CPSC 465/565 Theory of Distributed Systems

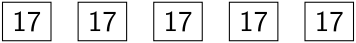
James Aspnes

2023-09-25

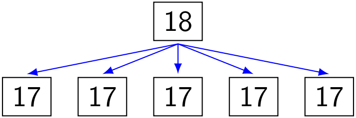
Today's exciting topics

- Replicated state machines
- Paxos

Replicated state machines (Lamport 1978)



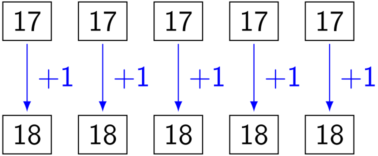
- Want multiple copies of data to protect against failures.
- But how to ensure identical copies?
- Naive approach: recopy all data:



But what if data is really big?

Replicated state machines

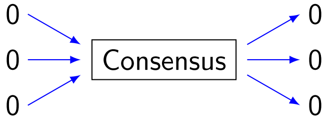
Update data by applying small operations:



- Still trying to defend against crash failures.
- Same operation at every copy $\Rightarrow$ same new states.
- Processes must agree on the sequence of changes.
- $\Rightarrow$ Must solve consensus!

It's a shame that FLP says we can't do this.

How to do asynchronous consensus despite FLP



Recall FLP says any asynchronous protocol with one crash failure violates at least one one of:

- Agreement: we need this
- Validity: we need this
- Termination: we might not need this

We'll look for a protocol that satisfies agreement and validity always, termination if things go well.

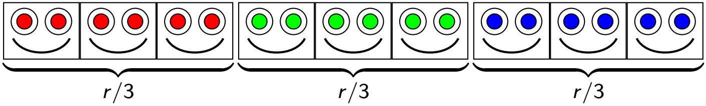
Paxos (Lamport 1990, 1998 2001)

- “The Part Time Parliament” 1990: joke?
- “Paxos Revisited” 1998: incomprehensible
- “Paxos Made Simple” 2001: best thing ever!

Idea: Use majority of **accepters** to ratify agreement.

- If majority S accepts v
- and majority T accepts v'
- then $S \cap T \neq \emptyset$.
- So $v = v'$ if each accepter only votes for one value.

What if we can't get a majority?



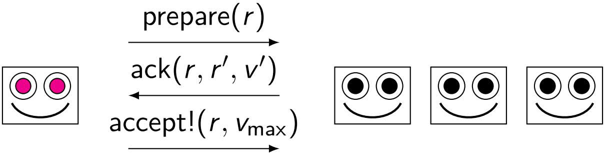
- Washington model: government shutdown, societal collapse.
- Westminster model: new election.

We'll go with Westminster: If one round fails, start a new round.

- Higher-numbered rounds override lower-numbered rounds.
- Majority in round r becomes only option in $r > r'$.

Note: *Higher* round does not necessarily mean *later* round.

Structure of round r



- Proposer (unique to this round) sends prepare(r).
  - "Can I start round r?"
- Accepters (fixed group) respond with ack(r, r', v') *if* they haven't seen a round $> r$.
  - "Fine with me, but I previously accepted v' in round $r' < r$."
- Proposer collects majority and finds highest $\langle r_{\max}, v_{\max} \rangle$.
- Proposer sends accept!($r, v_{\max}$) to accepters.
  - "Please accept $v_{\max}$ in round r."
- Each accepter accepts unless it saw a round $> r$.

Core mechanism doesn't have accepters tell anybody they accepted, but we can add accepted(r, v) messages if needed.

Paxos algorithm: proposer code

```
procedure propose(r, v)
    // Issue proposal number r with value v
    // Assumes r is unique
    send prepare(r, v) to all accepters
    wait to receive ack(r, v', r_{v'}) from a majority of accepters
    if some v' is not ⊥ then
        v ← v' with maximum r_{v'}
    send accept!(r, v) to all accepters
```

Paxos algorithm: accepter code

procedure accepter()
  initially do
    $r_{\text{ack}} \leftarrow -\infty$; $v \leftarrow \perp$; $r_v \leftarrow -\infty$
  upon receiving prepare(r) *from* p **do**
    if $r > \max(r_{\text{ack}}, r_v)$ **then**
      `// Respond to proposal`
      send ack(r, v, r_v) to p
      $r_{\text{ack}} \leftarrow r$ `// max round we've acknowledged`
  upon receiving accept!(r, v') **do**
    if $r \geq \max(r_{\text{ack}}, r_v)$ **then**
      `// Accept proposal`
      send accepted(r, v') to all learners
      if $r > r_v$ **then**
        `// Update highest accepted proposal`
        $\langle r_v, v \rangle \leftarrow \langle r, v' \rangle$

Paxos decision value

- By default, Paxos doesn't have individual processes decide.
- Instead, decision value is whatever is accepted by majority.
- Can have "learners" detect this through notification.

