Paxos and Co.

The original Paxos

ACM TOCS

- ☆ Transactions on Computer Systems
- ☆ Submitted 1990. Accepted 1998

Leslie Lamport

Turing Award 2013

Abstract

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time leg islators. The legislators maintained consistent copies of the Parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems …

Easier Versions: Paxos made simple / Paxos made live

At the same time (and nearly equivalent)

Viewstamped Replication (VR)

- ☆ Barbara Liskov / Brian Oki 1988

Raft

- ☆ Diego Ongar, John Ousterhout et. Al. 2014

Barbara Liskov

Turing Award

2008

VR provides the same properties

- ☆ Paxos follows *active* replication approach
- ☆ VR/Raft follows *passive* replication approach

Consensus Problem

❑ Model:

- ☆ set of processes P_1, …. P_N
- ☆ communication reliable but asynchronous
  - each message sent is eventually delivered to all correct recipients
    - ▲ (retransmission, network partitions eventually heal, ….)
- ☆ processes only fail by crashing and then stop executing
  - correct process: exhibits no failures at any point in the execution under consideration
  - faulty process: opposite

Consensus II

- Problem Definition:
  - ☆ Start
    - every process P_i begins in undecided state and proposes a value v_i.
  - ☆ Protocol:
    - processes communicate with each other exchanging values.
  - ☆ End
    - Each process then sets a decision variable d_i and enters decided state, and does not change d_i anymore.
  - ☆ Conditions:
    - *Termination (Liveness Property)*: Eventually each correct process sets its decision variable
    - *Agreement (Safety Property)*: The decision variable of all correct processes is the same: if p_i and p_j are correct and have entered the decided state, then $d_i=d_j$
    - *Integrity*: (different options)
      - ▲ If the correct processes all proposed the same value, then every correct process in the decided state has chosen that value.
      - ▲ The chosen value must be one of the proposed values

2PC and Paxos: Similarities

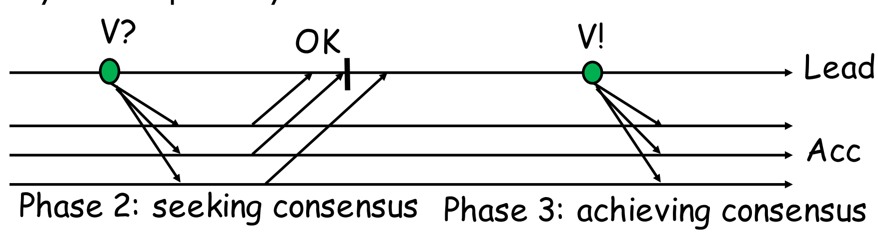
- Agreement
  - ☆ Agreement stronger than with Consensus
  - ☆ Consensus: the decision variables of all *correct processes* that have decided, have the same value
  - ☆ 2PC/Paxos: the decision variable of *all processes (correct or faulty)* that have decided, have the same value
  - ☆ Compare with *uniform delivery* of messages: as soon as one process (faulty or correct) delivers a message, all correct processes deliver the message
- Failure Model
  - ☆ 2PC and Paxos: Crash + recovery + Stable storage survives crash
- Handling Impossibility Result
  - ☆ 2PC and Paxos: Blocking (safety over liveness)
    - 2PC: in case coordinator crashes decision might be delayed until coordinator recovers
    - Paxos: if coordinator (called leader) crashes, another leader takes over and tries to finish the job

2PC and Paxos: Differences

- Integrity
  - ☆ 2PC:
    - Commit only decided if all propose yes
    - Needed because 2PC developed for distributed databases: different storage managers have different data: all must prepare and be willing to commit
  - ☆ Paxos
    - decide on one of the proposed values
    - Usage: replication of an object x:
      - ▲decide on the next update for all replicas
      - ▲Doesn't matter which update of several is taken as long as all decide on the same
    - Not all need to be asked if ok → turns out majority is enough…

Paxos preliminaries

- N acceptors
  - ☆ Majority required for consensus
- Leader/proposer/coordinator
  - ☆ Presents a consensus value to the acceptors and counts the the number of accepts (majority needed)
  - ☆ Notifies the acceptors of success
- Any node/replica may serve either/both roles



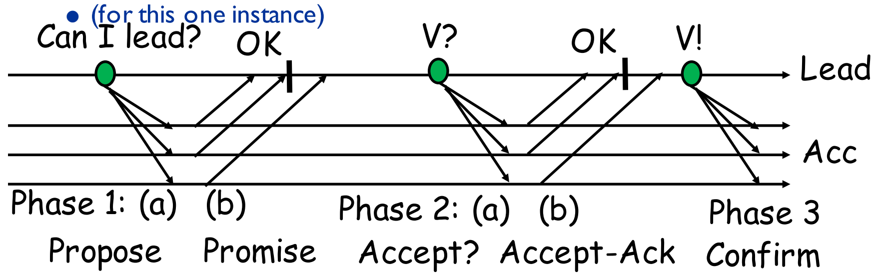
- Looks like 2PC but only a majority is needed because of difference in integrity requirement

Leader Election

- ❑ How to choose a new leader if the old fails
  - ☆ Leader election is a consensus problem!
- ❑ Paxos is safe with multiple leaders
  - ☆ Leader election is built in
    - **Phase I: try to be leader**
    - Phase II: get votes
    - Phase III: confirm decision
  - ☆ If consensus appears stalled, anyone can try to take over as leader
    - Initiate a new pre-phase
  - ☆ Live-lock can occur and things never terminate….

