Fault Tolerance

15-719/18-709: Advanced Cloud Computing

George Amvrosiadis
Greg Ganger
Majd Sakr

Agenda

• Failures in the field: severity, factors, ...
• Failure types: hard and soft
• Handling soft failures
  • Transactions
  • Two-phase locking
  • The Replicated State Machine model
• Designing recovery models
Failures are expensive

- Failures can cost millions of $ per hour of downtime
  - Mostly due to reputation damages, lost revenue
- 56% due to equipment failures
  - +22% due to cyber crime
  - +22% due to human error

![Chart: Total cost per minute of an unplanned outage](image)

Source: Ponemon Institute study, 2016

Understanding failures thru data

- Rich literature (especially in the last decade)
  - 2007: Studies of 100K+ disks in HPC [Schroeder07], 1.5M+ at NetApp [Bairavasundaram07], Google [Pineiro07]
  - 2009: Studies of DRAM errors at Google [Schroeder09]
  - 2010: Microsoft studies 100K machines over 4 months [Viswanath10]
  - 2012: Effect of temperature on hardware reliability [ElSayed12], DRAM failure studies [Hwang12]
  - 2016: Effect of humidity on storage reliability [Manousakis16], SSD failures at Google [Schroeder16]

- USENIX Computer Failure Data Repository (https://www.usenix.org/cfdr)
  - 30+ clusters, 13+ sites, HPC & Internet services (1996-2009)
  - > 23,000 failures, > 100,000 disk drives
## USENIX failure data: Hardware replacement logs

<table>
<thead>
<tr>
<th>Organization</th>
<th>Drive types</th>
<th>Drive count</th>
<th>Trace duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pittsburgh Supercomputing Center</td>
<td>18GB 10K RPM SCSI 36GB 10K RPM SCSI</td>
<td>3,400</td>
<td>5 years</td>
</tr>
<tr>
<td>HPC1</td>
<td>36GB 10K RPM SCSI</td>
<td>520</td>
<td>2.5 years</td>
</tr>
<tr>
<td>Supercomputing X</td>
<td>HPC3 15K RPM SCSI 15K RPM SCSI 7.2K RPM SATA</td>
<td>14,208</td>
<td>1 year</td>
</tr>
<tr>
<td>HPC2</td>
<td>36GB 10K RPM SCSI</td>
<td>520</td>
<td>2.5 years</td>
</tr>
<tr>
<td>Various HPCs</td>
<td>COM1 10K RPM SCSI</td>
<td>26,734</td>
<td>1 month</td>
</tr>
<tr>
<td>Internet Services Y</td>
<td>COM2 15K RPM SCSI</td>
<td>39,039</td>
<td>1.5 years</td>
</tr>
<tr>
<td>HPC4</td>
<td>COM3 10K RPM FC-AL 10K RPM FC-AL 10K RPM FC-AL</td>
<td>3,700</td>
<td>1 year</td>
</tr>
</tbody>
</table>

### Popular replaced components

#### The top ten of replaced components

<table>
<thead>
<tr>
<th>Component</th>
<th>HPC1</th>
<th>COM1</th>
<th>COM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard drive</td>
<td>30.6</td>
<td>34.8</td>
<td>49.1</td>
</tr>
<tr>
<td>Memory</td>
<td>28.5</td>
<td>20.1</td>
<td>23.4</td>
</tr>
<tr>
<td>Misc/Unk</td>
<td>14.4</td>
<td>8.0</td>
<td>11.4</td>
</tr>
<tr>
<td>CPU</td>
<td>12.4</td>
<td>2.0</td>
<td>10.1</td>
</tr>
<tr>
<td>PCl motherboard</td>
<td>4.9</td>
<td>1.2</td>
<td>4.1</td>
</tr>
<tr>
<td>Controller</td>
<td>2.9</td>
<td>0.6</td>
<td>3.4</td>
</tr>
<tr>
<td>QSW</td>
<td>1.7</td>
<td>0.6</td>
<td>2.2</td>
</tr>
<tr>
<td>Power supply</td>
<td>1.6</td>
<td>0.6</td>
<td>2.2</td>
</tr>
<tr>
<td>MLB</td>
<td>1.0</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>SCSI BP</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All hardware fails, but disk failures often common
Disk Failures in practice

- HDDs fail often
  - 1-13% replaced annually
- SSDs seem more reliable
  - 1-3% replaced annually
- Studies look at replacement events only
  - Disks may not have suffered complete failure
  - Depends on administrator replacement policies

Definitions are important!

