Fault Tolerance, Replication, and Consistency
Motivation: Hadoop Cluster.
Motivation: Hadoop Cluster

Mostly retired desktops

Intel Core 2: launched in 2008

Support is gathering old servers
Motivation: Hadoop Cluster

Mostly retired desktops

Intel Core 2: launched in 2008

Support is gathering old servers

Test case for fault tolerance!
Fault Tolerance

In any sufficiently large cluster, machines will fail.
In any sufficiently large job, machines will fail.
Defining Failure

Crashed: Node disappeared
Defining Failure

**Crashed:** Node disappeared

**Slow:** Too many students logged in, ...
Defining Failure

**Crashed:** Node disappeared

**Slow:** Too many students logged in, . . .

**Omission:** Drops a request
   Hard drive bad sector $\Rightarrow$ drops request for that file
   Intermittent network cable
Defining Failure

**Crashed:** Node disappeared

**Slow:** Too many students logged in, . . .

**Omission:** Drops a request
- Hard drive bad sector $\implies$ drops request for that file
- Intermittent network cable

**Wrong:** Returns bad data/does not follow protocol
- Defective RAM
- Undetected disk errors
- Wrong software version

**Byzantine:** Many untrustworthy nodes, worst-case behavior

**Hacked** Volunteer nodes (Tor, BitTorrent, Bitcoin)
Defining Failure

**Crashed:** Node disappeared

**Slow:** Too many students logged in, ... 

**Omission:** Drops a request
- Hard drive bad sector $\implies$ drops request for that file
- Intermittent network cable

**Wrong:** Returns bad data/does not follow protocol
- Defective RAM
- Undetected disk errors
- Wrong software version

**Byzantine:** Many untrustworthy nodes, worst-case behavior
- Hacked
- Volunteer nodes (Tor, BitTorrent, Bitcoin)
Failure: An Outline

1. Timeouts
2. Replication
3. Consistency
4. Consensus
5. Recovery
## Timeouts and Health Reports

Detects *crashed* and possibly *slow* nodes. A node might *omit* specific requests, but pass health.

<table>
<thead>
<tr>
<th>Node HTTP Address</th>
<th>Last health-update</th>
</tr>
</thead>
<tbody>
<tr>
<td>faulks.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:27:09 +0100 2015</td>
</tr>
<tr>
<td>tesserini.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:27:13 +0100 2015</td>
</tr>
<tr>
<td>blundell.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:27:08 +0100 2015</td>
</tr>
<tr>
<td>dancla.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:27:09 +0100 2015</td>
</tr>
<tr>
<td>bw1425n01.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:26:17 +0100 2015</td>
</tr>
<tr>
<td>glendora.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:27:09 +0100 2015</td>
</tr>
<tr>
<td>strathisla.inf.ed.ac.uk:8042</td>
<td>Thu Oct 08 09:27:12 +0100 2015</td>
</tr>
</tbody>
</table>
So A Node Times Out

Mark the node offline, ask another?
So A Node Times Out

Mark the node offline, ask another?

“on Sunday morning, a portion of the metadata service responses exceeded the retrieval and transmission time allowed by storage servers.” –Amazon AWS outage
So A Node Times Out

Mark the node offline, ask another?

“on Sunday morning, a portion of the metadata service responses exceeded the retrieval and transmission time allowed by storage servers.” –Amazon AWS outage

Service is loaded
→
Timeouts
→
Nodes marked offline
→
More load on remaining servers
→
Repeat.
Avoid cascading failure: drop incoming requests.

Twitter is over capacity.
Avoid cascading failure:

- Capacity planning!
- Rate-limit machine failure
- Heuristics for small failures can backfire in larger failures
Replication

Store several copies of the same data!
In HDFS: 3 copies by default.

Read from any copy $\implies$ better read performance.
Replicas for Fault Tolerance

Crashed, slow, or omission: read from another replica
Replicas for Fault Tolerance

Crashed, slow, or omission: read from another replica
Wrong: checksums on server side or client side, try another

BitTorrent: checksums in torrent file
Replicas for Fault Tolerance

Crashed, slow, or omission: read from another replica
Wrong: checksums on server side or client side, try another

BitTorrent: checksums in torrent file

Fine for read-only. What if the data changes?
Consistency?

