The Paxos Algorithm

Based on material by B. Liskov,
B. Lampson and J. Lin

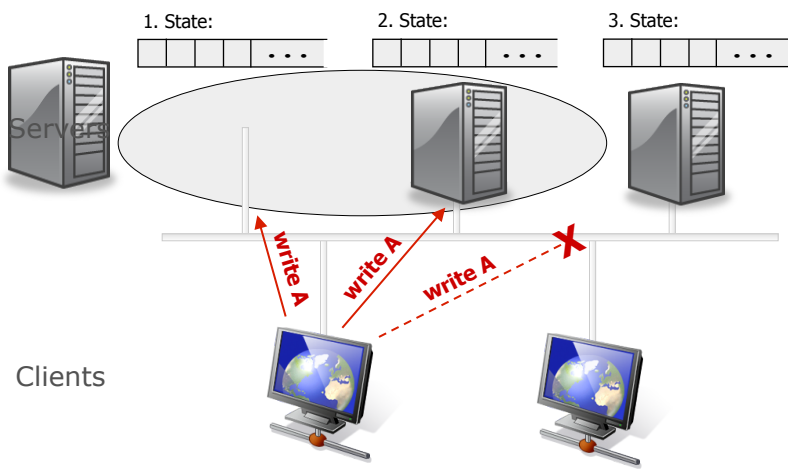
1

Replicated state machine (RSM)

- RSM is a general replication method
- RSM Rules:
  - All replicas start in the same initial state
  - Every replica apply operations in the same order
  - All operations must be deterministic
- All replicas end up in the same state

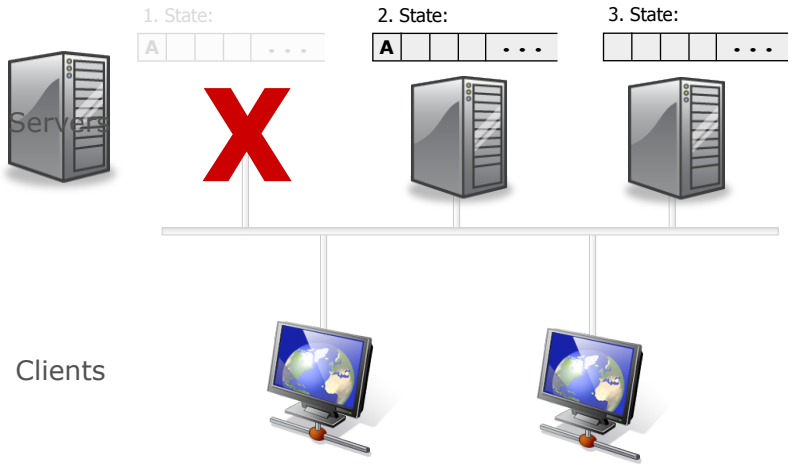
2

Quorums



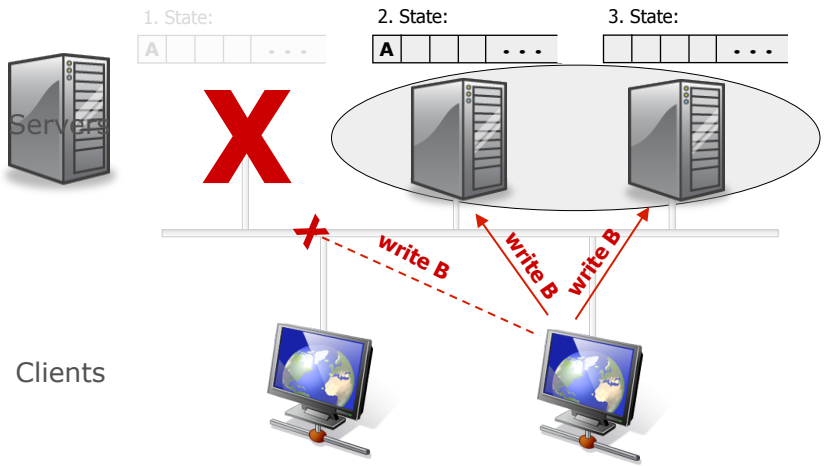
3

Quorums



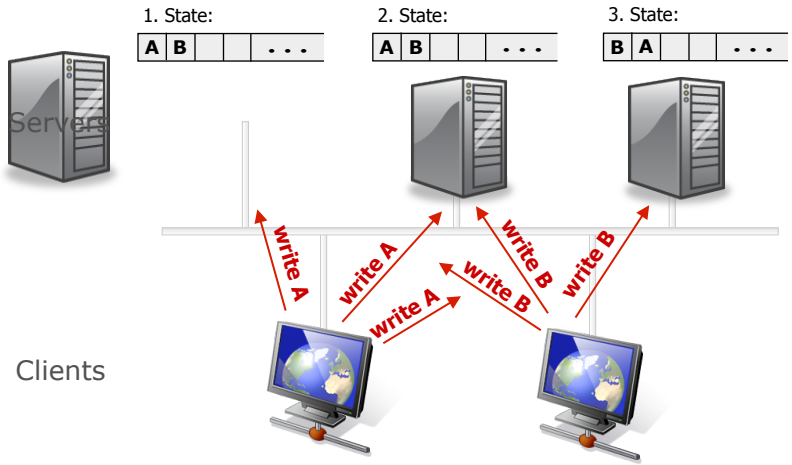
4

Quorums



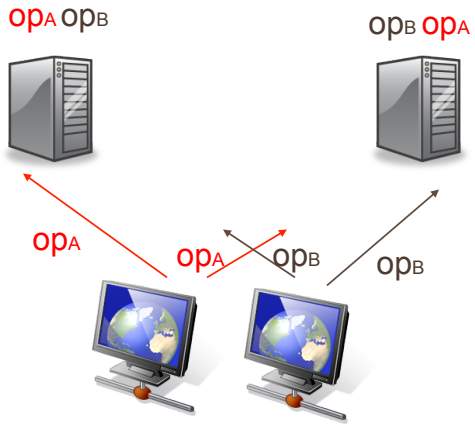
5

Concurrent Operations



6

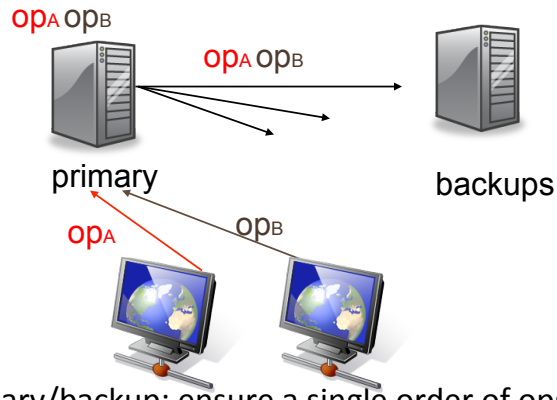
RSM



- How to maintain a single order in the face of concurrent client requests?

