The Byzantine Generals Problem
Leslie Lamport, Robert Shostak, and Marshall Pease
ACM TOPLAS 1982

Practical Byzantine Fault Tolerance
Miguel Castro and Barbara Liskov
OSDI 1999

A definition

- Byzantine (www.m-w.com):
  1: of, relating to, or characteristic of the ancient city of Byzantium
  …
  4b: intricately involved : labyrinthine <rules of Byzantine complexity>

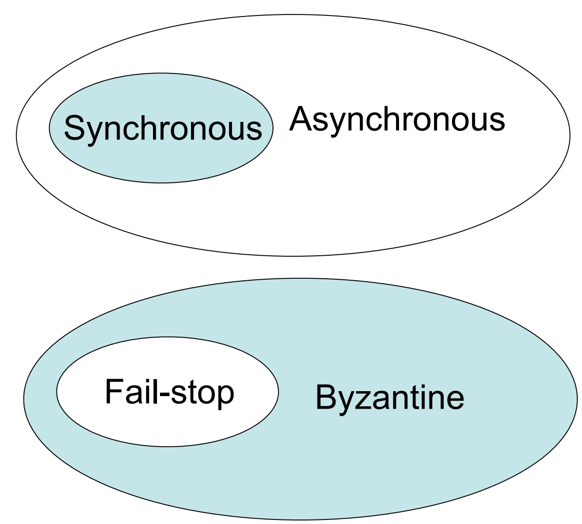
- Lamport's reason:
  "I have long felt that, because it was posed as a cute problem about philosophers seated around a table, Dijkstra's dining philosopher's problem received much more attention than it deserves."
  (http://research.microsoft.com/users/lamport/pubs/pubs.html#byz)

Byzantine Generals Problem

- Concerned with (binary) atomic broadcast
  – All correct nodes receive same value
  – If broadcaster correct, correct nodes receive broadcasted value

- Can use broadcast to build consensus protocols (aka, agreement)
  – Consensus: think Byzantine fault-tolerant (BFT) Paxos

Synchronous, Byzantine world



Cool note

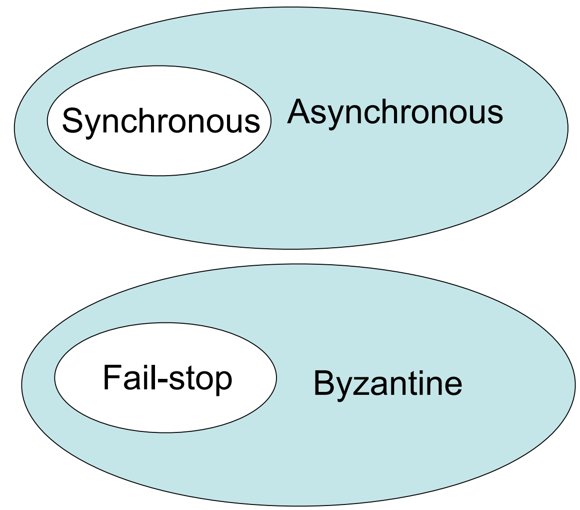
Example Byzantine fault-tolerant system:

⇒ Seawolf submarine's control system

Sims, J. T. 1997. *Redundancy Management Software Services for Seawolf Ship Control System.* In Proceedings of the 27th international Symposium on Fault-Tolerant Computing (FTCS '97) (June 25 - 27, 1997). FTCS. IEEE Computer Society, Washington, DC, 390.

But it remains to be seen if commodity distributed systems are willing to pay to have so many replicas in a system

Practical Byzantine Fault Tolerance: Asynchronous, Byzantine



Practical Byzantine Fault Tolerance

- Why async BFT? BFT:
  - Malicious attacks, software errors
  - Need N-version programming?
  - Faulty client can write garbage data, but can't make system *inconsistent* (violate operational semantics)
- Why async?
  - Faulty network can violate timing assumptions
  - But can also prevent liveness

[For different liveness properties, see, e.g., Cachin, C., Kursawe, K., and Shoup, V. 2000. Random oracles in constantipole: practical asynchronous Byzantine agreement using cryptography (extended abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing* (Portland, Oregon, United States, July 16 - 19, 2000). PODC '00. ACM, New York, NY, 123-132.]

