CSE 486/586 Distributed Systems
Paxos

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Paxos

- A consensus algorithm
  - Known as one of the most efficient & elegant consensus algorithms
  - If you stay close to the field of distributed systems, you'll hear about this algorithm over and over.
- What? Consensus? What about FLP (the impossibility of consensus)?
  - Obviously, it doesn't solve FLP.
  - It relies on failure detectors to get around it.
- Plan
  - Brief history (with a lot of quotes)
  - The protocol itself
  - ~~How to "discover" the protocol~~ (this is now optional in the schedule).

Brief History

- Developed by Leslie Lamport (from the Lamport clock)
- *"A fault-tolerant file system called Echo was built at SRC in the late 80s. The builders claimed that it would maintain consistency despite any number of non-Byzantine faults, and would make progress if any majority of the processors were working."*
- *"I decided that what they were trying to do was impossible, and set out to prove it. Instead, I discovered the Paxos algorithm."*
- *"I decided to cast the algorithm in terms of a parliament on an ancient Greek island (Paxos)."*

Brief History

- The paper abstract:
  - *"Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. 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."*
- *"I gave a few lectures in the persona of an Indiana-Jones-style archaeologist."*
- *"My attempt at inserting some humor into the subject was a dismal failure. People who attended my lecture remembered Indiana Jones, but not the algorithm."*

Brief History

- People thought that Paxos was a joke.
- Lamport finally published the paper 8 years later in 1998 after it was written in 1990.
  - Title: "The Part-Time Parliament"
- People did not understand the paper.
- Lamport gave up and wrote another paper that explains Paxos in simple English.
  - Title: "Paxos Made Simple"
  - Abstract: "The Paxos algorithm, when presented in plain English, is very simple."
- Still, it's not the easiest algorithm to understand.
- So people started to write papers and lecture notes to explain "Paxos Made Simple." (e.g., "Paxos Made Moderately Complex", "Paxos Made Practical", etc.)

Review: Consensus

- How do people agree on something?
  - Q: should Steve give an A to everybody taking CSE 486/586?
  - Input: everyone says either yes/no.
  - Output: an agreement of yes or no.
  - FLP: this is impossible even with one-faulty process and arbitrary delays.
- Many distributed systems problems can cast into a consensus problem
  - Mutual exclusion, leader election, total ordering, etc.
- Paxos
  - How do multiple processes agree on a value?
  - Under failures, network partitions, message delays, etc.

Review: Consensus

- People care about this!
- Real systems implement Paxos
  - Google Chubby
  - MS Bing cluster management
  - Etc.
- Amazon CTO Werner Vogels (in his blog post "Job Openings in My Group")
  - *"What kind of things am I looking for in you?"*
  - *"You know your distributed systems theory: You know about logical time, snapshots, stability, message ordering, but also acid and multi-level transactions. You have heard about the FLP impossibility argument. You know why failure detectors can solve it (but you do not have to remember which one diamond-w was). You have at least once tried to understand Paxos by reading the original paper."*

CSE 486/586 Administrivia

- Midterm & PA2B grades will be posted soon.
- UB Climate Survey
  - UB is assessing the current campus culture.
  - http://www.buffalo.edu/inclusion/campus-culture-survey.html
  - Please participate!

Paxos Assumptions & Goals

- The network is *asynchronous* with message delays.
- The network can *lose or duplicate* messages, but *cannot corrupt* them.
- Processes can *crash*.
- Processes are *non-Byzantine* (only crash-stop).
- Processes have *permanent storage*.
- Processes can *propose* values.

- The goal: every process agrees on a value out of the proposed values.

Desired Properties

- Safety
  - Only a value that has been proposed can be chosen
  - Only a single value is chosen
  - A process never learns that a value has been chosen unless it has been
- Liveness
  - Some proposed value is eventually chosen
  - If a value is chosen, a process eventually learns it

Roles of a Process

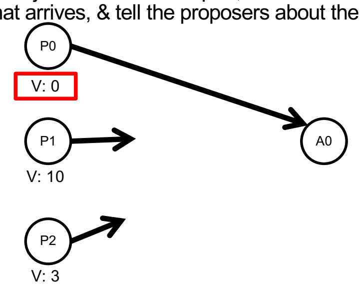
- Three roles
- Proposers: processes that propose values
- Acceptors: processes that accept (i.e., consider) values
  - "Considering a value": the value is a candidate for consensus.
  - Majority acceptance → choosing the value
- Learners: processes that learn the outcome (i.e., chosen value)

Roles of a Process

- In reality, a process can be any one, two, or all three.
- Important requirements
  - The protocol should work under process failures and with delayed and lost messages.
  - The consensus is reached via a majority (> ½).
- Example: a replicated state machine
  - All replicas agree on the order of execution for concurrent transactions
  - All replica assume all roles, i.e., they can each propose, accept, and learn.

