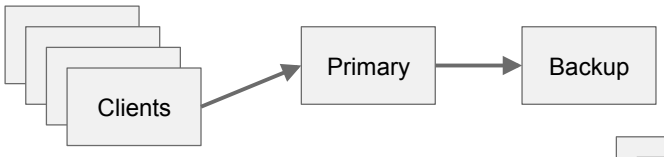
Byzantine Fault Tolerance

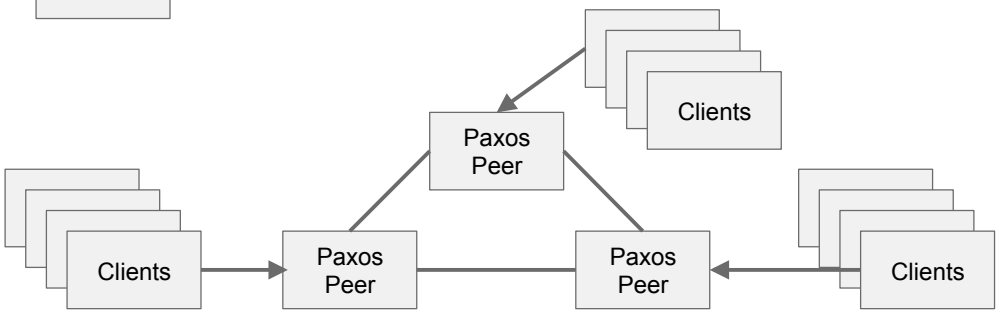
Raymond Cheng
CSE452
ryscheng@cs.washington.edu

Quick Recap

Primary-backup:



Paxos



Failure Model So Far: Fail-stop

Assume servers follow your protocol
e.g. power failures, network failure, network partition

This is hard enough!
e.g. crash vs network failure

Byzantine Generals Problem

“We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement...”

- Lamport, Shostak, and Pease, 1980-2

Byzantine Generals Problem

Byzantine Faults

Buggy servers - potentially computing incorrect results or maliciously modified

Byzantine Agreement

Replicated state machine

Assume 2f+1 of 3f+1 are honest/non-faulty (f are faulty)

Use voting to come to agreement

Paxos Pseudocode

```
proposer(v):
  while not decided:
    choose n, unique and higher than any n seen
so far
    send prepare(n) to all servers including self
    if prepare_ok(n_a, v_a) from majority:
      v' = v_a with highest n_a; choose own v
otherwise
      send accept(n, v') to all
      if accept_ok(n) from majority:
        send decided(v') to all
```

```
acceptor's state:
  n_p (highest prepare seen)
  n_a, v_a (highest accept seen)

acceptor's prepare(n) handler:
  if n > n_p
    n_p = n
    reply prepare_ok(n_a, v_a)
  else
    reply prepare_reject

acceptor's accept(n, v) handler:
  if n >= n_p
    n_p = n
    n_a = n
    v_a = v
    reply accept_ok(n)
  else
    reply accept_reject
```

What can the attacker do?

- Control all faulty nodes (e.g. supply code)
- Aware of faulty node's crypto keys
- Can read all network messages
- Can temporarily force messages to be delayed (e.g. via DoS)

What can't the attacker do?

- Break cryptography primitives

Simple example:

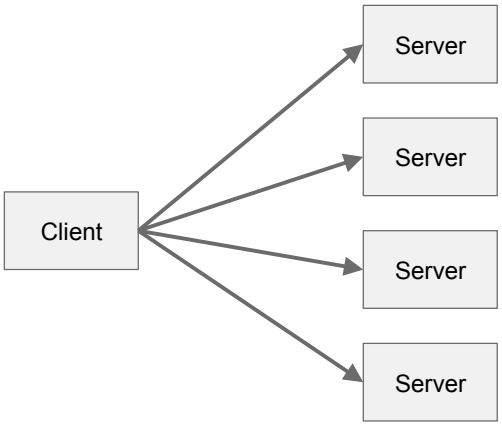
2 clients: Alice & Bob

Alice::
```
echo A > grade
echo B > grade
tell YM "grade file ready"
```
Bob::
```
cat grade
```

a faulty system could:

