Availability, Reliability, and Fault Tolerance

Guest Lecture for Software Systems Security

Tim Wood
Distributed Systems have Problems

- Hardware breaks
- Software is buggy
- People are jerks

- But software is increasingly important
  - Runs nuclear reactors
  - Controls engines and wings on a passenger jet
  - Runs your facebook wall!

- How can we make these systems more reliable?
  - Particularly large distributed systems
Inside a Data Center

- Giant warehouse filled with:
- Racks of servers
- Disk arrays
- Cooling infrastructure
- Power converters
- Backup generators
Modular Data Center

• ...or use shipping containers

• Each container filled with thousands of servers

• Can easily add new containers
  • “Plug and play”
  • Just add electricity

• Allows data center to be easily expanded

• Pre-assembled, cheaper
Definitions

• **Availability**: whether the system is ready to use at a particular time
• **Reliability**: whether the system can run continuously without failure
• **Safety**: whether a disaster happens if the system fails to run correctly at some point
• **Maintainability**: how easily a system can be repaired after failure
Availability and Reliability

- System 1: crashes for 1 millisecond every hour

- System 2: never crashes, but has to be shutdown two weeks a year
Availability and Reliability

• System 1: crashes for 1 millisecond every hour
  • Better than 99.9999% availability
  • Not very good reliability...

• System 2: never crashes, but has to be shutdown two weeks a year
  • "Perfectly" reliable
  • Only 96% availability

Is one more important?
Quantifying Reliability

- **MTTF**: Mean Time To Failure
  - The average amount of time until a failure occurs

- **MTTR**: Mean Time To Repair
  - The average amount of time to repair after a failure

- **MTBF**: Mean Time Between Failures
Real Failure Rates

- Standard hard drive MTBF
  - 600,000 hours = 68 years!
  - $1/68 = 1.5\%$ chance of failure per year
Real Failure Rates

• Standard hard drive MTBF
  • 600,000 hours = 68 years!
  • 1/68 = 1.5% chance of failure per year

• A big Google data center:
  • Has 200,000+ hard drives
  • 1.5% x 200,000 = 2,921 drive crashes per year
  • or about 8 disk failures per day
Real Failure Rates

• Standard hard drive MTBF
  • 600,000 hours = 68 years!
  • $1/68 = 1.5\%$ chance of failure per year

• A big Google data center:
  • Has 200,000+ hard drives
  • $1.5\% \times 200,000 = 2,921$ drive crashes per year
  • or about 8 disk failures per day

• Failures happen a lot
  • Need to design software to be resilient to all types of hardware failures
  • Actual failure rates are closer to 3% per year
Reliability Challenges

• Typical failures in one year of a google data center:
  • 1000 individual machine failures
  • thousands of hard drive failures
  • 1 PDU (Power Distribution Unit) failure (about 500-1000 machines suddenly disappear, budget 6 hours to come back)
  • 1 rack-reorganization (You have plenty of warning: 500-1000 machines powered down, about 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% packet loss)
  • 8 network maintenances (4 might cause ~30-minute random connectivity losses)
  • 12 router reloads (takes out DNS and external virtual IP address (VIPS) for a couple minutes)
  • 3 router failures (have to immediately pull traffic for an hour)
  • 0.5% overheat (power down most machines in under five minutes, expect 1-2 days to recover)
  • dozens of minor 30-second blips for DNS

Types of Failures

• Systems can fail in different ways

• How?
Types of Failures

• Systems can fail in different ways
  • Crash failure
  • Timing failure
  • Content failure
  • Malicious failure

• Are some easier to deal with than others?
Fault Tolerance through Replication

- We can handle failures through redundancy

- Have multiple replicas run the program
  - May want to keep them away from each other
  - May want to use different hardware platforms
  - May want to use different software implementations
Fault Detection

• Detecting a fault can be difficult
  • Crash failure
  • Timing failure
  • Content failure
  • Malicious failure
Fault Detection

• Detecting a fault can be difficult
  • Crash failure
  • Timing failure
  • Content failure
  • Malicious failure

• Approaches:
  • Heartbeat messages
  • Adaptive timeouts
  • Voting / Quorums
  • Authentication / signatures
Detection is Hard

• Or maybe even impossible

• How long should we set a timeout?

• How do we know heart beat messages will go through?
Two Generals Problem

• The Ninja general and the Pirate general need to coordinate an attack
• Can (try to) send messengers back and forth
  • Messengers can be shot
• How can they guarantee they will both attack at the same time?
Two Generals Problem

• We need to worry about physical characteristics when we build systems
• Packets can be lost, delayed, reordered
• Disks can be slow, fail, or crash

• Or things can be actively malicious
  • Big trouble...

• What kinds of assumptions do we need to make?
  • Network is ordered and reliable, but may be slow
  • We can quantify the expected number of nodes that will fail
Fault Tolerance through Replication

• How to tolerate a crash failure?

• How to tolerate a content failure?

• How many replicas to tolerate $f$ such failures?
Fault Tolerance through Replication

• How to tolerate a **crash** failure?

