Paxos, Agreement, Consensus

CS 739
Spring 2012
Lecture 9

Notes from reviews

- PtP:
  - How bring nodes up to date?
    - Wait through a whole round of voting?
  - How are decrees propagated to legislators?
    - Can ask for piggy-back up past decrees
  - How select quorum for a ballot?
    - Tend to use all alive nodes
  - Why wait three decrees for changing membership
    - Cannot be immediate (would affect that ballot)
    - Could be next one, but could be outstanding proposals using old membership
    - Gives time to finish ballot before starting next.

Questions

- PmS
  - Will elections terminate
    - PBFT is byzantine paxos, provide time guarantees on state transitions assuming limited number of failures
    - Generally, unsolvable in the presence of slow machines and failures to guarantee termination
- Unique proposal numbers?
  - Local number + node ID is unique

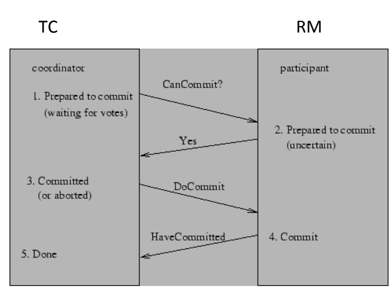
Core problem

- Want multiple nodes to agree on something
  - example: change the primary site for replication to a new node
- Challenge:
  - Make it fault tolerant
- Approaches
  - 2-phase commit
  - (3-phase commit)
  - Paxos

2 phase commit

- Developed for distributed databases
- Model:
  - Resource managers (RM) manage individual resources on different nodes
  - Transaction coordinator (TC) centrally coordinates operations that span multiple nodes
  - Operations (replicated or different) are sent to nodes in a transaction
    - would like to have atomicity: either everybody commits transaction, or nobody

2PC diagram



2PC Protocol: Phase 1

- TC sends out "prepare" message to all RMs
- RMs save enough state that they are guaranteed to be able to prepare if necessary
  - Any transient changes must be written to stable storage
  - Often done with a log
- RMs must still be able to abort
  - Don't erase old data yet
  - Don't know whether all other RMs will vote to commit

2PC protocol: phase 1.5

- RMs log the prepare message and the vote
- RMs send back a vote
  - "Commit" – RM is prepared to commit
  - "Abort" – RM is not able to commit and wants everyone else to abort
- Phase 2: TC sends out "Commit" or "abort"
  - Log result first at TC
  - RMs do the appropriate thing

Failure in 2PC

- RM failure
  - It it fails before/during prepare, TX is aborted (need unanimity)
  - If it fails after prepare, wakes up knowing it was prepared and can ask TC for outcome
- TC failure:
  - Before logging outcome, abort
  - TC aborts TX in prepared stage, resends outcome for Commit/abort
  - THIS BLOCKS
    - RMs don't communicate, so cannot ask each other what happened

2PC vs. Replication

- 2PC works well if different nodes play different roles (e.g., Bank A, Bank B)
- 2PC isn't perfect
  - Must wait for all sites and TC to be up
  - Must know if each site voted yes or no
  - TC must be up to decide
  - Doesn't tolerate faults well; must wait for repair
- Can clients make progress when some nodes unreachable?
  - Yes! When data replicated.

10

Can we fix 2PC

- Yes: 3-phase commit
  - Add another stage (pre-prepare)
  - Allow electing a new coordinator if it fails
- No: 3pc protocols don't handle partition
  - two coordinators may be elected on different sides of the network
- No
  - Known 3pc protocols have flaws

Paxos

- Developed indepdendently by Leslie Lamport and Barbara Liskov (View Stamped Replication)
  - Widely seen as the only solution to this problem
  - Widely used in real systems – Google, Microsoft
- Written in 1990, but lost & not published until 1998
- Solves consensus in asynchronous system
  - Never makes the wrong choice, but may not make progress (consistency not availability)

Problem

- How to reach consensus/data consistency in distributed system that can tolerate non-malicious failures?