- Datasheet MTTFs are 1,000,000 to 1,500,000 hours
- Expected annual replacement rate of 0.58 – 0.88% but:
  - Vendor says “no fault found” for 43% of returns
  - Customer breaks warranty: heat, moisture, vibration, workload, …
Microsoft datacenter study

• Studied 100K+ servers for 14 months
  • Unreported socket/chip count, usage
• 8% machine hardware fails annually
  • 50% repaired once, 85% < 4 times
  • 78% failures caused by hard disks
    • 2.7% disks repaired per year
  • 5% were RAID controllers
  • 3% were memory DIMM failures
  • 20% repeat failures within 1 day
    • 50% repeat failures within 2 weeks

Failure types

• How would the faulty component behave?
  • Byzantine behavior: faulty component may send arbitrary messages
  • Fail-stop behavior: faulty component stops and does not send

• Examples of fail-stop failures
  • Media failure: IO device code bugs, disk HW failures
    • Loss of durable data
  • System failure: DB bug, OS fault, HW failure
    • Loss of volatile data but durable memory (disk) survives
How does software handle **hard** failures?

**Storage redundancy + Repeatable computation**

- Short, non-shared, deterministic programs
  - OS, framework or user destroys partial changes, then reruns program
  - Builds on external storage independently protected (RAID/Replicas)
- Long running, non-shared, deterministic programs
  - Examples: scientific simulations, Extract/Transform/Load jobs
  - Periodic state checkpoints to durable, independent, protected storage
    - Components/tasks may checkpoint independently (less synchronization)
  - On failure: isolate failed component/system, restart from checkpoint
    - Dependent components/systems can trigger failure detection

How does software handle **soft** failures?

- **Micro-reboot** restarts individual long-running system software components
  - Fine-grained, transactional components: restart and reinitialize fast
  - State segregation: prevent corruption by storing important state externally
  - Loosely coupled components: well-defined, well-enforced boundaries
  - Retry-able requests: inter-component interactions use timeouts
  - Expiring locks (leases): clean-up simplification
- Concurrent, **shared data** (database) multi-app systems
  - Shared state interaction through **ACID transactions** and write-ahead logging
  - External state independently protected
- Concurrent, **shared-nothing** replicated systems (maybe no external state)
  - Replicated state machines driven by coordinating replica changes
Transactions

• **Goal:** multiple users manipulate shared data safely

• **ACID** properties of a transaction
  • **Atomicity:** an operation is done all-or-nothing
    • Partial changes must be tentative until one committing change
  • **Consistency:** user-specified (correctness) constraints applied before commit
  • **Isolation:** partial changes not visible to other users’ code (less complex)
  • **Durability:** changes survive subsequent failures (storage and process redundancy)

• **AID** provided by database system, C (mostly) by programmer
  • Database is consistent iff contents result only from successful transactions
  • Integrity constraints (partial consistency) may be enforced by DB
    • Replica consistency is an important special case (later)

---

Isolation: Two-Phase Locking (2PL)

• **Goal:** isolation mechanism guaranteeing serializability
  • Why? Simple for users to consider transactions running one at a time

• Assuming well-formed/consistent transactions seeking isolation
  • Simple locking: hold a (shared) lock to read and exclusive lock to write
  • Simple locking fails to provide isolation if transactions interleave mutation/locking

• 2PL: acquire no lock after releasing any
  • Strict 2PL: release no lock before committing, avoids cascading aborts
  • Locks held a long time increase blocking; decrease concurrency

• Optimistic methods don’t lock but may abort and retry
  • Record all variables touched and check for conflicts on commit
  • Faster if conflict is rare, but risks livelock if not
Failure types

- How would the faulty component behave?
  - Byzantine behavior: faulty component may send arbitrary messages
  - Fail-stop behavior: faulty component stops and does not send

- Examples of fail-stop failures
  - Media failure: IO device code bugs, disk HW failures
  - System failure: DB bug, OS fault, HW failure