First Attempt

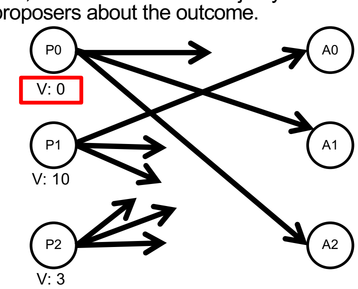
- Let's just have one acceptor, choose the first one that arrives, & tell the proposers about the outcome.



- What's wrong?
  - Single point of failure!

Second Attempt

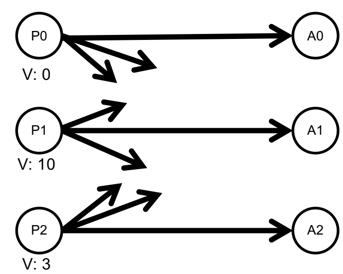
- Let's have multiple acceptors; each accepts the first one; then all choose the majority and tell the proposers about the outcome.



- What's wrong? (next slide)

Second Attempt

- One example, but many other possibilities



Paxos

- Let's have multiple acceptors each accept (i.e., consider) *multiple proposals*.
  - An acceptor accepting a proposal doesn't mean it will be chosen. A majority should accept it.
  - Make sure one of the multiple accepted proposals will have a vote from a majority (will get back to this later)
- Paxos: how do we select one value when there are multiple acceptors accepting multiple proposals?

Paxos Protocol Overview

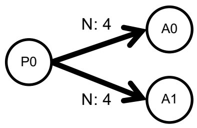
- A proposal should have an ID (since there's multiple).
  - (proposal #, value) == (N, V)
  - The proposal # strictly increasing and globally unique across all proposers, i.e., there should be no tie.
  - E.g., (per-process number).(process id) == 3.1, 3.2, 4.1, etc.
  - This proposal number determines the ordering of all proposals.
- Three phases
  - Prepare phase: a proposer learns previously-accepted proposals from the acceptors.
  - Propose phase: a proposer sends out a proposal.
  - Learn phase: learners learn the outcome.

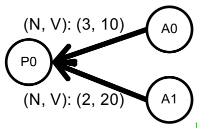
Paxos Protocol Overview

- Rough description of the proposers
  - Before a proposer proposes a value, it will ask acceptors if there is any value proposed before already.
    - » "before": according to the proposal ordering, not time
  - If there is, the proposer will propose the same value, rather than proposing another value.
  - Even with multiple proposals, the value will be the same.
  - The behavior is altruistic: the goal is to reach a consensus, rather than making sure that "my value" is chosen.
- Rough description of the acceptors
  - The goal for acceptors is to accept the highest-numbered proposal coming from all proposers.
  - An acceptor tries to accept a value V with the highest proposal number N.
- Rough description of the learners
  - All learners are passive and wait for the outcome.

Paxos Phase 1

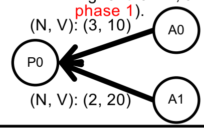
- A proposer chooses its proposal number N and sends a *prepare request* to acceptors.
  - "Hey, have you accepted any proposal before?"
- Note: Acceptors keep the history of proposals.
- An acceptor needs to reply:
  - If it accepted anything before N, the accepted proposal and its value (i.e., the highest proposal number less than N)
  - Extra action: The acceptor stops accepting any proposal numbered less than N any more (to make sure that it doesn't alter the result of the reply).

