CSE 486/586 Distributed Systems
Paxos

Slides by 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.

• This lecture
  – Brief history (with a lot of quotes)
  – The protocol itself
Brief History

• Developed by Leslie Lamport (of 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 published it 8 years after it was written in 1990.
  – Title: *The Part-Time Parliament* [1]
• People did not understand the paper.
• Lamport gave up and wrote another paper that explains Paxos in simple English.
  – Title: *Paxos Made Simple* [2]
  – Abstract: “The Paxos algorithm, when presented in plain English, is very simple.”
• It’s still not the easiest algorithm to understand.
• People have written papers and lecture notes to explain *Paxos Made Simple*. (e.g., *Paxos Made Moderately Complex* [4], *Paxos Made Practical* [5], etc.)
Review: Consensus

• How do processes agree on something?
  - Q: should Ethan give an A to everyone taking CSE 486/586?
  - Input: everyone says either yes or no.
  - Output: an agreement of yes or no.
  - FLP: this is impossible with even one faulty process and arbitrary delays.

• Many distributed systems problems can be cast as 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
- Amazon CTO Werner Vogels (in his blog post “Job Openings in My Group”, February 2, 2005)
  - “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.”
Paxos Assumptions & Goals

Assumptions:

- The network is asynchronous, with message delays.
- Messages can be lost or duplicated, but not corrupted.
- Processes can crash.
- Processes are non-Byzantine (only crash-stop).
- Processes have permanent storage.
- Processes can propose values.

Goal:

- Every process agrees on a value from the set of 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 actually been chosen

• 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 (or *consider*) values
  - “Considering a value”: the value is a candidate for consensus.
  - Majority acceptance $\rightarrow$ choosing the value
- **Learners**: processes that learn the outcome
Roles of a Process

• In reality, a process can inhabit any combination of roles.
• Important requirements
  – The protocol should work under process failures and with delayed and lost messages.
  – Consensus is reached via a majority (> ½).
• Example: a replicated state machine
  – All replicas agree on the order of execution for concurrent transactions
  – All replicas assume all roles, i.e., they can each propose, accept, and learn.
First Attempt

• Let’s have just one acceptor, choose the first proposal that arrives, and 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

- P0 → A0
  - V: 0

- P1 → A1
  - V: 10

- P2 → A2
  - V: 3
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 must accept it to be chosen.
  – 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 must 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.

- 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 proposers
  - Before a proposer proposes a value, it will ask the acceptors if there is already any proposed value.
  - If there is, the proposer will propose the same value, rather than proposing another value.
  - Even with multiple concurrent proposals, each proposed value will be the same.
  - The behavior is altruistic: the goal is to reach consensus, rather than making sure that “my value” is chosen.
Paxos Protocol Overview

• Rough description of acceptors
  – The goal for acceptors is to accept the highest-numbered proposal from any proposer.
  – An acceptor tries to accept a value V with the highest proposal number N.

• Rough description of learners
  – All learners are passive and wait for the outcome.
Paxos Phase 1

- A proposer chooses a proposal number N and sends a *prepare request* to acceptors.
  - “Hey, have you accepted any proposal yet?”
  - Note: Acceptors keep a history of proposals.
- If an acceptor has accepted anything, it replies with the accepted proposal and its value for the highest proposal number less than N.
- In addition, the acceptor will no longer accept any proposal numbered less than N (to make sure that it wouldn’t alter the result of its reply).

![Diagram showing Paxos Phase 1]

- P0 sends a *prepare request* to A0 and A1 with N: 4.
- A0 replies with (N, V): (3, 10).
Paxos Phase 2

- If a proposer receives a reply from a majority of acceptors, it sends an accept request for proposal \((N, V)\).
  - \(V\) is the value from the highest proposal number received.
- If no accepted proposal was returned in phase 1, it sends an accept request for the new proposal \((N, V)\).
- Upon receiving \((N, V)\), acceptors either:
  - Accept it
  - Reject it if there was another prepare request with \(N'\) higher than \(N\), and it has replied to it (due to the promise in phase 1).

\[
\begin{align*}
(P0, (4, 10)) & \rightarrow (A0, (4, 10)) \\
(P0, (2, 20)) & \rightarrow (A1, (2, 20)) \\
(P0, (3, 10)) & \rightarrow (A0, (3, 10)) \\
(P0, (3, 10)) & \rightarrow (A1, (3, 10))
\end{align*}
\]
Paxos Phase 3

• Learners need to find out which value has been chosen.
• Many possibilities:
  − Have each acceptor notify all learners when it accepts a proposal:
    • Learners will know if a majority has accepted a proposal
    • May be effective, but will be expensive
  − 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

- $(N, V): (2, 20)$
  - $N: 4$
  - $N: 4$
  - $(N, V): (4, 10)$

- $(N, V): (3, 10)$
  - $N: 4$
  - $N: 4$
  - $(N, V): (4, 10)$

- $(N, V): (2, 20)$
  - $N: 4$
  - $N: 4$
  - $(N, V): (4, 10)$
Problem: Progress (Liveness)

- A problematic run

```
0

N: 4

P0

A0

N: 4

P1

A1

(N, V): (2, 20)

P0

A0

(N, V): (3, 10)

P1

A1

(N, V): (2, 20)

P0

A0

(N, V): (3, 10)

P1

A1

(N, V): (2, 20)

P0

A0

(N, V): (3, 10)

P1

A1

(N, V): (2, 20)
```
Problem: Progress (Liveness)

• A problematic run (cont.)

\[
\begin{align*}
P0 & \xrightarrow{(N, V): (4, 10)} A0 \\
& \xrightarrow{(N, V): (4, 10)} P1 \\
& \xrightarrow{(N, V): (3, 10)} P0 \\
& \xrightarrow{(N, V): (2, 20)} P1 \\
P0 & \xrightarrow{(N, V): (5, 10)} A0 \\
& \xrightarrow{(N, V): (5, 10)} P1 \\
& \xrightarrow{(N, V): (4, 10)} A0 \\
& \xrightarrow{(N, V): (4, 10)} A1 \\
& \xrightarrow{(N, V): (5, 10)} A1
\end{align*}
\]
Problem: Progress (Liveness)

- There’s a race condition for proposals.
- P0 completes phase 1 with a proposal number $N_0$.
- Before P0 starts phase 2, P1 starts and completes phase 1 with a proposal number $N_1 > N_0$.
- P0 performs phase 2, acceptors reject.
- Before P1 starts phase 2, P0 restarts and completes phase 1 with a proposal number $N_2 > N_1$.
- P1 performs phase 2, acceptors reject.
- ...(this can go on forever)
Providing Liveness

• Solution: elect a distinguished proposer
  – *i.e.*, have only one proposer *at a time*

• If the distinguished proposer can successfully communicate with a *majority of acceptors*, the protocol guarantees liveness.
  – *i.e.*, if a process plays all three roles, Paxos can tolerate *f* failures where *f* < *N*/2.

• Still needs to get around FLP for the leader election, *e.g.*, having a failure detector
Summary

• **Paxos**
  - A consensus algorithm
  - Handles crash-stop failures \((f < N/2)\)

• **Three phases**
  - Phase 1: prepare request/reply
  - Phase 2: accept request/reply
  - Phase 3: learning of the chosen value
References


References

Acknowledgements

• These slides by Steve Ko, lightly modified and used with permission by Ethan Blanton
• These slides contain material developed and copyrighted by Indranil Gupta (UIUC).