DISTRIBUTED SYSTEMS: PAXOS

Hakim Weatherspoon

Slides borrowed liberally from past presentations from Robert Surton, Cecchetti, Burcu Canakci and Matt Burke
Timeline

Time, Clocks and Ordering

- 1978
- 1984
- 1989
Timeline

Time, Clocks and Ordering

- 1978
- 1984
- 1989
- 1998
Timelines

**Time, Clocks and Ordering**

- 1978: State Machine Replication
- 1984: Paxos Published
- 1989: Paxos Published In Journal
- 1998: Paxos Made Simple
- 2001: Paxos Made Simple
What is consensus?

- Assume a collection of processes that can propose values. A consensus algorithm ensures that a single one among the proposed values is chosen . . . We won’t try to specify precise liveness requirements.

- The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value . . . every protocol for this problem has the possibility of nontermination . . .
What is consensus?

- Only a proposed value may be chosen.
- Only one, unique value may be chosen.
- All correct processes must eventually choose that value.
Paxos

Leslie Lamport

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.
The Part-Time Parliament
Finally published in 1998 after it was put into use

Published as a “lost manuscript” with notes from Keith Marzullo

“This submission was recently discovered behind a filing cabinet in the TOCS editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.”

“Paxos Made Simple” simplified the explanation...a bit too much

Abstract: The Paxos algorithm, when presented in plain English, is very simple.
Assumptions about our model

- Processes can fail by **crashing**
  - No indication of failure; simply stops responding to messages
  - Failed processes cannot arbitrarily transition or send arbitrary messages

- **Asynchronous**, but **reliable**, network
  
  Messages can be
  - lost
  - duplicated
  - reordered
  - held arbitrarily long
  - If a msg is sent infinitely many time, it will be delivered infinitely many times.
Processes

Proposers

Acceptors

Learners
Any process might fail

- There must be multiple acceptors.
Only choose a single value

- A majority of acceptors must agree on the choice.
Property 1

- An acceptor must accept the first proposal it receives.
Wait—what?

- **Majority-must-agree + Must-accept-first =**
  Acceptors must be able to accept multiple proposals
Wait—what?

- Majority-must-agree + Must-accept-first =
  Acceptors must be able to accept multiple proposals
  - Number all proposals uniquely to distinguish them
Property 2

- If a proposal with value $v$ is chosen, then every higher-numbered proposal that is chosen has value $v$. 
Property 2a

- If a proposal with value $v$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $v$. 
Property 2b

- If a proposal with value $v$ is chosen, then every higher-numbered proposal issued by any proposer has value $v$. 
Property 2c

- For any $v$ and $n$, if a proposal with value $v$ and number $n$ is issued, then there is a set $S$ consisting of a majority of acceptors such that either
  - no acceptor in $S$ has accepted any proposal numbered less than $n$, or
  - $v$ is the value of the highest-numbered proposal among all proposals numbered less than $n$ accepted by the acceptors in $S$. 
Proposers
Proposers
Instead of predicting the future

- Proposer sends prepare $n$ to acceptors
- Each acceptor replies with
  - A promise to reject lower proposals in future
  - If any, the highest accepted lower proposal
Accept request

- If a majority promise
  - Proposer sends `propose n, v`
- If there were accepted proposals
  - `v` must match the highest one
    (Otherwise, `v` can be arbitrary.)
Acceptors
Property 1a

- An acceptor can accept a proposal numbered $n$ iff it has not responded to a prepare request having a number greater than $n$. 
Responding to prepare requests

- An acceptors may respond to any prepare request
- To optimize, ignore requests lower than promised
Learners

Choose majority

Broadcast choices

Learners
Distinguished learner (optimization)
Progress

- $P_1$ receives promises for $n_1$
- $P_2$ receives promises for $n_2 > n_1$
- $P_1$ sends proposal numbered $n_1$, rejected
- $P_1$ receives promises for $n_1' > n_2$
- $P_2$ sends proposal numbered $n_2$, rejected
- $P_1$ receives promises for $n_2' > n_1'$
- $P_1$ sends proposal numbered $n_1'$, rejected
- *ad infinitum*...
Paxos Made Moderately Complex

Robbert van Renesse and Deniz Altinbuken (Cornell University)
ACM Computing Surveys, 2015

“The Part-Time Parliament” was too confusing
“Paxos Made Simple” was overly simplified
Better to make it moderately complex!
Much easier to understand
Paxos Structure

Figure from James Mickens. ;login: logout. *The Saddest Moment*. May 2013
Paxos Structure

Proposers

Acceptors

Learners
Moderate Complexity: Notation

Store data and propose to proposers

Function as proposers and learners without persistent storage

Communication pattern between types of processes in a setting where $f = 2$.