Distributed systems

- Async BFT consensus: Need 3f+1 nodes
  - Sketch of proof: Divide 3f nodes into three groups of f, left, middle, right, where middle f are faulty. When left+middle talk, they must reach consensus (right may be crashed). Same for right+middle. Faulty middle can steer partitions to different values!

[See Bracha, G. and Toueg, S. 1985. Asynchronous consensus and broadcast protocols. J. ACM 32, 4 (Oct. 1985), 824-840.]

- FLP impossibility: Async consensus may not terminate
  - Sketch of proof: System starts in "bivalent" state (may decide 0 or 1). At some point, the system is one message away from deciding on 0 or 1. If that message is delayed, another message may move the system away from deciding.
  - Holds even when servers can only crash (not Byzantine)!
  - Hence, protocol cannot always be live (but there exist randomized BFT variants that are probably live)

[See Fischer, M. J., Lynch, N. A., and Paterson, M. S. 1985. Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (Apr. 1985), 374-382.]

Byzantine fault tolerance

Jinyang Li
With PBFT slides from Liskov

What we've learnt so far: tolerate fail-stop failures

- Traditional RSM tolerates benign failures
  - Node crashes
  - Network partitions
- A RSM w/ 2f+1 replicas can tolerate f simultaneous crashes

Byzantine faults

- Nodes fail arbitrarily
  - Failed node performs incorrect computation
  - Failed nodes collude
- Causes: attacks, software/hardware errors
- Examples:
  - Client asks bank to deposit $100, a Byzantine bank server substracts $100 instead.
  - Client asks file system to store f1="aaa". A Byzantine server returns f1="bbb" to clients.

Strawman defense

- Clients sign inputs.
- Clients verify computation based on signed inputs.
- Example: C stores signed file f1="aaa" with server. C verifies that returned f1 is signed correctly.
- Problems:
  - Byzantine node can return stale/correct computation
    - E.g. Client stores signed f1="aaa" and later stores signed f1="bbb", a Byzantine node can always return f1="aaa".
  - Inefficient: clients have to perform computations!

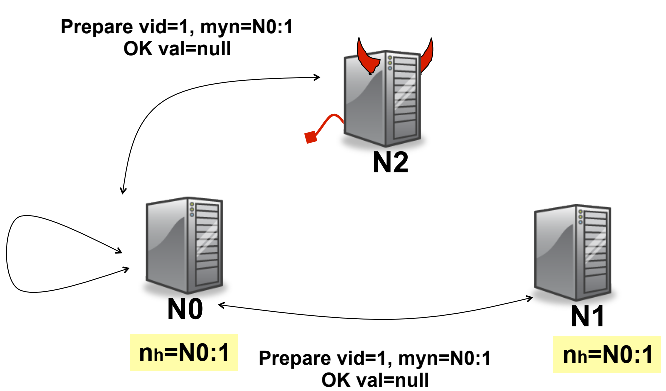
PBFT ideas

- PBFT, “Practical Byzantine Fault Tolerance”, M. Castro and B. Liskov, SOSP 1999
- Replicate service across many nodes
  - Assumption: only a small fraction of nodes are Byzantine
  - Rely on a super-majority of votes to decide on correct computation.
- PBFT property: tolerates <=f failures using a RSM with 3f+1 replicas

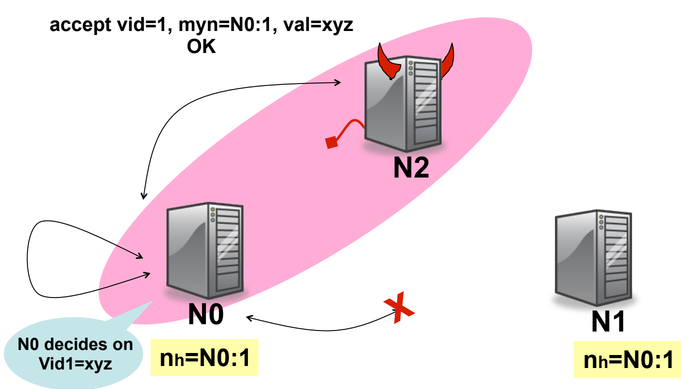
Why doesn’t traditional RSM work with Byzantine nodes?

