- slow disks, bad memory, misconfigured machines, flaky machines, etc.
Real-world Service Disruptions

Source of “disruptions events” at Google

Source of enterprise “disruption events”

- Large number of techniques on hardware fault-tolerance
- Software, operator, maintenance-induced faults
  - Affect multiple systems at once, correlated failure FT harder

(disruption event = service degradation that triggered operations team scrutiny)
Exercise

- A 2400-server Google cluster has the following events per year:
  - Cluster upgrades: 4 (need to fix)
  - Hard drive failures: 250 (need to fix)
  - Bad memories: 250 (need to fix)
  - Misconfigured machines: 250 (need to fix)
  - Flaky machines: 250 (need to reboot)
  - Server crashes: 5000 (need to reboot)

- Assume: time to reboot SW is 5 minutes, time to fix HW is 1 hour

- What is service availability?
Answer

- Hours outage
  - \((4 + 250 + 250 + 250) \times 1\) hour
  - \(+ (250 + 5000) \times 5\) minutes =
  - \(754 + 438 = 1192\) hours

- Hours in year = \(365 \times 24 = 8760\)
- Availability = \(1 - \frac{1192}{8760} = 86\%\)

Did Google have only 86% uptime last year?
Faults ≠ Failure

- Failure: service is unavailable or data integrity is lost
  - The user can really tell

- Possible effects of a fault (increasing severity)
  - Masked (examples?)
  - Degraded service
  - Unreachable service (service not available)
  - Data loss or corruption (data are not durable)
Techniques for Availability
Techniques for Availability

- Steady state availability = $\frac{MTTF}{MTTF + MTTR}$

- For higher availability, you can work on
  - Very high MTTF (reliable computing, fault prevention)
  - Very high MTTF (fault-tolerant computing)
  - Very low MTTR (recovery-oriented computing)
Improving MTTF & MTTR

- Two issues: error detection and error correction

- Observations
  - Both are useful (e.g., fail-safe operation after detection)
  - Both add to cost so use carefully
  - Can be done at multiple levels (chip/system/DC, HW/SW)

- Some terminology
  - Fail-fast: either function correctly or stop when error detected
  - Fail-silent: system crashes on failure (not necessarily immediately)
  - Fail-safe: automatically counteracting a failure

- Following slides: example techniques
  - General, disks, memories, networks, processors, system...
General: “Infant Mortality”

- Many failures happen in early stages of use
  - Marginal components, design/SW bugs, etc
- Use “burn-in” to screen such issues
  - E.g., stress test HW or SW before deployment
  - See next slide for some basic approaches
RAID: Dealing with Faults in Storage Systems

- Redundant arrays of inexpensive disks (RAID) (1987)
  - A collection of disks that behaves like a single disk with
    - High capacity, high bandwidth, high reliability disk
  - Key idea in RAID: error correcting information across disks
  - Many organizations; two distinguishing features:
    - The granularity of the interleaving (bit, byte, block)
    - The amount and distribution of redundant information
  - Patterson’s classification: RAID levels 0 to 6

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAID 0</td>
<td>Block-level striping without parity mirroring</td>
</tr>
<tr>
<td>RAID 1</td>
<td>Mirroring without parity striping</td>
</tr>
<tr>
<td>RAID 2</td>
<td>Bit-level striping with dedicated parity</td>
</tr>
<tr>
<td>RAID 3</td>
<td>Byte-level striping with dedicated parity</td>
</tr>
<tr>
<td>RAID 4</td>
<td>Block-level striping with dedicated parity</td>
</tr>
<tr>
<td>RAID 5</td>
<td>Block-level striping with distributed parity</td>
</tr>
<tr>
<td>RAID 6</td>
<td>Block-level striping with double-distributed parity</td>
</tr>
<tr>
<td>RAID 1+0</td>
<td>Disk mirroring and data striping without parity</td>
</tr>
</tbody>
</table>
### RAID 0 & 1: Block Duplication

<table>
<thead>
<tr>
<th>Disk 0</th>
<th>Disk 1</th>
<th>Disk 2</th>
<th>Disk 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
</tr>
<tr>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
</tr>
<tr>
<td>D8</td>
<td>D9</td>
<td>D10</td>
<td>D11</td>
</tr>
<tr>
<td>D12</td>
<td>D13</td>
<td>D14</td>
<td>D15</td>
</tr>
</tbody>
</table>

- **RAID 0**: Non-redundancy (sharding)
  - Best write performance
  - Not necessarily the best read latency!
  - Worst reliability

<table>
<thead>
<tr>
<th>Disk 0</th>
<th>Disk 1</th>
<th>Disk 2</th>
<th>Disk 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D1</td>
<td>D0</td>
<td>D1</td>
</tr>
<tr>
<td>D2</td>
<td>D3</td>
<td>D2</td>
<td>D3</td>
</tr>
<tr>
<td>D4</td>
<td>D5</td>
<td>D4</td>
<td>D5</td>
</tr>
<tr>
<td>D6</td>
<td>D7</td>
<td>D6</td>
<td>D7</td>
</tr>
</tbody>
</table>

- **RAID 1/1+0**: Mirroring (shadowing)
  - Half the capacity
  - Faster read latency: schedule the disk with the lowest queuing, seek, and rotational delays
  - What about write?
## RAID 5: Block-level, Distributed Parity