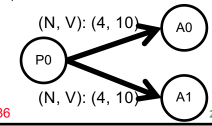




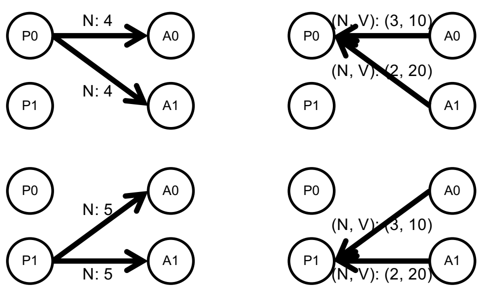
Paxos Phase 2

- If a proposer receives a reply from a majority, it sends an *accept request* with the proposal (N, V).
  - V: the value from the highest proposal number N from the replies (i.e., the accepted proposals returned from acceptors in phase 1)
  - Or, if no accepted proposal was returned in phase 1, a new value to propose.
- Upon receiving (N, V), acceptors either:
  - Accept it
  - Or, reject it if there was another prepare request with N' higher than N, and it replied to it (due to the extra action in phase 1).

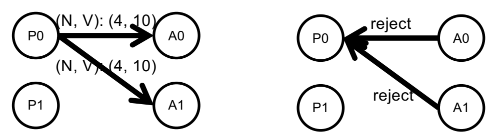




Two Parallel Proposers



Two Parallel Proposers

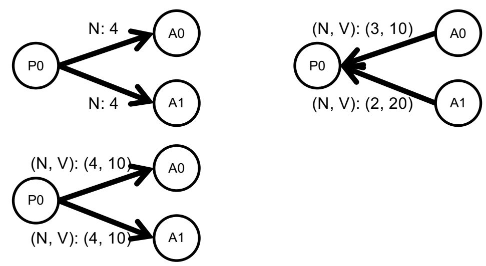


Paxos Phase 3

- Learners need to know which value has been chosen.
- Many possibilities
- One way: have each acceptor respond to all learners, whenever it accepts a proposal.
  - Learners will know if a majority has accepted a proposal.
  - Might be effective, but expensive
- Another way: elect a "distinguished learner"
  - Acceptors respond with their acceptances to this process
  - This distinguished learner informs other learners.
  - Failure-prone
- Mixing the two: a set of distinguished learners

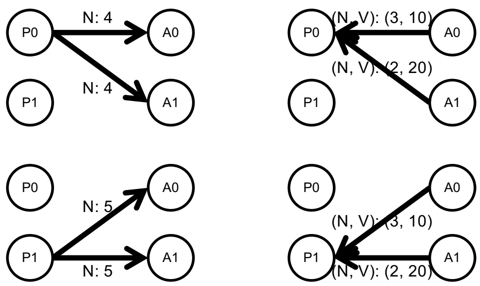
Problem: Progress (Liveness)

- A simple run



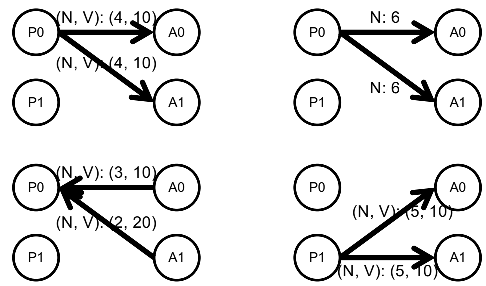
Problem: Progress (Liveness)

- A problematic run



Problem: Progress (Liveness)

- A problematic run (cont.)



Problem: Progress (Liveness)

- *There's a race condition for proposals.*
- P0 completes phase 1 with a proposal number N0
- Before P0 starts phase 2, P1 starts and completes phase 1 with a proposal number N1 > N0.
- P0 performs phase 2, acceptors reject.
- Before P1 starts phase 2, P0 restarts and completes phase 1 with a proposal number N2 > N1.
- P1 performs phase 2, acceptors reject.
- …(this can go on forever)

Providing Liveness

- Solution: elect a distinguished proposer
  - I.e., have only one proposer
- If the distinguished proposer can successfully communicate with a majority, the protocol guarantees liveness.
  - I.e., if a process plays all three roles, Paxos can tolerate failures $f < 1/2 * N$.
- Still needs to get around the problem of having a single point of failure

Summary

- Paxos
  - A consensus algorithm
  - Handles crash-stop failures (f < 1/2 * N)
- Three phases
  - Phase 1: prepare request/reply
  - Phase 2: accept request/reply
  - Phase 3: learning of the chosen value

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).