Figure from van Renesse and Altinbuken 2015
Single-Decree Synod

Decides on one command

System is divided into proposers and acceptors

The protocol executes in phases:

a. Proposer proposes a ballot $b$
   1. Acceptor $i$ responds with $(b', c_i)$
      a. If $b' > b$, update $b$ and abort
      Else wait for majority of acceptors
         Request received $c_i$ with highest ballot number
   b. If $b'$ has not changed, accept

A learner learns $c$ if it receives the same $(p2b, b', c)$ from a majority of acceptors
Optimizations: Distinguished Learner

Proposers

Acceptors

Distinguished Learner

Other Learners
Optimizations: Distinguished Proposer

- Distinguished Proposer
- Other Proposers
- Acceptors
- Learners
What can go wrong?

- A bunch of preemption
  - If two proposers keep preempting each other, no decision will be made

- Too many faults
  - Liveness requirements
    - majority of acceptors
    - one proposer
    - one learner
  - Correctness requires one learner
Deciding on Multiple Commands

Run Synod protocol for multiple slots

Sequential separate runs
  Slow

Parallel separate runs
  Broken (no ordering)

One run with multiple slots
  Multi-decree Synod!
Paxos with Multi-Decree Synod

- Like single-decree Synod with one key difference: Every proposal contains a both a ballot and slot number.
- Each slot is decided independently.
- On preemption (`if (b' > b) {b = b'; abort;}`), proposer aborts active proposals for all slots.
Moderate Complexity: Leaders

Leader functionality is split into pieces

- Scouts – perform proposal function for a ballot number
  - While a scout is outstanding, do nothing

- Commanders – perform commit requests
  - If a majority of acceptors accept, the commander reports a decision

- Both can be preempted by a higher ballot number
  - Causes all commanders and scouts to shut down and spawn a new scout
Moderate Complexity: Optimizations

- **Distinguished Leader**
  - Provides both distinguished proposer and distinguished learner

- **Garbage Collection**
  - Each acceptor has to store every previous decision
  - Once $f + 1$ have all decisions up to slot $s$, no need to store $s$ or earlier
Paxos Questions?
Backup
What is consensus?

Consensus is the problem of getting a set of processors to agree on some value.
What is consensus?

More formally, *consensus* is the problem of satisfying the following properties:

- Validity
- Agreement
- Integrity
- Termination
What is consensus?

More formally, consensus is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose $v$, then all correct deciding processes eventually decide $v$

- **Agreement**
- **Integrity**
- **Termination**
What is consensus?

More formally, consensus is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose $v$, then all correct deciding processes eventually decide $v$

- **Agreement**
  - If a correct deciding process decides $v$, then all correct deciding processes eventually decide $v$

- **Integrity**

- **Termination**
What is consensus?

More formally, *consensus* is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose $v$, then all correct deciding processes eventually decide $v$

- **Agreement**
  - If a correct deciding process decides $v$, then all correct deciding processes eventually decide $v$

- **Integrity**
  - Every correct deciding process decides at most one value, and if it decides $v$, then some process must have proposed $v$

- **Termination**
What is consensus?

More formally, consensus is the problem of satisfying the following properties:

- **Validity**
  - If all processes that propose a value propose \( v \), then all correct deciding processes eventually decide \( v \)

- **Agreement**
  - If a correct deciding process decides \( v \), then all correct deciding processes eventually decide \( v \)

- **Integrity**
  - Every correct deciding process decides at most one value, and if it decides \( v \), then some process must have proposed \( v \)

- **Termination**
  - Every correct deciding process eventually decides a value, with the value decided by some process.