Stellar Consensus Protocol

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

September 25, 2018

Lecture Plan

- Consensus Protocol Terminology
- Related Protocols for Context
  - Paxos
  - PBFT
- Federated Byzantine Agreement Model
- Federated Voting
- Stellar Consensus Protocol (in brief)

Consensus Protocol Terminology

- **Agents**: Parties interested in achieving consensus
- Each agent has an input
- Agents use protocol to agree on one of the inputs
- Each agent decides on a chosen value
- Agent failure modes
  - Stopping failure
  - Byzantine failure
- **Safety**
  - **Agreement**: No two non-faulty agents decide on different values
  - **Validity**: If all non-faulty agents have the same input v, then v is the only possible decision value
- **Liveness**
  - **Termination**: All non-faulty agents eventually decide
- Asynchronous network model
  - Messages may be delayed, duplicated, lost, reordered
  - No corrupted messages

Paxos

Paxos

- Consensus protocol for non-Byzantine agents and asynchronous network
- Proposed by Leslie Lamport in 1989
- Number of agents is known
- Agents act as proposers, acceptors, or learners (multiple roles allowed)
- Proposers propose values
- Acceptors accept a value if requested by a proposer
- Once a majority of acceptors has accepted a value, consensus has been achieved
- Learners are interested in learning about consensus values
- Challenges
  - Messages indicating acceptance may be lost
  - Consensus may be achieved without proposers finding out
  - Multiple proposers may be simultaneously proposing values

Paxos Protocol Phase 1

- Proposal made by proposers have a proposal number n from a totally ordered set
- Phase 1
  - Proposer sends a **prepare** request with number n to all acceptors
  - If acceptor receives a prepare request with number higher than any other previous prepare request, then
    1. it promises to not accept any more proposals with number less than n and
    2. returns highest-numbered proposal value (if any) it has accepted
- Example

Prop. No.	Value	Agent 1	Agent 2	Agent 3
1	7	7	$\langle\rangle$	$\langle\rangle$
2	8	8	$\langle\rangle$	$\langle\rangle$
3	9	$\langle\rangle$	$\langle\rangle$	9

For proposal 4, highest-numbered proposal accepted among all responses is used

Paxos Protocol Phase 2

- Phase 2
  - If proposer receives a response to its prepare request from a majority of acceptors, then it **either**
    - sends an **accept** request to each these acceptors with value v which is the highest-numbered proposal among the responses or
    - sends an **accept** request with any value if responses reported no proposals.
  - If acceptor receives an accept request for a proposal number n, it accepts the proposal unless it has already responded to a prepare request having number greater than n.
- Example 1

Prop. No.	**Value**	**Agent 1**	**Agent 2**	**Agent 3**
1	7	7	$\langle\rangle$	$\langle\rangle$
2	8	8	$\langle\rangle$	$\langle\rangle$
3	9	$\langle\rangle$	$\langle\rangle$	9

- For proposal 4, proposer can send accept request with
  - 8 if only agents 1 and 2 respond
  - 9 if only agents 2 and 3 respond

Paxos Protocol Phase 2

- Phase 2
  - If proposer receives a response to its prepare request from a majority of acceptors, then it **either**
    - sends an **accept** request to each these acceptors with values v which is the highest-numbered proposal among the responses or
    - sends an **accept** request with any value if responses reported no proposals.
  - If acceptor receives an accept request for a proposal number n, it accepts the proposal unless it has already responded to a prepare request having number greater than n.
- Example 2

Prop. No.	Value	Agent 1	Agent 2	Agent 3
1	8	8	$\langle\rangle$	$\langle\rangle$
2	9	9	$\langle\rangle$	9
3	9	$\langle\rangle$	$\langle\rangle$	9

- For proposal 4, proposer can send accept request with only value 9

Paxos Protocol

- Phase 1
  - Proposer sends a **prepare** request with number *n* to all acceptors
  - If acceptor receives a prepare request with number higher than any other previous prepare request, then
    1. it promises to not accept any more proposals with number less than *n* and
    2. returns highest-numbered proposal value (if any) it has accepted
- Phase 2
  - If proposer receives a response to its prepare request from a majority of acceptors, then it **either**
    - sends an **accept** request to each these acceptors with values *v* which is the highest-numbered proposal among the responses or
    - sends an **accept** request with any value if responses reported no proposals.
  - If acceptor receives an accept request for a proposal number *n*, it accepts the proposal unless it has already responded to a prepare request having number greater than *n*.
- Learners need messages from a majority of acceptors to find out about consensus value

Proposer Selection

- Lamport describes a method using timeouts
  - Each agent broadcasts its ID and the one with the highest ID is the proposer
- Presence of multiple proposers cannot violate safety but can affect liveness
  - Proposer p completes phase 1 for proposal number n_1
  - Proposer q completes phase 1 for proposal number $n_2 > n_1$
  - Proposer p's phase 2 messages are ignored
  - Proposer p completes phase 1 for new proposal with number $n_3 > n_2$
  - Proposer q's phase 2 messages are ignored
  - And so on
- **FLP Impossibility Theorem**: No deterministic consensus algorithm can guarantee all three of safety, liveness, and fault-tolerance in an asynchronous system.