7

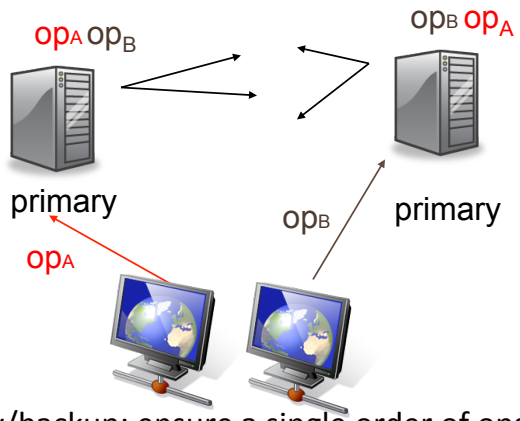
RSM: primary/backup



- Primary/backup: ensure a single order of ops:
  - Primary orders operations
  - Backups execute operations in order

8

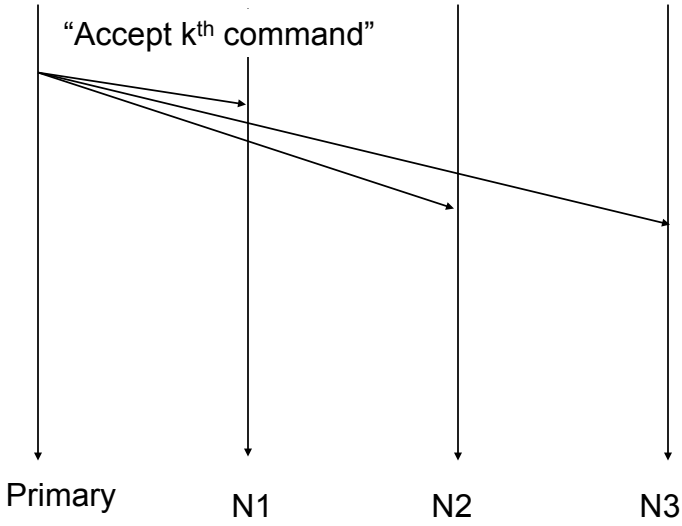
RSM: split-brain



- Primary/backup: ensure a single order of ops:
  - Primary orders operations
  - Backups execute operations in order

