Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~magoutis
Order on state updates
Paxos algorithm

• Way to build fault-tolerant distributed systems
  – Replicated state machines (RSM)

• Consensus via message exchange
  – Asynchronous: no timing guarantees
  – Network can delay, reorder, lose (but not corrupt) packets

• Can guarantee safety
  – Replicas will agree on a single value

• Need additional assumptions to ensure progress
Informally

• Three roles: Proposer, acceptor, learner

• Simplest, but fault-intolerant solution: single acceptor

• With >1 acceptors, agreement by a majority required

• If single value proposed, that value should be chosen
  – Thus, an acceptor must accept the first value proposed to it

• However, this may lead to fragmented electorate
  – Multiple proposals by each proposer should be possible
  – Identify each proposal by a unique integer N
Informally

• After consensus, an acceptor cannot change its mind
  – A value is chosen when single proposal with that value
    accepted by a majority of the acceptors

• Allow multiple proposals to be chosen, but guarantee
  that all chosen proposals have the same value
Paxos setup

- Be able to agree in the presence of up to $f$ failures
- $2f+1$ nodes
- Agreement when majority $(f+1)$ agrees on a value
Need to try to get a majority to accept
Informally

- Allow multiple proposals to be chosen, but guarantee that all chosen proposals have the same value.

- If proposal $N$ with value $\nu$ is chosen, every higher numbered proposal issued by any proposer should have value $\nu$.

- A proposer wanting to issue a proposal numbered $N$ must learn the highest-numbered proposal $< N$ (if any) that has been or will be accepted by a majority.
Informally

- A proposer wanting to issue a proposal numbered $N$ must learn the highest-numbered proposal $< N$ (if any) that has been or will be accepted by a majority
  - Easy to learn about values already accepted
  - Hard to predict the future

- Control the future by extracting a promise that there will not be any acceptances of proposals $< N$
Paxos – phase 1

- Client initiates a prepare request.
- Proposer receives the prepare request.
- Acceptors and learners acknowledge the prepare request.
- The highest-numbered prepare request is acknowledged.
- The highest-numbered proposal is accepted.
- The value $v'$ is written to stable store.
Paxos – phase 2

propose $N, \nu$

$N, _$

$N, \nu$

value $\nu'$
Paxos – communicate agreement

decide $N, \nu$

client

proposer

acceptor

learner

$N, \nu$

decide $N, \nu$

value $\nu'$

client

proposer

acceptor

learner

$N, \nu$

decide $N, \nu$

proposer

acceptor

learner

$N, \nu$

decide $N, \nu$

proposer

acceptor

learner

$N, \nu$
Paxos – majority learns outcome

1. Client sends a request to the proposer.
2. The proposer sends a value to the acceptor.
3. The acceptor sends the value to the learner.
4. The learner sends the value to the client.
5. The client learns the outcome.

Value $v'$
Paxos – learning chosen value

client

proposer

acceptor

learner

prepare $N'$

value $v'$

learner

prepare $N'$

$N', v$

$N', _$

$N', _$

$N', _$

proposer

acceptor

learner

prepare $N'$

prepare $N'$

prepare $N'$

prepare $N'$
Paxos – propagate chosen value
Paxos – everyone learns outcome
Example

ballots: xxxx00 xxxx01 xxxx02

proposers

\( v \)

\( v' \)

\( v'' \)

acceptors

\( (.,.) \)

\( (.,.) \)

\( (.,.) \)

proposers

\( v \)

\( v' \)

\( v'' \)

\( \text{ACK-pre} \ (0,.) \)

\( \text{ACK-pre} \ (1,.) \)

\( \text{ACK-pre} \ (1,.) \)

acceptors

\( (0,v_0) \)

\( (1,.) \)

\( (1,.) \)

proposers

\( v \)

\( v' \)

\( v'' \)

\( \text{ACK-pre} \ (0,.) \)

\( \text{ACK-pre} \ (0,.) \)

\( \text{ACK-pre} \ (0,.) \)

acceptors

\( (0,.) \)

\( (0,.) \)

\( (.,.) \)

proposers

\( v \)

\( v' \)

\( v'' \)

\( \text{ACK-pre} \ (1,.) \)

\( \text{ACK-pre} \ (2,.) \)

\( \text{ACK-pre} \ (2,.) \)

acceptors

\( (0,v_0) \)

\( (1,.) \)

\( (1,.) \)

proposers

\( v \)

\( v' \)

\( v'' \)

\( \text{pre} \ (2) \)

\( \text{pre} \ (2) \)

\( \text{pre} \ (2) \)

acceptors

\( (0,v_0) \)

\( (1,.) \)

\( (1,.) \)
Example (contd.)

proposers

\[ v \]
\[ v' \]
\[ v'' \]

acceptors

\[ (0,v_0) \]
\[ (2,v'_1) \]
\[ (2,\_\_) \]

proposers

\[ v \]
\[ v' \]

acceptors

\[ (3,v_0) \]
\[ (3,v'_2) \]

proposers

\[ v \]
\[ v' \]
\[ v'' \]

acceptors

\[ (0,v_0) \]
\[ (2,v'_2) \]
\[ (2,v'_2) \]

proposers

\[ v \]
\[ v' \]

acceptors

\[ (3,v_0) \]
\[ (3,v'_2) \]

proposers

\[ v \]
\[ v' \]

acceptors

\[ (0,v_0) \]
\[ (2,v'_2) \]

proposers

\[ v \]
\[ v' \]

acceptors

\[ (3,v'_3) \]
\[ (3,v'_3) \]

\[ \text{pro (2,v')} \]
\[ \text{pro (2,v')} \]
\[ \text{ACK-pre (3,v_0)} \]
\[ \text{ACK-pre (3, v'_2)} \]
\[ \text{ACK-pro (2,v'_2)} \]
\[ \text{ACK-pro (2,v'_2)} \]
\[ \text{pro (3,v')} \]
\[ \text{pro (3,v')} \]
\[ \text{pre (3)} \]
\[ \text{pre (3)} \]
\[ \text{pre (3)} \]

\[ \text{ACK-pro (3,v'_3)} \]
\[ \text{ACK-pro (3,v'_3)} \]
Lamport: implementing a state machine

- **How to run multiple instances of Paxos**
  - Assume the existence of a distinguished proposer (leader)
  - A leader will run Paxos for a number of instances
  - The leader may crash, at which point there may be gaps in the chosen instances (1-134, 138, ..)
  - A new leader will try to fill in those slots or propose *no-op*
  - As soon as gap fills, commands can be executed

- **Multi-Paxos**
  - New leader: execute phase 1 for infinitely many instances
  - Acceptors can respond with reasonably short messages
  - Cost of Paxos effectively the cost of executing phase 2
Multi-Paxos

Block acceptance of proposal # < N & learn accepted values

If a majority has not accepted anything for instances > I

Skip prepare phase until a propose is rejected!
Multi-Paxos

Servers play all roles

Replicas write to disk prior to sending ACK