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

Paxos: general approach

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

Paxos requirement

- Correctness (safety):
  - All nodes agree on the same value
  - The agreed value X has been proposed by some node
  - **NOTE: Like BGP**
- Fault-tolerance:
  - If less than N/2 nodes fail, the remaining nodes should reach agreement *eventually w.h.p*
  - Liveness is not *guaranteed* if there are a steady stream of failures

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?

Paxos setup

- Each node runs as a *proposer*, *acceptor* and *learner*
- Proposer (leader) proposes a value and solicits acceptance from acceptors
- Leader announces the chosen value to learners
  - Roles are transient (can be reassigned or float around), just someone has to do it in the protocol
  - Acceptor generally is the set of nodes that want to agree

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!

Strawman 2: multiple acceptors

- Each proposer (leader) propose to all acceptors
- Each acceptor accepts the first proposal it receives and rejects the rest
- 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? (not live!)

Paxos solution

- Proposals are ordered by proposal #
  – a node can choose an arbitrarily high number to try to have their proposal accepted …
- Each acceptor may accept multiple proposals
  – If a proposal with value v is chosen, all higher proposals have value v
  – Ensures that proposed values converge

Paxos operation: node state

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

Paxos algorithm

- Phase 1 (prepare):
  – A proposer selects a proposal number n and sends a *prepare request* with number n to majority of acceptors.
  – If an acceptor receives a prepare request with number n greater than that of any prepare request it saw, it responses YES to that request with a promise not to accept any more proposals numbered less than n and include the highest-numbered proposal (if any) that it has accepted.

Paxos operation: 3P protocol

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

    If $n < n_h$

      reply <prepare-reject>

    Already seen a higher-numbered proposal

    Else

      $n_h = n$

      reply <prepare-ok, n_a,v_a>

    This node will not accept any proposal lower than n

    Send back previous number, value

Paxos algorithm

- Phase 2 (accept):
  - If the proposer receives a response YES to its prepare requests from a majority of acceptors, then it sends an *accept request* to each of those acceptors for a proposal numbered n with a values *v* which is the value of the highest-numbered proposal among the responses.
  - If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n.

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>

Reuse most recent chosen value (ensures convergence)

Paxos operation

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

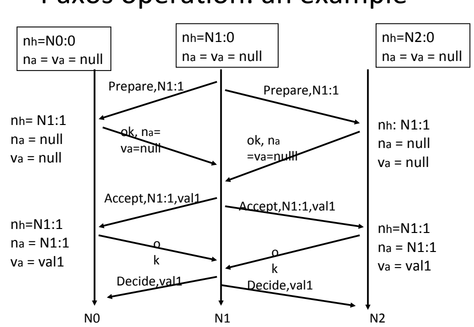
Paxos's properties

- P1: Any proposal number is unique.
- P2: Any two set of acceptors have at least one acceptor in common.
- P3: the value sent out in phase 2 is the value of the highest-numbered proposal of all the responses in phase 1.

Learning a chosen value

- There are some options:
  - Ask a single acceptor
    - May not know all outcomes or result of a vote
  - Each acceptor, whenever it accepts a proposal, informs all the learners.
  - Acceptors informs a distinguished learner (usually the proposer) and let the distinguished learner broadcast the result.
    - Partition results between multiple learners
    - Use timeouts to fail over between learners

Paxos operation: an example



Reading the result of an agreement

- Without designated learners/decide message:
  - Must run Paxos to learn what all nodes agreed
  - Otherwise cannot learn that a majority agreed
- With designated learner:
  - it gets notified of every decision
- Leases: allow fault-tolerant learners
  - promise a single learner for a while (with timeout), must be renewed or else a new learner will be found
  - Avoids paxos for learning

Paxos properties

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

Definition of chosen

- A value is chosen at proposal number n iff majority of acceptor accept that value in phase 2 (accept message) of the proposal number.
- Note: this is a stable property
  - Once majority accepts, all future majorities will

What About Omissions?