Pre-Phase: choosing the leader

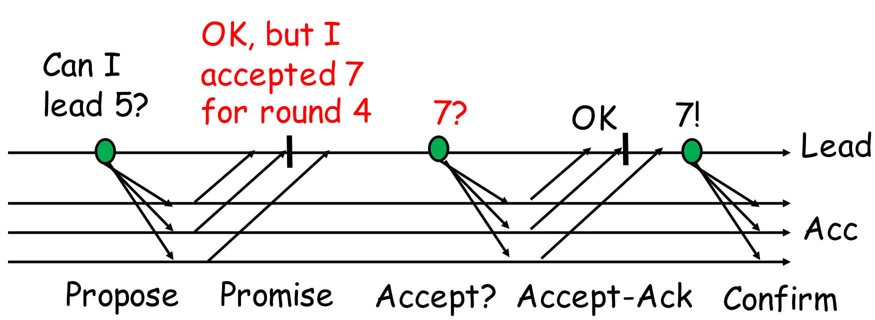
- ❑ Would-be leader chooses a unique ballotID
  - ☆ Try bigger and bigger (unique) numbers
- ❑ Proposes to the acceptors: accept me as leader
- ❑ Acceptors return highest ballotID seen so far
  - ☆ At least one is higher than yours: lost
  - ☆ If a majority responds and know of no higher ballot, it's you!
    - (for this one instance)



Details at Acceptors

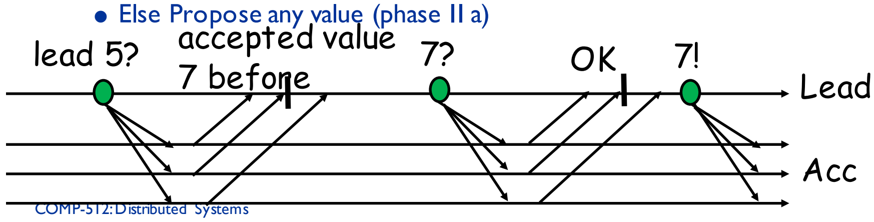
- Upon receiving ballot proposal (Phase I a)
  - ☆ If ballotID higher than any you have seen so far, *promise*
    - Log ballotID in persistent memory
    - Return to would-be leader (Phase I b)
  - ☆ Else Return highest ballotID so far (Phase I b)
    - If already accepted value for this ballotID, return value, too
- Upon receiving value to accept (tagged with ballotID of leader) (Phase II a)
  - ☆ if still the latest accepted ballotID
    - Accept value, store in log in persistent memory
    - Discard any previously accepted value
    - Return accept to would be leader (Phase II b)
  - ☆ Else (has accepted higher ballotID since then)
    - Return deny

Overview at Leader



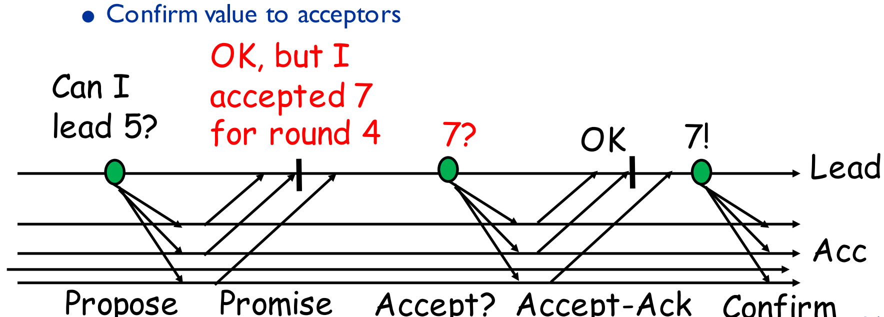
Details at Leader

- Upon receiving answers to ballot propose (Phase I b)
  - ☆ If don't receive a majority of promises,
    - give up the instance
    - Possibly restart with a new ballotID
  - ☆ Else (majority agrees to ballotID)
    - If someone had already agreed to another (smaller) ballotID before AND accepted a value for that, the information is in the response
      - ▲ Find the most recent value that any responding acceptor accepted (the one with the highest ballotID smaller than yours where a value was accepted), and ask acceptors to accept this value (phase II a)
    - Else Propose any value (phase II a)

