Distributed Consensus

Paxos

... 

Ethan Cecchetti

October 18, 2016
CS6410

Some structure taken from Robert Burgess’s 2009 slides on this topic
State Machine Replication (SMR)

View a server as a state machine.

To replicate the server:
1. Replicate the initial state
2. Replicate each transition
Paxos: Fault-Tolerant SMR

- Devised by Leslie Lamport, originally in 1989
- Written as “The Part-Time Parliament”
  - 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.
- Rejected as unimportant and too confusing
Paxos: The Lost Manuscript

- 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.
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:

1a. Proposer proposes a ballot b
1b. Acceptor_i responds with (b', c_i)
2a. If b' > b, update b and abort
   Else wait for majority of acceptors
   Request received c_i with highest ballot number
2b. 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 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?
CORFU: A Distributed Shared Log

Mahesh Balakrishnan†, Dahlia Malkhi†, John Davis†, Vijayan Prabhakaran†, Michael Wei‡, and Ted Wobber†
†Microsoft Research, ‡University of California, San Diego
TOCS 2013

Distributed log designed for high throughput and strong consistency.

- Breaks log across multiple servers
- “Write once” semantics ensure serializability of writes
CORFU: Conflicts

What happens on concurrent writes?

- The first write wins and the rest must retry
  - Retrying repeatedly is very slow.
- Use sequencer to get write locations first
CORFU: Holes and *fill*

What if a writer fails between getting a location and writing?

- **Hole in the log!**
  - Can block applications which require complete logs (e.g. SMR)

- **Provide a *fill* command to fill holes with junk**
  - Anyone can call *fill*
  - If a writer was just slow, it will have to retry
CORFU: Replication

- Shards can be replicated however we want
  - Chain replication is good for low replication factors (2-5)
- On failure, replacement server can take writes immediately
  - Copying over the old log can happen in the background.
Thank You!