“Paxos Made Moderately Complex”
Made Moderately Simple
State machine replication

Reminder: want to agree on order of ops

Can think of operations as a log
Put k1 v1  Put k2 v2

Paxos

Phase 1
- Send prepare messages
- Pick value to accept

Phase 2
- Send accept messages
Can we do better?

Phase 1: “leader election”
- Deciding whose value we will use

Phase 2: “commit”
- Leader makes sure it’s still leader, commits value

What if we split these phases?
- Lets us do operations with one round-trip
Roles in PMMC

Replicas (like learners)
  - Keep log of operations, state machine, configs

Leaders (like proposers)
  - Get elected, drive the consensus protocol

Acceptors (*simpler* than in Paxos Made Simple!)
  - “Vote” on leaders
A note about ballots in PMMC

(leader, seqnum) pairs

Isomorphic to the system we discussed earlier

- **0**: 0, 4, 8, 12, 16, ...
- **1**: 1, 5, 9, 13, 17, ...
- **2**: 2, 6, 10, 14, 18, ...
- **3**: 3, 7, 11, 15, 19, ...
A note about ballots in PMMC

*(leader, seqnum)* pairs

Isomorphic to the system we discussed earlier

<table>
<thead>
<tr>
<th>0</th>
<th>0.0, 1.0, 2.0, 3.0, 4.0, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1, 1.1, 2.1, 3.1, 4.1, ...</td>
</tr>
<tr>
<td>2</td>
<td>0.2, 1.2, 2.2, 3.2, 4.2, ...</td>
</tr>
<tr>
<td>3</td>
<td>0.3, 1.3, 2.3, 3.3, 4.3, ...</td>
</tr>
</tbody>
</table>
Paxos Made Moderately Complex Made Simple

- **client**
- **replicas**
- **leader**
- **acceptors**

- **request** → **propose** → **decision** → **response**
- **scout** → **adopted**
- **commander**
- **p1a** → **p1b** → **p2a** → **p2b**
Paxos Made Moderately Complex Made Simple
Acceptors

Acceptor

```
ballet_num: _
accepted: []
```
Acceptors

p1a(0.1)

Acceptor

ballot_num: 
accepted:[]
Acceptors

p1a(0.1) → Acceptor

ballot_num: 0.1
accepted: []
Acceptors

p1a(0.1)  
p1b([])
Acceptors

Acceptor

| ballot_num: 0.1 |
| accepted: []   |
Acceptors

p1a(0.0)

Acceptors

ballot_num: 0.1
accepted:[]
Acceptors

p1a(0.0)

Nope!

Acceptors

ballot_num: 0.1
accepted: []
Acceptors

Acceptor

ballot_num: 0.1
accepted:[]
Acceptors

$p2a(<0.1, 0, A>)$

Acceptor

ballot_num: 0.1
accepted: []
Acceptors

\[ p_{2a}(<0.1, 0, A>) \]

Acceptors

ballot_num: 0.1
accepted: [ <0.1, 0, A> ]
Acceptors

\[ \text{p2a(}<0.1, 0, A>\text{)} \]

\[ \text{OK!} \]

Acceptors

\[ \text{ballot\_num: 0.1} \]
\[ \text{accepted:}\[<0.1, 0, A>\text{]} \]
Acceptors

Acceptor

ballot_num: 0.1
accepted:[<0.1, 0, A>]
Acceptors

\[ p2a(<0.0, 0, B>) \]

Acceptor

- \text{ballot\_num}: 0.1
- \text{accepted}: [<0.1, 0, A>]
Acceptors

\[ p2a(<0.0, 0, B>) \]

Nope!

\[
\text{Acceptee} \\
\text{ballot\_num: 0.1} \\
\text{accepted: [<0.1, 0, A>]}
\]
Acceptors

Acceptor

ballot_num: 0.1
accepted:[<0.1, 0, A>]
Acceptors

- Ballot numbers increase
- Only accept values from current ballot
- Never remove ballots
- If a value $v$ is chosen by a majority on ballot $b$, then any value accepted by any acceptor in the same slot on ballot $b' > b$ has the same value
Paxos Made Moderately Complex Made Simple

- **client**
  - k

- **replicas**
  - \( \rho_1 \)
  - \( \rho_2 \)

- **leader**
  - \( \lambda \)
  - scout
  - adopted

- **acceptors**
  - \( \alpha_1 \)
  - \( \alpha_2 \)
  - \( \alpha_3 \)

- **request**
- **propose**
- **commander**
- **decision**
- **response**
Paxos Made Moderately Complex Made Simple
Leader: Getting Elected

Leader

active: false
ballot_num: 0.0
proposals: []
Leader: Getting Elected

Leader

active: false
ballot_num: 0.0
proposals: []

Acceptors

p1a(0.0)
Leader: Getting Elected

Leader

active: false
ballot_num: 0.0
proposals: []

Nope!

