Distributed Systems
11. Consensus: Paxos

Paul Krzyzanowski
Rutgers University
Fall 2015

Consensus Goal
Allow a group of processes to agree on a result
- All processes must agree on the same value
- The value must be one that was submitted by at least one process
  (the consensus algorithm cannot just make up a value)

We saw versions of this
- Mutual exclusion
  - Agree on who gets a resource or who becomes a coordinator
- Election algorithms
  - Agree on who is in charge
- Other uses of consensus:
  - Manage group membership
  - Synchronize state to manage replicas: make sure every group
    member agrees on a (key, value) set
  - Distributed transaction commit
- General consensus problem:
  - How do we get unanimous agreement on a given value?

Achieving consensus seems easy!
Designate a system-wide coordinator to determine outcome

But … this assumes there are no failures
… or we are willing to wait indefinitely for recovery

Consensus algorithm goal
- Create a fault-tolerant consensus algorithm that does not
  block if a majority of processes are working
- Goal: agree on one result among a group of participants
  - Processors may fail (some may need stable storage)
  - Messages may be lost, out of order, or duplicated
  - If delivered, messages are not corrupted

Consensus requirements
- Validity
  - Only proposed values may be selected
- Uniform agreement
  - No two nodes may select different values
- Integrity
  - A node can select only a single value
- Termination (Progress)
  - Every node will eventually decide on a value

Consensus requirements
Consensus: Paxos

• Paxos ensures a consistent ordering in a cluster of machines
  – Events are ordered by sequential event IDs (N)
  – Client wants to log an event: sends request to a Proposer
    – E.g., value, v = "add $100 to my checking account"

  • Proposer
    – Increments the latest event ID it knows about
    – ID = sequence number
    – Asks all the acceptors to reserve that event ID

  • Acceptors
    – A majority of acceptors have to accept the requested event ID

Paxos

• Fault-tolerant distributed consensus algorithm
  – Does not block if a majority of processes are working
  – The algorithm needs a majority (2P+1) of processors survive the simultaneous failure of P processors

• Goal: provide a consistent ordering of events from multiple clients
  – All machines running the algorithm agree on a proposed value from a client
  – The value will be associated with an event or action
  – Paxos ensures that no other machine associates the value with another event

• Abortable consensus
  – A client’s request may be rejected
  – It then has to re-issue the request

A Programmer’s View

Think of R as a key:value pair in a database

while (submit_request(R) != ACCEPTED) ;

Proposal Numbers

• Each proposal has a number (created by proposer)
  – Must be unique (e.g., <sequence>, <process_id>)

  • Newer proposals take precedence over older ones

  • Each acceptor
    – Keeps track of the largest number it has seen so far

  • Lower proposal numbers get rejected
    – Acceptor sends back the (number, value) of the currently accepted proposal

    – Proposer has to "play fair":
      • It will ask the acceptors to accept the (number, value)
      • Either its own or the one it got from the acceptor

Paxos players

• Client: makes a request

  • Proposers:
    – Get a request from a client and run the protocol
    – Leader: elected coordinator among the proposers
    – Not necessary but simplifies message numbering and ensures no contention
    – We don’t need to rely on the presence of a single leader

    • Acceptors:
      – Multiple processes that remember the state of the protocol
      – Quorum = any majority of acceptors

    • Learners:
      – When agreement has been reached by acceptors, a Learner executes the request and/or sends a response back to the client

Proposal Numbers

• Each proposal has a number (created by proposer)
  – Must be unique (e.g., <sequence>, <process_id>)

  • Newer proposals take precedence over older ones

  • Each acceptor
    – Keeps track of the largest number it has seen so far

  • Lower proposal numbers get rejected
    – Acceptor sends back the (number, value) of the currently accepted proposal

    – Proposer has to "play fair":
      • It will ask the acceptors to accept the (number, value)
      • Either its own or the one it got from the acceptor
Paxos in action

One of the proposers is chosen to be a leader; it gets all the requests.

Paxos nodes: one machine may serve several roles.

Proposer

Acceptor

Client

Leader

Quorum

Learner

Paxos in action: Phase 0

Client sends a request to a proposer.

Paxos in action: Phase 1a – PREPARE

Proposer: creates a proposal \( N \) (\( N \) acts like a Lamport time stamp), where \( N \) is greater than any previous proposal number used by this proposer.

