6.5840: Byzantine Fault Tolerance

Lecture 22

Last week: security

Lecture 20: Fork Consistency → SUNDR

Lecture 21: Decentralized payments → Bitcoin

Last week: security

Lecture 20: Fork Consistency → SUNDR

Lecture 21: Decentralized payments → Bitcoin

- Can we replicate state in an open system?
- Solved problem thought to be impossible!

Last week: security

Lecture 20: Fork Consistency → SUNDR

Lecture 21: Decentralized payments → Bitcoin

- Can we replicate state in an open system?
- Solved problem thought to be impossible!

But, limitations…

- Throughput: ~1K txns/min
- Latency: ~1hr (6-block depth)
- No linearizability guarantee!

Idea: RSM? (Raft)

But Raft has a linearizability guarantee…

- When insufficient connections, wait to recover

Can we implement RSM with malicious replicas?

- Throughput: ~1K txns/min
- Latency: ~1hr (6-block depth)
- No linearizability guarantee!

Can we implement RSM with <u>malicious</u> replicas?

"*Byzantine*"

Practical Byzantine Fault Tolerance (Castro + Liskov ’99)

Can we implement RSM with <u>malicious</u> replicas?

“*Byzantine*”

Practical Byzantine Fault Tolerance (Castro + Liskov ’99)

- “Academic problem” in 1999
- Ancestor of many of today’s cryptocurrency protocols

Can we implement RSM with <u>malicious</u> replicas?

“*Byzantine*”

Aside: About me

- Class project: Implement PBFT
  (6.5840 lab 5 final project)

- Job: Implement BFT protocol at company
  (Algorand, Inc.)

- Ph.D. project: Implement BFT without bugs
  (Formal verification with Frans + Nickolai Zeldovich)

PBFT solves harder problem than Raft

Similar idea: RSM, but with malicious nodes

- Leaders ≈ Primaries
- Terms ≈ Views
- Timeouts

PBFT solves harder problem than Raft

Similar idea: RSM, but with malicious nodes

- Leaders ≈ Primaries
- Terms ≈ Views
- Timeouts

Additional ingredients needed

1. Authenticity of messages
2. More honest nodes
3. Leadership “fairness”

Protocol rules

Goal: RSM under the following assumptions

- Nodes
    - Attacker controls f machines
        - Detail: client honest in paper
    - Cryptography protects messages of honest machines
- Network
    - Attacker can reorder messages
    - Attacker can delay messages for limited time (denial of service)

Protocol rules

Goal: RSM under the following assumptions

Are these realistic?

- Nodes
    - Attacker controls f machines
        - Detail: client honest in paper
    - Cryptography protects messages of honest machines
- Network
    - Attacker can reorder messages
    - Attacker can delay messages for limited time (denial of service)

Let’s build a PBFT

Start with one client

One-client protocol

For consistency and progress:

Can tolerate <N/3 faults! (i.e., N >= 3f+1)

Multiple clients?

Need to elect a *primary*

Problem: what if the primary is bad?

Bad primaries

These make PBFT expensive

Recovery example

Bad primaries impose requirements

- Need to elect a good leader: all-to-all communication
- Prepare messages might be lost: new commit round
- New view messages must be justified: signature *stapling*
  - Op executed ⇒

    Commit received by some node ⇒

    Prepares received by honest majority ⇒

    View-change includes value ⇒

    Future new-views include value ⇒

Additional details

- No value: use special value *null*
- Multiple views: new-view guaranteed to have max view with 2f+1 prepares (ensures no commit missed)
- If timeout wrong, exponential backoff
    - Subtle detail: Can timeout early with f+1 nodes

Extending from op to RSM

- (Like Raft): Primary pipelines many client requests
  - Low- and high-water mark prevent sequence # exhaustion
- Checkpoints allow log compaction (c.f. Raft)
- Clients get f+1 replies

Optimizations

- Hash of values (c.f. Bitcoin)
- Tentative replies
- Read-only operations: don’t need to hit log
- MACs vs. signatures
- Network NACKs

Questions