- One extra disk to provide storage for parity blocks
  - Given N block + parity, we can recover from 1 disk failure
- Parity distributed across all disks to avoid write bottleneck
  - Good small/large read, and best large write performance of any RAID (cache)
  - Small write performance worse than, say, RAID-1

<table>
<thead>
<tr>
<th>Disk 0</th>
<th>Disk 1</th>
<th>Disk 2</th>
<th>Disk 3</th>
<th>Disk 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>p0-3</td>
</tr>
<tr>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>p4-7</td>
<td>D4</td>
</tr>
<tr>
<td>D10</td>
<td>D11</td>
<td>p8-11</td>
<td>D8</td>
<td>D9</td>
</tr>
<tr>
<td>D15</td>
<td>p12-15</td>
<td>D12</td>
<td>D13</td>
<td>D14</td>
</tr>
</tbody>
</table>
RAID 5: Block-level, Distributed Parity

- One extra disk to provide storage for parity blocks
  - Given N block + parity, we can recover from 1 disk failure
- Parity distributed across all disks to avoid write bottleneck
  - Good small/large read, and best large write performance of any RAID (cache)
  - Small write performance worse than, say, RAID-1
RAID 4: Block-level, Dedicated Parity

- One extra disk to provide storage for parity blocks
  - Given N block + parity, we can recover from 1 disk failure
- Parity aggregated in a single disk
- Issues?
RAID 6: Multiple Types of Parity

- Two extra disks protect against 2 disk failures
  - E.g., row and diagonal parities
- Parity blocks can be distributed like in RAID-5
  - Separate parity disks shown for clarity
- Also called RAID/ADG
RAID Discussion

- RAID layers tradeoffs
  - Space, fault-tolerance, read/write performance

- HW-based RAID-5 is becoming less popular
  - RAID1+0 is often used for large disk arrays

- RAID can be done in SW & across servers
  - RAID-5 like approaches are popular
Dealing with Faults in Memories

- **Permanent faults (stuck at 0/1 bits)**
  - Address with redundant rows/column (spares)
  - Built-in-self-test & fuses to program decoders
  - Done during manufacturing test

- **Transient faults**
  - Bits flip 0 → 1 or 1 → 0
  - Detect using parity (i.e., a 9\textsuperscript{th} bit per byte)
    - Even parity: make the 9\textsuperscript{th} bit 1 if the number of 1s in the byte is odd
ECC for Transient Faults

- Error correcting codes
  - Error correction using Hamming codes
    - E.g., add a 8-bit code to each 64-bit word
  - Common case for DRAM: SECDED (y=2, x=1)
    - Can buy DRAM chips with (64+8)-bit words to use with SECDED (single error correct double error detect)
  - Codes calculated/checked by memory controller
Error Correcting Codes (ECC)

- P extra parity bits for N bits of data
  - Distance: the # of bits that are different between two words
  - Example: 0001 and 1000 have distance D=2
  - We can detect any single-bit error given these two words

- In general, codes where valid words are at least D bits apart
  - We can detect D-1 errors
  - To correct x errors, we need the D \geq 2x+1
ECC Issues

- Performance issue
  - Every subword write is a read-modify-write
  - Necessary to update the ECC bits

- Reliability issue: double-bit errors
  - Likely if we take a long time to re-read some data
  - Solution: background scrubbing

- Reliability issue: a whole-chip failure
  - Possible if it affects interface logic of the chip
  - Solutions?
ChipKill: RAID for DRAM

Memory Organization

- Chipkill DIMMs: ECC across chips (or even DIMMS)
  - Instead of 8 use 9 regular DRAM chips (64-bit words)
  - Can tolerate errors both within and across chips
Dealing with Network Faults

- Use error detecting codes and retransmissions
  - CRC: cyclic redundancy code
  - Receiver detects error and requests retransmission
    - Requires buffering at the sender side
  - An ack/nack protocol is typically used
    - To indicate when the receiver received correct data (or not)
    - Time-outs to deal with the case of lost messages
      - Error in control signals or with acknowledgements

- Permanent faults
  - Use network with path diversity
Dealing with Faults in Logic

- **Triple module redundancy (TMR)**
  - 3 copies of compute unit + voter
  - Issues: synchronization & common mode errors

- **Dual modular redundancy (DMR)**
  - 2 copies of compute unit + comparator
  - Can use simpler 2\textsuperscript{nd} copy (e.g., parity predictor)

- **Checkpoint & restore**
  - Periodic checkpoints of state
  - On error detection, rollback & re-execute from checkpoint
  - Issues: checkpoint interval, detection speed, # of checkpoints, recovery time, …
Data Center Availability

- Mostly system-level, SW-based techniques
  - Using clusters for high-availability

- Reasons
  - High cost of server-level techniques
    - Cost of failures vs cost of more reliable servers
  - Cannot rely on all servers working reliably anyway!
    - Example: with 10K servers rated at 30 years of MTBF, you should expect to have 1 failure per day

- But components need to be reliable enough…
  - ECC based memory used (detection is important)
## DC Availability Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Performance</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Partitioning (sharding)</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Load-balancing</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Watchdog timers</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Integrity checks</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>App-specific compression</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Eventual consistency</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

### Other techniques
- Fail-safe behavior, admission control, spare capacity
- Use monitoring/deployment mgmt system to handle failures as well