Byzantine Failures

Failure Models

- Showed up in Partial Synchrony

- Fail stop: Failed process takes no further steps

- Omission: Some messages might never be processed

[Aside: DiSkstra Prize This Year

Impossibility of consensus in synch. model under enough omissions]

Today's Focus!!!

- Byzantine [Lamport's Oral Messages]

- Byzantine w/ [Lamport's Signed Messages] Authentication

Why care about Byzantine Failures

- Systems where some participants are malicious
  - PBFT. Malicious clients? Not our care
Cryptocurrency? focus today.

PBFT handling of malicious clients

2 Bugs

- Fail stop (and # of likely failures) are assumptions.
  But these assumptions might not hold
  
  - Failed process might not stop sending msgs
  
  - Network might corrupt messages, replay previously sent messages, etc.

- Byzantine makes fewer assumptions
  - Safe (at hopefully live) even under arbitrary behavior

Where

- Space (the final frontier)
  - Airplanes, (Boeing 777, 787 flight control

[The Real Byzantine Generals - Driscoll et al., 2004]
- Submarines
- Some cloud services

What faulty nodes can do under Byzantine failures

- Pretty much anything
  - But visibly
    - Not send messages
    - Send malformed messages
  - Equivocate

![Diagram: I ❤️ Pizza. Pizza Is The Worse Food]

- Send messages to a subset of processes

Desirable Properties

- Agreement: All non-faulty nodes agree on value (condition $1/|I^c|)$
- Validity is a bit weird:
  - Assume single proposer [Leader]
    If proposer is correct =>
    Value that correct nodes agree on is proposed value
  - Alto Weak Unanimity [Past Synchrony]
    No Byz. failure =>
    Validity from fail-stop, etc.

- Termination / Liveness
  Of particular importance in BFT
  Want to ensure that Byz. processes cannot impede progress.

Authentication vs Not

[Note, some papers also refer to this as transferable auth]

Core Problem

$i$ loves Pizza $\rightarrow$ 😁
Auth \Rightarrow Can prove who sent a message.

How? Cryptography

\[ f_{\text{sig}_A}(m) \rightarrow \mathcal{B} \]

\[ f_{\text{ver}_A}(\mathcal{B}) \rightarrow \text{Yes on N} : \text{Public} \]

Hardness assumption:

Guessing/computing \( f_{\text{sig}_A}(m) \) impossible for anyone other than A
Lamport: Core result:

Without authentication, need \( \geq 3f+1 \) nodes to tolerate \( f \) failures

Q1. Async OR Partially Synchronous OR Synchrony?

Why? Start with 3 nodes, possibly 1 faulty
Focus on Node C

- Agreement: All honest nodes agree.

\( B \& C \)

\( \rightarrow \) Case P: B & C must agree / communicate

- Validity: If Prop is honest, must accept its proposal

Situations appear indistinguishable to C.

Aside: Why does auth help?
OM(n): Protocol for agreement in Byz. w/o authentication w/ up to n failures.

- Core idea: Reach agreement on what was proposed
  Agreement how? Majority vote.

OM(0)

No failures, hence safe!

OM(1)
Does this work?

- **Validity**: if \( p \) is honest
  \[
  m_1 = m_2 = m_3 = m
  \]
  since at most one of \( A, B, C \) dishonest

  \( m \) is majority

- **Agreement**: Easier: Pick deterministically if no majority
- Termination?

- $OM(2) \ldots OM(n)$

Simple induction proof

<table>
<thead>
<tr>
<th>$om(2)$</th>
<th>P</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>$m_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>$m_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>$m_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Moving on to partial synch/asynch.

- Quorums for fail stop
  - Assume $f$ failures
  - At any step
    - No failures so far
    - Safety: Enough nodes to ensure correctness
ensure at least $f$ correct node is involved

$\Rightarrow f+1$ participants

$\Rightarrow f$ failures so far

**Liveness:** Must have enough nodes to ensure progress.

Need $n = f + f+1$ participants = $2f+1$ participants

- Quorums for BFT: Up to $f$ nodes
  - A failed node can participate or not.

**Safety:** Failed nodes should not be able to decide next protocol state.

Decision requires $f + f+1$ nodes

[correct nodes always outvote faulty ones]

**Liveness:** Must make progress if failed nodes do not participate.
as far participate

\[ N \geq f + 2f + 1 = 3f + 1 \]

PBFT

- System assumptions
  - Partial synchrony
  - Authentication [kind of]

- Core technique: Quorum intersection

  \( \rightarrow \) Multipaxos/Viewstamps replication like

  \( \rightarrow \) Differences for BFT

Differences

- Larger quorum

- Leader is less special

  \( \rightarrow \) Decides what to propose

  \( \rightarrow \) Protocol is symmetric otherwise

- Changing leader requires \( 2f + 1 \) nodes to suspect leader

  [Leader Stability]
- Gather proof that commands are committed [safety]
- Can trigger leadership change [liveness]

Replication

Client request

Prepare

- `<c>δ`
- `<View (Term), n (slot), d> A`
- Hash of `<c>δ`

Prepare

`<view, n, d, B> B`
On receiving $\geq 2f$

Prepare for a prev. pre prepared command: Commit

$\langle \text{view, n, d, c} \rangle_c$

On receiving $\geq 2f$

Commit for command that we have seen $\geq f$

Prepare messages for Respond to client.

Client counts and decides.

**Leader Election/View Change**

- Goal: Faulty nodes should not disrupt correct leader

**How:**

Current view (term) remains active until $2f+1$ nodes vote to move to new view

- Goal: Faulty leaders should not be able to monopolize leadership

**How:** Associate views with leaders
Beyond this, view change is Multi Paxos like with the new leader merging logs from the 2f nodes proposing view change.

Client Triggered View Change

Clients can notify other nodes about commands which have not received responses.

- If node has not seen command it forwards it to leader
- Node sets a timer.

PBFT in Practice

- Not widely used
- Performance
- # of machines
- Tradeoff
  - effect of bugs
- costs