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

value $v$

value $v'$
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
- Proposed prepare N
- Accept prepare N
- Written to stable store

Highest-numbered proposal accepted
Highest-numbered prepare request acknowledged
Written to stable store
Paxos – phase 2

client

propose $N, v$

client

value $v'$

propose $N, v$

proposer

acceptor $N, _$

learner

proposer

acceptor $N, v$

learner

propose $N, v$

proposer

acceptor

learner

propose $N, v$

proposer

acceptor

learner
Paxos – communicate agreement
Paxos – majority learns outcome

client

proposer
acceptor
learner

N, v

client

value v'

proposer
acceptor
learner

N, v

proposer
acceptor
learner

N, v
Paxos – learning chosen value

N', v

prepare N'

proposer
acceptor
learner

N', _

prepare N'

proposer
acceptor
learner

N', _

prepare N'

proposer
acceptor
learner

value v'

prepare N'

client
Paxos – propagate chosen value

client

proposer
acceptor
learner

propose \(N', v\)

proposer
acceptor
learner

propose \(N', v\)

proposer
acceptor
learner

propose \(N', v\)

propose \(N', v\)
Paxos – everyone learns outcome
Example

ballots: xxxx00  xxxx01  xxxx02
proposers
\[ v \quad v' \quad v'' \]
acceptors
\[ (\_\_\_\_\_) \quad (\_\_\_\_\_) \quad (\_\_\_\_\_) \]
proposers
\[ v \quad v' \quad v'' \]
acceptors
\[ (0,v_0) \quad (1,\_\_\_) \quad (1,\_\_\_) \]

proposers
\[ v \quad v' \quad v'' \]
acceptors
\[ (0,\_\_\_) \quad (0,\_\_\_) \quad (\_\_\_\_\_) \]
proposers
\[ v \quad v' \quad v'' \]
acceptors
\[ (0,v_0) \quad (1,\_\_\_) \quad (1,\_\_\_) \]

proposers
\[ v \quad v' \quad v'' \]
acceptors
\[ (0,\_\_\_) \quad (0,\_\_\_) \quad (\_\_\_\_\_) \]
proposers
\[ v \quad v' \quad v'' \]
acceptors
\[ (0,v_0) \quad (2,v_1') \quad (2,\_\_\_) \]
Example (contd.)

proposers

\(v\)  \(v'\)  \(v''\)

\(\text{pro } (2,v')\)

\(\text{pro } (2,v')\)

acceptors

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

proposers

\(v\)  \(v'\)

\(\text{ACK-pre } (3,v_0)\)

\(\text{ACK-pre } (3, v'_2)\)

acceptors

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

proposers

\(v\)  \(v'\)  \(v''\)

\(\text{ACK-pro } (2,v'_2)\)

\(\text{ACK-pro } (2,v'_2)\)

acceptors

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

proposers

\(v\)  \(v'\)

\(\text{pro } (3,v')\)

\(\text{pro } (3,v')\)

acceptors

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

proposers

\(v\)  \(v'\)

\(\text{pre } (3)\)

\(\text{pre } (3)\)

acceptors

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

proposers

\(v\)  \(v'\)

\(\text{ACK-pro } (3,v'_3)\)

\(\text{ACK-pro } (3,v'_3)\)

acceptors

\((3,v'_3)\)  \((3,v'_3)\)
How to run multiple instances of Paxos

- A new leader may know the outcome of only some instances
- 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