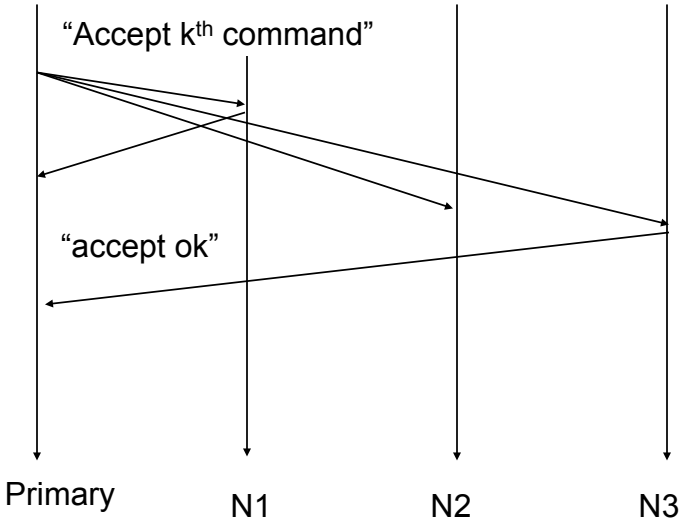
9

Paxos: Normal Execution



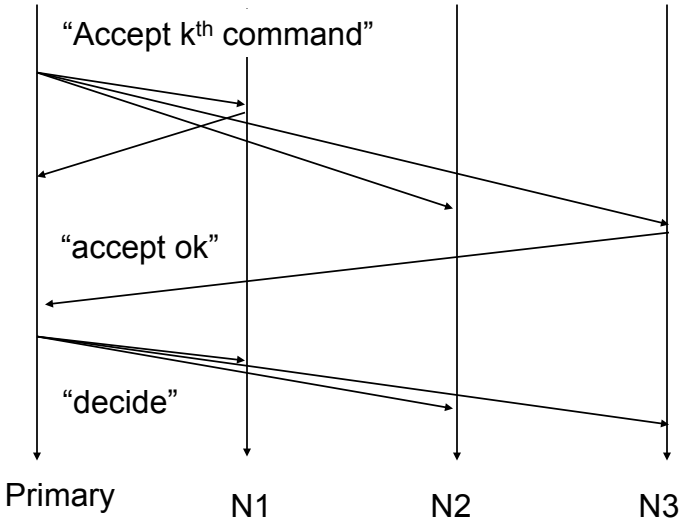
10

Paxos: Normal Execution



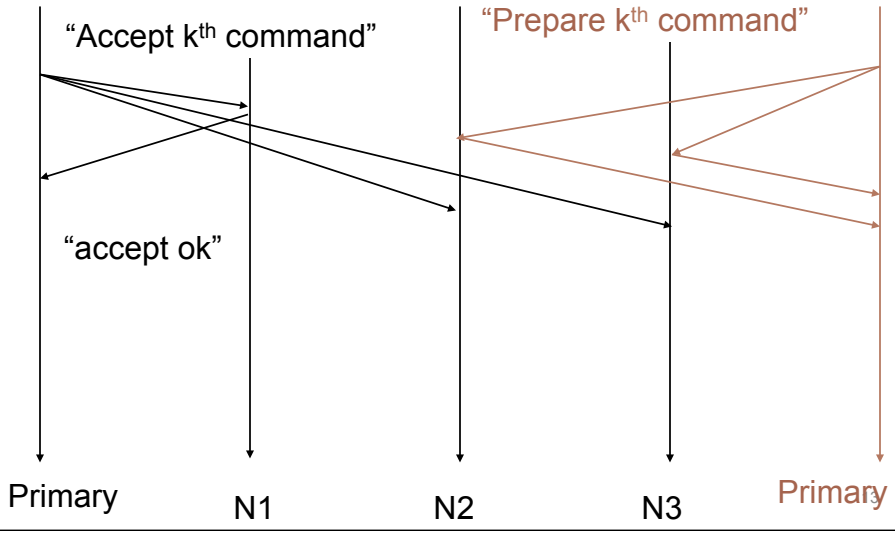
11

Paxos: Normal Execution

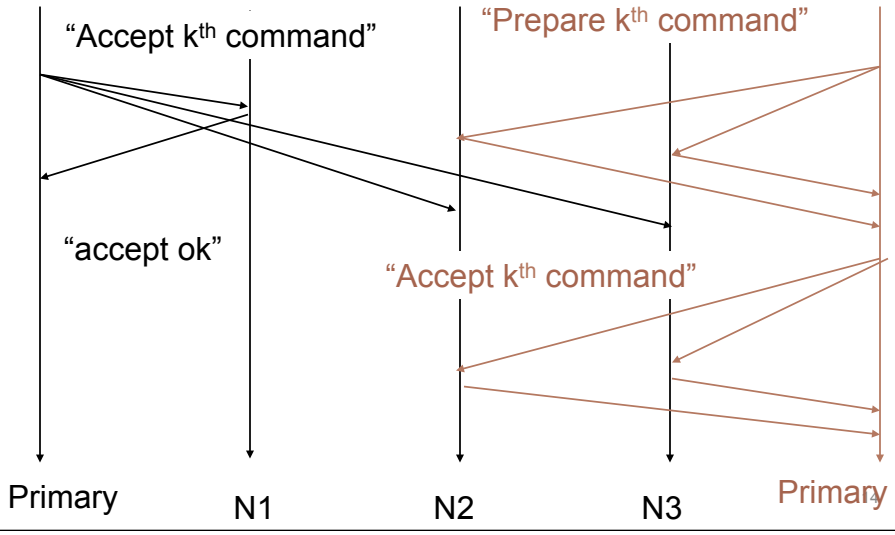


12

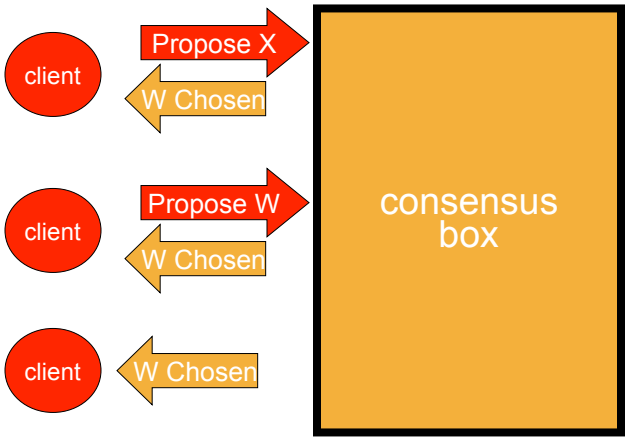
Paxos: Split Brain



Paxos: Split Brain



Consensus



- Collects proposed values
- Picks one proposed value
- Remembers it forever

15

Paxos: fault tolerant agreement

- Paxos lets all nodes agree on the same value despite node failures, network failures and delays
- Extremely useful:
  - e.g. Nodes agree that X is the primary
  - e.g. Nodes agree that Y is the last operation executed

16

Paxos: general approach

- One (or more) node decides to be the leader
- Leader proposes a value and solicits acceptance from others (acceptors)
- Leader announces result or tries again

17

Paxos requirement

- Correctness (safety):
  - All nodes agree on the same value
  - The agreed value X has been proposed by some node
- Fault-tolerance:
  - If less than N/2 nodes fail, the rest nodes should reach agreement *eventually w.h.p*
  - Liveness is not *guaranteed*

18

Why is agreement hard?

- What if >1 nodes become leaders simultaneously?
- What if there is a network partition?
- What if a leader crashes in the middle of solicitation?
- What if a leader crashes after deciding but before announcing results?
- What if the new leader proposes different values than already decided value?

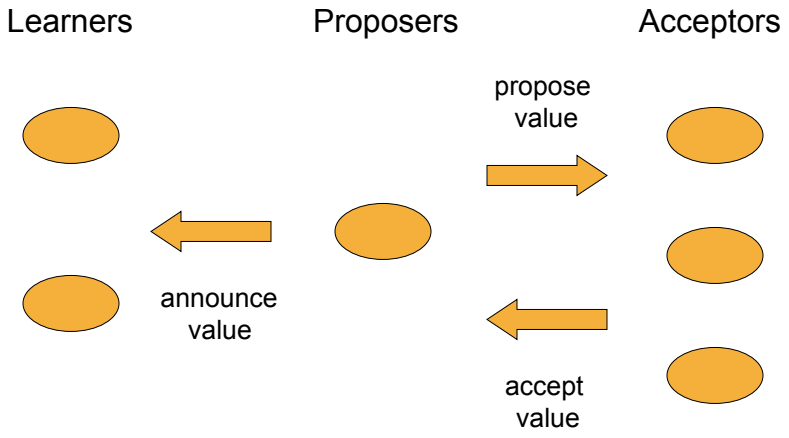
19

Paxos setup

- Each node runs as a *proposer*, *acceptor* and *learner*
- Proposer (leader) proposes a value and solicit acceptance from acceptors
- Leader announces the chosen value to learners

20

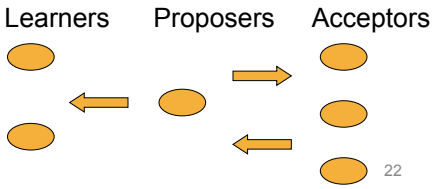
Paxos setup



21

Strawman 1: Single Acceptor

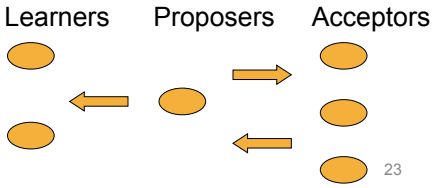
- Designate a single node X as acceptor (e.g. one with smallest id)
  - Each *proposer* sends its value to X
  - X decides on one of the values
  - X announces its decision to all *learners*



22

Strawman 1: Single Acceptor

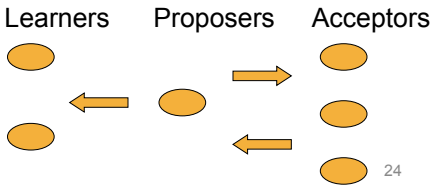
- Designate a single node X as acceptor (e.g. one with smallest id)
  - Each *proposer* sends its value to X
  - X decides on one of the values
  - X announces its decision to all *learners*
- Problem?
  - Failure of the single acceptor halts decision
  - Need multiple acceptors!