Web Pages
Stale pages might be fine, but don’t mix old and new in one page. If somebody shares a link, it should work.

Domain Names
Caching with a time limit. Inconsistent answers are ok with time limit.

Banking
Reorder transactions to charge customers the most fees. A transaction succeeds or fails.

E-Commerce
Don’t assign the same seat on a plane (or do . . . )
Consistency?

Web Pages
Stale pages might be fine, but don’t mix old and new in one page. If somebody shares a link, it should work.

Domain Names
Caching with a time limit. Inconsistent answers are ok with time limit.

Banking
Reorder transactions to charge customers the most fees. A transaction succeeds or fails.

E-Commerce
Don’t assign the same seat on a plane (or do...)

Consistency needs depend on the application!
Models for Consistency

**Strict**: Absolute ordering of all accesses by time

**Linearisability**: There exists some linear story (like a bank statement)

**Sequential**: Nodes read in a consistent order
### Example

<table>
<thead>
<tr>
<th>Time 1</th>
<th>Time 2</th>
<th>Time 3</th>
<th>Time 4</th>
<th>Time 5</th>
<th>Time 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writes A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Writes B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reads B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reads A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reads B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reads A</td>
</tr>
</tbody>
</table>

- **X** Strict
- ✓ Linearisable
- ✓ Sequential: Carol and Dan saw the same order.
Example

<table>
<thead>
<tr>
<th>Time 1</th>
<th>Time 2</th>
<th>Time 3</th>
<th>Time 4</th>
<th>Time 5</th>
<th>Time 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Writes A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>Writes B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
<td>Reads B</td>
<td>Reads A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td></td>
<td></td>
<td>Reads B</td>
<td>Reads A</td>
<td></td>
</tr>
<tr>
<td>Eve</td>
<td></td>
<td></td>
<td>Reads A</td>
<td>Reads B</td>
<td></td>
</tr>
</tbody>
</table>

× Strict
× Linearisabile
× Sequential: Eve saw a different order.
Models for Consistency

**Strict:** Absolute ordering of all accesses by time

**Linearisability:** There exists some linear story (like a bank statement)

**Sequential:** Nodes read in a consistent order

**Causal:** Causally related events are ordered correctly

**FIFO:** Writes from same node are ordered consistently
   - But writes from different nodes can be inconsistently ordered
Explicit Consistency Options (sync)

- **Weak**: Only when programmer says so
- **Entry**: When a lock is acquired
- **Release**: When a lock is released
Eventual Consistency

Update one replica, let the others update lazily.

Some algorithms guarantee consistency eventually, despite some failures.
Consistency: Two Generals Problem

Two generals leading armies on opposite sides of a city. Need to both attack or both retreat. Only communication is messengers, who might be captured.
Consistency: Two Generals Problem

Two generals leading armies on opposite sides of a city. Need to both attack or both retreat. Only communication is messengers, who might be captured.

Theorem: no protocol ensures consensus.
Byzantine Generals Problem

Multiple generals, majority vote: message exchange has to be $3x$ number of lost messages.

Byzantine Fault Tolerance: need $3m + 1$ nodes to agree on a bit if $m$ nodes are faulty.

Want more/proof? Take distributed systems!
CAP Theorem: Consistency, Availability, Partition tolerance

- **Consistency**: Nodes see same data at the same time
- **Availability**: Node failures do not prevent system operation
- **Partition Tolerance**: Network failures do not prevent system operation

Conjecture: pick two of the above.
Related theorem for a special case.
Recovery

Something failed, now what?

Backward Recovery
Checkpointing: return to previous. Can be expensive to store.
Packet retransmission (when client does not ACK).

Forward recovery
Plan for some loss e.g. error correcting codes

Backward recovery is more common.
Fail! Summary

Ways to fail
Ways to be consistent
Redundancy by replicas or recomputing