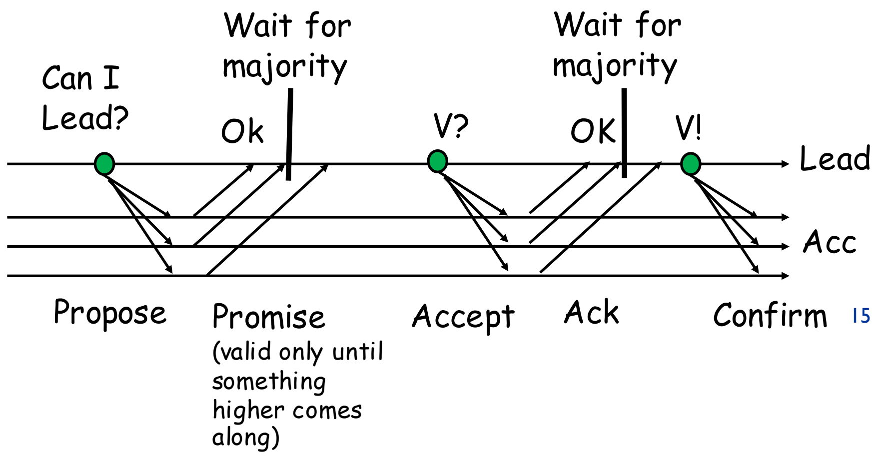


Details at Leader

- Upon receiving responses to val accept request(Phase II b)
  - ☆ If don't receive a majority of accept acknowledgments,
    - give up the instance
    - Possibly restart with a new ballotID
  - ☆ Else (majority accepts value)
    - Log value
    - Confirm value to acceptors



A successful Paxos *Round/Instance*



- Where is the consensus point of no return?

Success and Failure

- A round succeeds if a majority of acceptors hear the accept command (Phase II a) and obey
- A round fails if too many acceptors fail, too many messages get lost, or another leader takes over
  - ☆ But some acceptors may survive and hear the accept request and accept even if the round as a total failed
- Liveness requires that acceptors are free to accept different values in subsequent rounds
- Safety requires that once a round succeeds, no subsequent round changes the value

Success of round but no completion

- Round succeeds:
  - ☆ Majority of acceptors accept value and have value logged
- Round not completed
  - ☆ Leader fails before making final decision
  - ☆ Accept/Confirm messages are lost
  - ☆ Acceptors fail after logging but before sending that they accepted
- On timeout: Start new round (possibly with new leader)
  - ☆ Must agree on the same value as successful but uncompleted round
- Key Invariant: If some round succeeds, then any subsequent round chooses the same value or fails

Why does Key invariant hold?

- Consider leader L of round R
  - ☆ If a previous round S succeeded with value v, then either L learns v or else R fails
  - ☆ S only succeeded if leader of S received responses from majority of acceptors
  - ☆ R only succeeds if leader of R receives responses from a majority
  - ☆ There is at least one acceptor that answered to S and R
  - ☆ When it answers to R, it tells L the value it has chosen for S

Paxos 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 1 is the value of the highest-numbered proposal of all the responses in the pre-phase

One Example

- Two leaders with BulletID = 10, and BulletID = 11
- Case 1: Proposer of 10 does not receive accept-ack from majority
  - ☆ Because nodes that have received bulletid = 11 will not sent it
  - ☆ Because proposer is in network partition that does not hold majority
  - ☆ No decision

One Example

- ❑ Two leaders with BulletID = 10, and BulletID = 11
- ❑ Case II: Proposer of 10 receives accept-ack from majority
  - ☆ Proposer might have sent decision
  - ☆ Majority of acceptors have seen 10's accept and value before agreeing to 11.
  - ☆ Thus, 11 must have received promise from at least one node that saw 10's accept
  - ☆ Thus, 11 must be aware of 10's value
  - ☆ Thus, 11 will use 10's value, rather then creating new one

Result: all agree on 10's value

Leader Fails

- ❑ Before accept (phase II a)
  - ☆ New node will become leader
  - ☆ Old leader hasn't sent decision, so no danger of disagreement
- ❑ After sending minority of accept of phase II a
  - ☆ Same as two leaders
- ❑ After sending majority of accept
  - ☆ i.e., potentially after reaching agreement
  - ☆ Same as two leaders…

Need for persistence

- Acceptor fails after receiving accept and after sending accept ack (phase II)
  - ☆ It must remember that it has accepted the value
  - ☆ write it to stable storage
- Overall:
  - ☆ Logging before every message: 5 logs!

Multi-Paxos

- Chubby Replicated Database
- Data items stored in key/value data store
- Each replica
  - ☆ Snapshot of the database
  - ☆ Replicated log of database commands (insert/update/…)
- Requests are appended to the log
- Paxos decides on the order logs are appended.
- Master/Secondary architecture
  - ☆ All requests are submitted to the presumed leader/master
  - ☆ Thus, it's likely always the same master that initiates a paxos instance
  - ☆ Thus, phase I can be omitted, if the master is stable
  - ☆ Down to 3 message rounds and logs (just like 2PC)