Paxos Made Moderately Complex

Jeremy Rubin

Simple State Machine

```
kv = {}
while(1){
    m, from = read_message()
    switch (m.type){
    PUT:
        kv[k] = v;
        from.reply(PUT_OK);
    GET:
        from.reply(kv[k);
    }
}
```

Replicated State Machine

- Run Two Copies

What if one sees:

Put(1, “A”),Put(1, “B”),Get(1)

And the other

Put(1, “B”),Put(1, “A”),Get(1)

We want a way to agree on arguments

What is Paxos

- Fault Tolerant Consensus
- Need a majority
  - 2F+1 nodes to tolerate F failures and make progress
  - F+1 nodes to just tolerate and not make progress

How Does Paxos Work

Paxos Roles

- Client
  - Simply communicates that it would like some action done to the cluster
  - Makes sure action goes through
  - Processes State Machine
- Proposer
  - Tries to get a value from a client accepted
- Acceptor
  - Persistent storage
- Learner
  - Gets results from the majority

Proposer

```
proposer(v):
  v’ = v
  while not decided:
      choose n, unique and higher than any n seen so far
                                  (eg, i*n_peers+my_id +1)
      send prepare(n) to all servers including self
      if prepare_ok(n, na, va) from majority:
          v' = va with highest na; choose own v otherwise
          send accept(n, v') to all
      if accept_ok(n) from majority:
          send decided(v') to all
```

Acceptor

Persistent state on each node:
np --- highest prepare seen
na, va --- highest accept seen

```
acceptor's prepare(n)
handler:
    if n > np
        np = n
        reply prepare_ok(n, na, va)
    else
        reply prepare_reject
```

```
acceptor's accept(n, v)
handler:
    if n >= np
        np = n
        na = n
        va = v
        reply accept_ok(n)
    else
        reply accept_reject
```

Why Paxos Doesn't Work

- This is an Academic protocol, not a battle-ready implementation
- How can we get it 'hardy'.

Single Instance v.s. Slots

- In the presented algorithm, it only works for a single argument, instead we can run many instances with an integer slot id

State Reduction

- What do we need to store?
- If we're running a state machine, once all machines have applied some argument we can forget it

Leader Election

- Rather than force Paxos consensus on every argument, we can elect a leader to have control for a certain duration
- If leader times out, we can run Paxos again to elect a new one and fill in the old slots with NOP
- Tradeoffs: letting leaders get work done v.s. quick detection after fault

Failure Recovery

- In standard Paxos, static cluster is assumed
- We may want to swap out faulty nodes
- Suppose node A fails, we can use Cluster-A to agree on a replacement for A, and then send them a snapshot of the state machine to catch up

Read Only Optimization

- We have to put read operations in log otherwise you could run into
  - Node[a].get(k) != Node[b].get(k)
- But what if we make a read operation leader and redirect all reads to them? They will be consistent. What if we shard reads based on key, etc.

Resources

- http://nil.csail.mit.edu/6.824/2015/notes/paxos-code.html
- http://people.csail.mit.edu/matei/courses/2015/6.S897/readings/paxos-moderately-complex.pdf