23

Strawman 2: multiple acceptors

- Each proposer (leader) proposes to all acceptors
- Each acceptor accepts the first proposal it receives and rejects other proposals
- If the leader receives positive replies from a majority of acceptors, it chooses its own value
  - There is at most 1 majority, hence only a single value is chosen
- Leader sends chosen value to all learners



24

Strawman 2: multiple acceptors

- Each proposer (leader) proposes to all acceptors
- Each acceptor accepts the first proposal it receives and rejects other proposals
- If the leader receives positive replies from a majority of acceptors, it chooses its own value
  - There is at most 1 majority, hence only a single value is chosen
- Leader sends chosen value to all learners
- Problem:
  - What if multiple leaders propose simultaneously so there is no majority accepting?

25

Paxos solution

- Proposals (for a value e.g. k^{th} command) are ordered by proposal #
- Each acceptor must accept the first proposal that it receives
- Each acceptor may accept multiple proposals
  - If a proposal with value v is chosen, all higher proposals chosen have value v

26

Paxos solution

- Proposals (for a value e.g. k^{th} command) are ordered by proposal #
- Each acceptor must accept the first proposal that it receives
- Each acceptor may accept multiple proposals
  - If a proposal with value v is chosen, all higher proposals chosen have value v
  - If a proposal with value v is chosen, all higher proposals accepted by any acceptor have value v

27

Paxos solution

- Proposals (for a value e.g. k^{th} command) are ordered by proposal #
- Each acceptor must accept the first proposal that it receives
- Each acceptor may accept multiple proposals
  - If a proposal with value v is chosen, all higher proposals chosen have value v
  - If a proposal with value v is chosen, all higher proposals accepted by any acceptor have value v
  - If a proposal with value v is chosen, all higher proposals issued by any proposer have value v

28

Paxos solution

- Proposals (for a value e.g. k^{th} command) are ordered by proposal #
- Each acceptor must accept the first proposal that it receives
- Each acc
  - If a pro
  - If a pro
  - If a proposal with value v is chosen, all higher proposals issued by any proposer have value v

Before proposing value v for proposal n, proposer will poll acceptors for
• Promise that they will not accept any future proposals < n
• What value if any that they accepted for highest numbered proposal < n

29

Paxos operation: node state

- Each node maintains:
  - n_a, v_a: highest proposal # and its corresponding accepted value
    - initially null
  - n_h: highest proposal # seen
  - my_n: my proposal # in current Paxos

30

Paxos operation: 3P protocol

- Phase 1 (Prepare)
  - A node decides to be leader (and propose)
  - Leader chooses $my_n > n_h$
  - Leader sends <prepare, my_n> to all nodes

31

Paxos operation: 3P protocol

- Phase 1 (Prepare)
  - A node decides to be leader (and propose)
  - Leader chooses $my_n > n_h$
  - Leader sends <prepare, my_n> to all nodes
  - Upon receiving <prepare, n>

    If $n < n_h$

    reply <prepare-reject>

    else /* $n > n_h$ */

    $n_h = n$

    reply <prepare-ok, n_a,v_a>

32

Paxos operation: 3P protocol

- Phase 1 (Prepare)
  - A node decides to be leader (and propose)
  - Leader chooses $my_n > n_h$
  - Leader sends <prepare, my_n> to all nodes
  - Upon receiving <prepare, n>

    If $n < n_h$

    reply <prepare-reject>

    else /* $n > n_h$ */

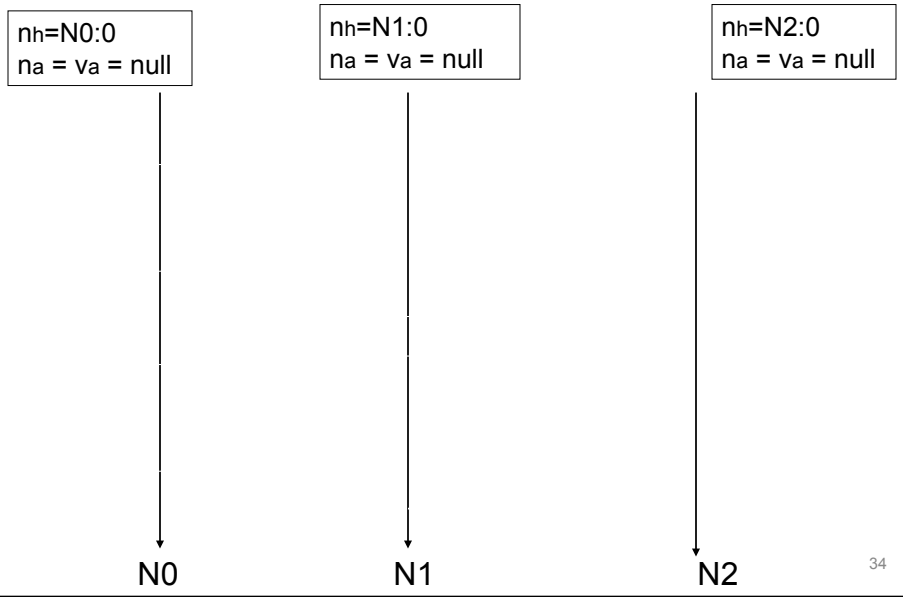
    $n_h = n$

    reply <prepare-ok, n_a,v_a>

This node will not accept any proposal lower than n

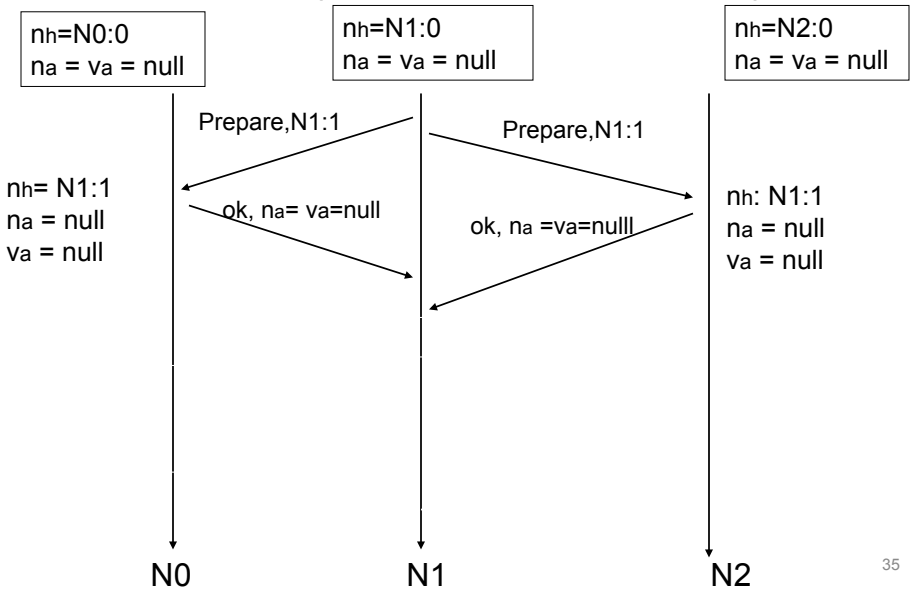
33

Paxos operation: an example



34

Paxos operation: an example



35

Paxos operation

- Phase 2 (Accept):
  - If leader gets prepare-ok from a majority
    V = non-empty value corresponding to the highest n_a received
    If V = null, then leader can pick any V
    Send <accept, my_n, V> to all nodes

