EECS 591
DISTRIBUTED SYSTEMS

Manos Kapritsos
Winter 2018
BYZANTINE FAULT TOLERANCE
A HIERARCHY OF FAILURE MODELS

- Fail-stop
- Crash
- Send omission
- Receive omission
- General omission
- Arbitrary (Byzantine) failures

○ = benign failures
What are Byzantine Failures

The short answer: they can be *anything!*
(they can even be crash/omission failures)

Examples of commission failures

- A bit flip in memory
  - Manufacturing defect
  - Alpha particles
- Network card malfunction
- Intentional behavior
  - Rational node: trying to game the system for personal gain
  - Malicious node: trying to bring the system down
The Byzantine Generals

- Synchronous communication
- One general may be a traitor
The Byzantine Generals

- Synchronous communication
- One general may be a traitor
- One of the generals is the commander C
  - The commander decides Attack or Retreat

Goals

1. If C is trustworthy, every trustworthy general must follow C’s orders
2. Every trustworthy general must follow the same battle plan
Remember when things were simpler?
YOU CAN’T TRUST ANYONE THESE DAYS...
You can’t trust anyone these days…

He said “retreat”

He said “attack”
You can’t trust anyone these days...

C

G1

G2

He said “retreat”

He said “attack”

Attack

Retreat

C

G1

G2

He said “retreat”

Attack

Attack
“But they were all of them deceived...”
A LOWER BOUND

Theorem
There is not algorithm that solves TRB for Byzantine failures if $n \leq 3f$

Lamport, Shostak and Pease, The Byzantine Generals Problem, 1982
Research project

- Email Boyu and me by tonight
  - project topic
  - team membership
- 2-page proposal by Friday, 02/23
  - Motivation
  - Advancement over state-of-the-art

Presentations

- Start on 02/14
PBFT: A Byzantine Renaissance

Practical Byzantine Fault Tolerance
(Castro, Liskov 1999-2000)

- First practical protocol for asynchronous BFT replication
- Like Paxos, PBFT is safe all the time, and live during periods of synchrony
THE SETUP

**System model**
- Asynchronous system
- Unreliable channels

**Crypto**
- Public/private key pairs
- Signatures
- Collision-resistant hashes

**Service**
- Byzantine clients
- Up to \( f \) Byzantine servers
- \( n = 3f + 1 \) total servers

**System goals**
- Always safe
- Live during periods of synchrony
The general idea

- One primary, 3f replicas
- Execution proceeds as a sequence of **views**
  - A view is a configuration with a well-defined primary
- Client sends signed commands to primary of current view
- Primary assigns sequence number to client’s command
- Primary is responsible for the command eventually being decided
What could possibly go wrong!?

- The primary could be faulty!
  - could ignore commands, assign same sequence number to different requests, skip sequence numbers, etc.
  - Backups monitor primary's behavior and trigger view changes to replace a faulty primary

- Replicas could be faulty!
  - could incorrectly forward commands received by a correct primary
  - any single request may be misleading; need to rely on quorums of requests
  - could send incorrect responses to the client
  - client waits for $f + 1$ matching responses before accepting
Certificates

Protocol steps are justified by certificates
  • Sets (quorums) of signed messages from distinct replicas proving that a property holds

Certificates are of size at least $2f + 1$
  • Any two quorums intersect in at least one correct replica (for safety)
  • There is always a quorum of correct replicas (for liveness)
PBFT: Normal Operation

Three phases:

- **Pre-prepare** assigns sequence number to request
- **Prepare** ensures consistent ordering of requests within views
- **Commit** ensures consistent ordering of requests across views

Each replica maintains the following state:

- Service state
- A *message log* with all messages sent or received
- An integer representing the replica’s current view
CLIENT ISSUES REQUEST

<REQUEST, o, t, c> $\sigma_c$

Primary

Replica 1

Replica 2

Replica 3
**CLIENT issues request**

- Primary

- Replica 1

- Replica 2

- Replica 3

<REQUEST, o, t, c> \( \sigma_c \)

state machine operation
CLIENT ISSUES REQUEST

.timestamp

<REQUEST, o, t, c> \( \sigma_c \)

Primary

Replica 1

Replica 2

Replica 3
Client issues request

<REQUEST, o, t, c>οc

Primary

Replica 1

Replica 2

Replica 3
CLIENT ISSUES REQUEST

<REQUEST, o, t, c> → $\sigma^c$

Primary

Replica 1

Replica 2

Replica 3

client signature
Primary sends $<\langle\text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m>$ to all replicas.
Primary sends $\langle\langle\text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m\rangle$ to all replicas.

- Primary
- Replica 1
- Replica 2
- Replica 3
Primary sends $\langle\langle$ PRE-PREPARE, v, n, d$\rangle_{\sigma_p}, m \rangle$ to all replicas.
Primary sends $<<\text{PRE-PREPARE}, v, n, d>_{\sigma_p}, m>>$ to all replicas

Primary sends <<PRE-PREPARE, v, n, d>_{\sigma_p}, m>> to all replicas
Primary sends \( <<\text{PRE-PREPARE, } v, n, \text{ digest of } m, \sigma_p, m > \) to all replicas.
Primary sends \(<\text{PRE-PREPARE}, v, n, d>_{\sigma_p}, m>\) to all replicas

Correct backup \(k\) accepts PRE-PREPARE if:

- message is well formed
- \(k\) is in view \(v\)
- \(k\) has not accepted another PRE-PREPARE message for \(v, n\) with a different \(d\)
- \(n\) is between two watermarks \(L\) and \(H\) (to prevent sequence number exhaustion)
Primary sends \(<\text{PRE-PREPARE, } v, n, d>^\sigma_p, m>\) to all replicas

Each accepted PRE-PREPARE message is stored in the accepting replica’s message log (including the primary’s)
Replica $k$ sends $\langle \text{PREPARE}, v, n, d, k \rangle_{\sigma_k}$ to all replicas

**Pre-prepare phase**
Replica $k$ sends $<\text{PREPARE}, v, n, d, k>_{\sigma_k}$ to all replicas.

Correct backup $k$ accepts PREPARE if:
- message is well formed
- $k$ is in view $v$
- $n$ is between two watermarks $L$ and $H$
Replica $k$ sends $<\text{PREPARE}, v, n, d, k>\sigma_k$ to all replicas

- Replicas that send a PREPARE accept the assignment of $m$ to sequence number $n$ in view $v$
- Each accepted PREPARE message is stored in the accepting replica’s message log
**PREPARE CERTIFICATE**

- P-Certificates ensure consistent order of requests within views.

- A replica produces a P-Certificate\((m,v,n)\) iff its log holds:
  - the request \(m\)
  - A PRE-PREPARE for \(m\) in view \(v\) with sequence number \(n\)
  - \(2f\) PREPARE from distinct backups that match the PRE-PREPARE

- A P-Certificate\((m,v,n)\) means that a quorum agrees to assign \(m\) to sequence number \(n\) in view \(v\)
  - No two non-faulty replicas with P-Certificate\((m,v,n)\) and P-Certificate\((m',v,n)\)