![Diagram](image)

- Inputs: P1, P2, P3
- Output: 2+2=4
- f+1 replicas

• How to tolerate a **content** failure?

![Diagram](image)

- Inputs: P1, P2, P3
- Output: 2+2=4
- Majority Voter
- 2f+1 replicas

19
Agreement without Voters

• We can't always assume there is a perfectly correct voter to validate the answers

• Better: Have replicas reach **agreement** amongst themselves about what to do
  • Exchange calculated value and have each node pick winner

<table>
<thead>
<tr>
<th>Replica</th>
<th>Receives</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4, 4, 5</td>
<td>= 4</td>
</tr>
<tr>
<td>B</td>
<td>4, 4, 5</td>
<td>= 4</td>
</tr>
<tr>
<td>C</td>
<td>4, 4, 5</td>
<td>= 4?</td>
</tr>
</tbody>
</table>
Byzantine Generals Problem

- There are $N$ generals making plans for an attack.
- They need to decide whether to Attack or Retreat.
- Send your vote to everyone (0=retreat, 1=attack).
  - But $f$ generals may be traitors that lie and collude.
- Can all correct replicas agree on what to do?
  - Take majority vote of planned actions.

<table>
<thead>
<tr>
<th>Replica</th>
<th>Receives</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1, 0, 1</td>
<td>Attack!</td>
</tr>
<tr>
<td>B</td>
<td>1, 0, 0</td>
<td>Retreat!</td>
</tr>
<tr>
<td>C</td>
<td>1, 0, ?</td>
<td>???</td>
</tr>
</tbody>
</table>

Majority voting doesn't work if a replica lies!
Byzantine Generals Solved!

- Need more replicas to reach consensus
  - Requires $3f+1$ replicas to tolerate $f$ byzantine faults
- Step 1: Send your plan to everyone
- Step 2: Send learned plans to everyone
- Step 3: Use majority of each column

<table>
<thead>
<tr>
<th>Replica</th>
<th>Receives</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(1,0,1,1)</td>
<td>A: 1</td>
</tr>
<tr>
<td></td>
<td>(1,0,0,1)</td>
<td>B: 0</td>
</tr>
<tr>
<td></td>
<td>(1,1,1,1)</td>
<td>C: 1</td>
</tr>
<tr>
<td></td>
<td>(1,0,1,1)</td>
<td>D: 1</td>
</tr>
<tr>
<td>B</td>
<td>(1,0,1,1)</td>
<td>A: 1</td>
</tr>
<tr>
<td></td>
<td>(1,0,0,1)</td>
<td>B: 0</td>
</tr>
<tr>
<td></td>
<td>(0,0,0,0)</td>
<td>C: 0</td>
</tr>
<tr>
<td></td>
<td>(1,0,0,1)</td>
<td>D: 1</td>
</tr>
</tbody>
</table>
Can we make this any easier?

• Fundamental challenge in BFT:
  • Nodes can misrepresent what they heard from other nodes

• How can we fix this?

• Have nodes **sign** messages!
  • Then liars can't forge messages with false information
Can we make this any easier?

• Fundamental challenge in BFT:
  • Nodes can misrepresent what they heard from other nodes

• How can we fix this?

• Have nodes **sign** messages!
  • Then liars can't forge messages with false information

• Crypto actually is useful!
Denial of Service

• Attack to reduce the *availability* of a service
  • Can also cause crashes if software is poorly written

• "Unsophisticated but effective"
  • Flood target with traffic
  • No easy way to differentiate between a valid request and an attacker
Sept 2012 DDoS

• Six US banks attacked
  • Attacks were announced in advance
  • Banks still could not prevent the damage
  • Attackers sent 65 gigabytes of data per second

• Iranian "Cyber Fighter" group claimed responsibility
  • Encouraged members to use Low Orbit Ion Cannon software to flood banks with traffic
  • Also used botnets as a traffic source
Sept 2012 DDoS

• But it's not clear if that was the real source...

• Botnet machines have relatively low bandwidth
  • Would need 65,000+ compromised machines

• Most traffic to the banks was coming from about 200 IP addresses
  • Appear to be a small set of compromised high powered web servers

• Not clear if Iranian hacker group did all of this or if some other group was the mastermind
  • Iranian government fighting against sanctions?
  • Eastern European crime groups make fraudulent purchases and then disrupt bank web activity long enough for them to go through
Anonymous (?)

• The Anonymous "hacktivist" group has used DDoS for various political causes
  • Members run LOIC software and target a specific site
  • "Volunteer bot net"

• But be careful...

• In March 2012 the LOIC software had a trojan
  • Ran a DDoS on the enemy...
  • And stole your bank and gmail account info
Defending against DDoS

- Some DDoS traffic can be easily distinguished
  - Most web apps can safely ignore ICMP and UDP traffic

- But performance impact will depend where filtering is performed
  - Firewall on server being attacked may limit impact on application, but still clogs network
  - Firewall at ISP is much better, but may be under someone else's control
Summary

- Software systems must worry about:
  - Hardware and software failures
  - Service availability
  - Malicious attacks that affect reliability and/or availability

- Approaches:
  - Redundancy
  - Fault mitigation
  - Fault detection