36

Paxos operation

- Phase 2 (Accept):
  - If leader gets prepare-ok from a majority
    - V = non-empty value corresponding to the highest n_a received
    - If V = null, then leader can pick any V
    - Send <accept, my_n, V> to all nodes
  - If leader fails to get majority prepare-ok
    - Delay and restart Paxos

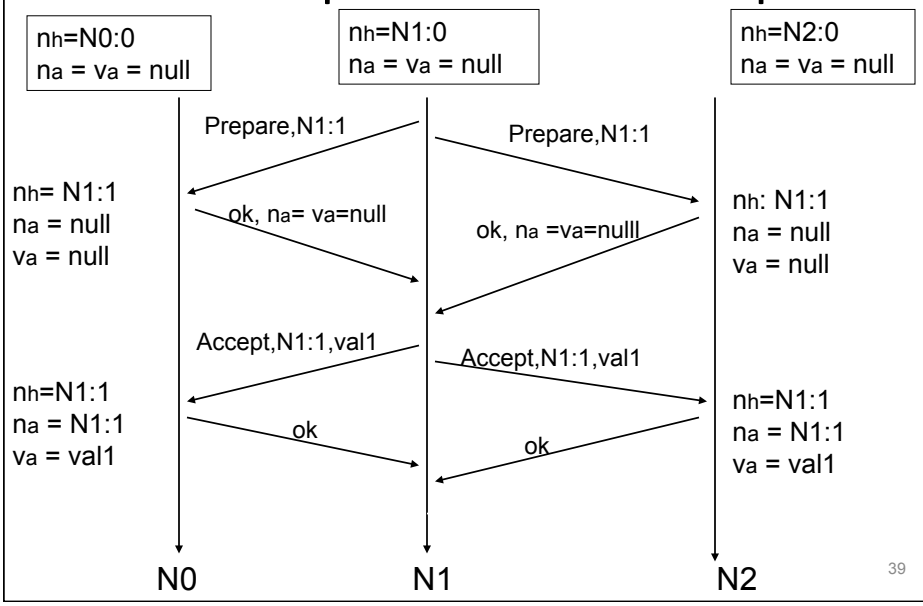
37

Paxos operation

- Phase 2 (Accept):
  - If leader gets prepare-ok from a majority
    - V = non-empty value corresponding to the highest n_a received
    - If V = null, then leader can pick any V
    - Send <accept, my_n, V> to all nodes
  - If leader fails to get majority prepare-ok
    - Delay and restart Paxos
  - Upon receiving <accept, n, V>
    - If $n < n_h$
      - reply with <accept-reject>
    - else
      - $n_a = n$; $v_a = V$; $n_h = n$
      - reply with <accept-ok>

38

Paxos operation: an example



39

Paxos operation

- Phase 3 (Decide)
  - If leader gets accept-ok from a majority
    - Send <decide, v_a> to all nodes

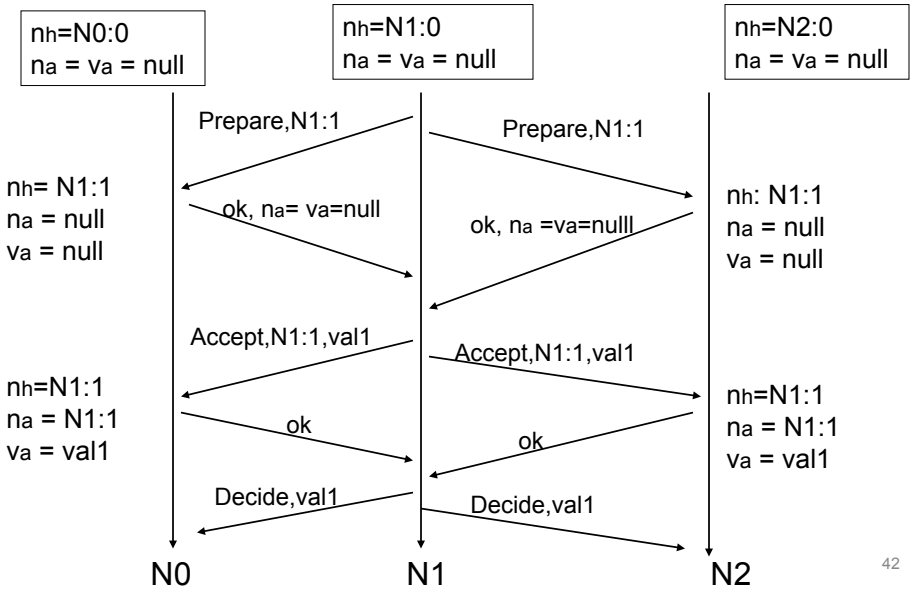
40

Paxos operation

- Phase 3 (Decide)
  - If leader gets accept-ok from a majority
    - Send <decide, v_a> to all nodes
  - If leader fails to get accept-ok from a majority
    - Delay and restart Paxos

41

Paxos operation: an example



42

Paxos properties

- When is the value V chosen?
  1. When leader receives a majority prepare-ok and proposes V

43

Paxos properties

- When is the value V chosen?
  1. When leader receives a majority prepare-ok and proposes V
  2. When a majority of nodes accept V

44

Paxos properties

- When is the value V chosen?
  1. When leader receives a majority prepare-ok and proposes V
  2. When a majority of nodes accept V
  3. When the leader receives a majority accept-ok for value V

45

Understanding Paxos

- What if more than one leader is active?
- Suppose two leaders use different proposal number, N0:10, N1:11
- Can both leaders see a majority of prepare-ok?

