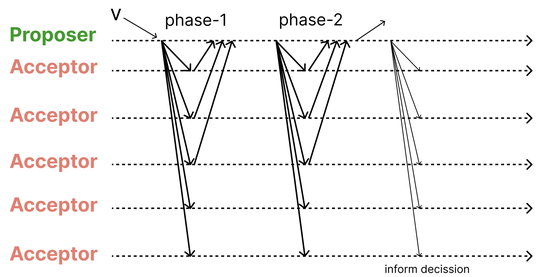
Distributed Consensus Over Secret-Shares 🤝🔐

Fadhil I. Kurnia and **Arun Venkataramani**
fikurnia@cs.umass.edu, arun@cs.umass.edu
University of Massachusetts Amherst, USA

🤝 Paxos - A Distributed Consensus Protocol

Goal: all the nodes agree on the same one value, even if some of the nodes are crashed.



Progress can be made if one proposer and a majority of acceptors are alive.

🕵️ Paxos Reveals the Values to all Acceptors

Issue: the value being agreed upon is revealed to all the acceptors. Agreeing on secret value becomes an important problem when we outsource some of the nodes in public cloud providers that are prone to data breach.

Example use cases: outsourced secret management systems that have to agree on a single consistent password or private key, private backups, outsourced private computation (MPC).

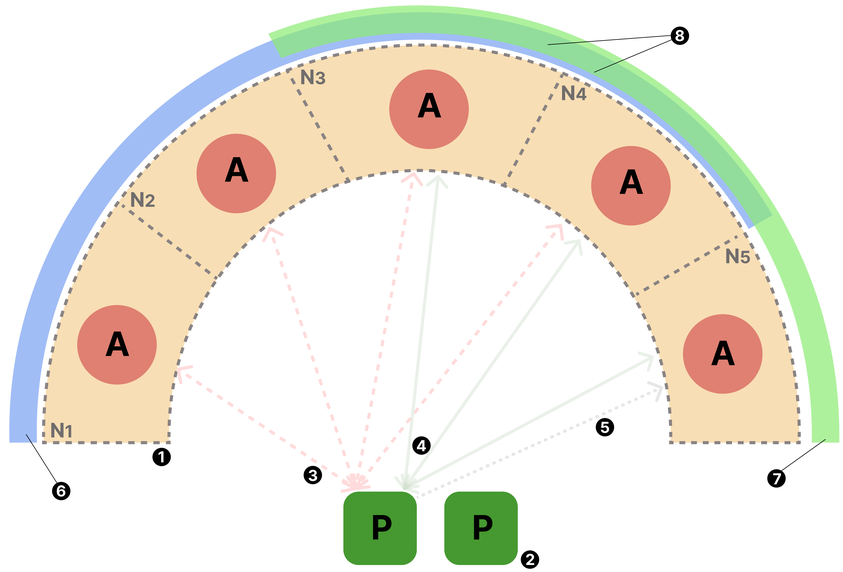
🔑 Encryption Requires Key Escrow

A straighforwad solution is using consensus to agree on the encrypted value (ciphertext).

- ✗ Require an external key management accessible by all the trusted parties.
- ✗ Prone to availability issues, especially if the key get lost.

Combine Secret-Sharing with Consensus

We design a distributed consensus protocol that enables agreement on a secret-shared value with low overhead. We use (***t***,***n***) threshold secret-sharing that generates ***n*** secret-shares, which ***t*** of them are enough to recover the secret.



System Design, Roles and Threat Model

Actors: Proposer, Acceptor, Learner

- **A** Acceptor (combined with Learner), deployed in the untrusted nodes
  Example: N_1, N_2, N_3, N_4, and N_5 in the figure ❶
- **P** Proposer that has unique ballot-number, deployed in the trusted nodes ❷

- Proposers are allowed to know the secret value
- Acceptors, which must not know the value, only know secret-share sent by the proposers.
- The secret value stay private as long as there is no ***t*** colluding Acceptors.
- The protocol metadata (e.g ballot-number, secret-value's id) is public.
- Need to works despite some nodes crashed.
- Acceptors and proposers run the protocol as specified in a non-byzantine manners.

Goal: Eventually, each learner learns a secret-share that represents the same secret value agreed by all the acceptors. The learners can not learn secret-shares of more than one value.