- totally make up the file contents
- execute write("A") but ignore write("B")
- show "B" to Alice and "A" to Bob
- execute write("B") only only

BFT: Design Attempt 1

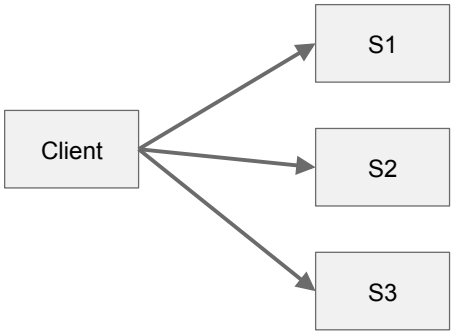


Client sends request to all n servers

Waits for all n servers to reply

Only proceeds if all n agree

BFT: Design Attempt 2

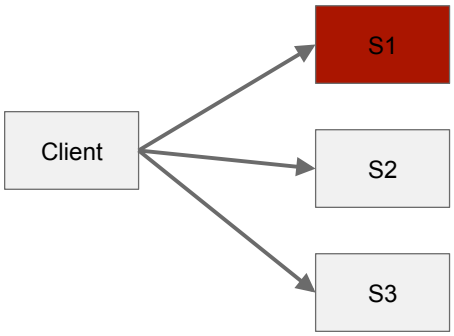


Client sends request to all 2f +1 servers

Assume f are faulty

Waits for f+1 matching replies (majority)

BFT: Design Attempt 2



write("A")

OK from {S1, S2, S3}

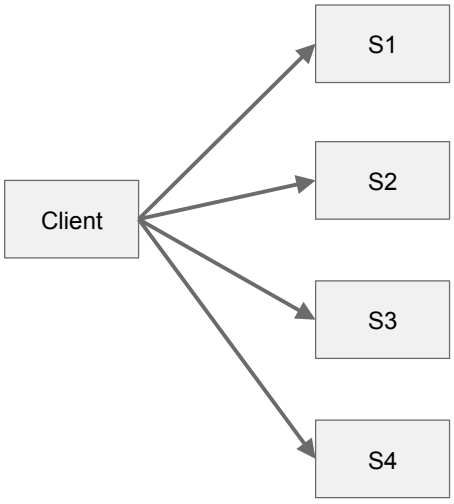
write("B")

OK from {S1, S2}

read()

S1 and S3 replies "A"

BFT: Design Attempt 3

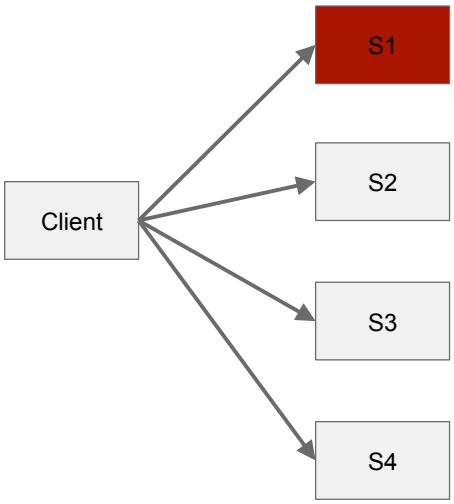


Client sends request to all 3f +1 servers

Assume f are faulty

Waits for 2f+1 matching replies

BFT: Design Attempt 3



write("A")
OK from {S1, S2, S3, S4}

write("B")
OK from {S1, S2, S3}

read()

S1 and S4 replies "A"

S2 and S3 replies "B"