Practical Byzantine Fault Tolerance

PBFT

- Proposed in 1999 as an algorithm for state machine replication
  - Each agent is a replica of a state machine
  - Replicas need to achieve consensus on state transitions
- Assumes Byzantine agent failures and weak synchrony
  - Messages may be delayed, duplicated, lost, reordered
  - Delays do not grow faster than t indefinitely
- Guarantees safety and liveness if at most $\lfloor \frac{n-1}{3} \rfloor$ out of n replicas are faulty
  - For f faulty replicas, $3f + 1$ is the minimum number of replicas required
- Let $\mathcal{R}$ be the set of replicas with cardinality $3f + 1$
- Each replica is identified using an integer in $0, 1, \ldots, |\mathcal{R}| - 1$
- The algorithm moves through a sequence of **views**
- Views are numbered sequentially
- In view v, replica with identity $v \bmod |\mathcal{R}|$ is the **primary** and the remaining replicas are **backups**

PBFT Algorithm

- Rough outline
  1. A client sends a request to the primary to invoke a state machine operation
  2. Primary multicasts the request to the backups
  3. Replicas execute the request and send a reply to the client
  4. The client waits for $f + 1$ replies from different replicas with same result
- Three phases in case of non-faulty primary
  - Pre-prepare
  - Prepare
  - Commit
- Pre-prepare phase
  - Primary in view v receives client request m
  - Primary assigns a sequence number n to m
  - Primary multicasts PRE-PREPARE message with m, v, n to all backups
  - Backup accepts PRE-PREPARE message if
    - it is in view v and
    - it has not accepted a PRE-PREPARE message for view v and sequence number n with different request

PBFT Prepare Phase

- Prepare
  - If backup i accepts the PRE-PREPARE message, it enters the prepare phase
  - Multicasts PREPARE message with v, n, m, i to all other replicas
  - Adds both PRE-PREPARE and PREPARE messages to its log
- Define predicate **prepared**(m, v, n, i) to be true if and only if replica i has inserted in its log
  1. a PRE-PREPARE message with m, v, n, and
  2. at least $2f$ PREPARE messages for m, v, n.
- Guarantees that non-faulty replicas agree on total order of requests in a view
  - **Invariant**: If **prepared**(m, v, n, i) is true, then **prepared**(m', v, n, j) is false for any non-faulty replica j where $m' \neq m$
  - **prepared**(m, v, n, i) true $\implies$ at least $f + 1$ non-faulty replicas have sent PREPARE or PRE-PREPARE messages for m, v, n
  - **prepared**(m', v, n, j) true $\implies$ $2f + 1$ replicas have sent PREPARE or PRE-PREPARE messages for m', v, n to j
  - At least one non-faulty replica has sent conflicting PREPAREs or PRE-PREPAREs $\implies$ contradiction

PBFT Commit Phase

- Commit
  - When **prepared**(m, v, n, i) becomes true, replica i multicasts a COMMIT message for m, v, n, i
  - Replicas accept COMMIT messages which match their view and insert them into their logs
  - Replica i executes the operation requested by m when **committed-local**(m, v, n, i) becomes true and all requests with lower sequence number have been executed
- **committed-local**(m, v, n, i) is true if and only if
  1. **prepared**(m, v, n, i) is true and
  2. replica i has accepted $2f + 1$ COMMITs (including its own) for m, v, n
- **committed**(m, v, n) is true if and only if **prepared**(m, v, n, j) is true for all j in some set of $f + 1$ non-faulty replicas
- **Invariant**: If **committed-local**(m, v, n, i) is true for some non-faulty i, then **committed**(m, v, n) is true
- At non-faulty replicas i and j, **committed-local**(m, v, n, i) and **committed-local**(m', v, n, j) cannot both be true for $m \neq m'$

PBFT View Change

- View changes are required when primary replica fails
- View-change algorithm
  1. If client does not receive replies before a timeout, it broadcasts the request to all replicas
  2. If request has already been processed, the replicas resend the reply to client
  3. If request was not received from primary, a backup starts a timer upon receiving the client's request
  4. If the timer expires while waiting for same request from primary, the backup multicasts a view-change message to all replicas
  5. When primary of view $v + 1$ receives $2f$ view-change messages, it multicasts a new-view message and enters view $v + 1$

References

- SCP talk `https://www.youtube.com/watch?v=vmwnhZmEZjc`
- SCP white paper `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`
- *Paxos Made Simple*, Leslie Lamport, `https://lamport.azurewebsites.net/pubs/paxos-simple.pdf`
- *How to Build a Highly Available System Using Consensus*, B. W. Lampson, `https://doi.org/10.1007/3-540-61769-8_1`
- PBFT paper `http://www.pmg.csail.mit.edu/papers/osdi99.pdf`