Send to Quorum of Acceptors (however many you can reach - but a majority).

\[ N = < \text{seq#}, \text{process_ID} > \]

Paxos in action: Phase 1b – PROMISE

Acceptor: if proposer’s ID > any previous proposal promise to ignore all requests with IDs < \( N \) and reply with info about highest accepted proposal, if there is one: \( \{ N, \text{value} \} \) else reject the proposal.

Paxos in action: Phase 2a

Proposer: if proposer receives promises from the quorum (majority):

Attach a value \( v \) to the proposal (the event).

Send Accept to quorum with the chosen value.

If promise was for another \( \{ N, v \} \), proposer MUST accept that instead.

Paxos in action: Phase 2b

Acceptor: if the promise still holds, then announce the value \( v \)

Send Accepted message to Proposer and every Learner.

else ignore the message (or send NACK).

If the acceptor returned any \( \{ N, v \} \) sets then the proposer must agree to accept one of those values instead of the value it proposed. It picks the \( v \) for the highest \( N \).
Paxos in action: Phase 3

Client sends a request to a proposer

Think of the request as an update to a specific key-value set, such as a field in a database or a specific key-value pair.

Paxos in action: Phase 0

Client sends a request to a proposer

Paxos in action: Phase 1a – PREPARE

Proposer: picks a sequence number: 5
Send to Quorum of Acceptors

Paxos in action: Phase 1b – PROMISE

Accept: Supposes 5 is the highest sequence # any acceptor has seen
Each acceptor PROMISES not to accept any lower numbers

Paxos in action: Phase 2a – ACCEPT

Proposer: Proposer receives the promise from a majority of acceptors
Proposer must accept that <seq, value>

Paxos: A Simple Example – All Good
Paxos

Paxos in action: Phase 2b – ANNOUNCE

Acceptors state that they accepted the request

Proposer
Accept
Accept
Accept
Announce(5, “e”)

Client

Learners

Paxos: A Simple Example – Higher Offer

One acceptor receives a higher offer before it gets this PREPARE message

Proposer: picks a sequence number: 5
Send to Quorum of Acceptors

Prepare(7: “g”)

Client

Paxos in action: Phase 0

Client sends a request to a proposer

Client

Proposer
Request(“e”)

Learners

Paxos in action: Phase 1a – PREPARE

Proposer: Suppose 5 is the highest sequence # any acceptor has seen
Each acceptor PROMISES not to accept any lower numbers

Proposer: Produces a new sequence # and MUST change its mind and accept the highest offer that it received from any acceptor.

Client

Paxos in action: Phase 1b – PROMISE

Accept
Accept
Accept

Client

Learners

Paxos in action: Phase 2a – ACCEPT

Client

Learners

Proposer
Accept(7, “g”)

Proposer

Accept
Accept
Accept

Quorum

Client

Paxos: A Simple Example – Higher Offer

Prepare(7: “g”)

Client

Paxos in action: Phase 1a – PREPARE
Paxos in action: Phase 2b – ANNOUNCE

Acceptors state that they accepted the request. A learner can propagate this information.

Paxos: Keep trying if you need to

- A proposal $N$ may fail because
  - The acceptor may have made a new promise to ignore all proposals less than some value $M > N$
  - A proposer does not receive a quorum of responses: either promise (phase 1b) or accept (phase 2b)

- Algorithm then has to be restarted with a higher proposal $N$

Paxos summary

- Paxos allows us to ensure consistent (total) ordering over a set of events in a group of machines
  - Events = commands, actions, state updates
- Each machine will have the latest state or a previous version of the state
- Paxos used in:
  - Cassandra lightweight transactions
  - Google Chubby lock manager / name server
  - Google Spanner, Megastore
  - Microsoft Autopilot cluster management service from Bing
  - VMware NSX Controller
  - Amazon Web Services

Paxos summary

To make a change to the system:
- Tell the proposer (leader) the event/command you want to add
  - Note: these requests may occur concurrently
  - Leader = one elected proposer. Not necessary for Paxos algorithm but an optimization to ensure a single, increasing stream of proposal numbers. Cuts down on rejections and retries.
- The proposer picks its next highest event ID and asks all the acceptors to reserve that event ID
  - If any acceptor sees has seen a higher event ID, it rejects the proposal & returns that higher event ID
  - The proposer will have to try again with another event ID
- When the majority of acceptors accept the proposal, accepted events are sent to learners, which can act on them (e.g., update system state)
  - Fault tolerant: need $2k+1$ servers for $k$ fault tolerance

The End