CMPSCI 677 Operating Systems **Spring 2024**

Lecture 19: April 17

Lecturer: Prashant Shenoy *Scribe: Aishwarya Bhave(2023), Yiming Huang(2024)*

Lecture Overview

- Part 1: Consensus
- Part 2: PAXOS
- Part 3 : RAFT

19.1 Consensus

Definition: get a group of processes to agree on something such as database replication etc. Consensus also means getting the set of processes to agree when some of the processes fail via atomic. More formally, we want to achieve reliability in presence of faulty processes:

- Requires processes to agree on data value needed for computation.
- **Examples**: The operation could be anything - whether to commit a transaction, agree on identity of a leader, atomic multicasts, atomic broadcasts, distributed locks.

The failures are in context of crash faults or Fail-stop failures i.e., a process produces correct output while it is running, but the process can hang/go-down and hence will not produce any results. **Note**: When there are no failures, there are protocols like 2 Phase commits that we discussed, to come to an agreement. For instance, committing a transaction.

Byzantine Consensus vs Consensus Byzantine consensus is when we need to come to an agreement in case of processes that are byzantine-faulty i.e., faulty processes continue to run and produce malicious outputs and prevent agreement. Whereas consensus is a benign scenario where some processes fail to respond.

Q: How to decide what consensus protocol to use? Depends on what we're trying to achieve. PAXOS, RAFT can be used for crash faults, for implementing a basic fault tolerance mechanism. Byzantine is more elaborate and used in case where we do not want malicious actors to cause confusion as in case of crypto-currency.

19.1.1 Properties of a Consensus Protocol

- **Agreement**: Every correct process *agrees* on the same value. (fundamental to consensus)
- **Termination**: Every correct process *decides* on some value.
- **Validity**: If all processes *propose* a value(v), all correct processes must *decide* on that value, v.

- **Integrity**:
  - Every correct process *decides* at most one value.
  - If a correct process *decides* on a value, a process must have *proposed* that value.

Q: What does 'all' in validity mean? We will have failures, if any process or say the co-ordinator crashes during agreement, we will not have consensus. But the protocols define that, if we have a majority(and not all) of the nodes up and running and they agree on a value, we have consensus.

Q: Difference between agreement and termination A safety and liveness question, Agreement can agree on null value, it's a safety property, and termination means we make agreement on actual value, so it's a liveness property, we need both for the system to go on.

19.1.2 2PC/3PC Problems

Both two phase commits and three phase commits experience problems in the presence of different types of failures. While the **safety** property can be ensured, the **liveness** properties cannot always be guaranteed due to node failures and network failures: the system will never perform an operation that leads to an inconsistent state (satisfying the safety property) but can still be deadlocked (violating the liveness property). We describe a few caveats associated with each type of commit below.

Two Phase Commit

- It must wait for the coordinator and the subordinates to be running.
- It requires all nodes to vote.
- It requires the coordinator to always be running.
- Either all commit or no one commit.

Three Phase Commit

- It can handle coordinator failures.
- But network failures are still a problem.
- Add an additional stage of pre-commit compared to 2PC when the coordinator receive the commit, and if the coordinator says commit again, all the nodes move to commit stage.

3PC never get confused for abort and commit because of the existence of pre-commit stage. There has been an implicit assumption that there could only be node failures instead of network failures during a two or three phase commit. While a node could crash in the network, the network would never experience any issues. Suppose, however, that the network was partitioned into two partitions due to some problem. Although both partitions will continue to function correctly, each partition cannot communicate with each other. **By definition, if the network is partitioned due to some problem, a two or three phase commit cannot work because every node is required to vote on the answer.**

In order to eliminate such an assumption, we have to revisit the definition of *agreement*. Rather than requiring the vote of every node, we can just require the vote of the majority of nodes. Therefore, if the network were to be separated into two partitions, the partition with the majority of nodes can still continue to function properly. This idea forms the basis of **Paxos**, a consensus protocol. **Instead of requiring every node to vote, Paxos only requires the majority of nodes to vote.**

Replication for Fault Tolerance Technique 1: split all incoming requests among replicas. (If one replica fails, other replicas take over its load) Technique 2: send each request to all replicas. use 2PC, 3PC, Paxos, a replica can produce wrong results

19.2 Paxos: Fault-tolerant agreement

Paxos lets nodes agree on the same value despite node failures, network failures and network delays. but cannot deal with Byzantine Fault **Use-cases include:**

- Nodes agree X is primary (or leader)
- Nodes agree Y is last operation (order operations)

The protocol is widely used in real systems such as Zookeeper, Chubby and Spanner. ***Leader*** is a process that tries to get other processes to agree on a value. For instance, a process says, I propose that the value after computation is X and gets other process to agree that the output after computation is X. Therefore, leader is essentially a proposer. If majority of the processes agree then, the value is agreed upon. If not, then either the leader tries again or some other process becomes a leader and attempts consensus. **Note:** There can be multiple leaders and can attempt to get others to agree on a value.