- Does not block in case of a lost message
  - Phase I can start with new proposal even if previous attempts never ended

Understanding Paxos

- What happens if the network is partitioned?
  - With one partition, will have a majority on one side, can come to agreement (if nobody else fails)

Paxos: Timeouts

- All nodes wait a maximum period (timeout) for messages they expect
- Upon timeout, a node declares itself a leader and initiates a new Phase 1 of algorithm

36

Paxos: Ensuring Agreement

- When would non-agreement occur?
  - When nodes with different v_a receive Decide
- Safety goal:
  - If Accept could have been sent, future Decide's guaranteed to reach nodes with same v_a

37

Risk: More Than One Leader

- Can occur after timeout during Paxos algorithm, partition, lost packets
- Two leaders must use different n in their Prepare()s, by construction of n
- Suppose two leaders proposed n = 10 and n = 11

38

More Than One Leader (2)

- Case 1: proposer of 10 didn't receive Accept-ok()s from majority of participants
  - Proposer never will receive accept-ok()s from majority, as no node will send accept-ok() for prepare(10,…) after seeing prepare(11,…)
  - Or proposer of 10 may be in network partition with minority of nodes

Result: 10's proposed not decided!

39

More than One Leader (3)

- Case 2: proposer of 10 (10) did receive accept-ok()s from majority of participants
  - Thus, 10's originator may have sent decide()!
  - But 10's majority must have seen 10's accept() before 11's prepare()
    - Otherwise, would have ignored 10's accept, and no majority could have resulted
  - Thus, 11 must receive prepare from at least one node that saw 10's accept
  - Thus, 11 must be aware of 10's value
  - Thus, 11 would have used 10's value, rather than creating one!

Result: agreement on 10's proposed value!

40

Risk: Leader Fails Before Sending accept()s

- Some node will time out and become a leader
- Old leader didn't send any decide()s, so no risk of non-agreement caused by old leader
- Good, but not required, that new leader chooses higher n for proposal
  - Otherwise, timeout, some other leader will try
  - Eventually, will find leader who knew old n and will use higher n

41

Risks: Leader Failures

- Suppose leader fails after sending minority of accept()s
  - Same as two leaders!
- Suppose leader fails after sending majority of accept()s
  - i.e., potentially after reaching agreement!
  - Also same as two leaders!

42

Risk: Node Fails After Receiving accept(), and After Sending accept-ok()

- If node doesn't restart, possible timeout in Phase 3, new leader
- If node does restart, it must remember v_a and n_a on disk!
  - Leader might have failed after sending a few Q3()s
  - New leader must choose same value
  - This failed node may be only node in intersection of two majorities!

43

Paxos and BFT

- The BFT protocol really is a byzantine version of Paxos
  - Signed messages
  - 2F+1 responses needed to make progress rather than a simple majority

Variants

- Multi-paxos
  - Once a leader has an established ballot number, it can pass multiple steps without sending out a new prepare
    - It already has a good idea what the other nodes ballot/proposal numbers are and what they will accept
  - just send "accept" and "decide" (like two-phase commit) with the correct numbers
- Change memberships
  - pass it in one ballot and use it later

Checkpoint+log recovery

- How does a dead node come up to speed?
  - Copy state from another node + replay log
  - Need to snapshot state periodically
- Complication: must synchronize snapshot (slow operation) with log, so set is consistent
  - Think copy-on-write

Real-world problems

- Disk corruption on failure recovery
  - Must checksum log
- Simplifying reads: master leases
  - Ensure no one else will try to propose
  - Replicas refuse prepare messages from anyone but master
  - Flip-flop from repeated master failover
  - Upgrades between protocol versions

Other problems

- Changing membership
  - Need a quorum to change membership with Paxos
  - Catastrophic failure may prevent this
- Otherwise:
  - Pass a decree with new membership for future ballots