Acceptor

Nope!

Acceptor

Nope!

Acceptor
Leader: Getting Elected

Leader

active: false
ballot_num: 1.0
proposals: []
Leader: Getting Elected

Leader

active: false
ballot_num: 1.0
proposals: []

Or...

Acceptor

Acceptor

Acceptor
Leader: Getting Elected

Leader

active: false
ballot_num: 0.0
proposals: []

OK([])!

Acceptor

OK([])!

Acceptor

Acceptor
Leader: Getting Elected

Leader

- active: true
- ballot_num: 0.0
- proposals: []

Acceptors:

- Acceptor
- Acceptor
- Acceptor
When to run for office

When should a leader try to get elected?

- At the beginning of time
- When the current leader seems to have failed

Paper describes an algorithm, based on pinging the leader and timing out

If you get preempted, don’t immediately try for election again!
Paxos Made Moderately Complex Made Simple

Diagram showing the Paxos protocol with client, replicas, leader, and acceptors. The process includes:
- Request from client to replicas
- Propose from replicas to leader
- Adopted from leader to acceptors
- Commander from leader to acceptors
- Decision from leader to replicas and client
- Response from client to replicas

The diagram illustrates the communication and decision-making process in the Paxos protocol.
PaxosMadeModeratelyComplexMadeSimple
Leader: Handling proposals

Leader

active: true
ballot_num: 0.0
proposals: []

Op1 should be A
(A = “Put k1 v1”)

Replica

Acceptor

Acceptor

Acceptor
Leader: Handling proposals

Leader
active: true
ballot_num: 0.0
proposals: [<1, A>]

Acceptors

Replica
Leader: Handling proposals

Leader:
- active: true
- ballot_num: 0.0
- proposals: [<1, A>]

Acceptors:
1. p2a(<0.0, 1, A>)
2. Replica
Leader: Handling proposals

active: true
ballot_num: 0.0
proposals: [<1, A>]
Leader: Handling proposals

Leader

active: false
ballot_num: 0.0
proposals: [<1, A>]

Acceptor

Acceptor

Acceptor

Replica
Leader: Handling proposals

Leader

active: false
ballot_num: 0.0
proposals: [<1, A>]

Or...

Acceptor

Acceptor

Replica

Acceptor

Acceptor
Leader: Handling proposals

Leader

active: true
ballot_num: 0.0
proposals: [<1, A>]

Acceptor

OK!

Acceptor

OK!

Acceptor

OK!

Replica
Leader: Handling proposals

active: true
ballot_num: 0.0
proposals: [<1, A>]

Op1 is A

Leader

Replica

Replica

Replica

Acceptor

Acceptor

Acceptor
Leaders

- Only propose one value per ballot and slot

- If a value \( v \) is chosen by a majority on ballot \( b \), then any value proposed by any leader in the same slot on ballot \( b' > b \) has the same value
Paxos Made Moderately Complex Made Simple
Paxos Made Moderately Complex Made Simple
Replicas

Put k1 v1
Put k2 v2
Replicas

Replica

slot_out

Put k1 v1
Put k2 v2
App k1 v1
App k2 v2

slot_in

Op1
Op2
Op3
Op4
Op5
Op6
Replicas

decision(3, “App k1 v1”)
Replicas

Leader

Replica

(slot_out slot_in)

Put k1 v1  Put k2 v2  App k1 v1  App k2 v2


Replicas

decision(4, “Put k3 v3”)
Replicas

propose(5, "App k2 v2")

slot_out slot_in

Put k1 v1  Put k2 v2  App k1 v1  Put k3 v3  App k2 v2

Reconfiguration

All replicas *must* agree on who the leaders and acceptors are.

How do we do this?
Reconfiguration

All replicas *must* agree on who the leaders and acceptors are

How do we do this?

- Use the log!
- Commit a special reconfiguration command
- New config applies after WINDOW slots
Reconfiguration

What if we need to reconfigure *now* and client requests aren’t coming in?
Reconfiguration

What if we need to reconfigure now and client requests aren’t coming in?

- Commit no-ops until WINDOW is cleared
Other complications

State simplifications
- Can track much less information, esp. on replicas

Garbage collection
- Unbounded memory growth is bad
- Lab 3: track finished slots across all instances, garbage collect when everyone has learned result

Read-only commands
- Can’t just read from replica (why?)
- But, don’t need their own slot
Questions

What should be in stable storage?
Question

What are the costs to using Paxos? Is it practical enough?