- Cannot rely on the primary to assign seqno
  - Malicious primary can assign the same seqno to different requests!
- Cannot use Paxos for view change
  - Paxos uses a majority accept-quorum to tolerate f benign faults out of 2f+1 nodes
  - Does the intersection of two quorums always contain one honest node?
  - Bad node tells different things to different quorums!
    - E.g. tell N1 accept=val1 and tell N2 accept=val2

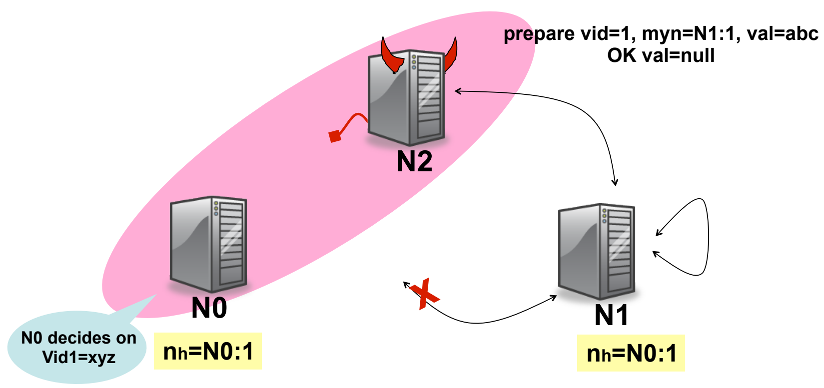
Paxos under Byzantine faults



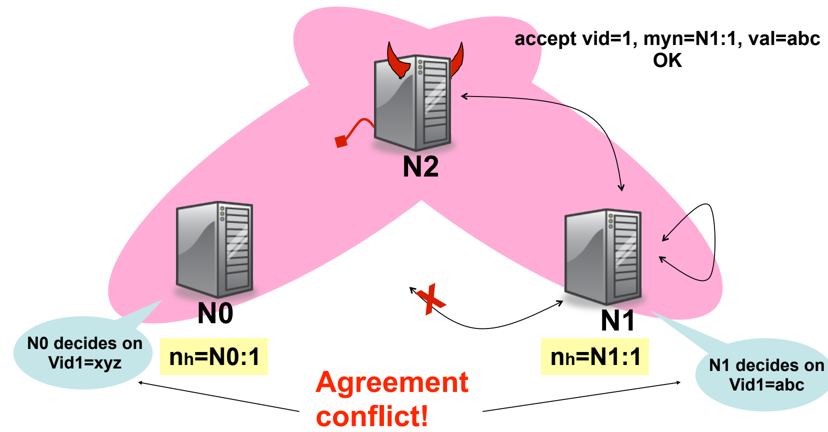
Paxos under Byzantine faults



Paxos under Byzantine faults



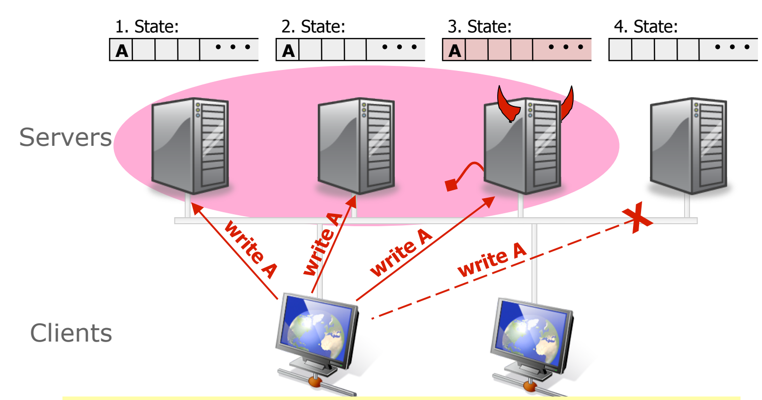
Paxos under Byzantine faults



PBFT main ideas

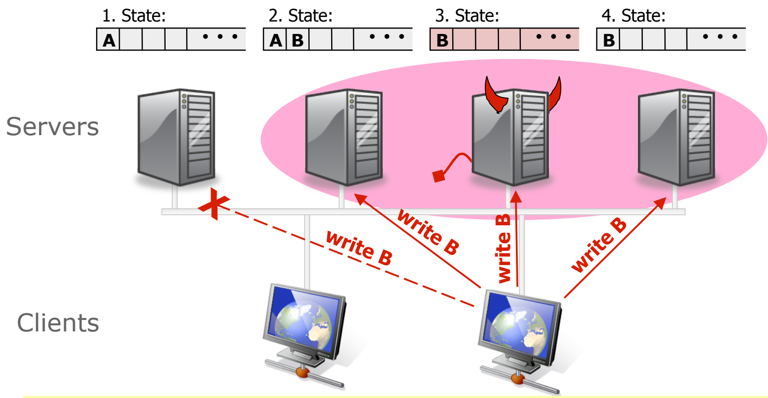
- Static configuration (same 3f+1 nodes)
- To deal with malicious primary
  - Use a 3-phase protocol to agree on sequence number
- To deal with loss of agreement
  - Use a bigger quorum (2f+1 out of 3f+1 nodes)
- Need to authenticate communications

BFT requires a 2f+1 quorum out of 3f+1 nodes



For liveness, the quorum size must be at most N - f

BFT Quorums



For correctness, any two quorums must intersect at least one honest node: (N-f) + (N-f) - N >= f+1 ➡ N >= 3f+1

PBFT Strategy

- Primary runs the protocol in the normal case
- Replicas *watch* the primary and do a view change if it fails

Replica state

- A replica id i (between 0 and N-1)
  - Replica 0, replica 1, …
- A view number v#, initially 0
- Primary is the replica with id
  i = v# mod N
- A log of <op, seq#, status> entries
  - Status = pre-prepared or prepared or committed

Normal Case

- Client sends request to primary
  - or to all

Normal Case

- Primary sends pre-prepare message to all
- Pre-prepare contains <v#,seq#,op>
  - Records operation in log as pre-prepared
  - Keep in mind that primary might be malicious