Requirements:

- Validity: Any value accepted by a majority was an input.
  - Proof: Show an invariant that any value v in the protocol started out as a proposer's input.
- Agreement:
  - If v is accepted by a majority in r,
  - and v' is accepted by a majority in r',
  - then $v = v'$.

Proof of agreement is trickier.

Proof of agreement

Claim: If v is accepted by a majority in round r, then for any $r' \geq r$, any accept!(r', v') message has $v' = v$.

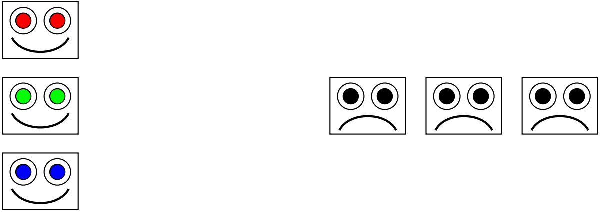
Proof: By induction on r'. Base case $r' = r$ is trivial as usual.

Induction step: Let $r' > r$:

- Round r' proposer $p_{r'}$ got ack$(r', -, -)$ from a majority S.
- But a majority T accepted v in r.
- So $\exists a \in S \cap T$ that accepted v in r and sent ack$(r', -, -)$.
- What did a tell $p_{r'}$?
  - $r' > \max(r^a_{\text{ack}}, r^a_v)$: ack$(r, v^a, r^a_v)$ where $r \leq r^a_v < r'$
    - Induction hypothesis says $v^a = v$.
  - $r' \leq \max(r^a_{\text{ack}}, r^a_v)$: nothing!
    - Contradicts assumption a sent ack$(r', -, -)$.
- $p_{r'}$ picks either v from r^a_v or picks some v' with bigger r'_v.
  - In second case, $r'_v < r$ or sender wouldn't have sent it.
  - Induction hypothesis again says $v' = v$.
- Any value accepted in r' = value sent by $p_{r'} = v$.

What about termination?

Maybe proposer crashes.



Maybe too many proposers.

- Round numbers keep going up.
- Each proposal is blocked by the next.

Standard asynchronous model doesn't let us avoid this.

Solution: Ω failure detector

Failure detector: **oracle** that directly informs processes of failures.

Ω failure detector:

- Every process always believes some process p is leader.
- Eventually all processes believe the same p is leader.
- Eventually p actually is non-faulty.

With Ω:

- Don't act as proposer unless I think I'm the leader.
- Use nack messages to learn about higher-numbered rounds.
- Pick a round number that is even higher.
- If I am the agreed-upon leader process:
  - No conflict with other proposers.
  - Proposer doesn't fail $\Rightarrow$ majority accepts (if majority non-faulty).

(We'll see more about failure detectors next lecture.)

How to implement Ω?

- Can't implement it directly in asynchronous model.
- In practice: use a lot of timeouts + a leader election algorithm.

Popular practical solution: Raft (Ongaro and Osterhout 2014).

Multipaxos

Recall we wanted consensus for replication:

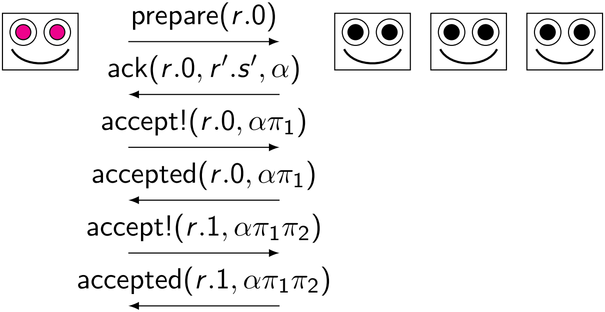
$$\pi_1 \longrightarrow \boxed{\text{Paxos}} \longrightarrow \pi_1$$

$$\pi_1\pi_2 \longrightarrow \boxed{\text{Paxos}} \longrightarrow \pi_1\pi_2$$

Two ideas:

- Don't use new instance of Paxos for each new operation
  - Have proposer add new op to old list.
  - Use existing prepare/ack mechanism to learn old list.
- Don't bother with prepare/ack if proposer unchanged.
  - Reduces round to just accept!/accepted.

Multipaxos: unchanged leader optimization



- If leader unchanged, no need for prepare.
- What if leader changes?
  - Use major.minor version numbering scheme $r.s$.
  - Same leader: increment minor version s.
  - New leader: pick larger major version r'.

Reduces time to one round-trip per operation in good case.

Next time

Failure detectors!