CSC2/458 Parallel and Distributed Systems
Paxos

Sreepathi Pai

April 19, 2018

URCS

Outline

State Machine Replication (SMR)

Paxos

Outline

State Machine Replication (SMR)

Paxos

Tolerating faults using SMR

- Use replicas
- Each replica runs a deterministic algorithm
- If all replicas:
  - starts in the same initial state
  - execute messages in the *same order*
  - then a majority vote among them will allow fault tolerance
- t faults can be tolerated by $2t + 1$ machines

Key problem

How can replicas always agree on the same order in the presence of faults?

Hint: this is the consensus problem

FLP

They can't: FLP theorem tells us there is no deterministic consensus algorithm that works even in the face of one failure.

What does “works” mean?

Recall: To prove FLP, we showed that all algorithms could always get trapped in states where they made no decision.

Liveness is impossible, what about safety?

Can we build a distributed system where if replicas agree, they will all agree on the same order? Even in the presence of failures?

Safety/Correctness properties

From Lamport (2001):

- Only a value that has been proposed may be chosen
- Only a single value is chosen
- A process never learns that a value has been chosen unless it actually has been

Assumptions

- Asynchronous communication
- Non-Byzantine Fail-stop failures
  - However, machines have stable memory that can tolerate failures (why?)
  - Byzantine failures: arbitrary failures
- Messages can take:
  - delayed,
  - duplicated
  - lost
  - but NOT corrupted

Outline

State Machine Replication (SMR)

Paxos

Setup

- Three classes of "agents"
  - Proposers: proposes a value v
  - Acceptors: accepts a value v that is proposed
  - Learners: learns that a value v was accepted
- These agents may be mapped to the same process, so a process could play all three roles

Accepting

- As an acceptor, you may accept a value that is not accepted (i.e. not *chosen*) by the majority.
  - Multiple proposers, different values
  - implies multiple rounds of acceptance
- But you must accept a value
  - Single proposer, one value
  - i.e., you don't know if there are other proposers

Requirement 1

As acceptor, accept the first proposal you receive, [but also accept multiple proposals, if they have the same value you accepted.]

Invariants

Let proposals have a unique number n in the system $n : v$.

Ensure that if a proposal with value v is chosen (i.e. accepted by majority), then every higher-numbered proposal that is chosen has the value v

You can ensure this by:

- Acceptors only accept higher-numbered proposals if these have the chosen value v
- Proposers only propose v in their higher-numbered proposals if v was chosen

- How do we ensure this (esp. if we're a proposer)?

Invariants

The two invariants hold, if when proposal $n : v$ is issued:

- no majority set of acceptors has accepted a proposal $< n$
  - otherwise, we couldn't propose $n : v$ unless v was chosen
- v has been chosen by a set of acceptors with a proposal $< n$

The Proposer's Algorithm

- Proposer sends a "prepare" request, using a value n
- An acceptor who responds to this message:
  - promises that it will not accept a proposal $< n$
  - if it has already accepted a proposal $< n$, it sends the value v that has been accepted
- If a majority of acceptors respond, a proposal is made ("accept"):
  - $n : v$ where v is the value of the highest-numbered proposal accepted so far
  - or is the value of the proposer itself

The Acceptor's algorithm

- An acceptor responds to a prepare if its n is greater than any n' it has seen so far
- An acceptor accepts a proposal numbered n iff:
  - it has not responded to a prepare request $> n$
- It can ignore:
  - prepare requests with n' where $n' < n$ and it has already responded to prepare n
  - duplicate prepare requests

Learners

- Learners learn chosen value by (e.g.) broadcast
- But could also use a single distinguished learner that communicates with other learners

Stable storage

- An acceptor must remember n, below which it will not accept
  - save this before responding to prepare
- An acceptor must remember $n' : v$, which it has accepted
  - save this before accepting
- A proposer must remember all proposal numbers it has used in the past
  - and must not reuse them
  - save this before proposing

Progress

- Proposal p prepares $n1$ and succeeds
- Proposal q prepares $n2 > n1$ and also succeeds
  - Causing accepts of p to fail, since majority will not accept $n1 < n2$
- Proposal p restarts with $n3 > n2$
  - Causing accepts of q to fail, since majority will not accept $n2 < n3$
- *ad infinitum*

Distinguished Proposers

- To ensure progress, elect a distinguished proposer
  - Must have access to a majority of acceptors
  - eventually will have a proposal accepted
- "If enough of the system (proposers, acceptors, communication network) is working properly", liveness can therefore be achieved...
  - This is not guaranteed
- But safety is guaranteed.

Acknowledgments

Lecture largely follows the treatment in Lamport (2001), "Paxos made simple"