46

Understanding Paxos

- What if leader fails while sending accept?
- What if a node fails after receiving accept?
  - If it doesn't restart …
  - If it reboots …
- What if a node fails after sending prepare-ok?
  - If it reboots …

47

BYZANTINE FAULT TOLERANCE

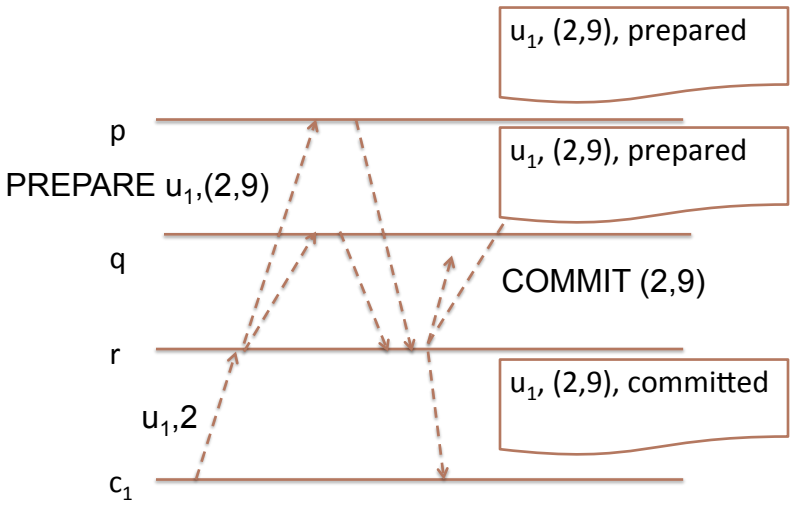
48

Recap

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

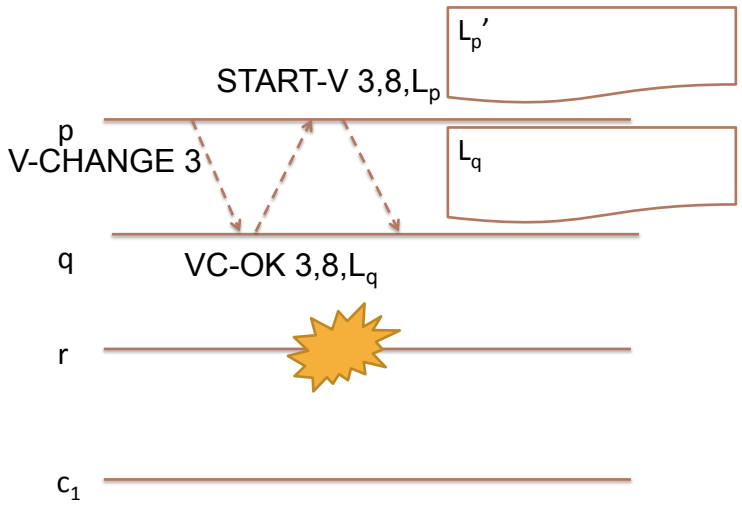
49

Viewstamp: Update



50

Viewstamp: View Change



51

Byzantine fault tolerance

- Nodes fail arbitrarily
  - they lie
  - they collude
- Causes
  - Malicious attacks
  - Software errors
- Seminal work is PBFT
  - Practical Byzantine Fault Tolerance, M. Castro and B. Liskov, SOSP 1999

52

What does PBFT achieve?

- Achieve sequential consistency (linearizability) if …
- Tolerate f faults in a 3f+1-replica RSM
- What does that mean in a practical sense?

53

Practical attacks that PBFT prevents (or not)

- Prevent consistency attacks
  - E.g. a bad node fools clients into accepting a stale bank balance
- Protection is achieved only when <= f nodes fail
  - Byzantine assumes independent failures
- Does not prevent attacks like:
  - Turn a machine into a botnet node
  - Steal SSNs from servers

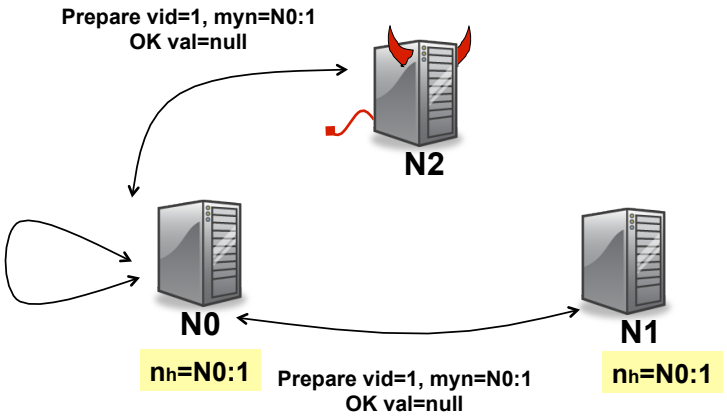
54

Why doesn't traditional RSM work with Byzantine nodes?

- Cannot use Paxos for view change
  - Majority accept-quorum to tolerate f benign faults
  - Bad node tells different things to different quorums!
- Cannot rely on the primary to assign seqno
  - E.g. assign same seqno to different requests!