- Transaction failure
  - Code aborts, based on input/database inconsistency
    [sometimes programmer is just escaping complex corner cases in code]
  - Mechanical aborts caused by concurrency control solutions to isolation
  - Frequent events, “instant” recovery needed

Recoverable Database System Model

- Log changes durably before database changes durable
  - Write-ahead logging
  - Once a committed transaction has been logged separately, multiple changes to database can be serialized (retry will “REDO” work if needed)

- REDO: repeat completed transaction on old DB data
  - Partial system or total media failure

- UNDO: rollback aborted transaction
  - Transaction or system failure
  - Only if uncommitted transaction allowed to change durable media
Replicated State Machines

• **State machine:**
  code + data + input command = deterministic output

• Assuming non-faulty replicas:
  same initial state + same input = same final state + same output
  • Fail-stop failures: 1 surviving replica is sufficient
  • Byzantine failures: non-faulty survivors must win a vote
    • Need 2t+1 replicas to survive t malicious (Byzantine) failures

• Common tools for replicated state machines
  • Part-time parliament or PAXOS [Lamport89], ZooKeeper, RAFT, ...

Agreement and Ordering

• Decompose Replicated State Machine protocol into two:
  • Agreement – deliver every request to all non-faulty machines
  • Ordering – ensure the same order of execution at all non-faulty machines
Agreement and Ordering

- Decompose Replicated State Machine protocol into two:
  - Agreement – deliver every request to all non-faulty machines
    - A coordinator/client specifies a request and the rest agree
  - Ordering – ensure the same order of execution at all non-faulty machines
    - Assign identifiers to requests and execute in identifier order
    - Use a clock – three kinds:
      - Logical clock: Client sends a logical clock with every message
      - Real-time clock: Every machine has and sends real-time clocks
      - Replica-generated clock: Replicas negotiate a clock/identifier for order

Concurrence and “Happens Before”

- Two events are not concurrent if one “happens before” the other
- E.g., P1 happens before R3, but P2 and R4 may be concurrent
- Replicated state machine wants same order of changes at all replicas
Logical clocks

- Every machine maintains a counter for its (orderable) events
- A message arrives with sender’s counter: receiver advances counter past the sender’s
  - $C' = \max(C, \text{msg}-C) + 1$
  - Resolve ties by adding machine/thread ID as lower order bits
- Define a total order that is consistent with “happen before”

Replicated State Machines + Logical Clocks

- **Problem:** To decide what request to execute next, we need to know no request with a lower logical clock may arrive in future
- Require messages between two machines **arrive in order** (e.g., TPC)
- **Delay execution** at a replica until it has seen a larger logical clock from all non-faulty machines
  - Requests being held all happened before the latest messages, so a smallest identifier can be selected and executed (following total order)
- **New problem:** waiting for later messages is undesirable
  - Forces heartbeat messages, and significant latency
  - Real-time clocks can fix this iff clock skew < message delivery
  - Replicas can negotiate an order by communicating among themselves
    - At the cost of extra messaging
Designing for disasters [Keaton04]

• Optimization framework to auto-design data dependability solutions
  • Data dependability metrics: reliability, availability, failure consequences
  • Models of data protection and recovery alternatives

Modeled protection techniques

• Primary copy protected by one or more secondary copies
• Data protection and recovery techniques modeled
  • Remote mirroring: sync, async, async with batching
  • Tape backup and vaulting
  • Failover vs. reconstruction
  • Resource sparing: hot vs. unconfigured, dedicated vs. shared
Next up

• Geo-replication!

Readings


• Optional: “Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?”, B. Schroeder, G. A. Gibson. USENIX Conference on File and Storage Technologies (FAST), 2007.
Annual replacement rate (ARR)

- Datasheet MTTFs are 1,000,000 to 1,500,000 hours
- Expected annual replacement rate: 0.58 – 0.88%
  - Vendor sees “no fault found” for 43% of returns
  - Customer breaks warranty: heat, moisture, vibration, workload, ...

System failure rate highly variable

- 4096 procs
- 1024 nodes
- 6152 procs
- 49 nodes
- 128 procs
- 32 nodes
- 2001
- 2003
- 1996
- 2004
Best model: failures track # of processor chips

![Graph showing failures per year per chip for different processor models.](image)