Quorum Size

Phase-1 quorum size $\left\lceil \frac{n+t}{2} \right\rceil$

Phase-2 quorum size $\left\lfloor \frac{n+t}{2} \right\rfloor$

Two quorums in the protocol: Quorum for Phase 1 and Quorum for Phase 2.

This ensures **t** intersection: the proposer P can always recover the previously chosen secret-value from any **t** secret-shares that P receive in Phase 1.

For example, with **n**=5, **t**=2:

❻ Example of a quorum for Phase 1, consists of 4 acceptors

❼ Example of a quorum for Phase 2, consists of 3 acceptors

❽ Quorum for Phase 1 and Quorum for Phase 2 intersects in 2 acceptors

Pairing Secret-share with Original-Ballot

<secret-share, original-ballot>

- ✗ Using hash or random number as the secret-value's id (**sid**) is prone to collission.
- ✓ Use *the original ballot-number* as the **sid**: the ballot-number of the first proposer that proposes the secret-value. Computationally cheaper & collission-free.

🤝🔐 Privacy-Preserving Consensus Protocol

❸ Phase-1: Prepare

P 1. Proposer sends prepare messages **Phase1A{b}** with unique ballot-number **b** to all **n** acceptors.

A 2. Acceptor A when receiving prepare message:
- 2.a. A ignores the prepare message if A previously already received messages with higher ballot-number.
- 2.b. Otherwise, A reply the prepare massage. A puts the highest ballot number **b** and the <secret-share, secret-value's id> pair that A already accepted previously, if any, in the response: **Phase1B{b, <ss, sid>}**

P 3. Proposer P when receiving a quorum of Phase1B messages containing P's ballot-number **b**:
- 3.a. If P receive at least **t** secret-shares **ss** with the same **sid**, and one of them has the highest *accepted ballot*, then P has to reconstruct the secret value: **v** ← Combine($\mathbf{ss}_1$, $\mathbf{ss}_2$, ..., $\mathbf{ss}_t$), and reuse the same **sid**.
- 3.b Otherwise, P can use any secret value **v** in the next phase and use P's current ballot-number as the secret-value's id: **sid** ← **b**. We call this **sid** as *the original ballot-number*.

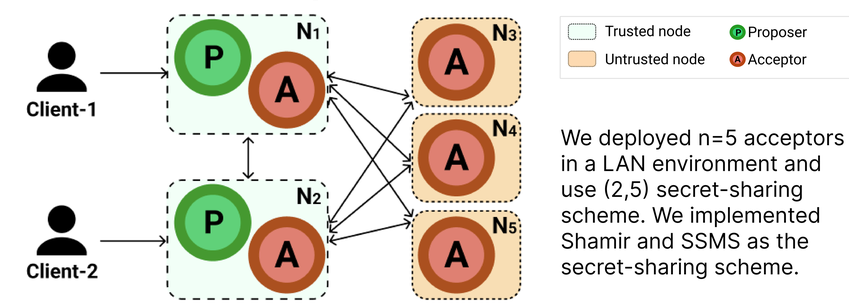
❹ Phase-2: Propose

Proposer P proposes **v**

❺ Commit

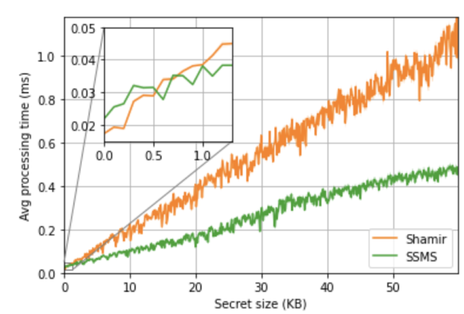
If a proposer P fail to get a quorum of responses, in Phase-1 or Phase-2, P can restart with higher ballot number.

Evaluation Setup

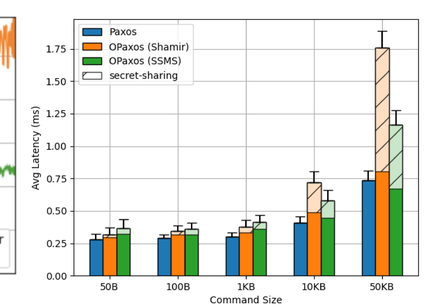


We deployed n=5 acceptors in a LAN environment and use (2,5) secret-sharing scheme. We implemented Shamir and SSMS as the secret-sharing scheme.

Measurement Results



For small values, secret-sharing process induces *small* latency



Integrating secret-sharing into consensus has *negligible* latency overhead