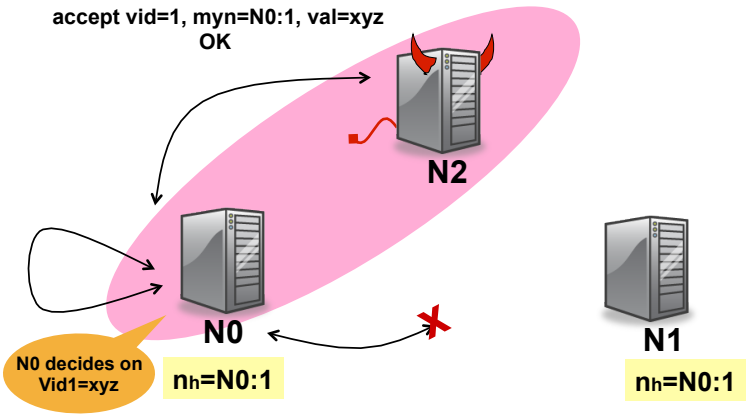
55

Paxos under Byzantine faults



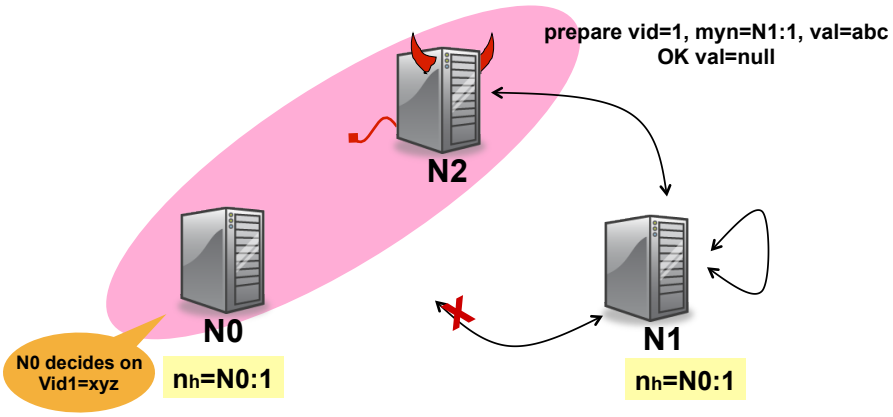
56

Paxos under Byzantine faults



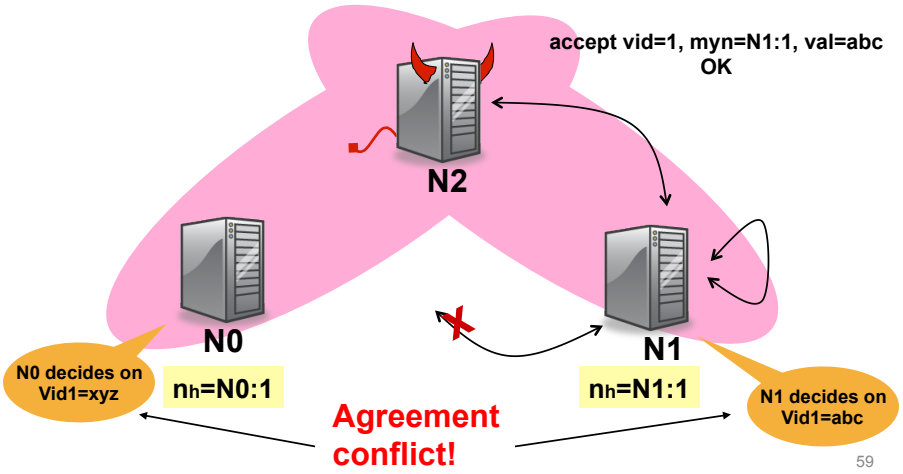
57

Paxos under Byzantine faults



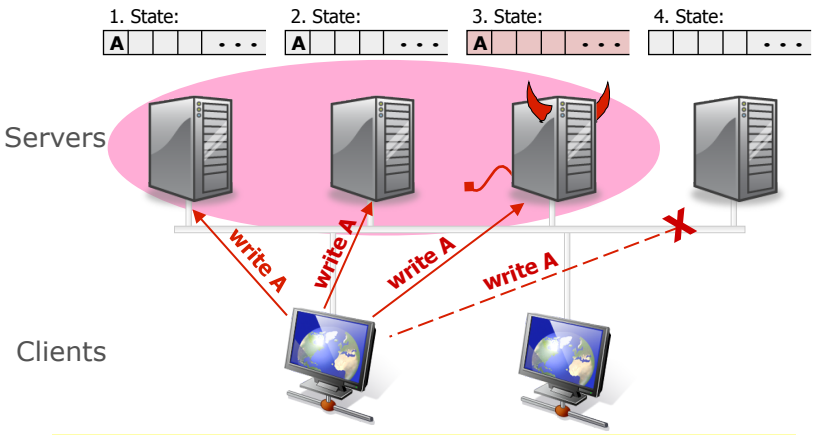
58

Paxos under Byzantine faults



59

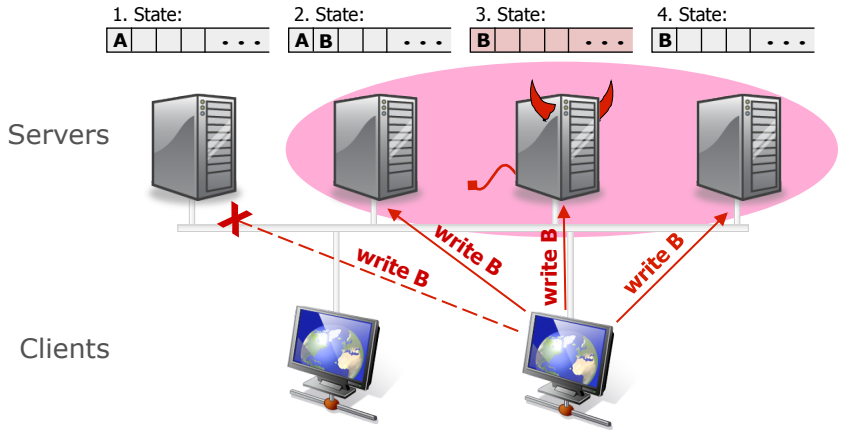
BFT requires 2f+1 quorum out of 3f+1 nodes



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

60

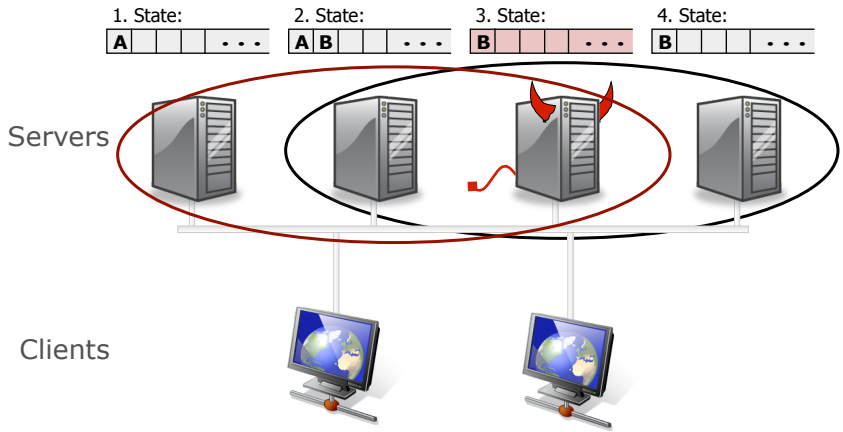
BFT requires 2f+1 quorum out of 3f+1 nodes