Multiple Clients

Remember: linearizability

- Non-faulty replicas must process operations in same order

Let's introduce a primary

- Picks an order for concurrent clients

Fault primaries can:

send wrong result to client

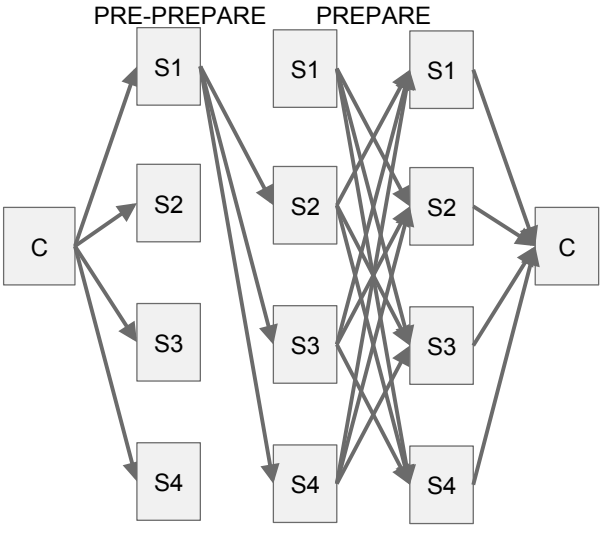
different ops to different replicas

ignore client requests

Handling a faulty primary

- Replicas send results directly to client
- Replicas exchange information about ops sent by primary
- Clients notify replicas of each operation, as well as primary
  - Each replica watches progress of each operation
  - If no progress, force change of primary

BFT: Design Attempt 4



C -> all 3f+1 servers

- One is primary
- Assume f are faulty

Primary chooses next op and n

Primary -> replicas
PRE-PREPARE(op, n)

Replicas broadcast
PREPARE(op, n)

If get matching PREPARE(op, n)

Fault Primary Scenarios

case 1: all good nodes get 2f+1 matching PREPAREs

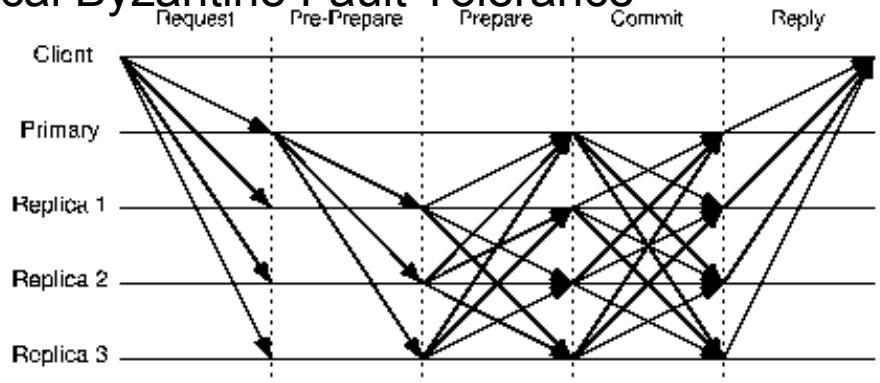
case 2: >= f+1 good nodes get 2f+1 matching PREPARES

case 3: < f+1 good nodes get 2f+1 matching PREPAREs

View Changes: Design Attempt 1

- replicas send VIEW-CHANGE requests to *new* primary
- new primary waits for enough view-change requests
- new primary announces view change w/ NEW-VIEW
  - includes the VIEW-CHANGE requests as proof that enough replicas wanted to change views
- new primary starts numbering operations at:
  (last n it saw) + 1

Practical Byzantine Fault Tolerance



client sends op to all

primary sends PRE-PREPARE(op, n) to all

all send PREPARE(op, n) to all

after replica receives 2f+1 matching PREPARE(op, n)

View Changes

- each replica sends new primary 2f+1 PREPAREs for recent ops
- new primary waits for 2f+1 VIEW-CHANGE requests
- new primary sends NEW-VIEW msg to all replicas with complete set of VIEW-CHANGE msgs
  - list of every op for which some VIEW-CHANGE contained 2f+1 PREPAREs
    i.e. list of final ops from last view

Practical Applications

Peer-to-peer applications (e.g. bitcoin)
Critical systems (e.g. aircraft)