Question: what method would you want to use this as opposed to other approaches (2PC/3PC) ? *Answer*: This can be used in case of failures in the node as well as multiple leader failures. Since it is a quoram based protocol, it allows leader election even if some processes fail as long as majority of them are up.

19.2.1 Paxos Requirements

Paxos satisfies the following properties:

- Safety (*Correctness*)
  - All nodes must agree on the same value.
  - The agreed upon value must be computed by some node.
  - **Note:** We do not want just trivial consistency i.e.; everyone agrees value is zero or null. Therefore, the value that is agreed upon must be computed by some node.
- Liveness (*Fault Tolerance*)
  - If less than $\frac{n}{2}$ nodes fail, the remaining nodes will eventually reach agreement. This allows the system to make progress in the presence of failures.
  - Note that that liveness is not guaranteed if there is a steady stream of failures as the protocol determines what to do. If a node fails in the middle of the protocol, it must be restarted.
- **Why is agreement hard?** Because even in the face of failures, we still need to reach agreement.
  - The network might be partitioned.

- The leader may crash during solicitation or before announcing the outcome of voting. While the current round will not produce any results, a new leader will be elected through leader election. All nodes will then vote again.
- A new leader may propose different values from the value that had been agreed upon originally.
- Several nodes may become a leader at the same time. This is possible when the network is partitioned due to a network failure. The left half will elect a new leader while the right half will have the old leader, and they will still continue to function properly. Both sides of the partition may agree on different things unfortunately.

19.2.2 Paxos Setup

- Entities: Proposer(leader), acceptor, learner:
  - *Leader* proposes value, solicits acceptance from acceptors.
  - *Acceptors* are nodes that want to agree; announce chosen value to learners
  - *Learners* do not play an active role, but agree on proposed value.
- Proposals are ordered by unique proposal numbers.
  - Node can choose any high number to try and get proposal accepted
  - An acceptor can accept multiple proposals.
    - If a proposal with value v is chosen, all higher proposals have value v.
- Each node maintains:
  - **n_a, v_a:** The highest proposal number and accepted value during that proposal.
  - **n_h:** The highest proposal number seen so far
  - **my_n:** the current proposal number that is in progress.

19.2.3 Paxos Operation : 3 Phase protocol

Phase 1: Prepare Phase Leader understands what other processes have seen or accepted before.

- A node decides to be leader and proposes a value
- Leader chooses my_n >n_h
- Leader sends **<prepare, my_n>** to all nodes. **Note that,** during this, the value proposed is not sent, it's just the prepare message with proposal number.
- Upon receiving **<prepare, n>** at acceptor:
  - If n <n_h: Reply with **<prepare-reject**. (Since, already seen a higher # proposal.)
  - Else:
    - n_h = n (Protocol will not accept proposal lower than n)
    - Reply **<prepare-ok, n_a, v_a >.** (Send back the most recently accepted proposal # and value)
    - Reply can be null, if you haven't seen any proposals yet and this is the first proposal.

Phase 2: Accept Phase

- If leader gets **<prepare-ok>** from majority *(Actions taken by leader)*
  - V = non empty value from the highest n_a received from prepare phase.
  - If V = null, leader can pick any V
  - Send <accept, my_n, V >to all nodes
- If leader fails to get majority **prepare-ok** : Delay and restart paxos.
- Upon receiving <accept, n, V>*(Actions taken by acceptor)*:
  - If n <n_h : Reply with **<accept-reject >**
  - Else : n_a = n; v_a = V, n_h = h; reply **<accept-ok >**

Phase 3: Decide

- If leader gets **<accept-ok >** from majority: Send **<decide, v_a >** to all learners.
- If leader fails to get **<accept-ok >** from a majority: Delay and restart Paxos.

Question: Are we assuming the number of nodes is fixed? Can new nodes join? *Answer*: It is not sure new nodes can join since we are assuming because as mentioned previously here that if you have a steady stream of failures and recovery liveness is not guaranteed so we can have failures but then if new nodes are joining and they suddenly start saying something that they're in participant participate that's a problem then that round will fail the nodes can join but that round will failure to return. It may also cause problems in majority voting.

Q: Can Proposals go on indefinitely? At the beginning, no one has agreed to anything, leader gets null and chooses a value V. Another proposer suggests a value and it gets accepted and so on. Essentially the value will not change and this is similar to electing the same leader over and over again. While anyone can start a proposal at any time, the agreed value will not get affected. However, the phase 3 or decide phase cannot happen if a new proposal with higher proposal number has started making rounds. Nodes may decide to reject the proposal and accept a new one. And this is possible since we can have multiple leaders. Therefore, there must be a gap between decide phase and new proposals for decide phase to happen. To re-iterate, this doesn't change the value however.

Q: What if you have same proposal numbers? Proposal numbers are unique, Paxos will not work if two proposals have same number. We can append PID (process id) to make it unique. This is similar to Lamport's clock ordering to convert partially ordered to fully-ordered events where we append process id.

Properties

- Property 1: any proposal number is unique.
- Property 2: two sets of acceptors have at least one node in common
- Property 3: value sent in phase 2 is value of the highest numbered proposal received in responses in phase 1.