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

61

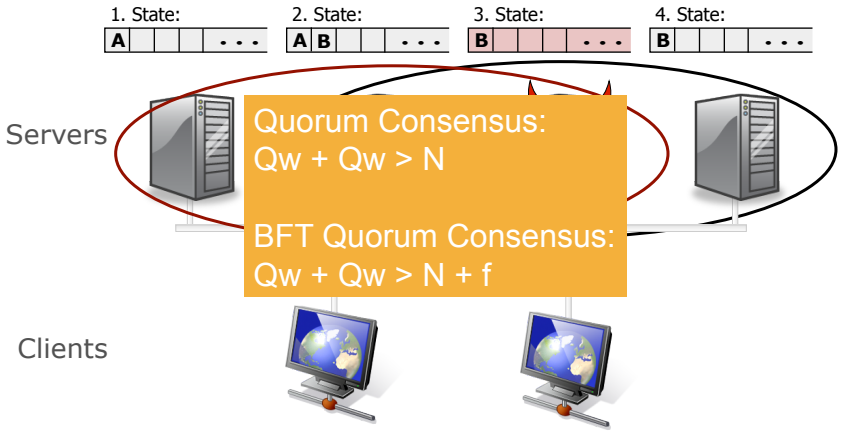
BFT requires 2f+1 quorum out of 3f+1 nodes



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

62

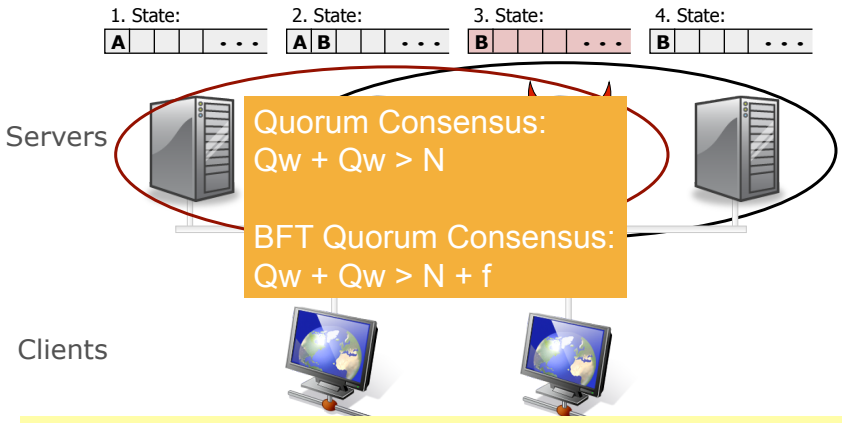
BFT requires 2f+1 quorum out of 3f+1 nodes



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

63

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



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

64

Why doesn't traditional RSM work with Byzantine nodes?

- Cannot use Paxos for view change
  - Majority accept-quorum to tolerate f benign faults
  - Bad node tells different things to different quorums!
- Cannot rely on the primary to assign seqno
  - E.g. assign same seqno to different requests!

65

PBFT main ideas

- To deal with loss of agreement
  - Use a bigger quorum (2f+1 out of 3f+1 nodes)
- To deal with malicious primary
  - 3-phase protocol to agree on sequence number
  - Viewstamp: Prepare, Commit
  - PBFT: PrePrepare, Prepare, Commit
- Need to authenticate communications

66

PBFT Strategy

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

67

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

68

Normal Case

- Client sends request to primary
  - or to all

69

Normal Case

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

70

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 <replica id i,v#,seq#,op>
- All to all communication

71

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>
- **Trust the group, not the individuals**

72

Normal Case

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

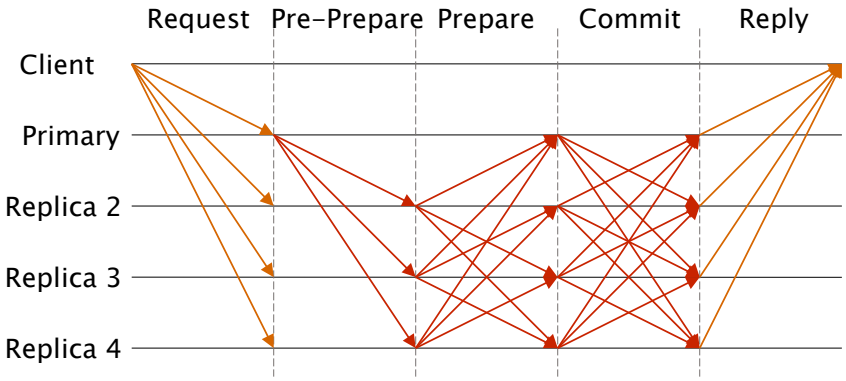
73

Normal Case

- Client waits for f+1 matching replies

74

BFT



75

View Change

- Replicas watch the primary
- Request a view change

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

76

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

77

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

78

BFT Performance

Phase	BFS-PK	BFS	NFS-sdt
1	25.4	0.7	0.6
2	1528.6	39.8	26.9
3	80.1	34.1	30.7
4	87.5	41.3	36.7
5	2935.1	265.4	237.1
total	**4656.7**	**381.3**	**332.0**

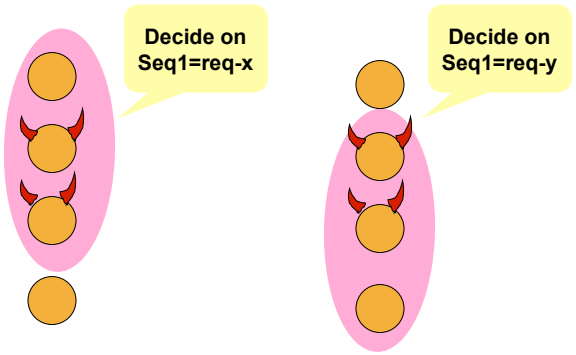
Table 2: Andrew 100: elapsed time in seconds

M. Castro and B. Liskov, *Proactive Recovery in a Byzantine-Fault-Tolerant System*, OSDI 2000

79

Attested Append-Only Memory

- Main worry in PBFT is that malicious nodes lie differently to different replicas



80

A2M proposal

- Introduce a trusted abstraction: **attested append-only-memory**
- A2M properties:
  - Trusted (Attacker can corrupt the RSM implementation, but not A2M itself)
  - Prevent malicious nodes from making different lies to different replicas

81

A2M's abstraction

- A2M implements a trusted log
  - Append: append a value to log
  - Lookup: lookup the value at position i
  - End: lookup the value at the end of log
  - Truncate: garbage collection old values
  - Advance: skip positions in log

82

What A2M achieves

- Smaller quorum size for BFT
  - Achieve correctness & liveness if ≤ f Byzantine faults with 2f+1 nodes

  Or
  - Achieve correctness if ≤ 2f nodes fail. Achieve liveness if ≤ f nodes fail

83

Conclusions

- Paxos
  - Crash failures in asynchronous system
  - Earlier approaches (voting, viewstamp)?
- PBFT
  - Byzantine failures in asynchronous system
  - Adaptation of earlier viewstamp

84