Paxos Week: Return of the State Machine

Doug Woos
Logistics notes

No in-class lecture Monday
Problem Set 2 due tonight
Lab 3 out
Paxos Made Simple Discussion

Paxos vs. Primary/Backup

Paxos vs. 2PC

What about reconfig?

The story of Paxos
State machine replication

Reminder: want to agree on order of ops
Can think of operations as a log
I want to do “Put k1 v1”

I want to do “Put k2 v2”
I want to do "Put k1 v1" for Op1

I want to do "Put k2 v2"
Put k1 v1


S1

S2

I want to do "Put k2 v2"

S3
Put k1 v1

I want to do “Put k2 v2”
Put k1 v1  Put k2 v2

Put k1 v1  Put k2 v2

Paxos?
Lab 3

Paxos = Paxos Made Simple
Lab 3

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
Put k1 v1
Put k2 v2
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 Mon, Wed

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 Mon, Wed

\[
\begin{align*}
0 & \quad (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), \ldots \\
1 & \quad (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), \ldots \\
2 & \quad (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), \ldots \\
3 & \quad (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), \ldots 
\end{align*}
\]
A note about ballots in PMMC

(leader, seqnum) pairs

Isomorphic to the system we discussed Mon, Wed

0

0.0, 1.0, 2.0, 3.0, 4.0, ...

1

0.1, 1.1, 2.1, 3.1, 4.1, ...

2

0.2, 1.2, 2.2, 3.2, 4.2, ...

3

0.3, 1.3, 2.3, 3.3, 4.3, ...
Paxos Made Moderately Complex Made Simple
Paxos Made Moderately Complex Made Simple
Acceptors

![Diagram of an acceptor with ballot_num: 0 and accepted:[]]

- ballot_num: 0
- accepted: []
Acceptors

p1a(0.1) → Acceptor

ballot_num: 0.1
accepted: []
Acceptors

p1a(0.1)

p1b([])

Acceptor

ballot_num: 0.1
accepted: []
Acceptors

Acceptor

ballot_num: 0.1
accepted:[]
Acceptors

p1a(0.0) → Acceptor

ballot_num: 0.1
accepted: []
Acceptors

p1a(0.0)

Acceptors

ballot_num: 0.1
accepted:[]

Nope!
Acceptors

acceptor

ballot_num: 0.1
accepted:[]
Acceptors

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

Acceptors

\[ \text{ballot_num}: 0.1 \]

\[ \text{accepted}:[\] \]
Acceptors

p2a(<0.1, 0, A>)

Acceptors

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

p2a(<0.1, 0, A>)

Acceptors

OK!

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

Acceptor

```
ballet_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>)

Acceptor

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

Nope!
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
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: []

Acceptor

Nope!

Acceptor

Nope!

Acceptor
Leader: Getting Elected

Leader

active: false
ballot_num: 1.0
proposals: []

Acceptor

Acceptor

Acceptor
Leader: Getting Elected

Leader

active: false
ballot_num: 1.0
proposals: []

Or...

Acceptors
Leader: Getting Elected

Leader

active: false
ballot_num: 0.0
proposals: []

Acceptor

OK([])!

Acceptor

OK([])!

Acceptor
Leader: Getting Elected

**Leader**

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

**Acceptors**

- Acceptor
- Acceptor
- Acceptor
Paxos Made Moderately Complex Made Simple
Paxos Made Moderately Complex Made Simple
Leader: Handling proposals

Leader

active: true
ballot_num: 0.0
proposals: []

Op1 should be A
(A = “Put k1 v1”)

Replica

Acceptors
Leader: Handling proposals

**Active:** true
**Ballot Num:** 0.0
**Proposals:** \([<1, A>]\)
Leader: Handling proposals

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

Acceptors
p2a(<0.0, 1, A>)

Replica
Leader: Handling proposals

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

Replica

Acceptor

Acceptor

Acceptor

Nope!

Nope!

Nope!
Leader: Handling proposals

**Leader**

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

**Acceptor**

**Replica**
Leader: Handling proposals

Leader

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

Or...

Acceptor

Acceptor

Acceptor

Replica
Leader: Handling proposals

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

- Acceptor

- Acceptor

- Acceptor

- Replica

OK!
Leader: Handling proposals

Leader

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

Op1 is A

Replica

Replica

Replica

Acceptor

Acceptor

Acceptor
Paxos Made Moderately Complex Made Simple
Election revisited

Leader

active: false
ballot_num: 3.0
proposals: [<1, B>]

Acceptator

ballot_num: 2.1
accepted: [<2.1, 1, A>]
Election revisited

Leader

\[
\begin{align*}
\text{active:} & \quad \text{false} \\
\text{ballot\_num:} & \quad 3.0 \\
\text{proposals:} & \quad [<1, B>] \\
\end{align*}
\]

\[
\text{p1a}(3.0)
\]

Acceptor

\[
\begin{align*}
\text{ballot\_num:} & \quad 2.1 \\
\text{accepted:} & \quad [<2.1, 1, A>] \\
\end{align*}
\]
Election revisited

Leader

active: false
ballot_num: 3.0
proposals: [<1, B>]

Acceptor

ballot_num: 3.0
accepted:[<2.1, 1, A>]
Election revisited

Leader  OK([<2.1, 1, A>])  Acceptor

active: false
ballot_num: 3.0
proposals: [<1, B>]

ballot_num: 3.0
accepted:[<2.1, 1, A>]
Election revisited

Leader

active: true
ballot_num: 3.0
proposals: [1, A]

Acceptor

ballot_num: 3.0
accepted: [2.1, 1, A]
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”)

Replica

slot_out

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

slot_in

Replicas

Leader

Replica

slot_out slot_in

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

Replicas

Leader

decision(4, “Put k3 v3”)

Replica

slot_out slot_in

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

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

Paxos Made Moderately Complex Made Simple

- Client $\kappa$
- Replicas $\rho_1$, $\rho_2$
- Leader $\lambda$
- Acceptors $\alpha_1$, $\alpha_2$, $\alpha_3$

- Request
- Propose
- Adopted
- Scout
- Commander
- Decision
- Response
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!
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
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 is ready

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