    - Send different seq# for the same op to different replicas
    - Use a duplicate seq# for op

Normal Case

- Replicas check the pre-prepare and if it is ok:
  - Record operation in log as pre-prepared
  - Send prepare messages to all
  - Prepare contains <i,v#,seq#,op>
- All to all communication

Normal Case:

- Replicas wait for 2f+1 matching prepares
  - Record operation in log as prepared
  - Send commit message to all
  - Commit contains <i,v#,seq#,op>
- What does this stage achieve:
  - All honest nodes that are prepared prepare the same value

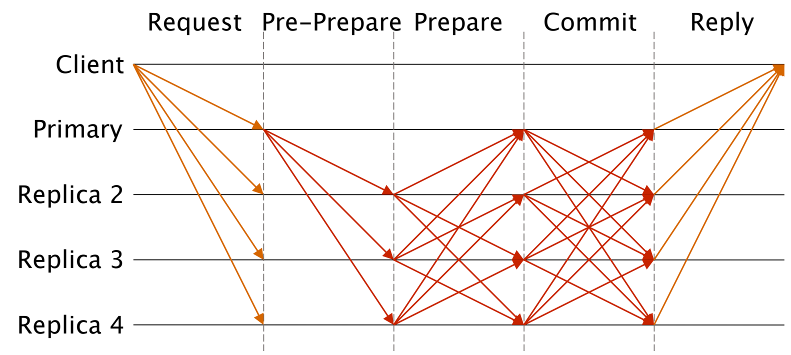
Normal Case:

- Replicas wait for 2f+1 matching commits
  - Record operation in log as committed
  - Execute the operation
  - Send result to the client

Normal Case

- Client waits for f+1 matching replies

BFT



View Change

- Replicas watch the primary
- Request a view change

- Commit point: when 2f+1 replicas have prepared

View Change

- Replicas watch the primary
- Request a view change
  - send a do-viewchange request to all
  - new primary requires 2f+1 requests
  - sends new-view with this certificate
- Rest is similar

Additional Issues

- State transfer
- Checkpoints (garbage collection of the log)
- Selection of the primary
- Timing of view changes

Possible improvements

- Lower latency for writes (4 messages)
  – Replicas respond at prepare
  – Client waits for 2f+1 matching responses
- Fast reads (one round trip)
  – Client sends to all; they respond immediately
  – Client waits for 2f+1 matching responses

Practical limitations of BFTs

- Expensive
- Protection is achieved only when <= f nodes fail
  – Is 1 node more or less secure than 4 nodes?
- Does not prevent many types of attacks:
  – Turn a machine into a botnet node
  – Steal SSNs from servers