Save this note as PDF

Introduction

Previously, we discussed databases with the assumption that all of our data resides on just one machine or that our data is partitioned over multiple machines. However, in practice, many databases employ replication as an important technique for performance at scale. Replication comes with its own set of challenges, and there are many design decisions that need to be carefully considered. For example, should the database provide strong consistency or eventual consistency? While it is often easier to achieve higher performance while providing only eventual consistency, more burden is placed on application programmers to ensure their wanted semantics. On the other hand, certain databases provide strong consistency in which all replicas behave the same. To ensure strong consistency, consensus protcols such as Paxos and RAFT are needed to keep replicas in accordance with one aother.

Consenus Definition

The consensus problem begins with a set of machines (nodes) that communicate by sending messages over a network. The machines want to all agree on a specific value. For example, the machines may want to agree on what the value of a specific record in the database is. Suppose each machine has a replica of the record. If write requests go to different replicas, it is important that each replica still sees the same values. To determine the chosen value for the record, a consenus protocol must be run within the participating machines. Three specific properties must be satisfied by any consensus protocol:

- Termination: All non-faulty processes eventually decide on a value (i.e., we will make progress)
- Agreement: All processes that decide do so on the same value
- Validity: The chosen value must have proposed by some process that participated in the protocol (i.e., none of the participants is compromised in the security sense).

System Model

Algorithms in distributed sytems typically provide a system model that describes properties of the system. Paxos, which we study in this class, assumes that messages are neither lost nor corrupted, and that messages eventually arrive, even though there is no upper bound on their delay in the network. Additionally, Paxos assumes that machines fail in a non-Byzantine fashion by stopping (i.e., they don't send random, corrupted messages). Furthermore, Paxos requires that in order to tolerate F such failures, at least 2F + 1 replicas are needed in the system. In other words, a majority of machines must not be faulty.

Paxos

The Paxos protocol is a consenus algorithm that works in a series of rounds. In each round, the goal is to achieve consensus by the end of the round. If consensus is not achived in this round, then a new round will be initiated. Each round consists of three phases:

- Phase 1: A leader is elected (Election)
- Phase 2: Leader proposes a value, processes ack (Bill)
- Phase 3: Leader multicasts final value (Law)

Phase 1 (Proposer Election)

The first phase of a Paxos round begins when a particular machine decides it wants to try and become the leader. To become the leader, the machine must first create a ballot with a specific ballot id. It is important that ballot ids are strictly increasing and that different machines do not generate the same ballot id. To guarantee the former, a strictly increasing counter may be used. To guarantee the latter, a typical way to do so is for each machine to append their machine id before the ballot id. Let's call this machine that wants to become the leader the proposer for the Paxos round. After generating the ballot id, the proposer multicasts a PREPARE(ballot_id) message to a majority (e.g., all) of participants, which are other machines in the system.

When a participant recieves the PREPARE(ballot_id) message it needs to determine if it wants to respond to the proposer. If the participant has already promised not to respond to this ballot_id, the participant ignores the PREPARE message. Since Paxos rounds are asynchronous, it is possible that consensus may have already been reached. Thus if the participant has already sent an ACCEPT message (discussed below) for a ballot (old_ballot_id, old_v), the participant must respond with to the proposer with a PROMISE(ballot_id, (old_ballot_id, old_v)) message. Otherwise, the participant sends a PROMISE(ballot_id) message back to the proposer. By sending this PROMISE messsage,

the participant is promising to ignore any ballot ids in future rounds whose ballot id is less than ballot_id. Before sending out the PROMISE message, the participant also logs the ballot id and flushes it to disk.

If the proposer receives PROMISE messages from a majority of participants, then the propser now becomes the leader. However, the proposer is not necessarily free to propose any value they want to be chosen. If any machine responded with a PROMISE(ballot_id, (old_ballot_id, old_v)) message, then the proposer must propose the value corresponding to the highest old_ballot_id. This is necessary to guarantee safety as consensus may have already been reached earlier.

Phase 2 (Proposer Introduces Bill)

After becoming the leader, the leader (proposer) now proposes value v by multicasting (broadcasting) a PROPOSE(ballot_id, v) to the same majority of participants as in the previous phase. When a participant receives the PROPOSE message, it must determine if it wants to respond to the proposer. If it has already responded to a PREPARE(higher_ballot_id) message where higher_ballot_id > ballot_id, then it must ignore this PROPOSE message. Otherwise, the participant decides to accept the proposed value v. To do so, it first adds (ballot_id, v) to its log and flushes the log to disk. Then it responds to the proposer with an ACCEPT(ballot_id, v) message.

Phase 3 (Bill Signed into Law)

Once a proposer has recieved ACCEPT messages from a majority of participants, it knows that the value it proposed was chosen as the consensed upon value. Thus it adds (ballot_id, v) to its log and flushes it to disk. It then needs to let other participants know that their value was chosen. This is done by the proposer broadcasting a COMMIT(ballot_id, v) message to all participants.

Paxos Summary

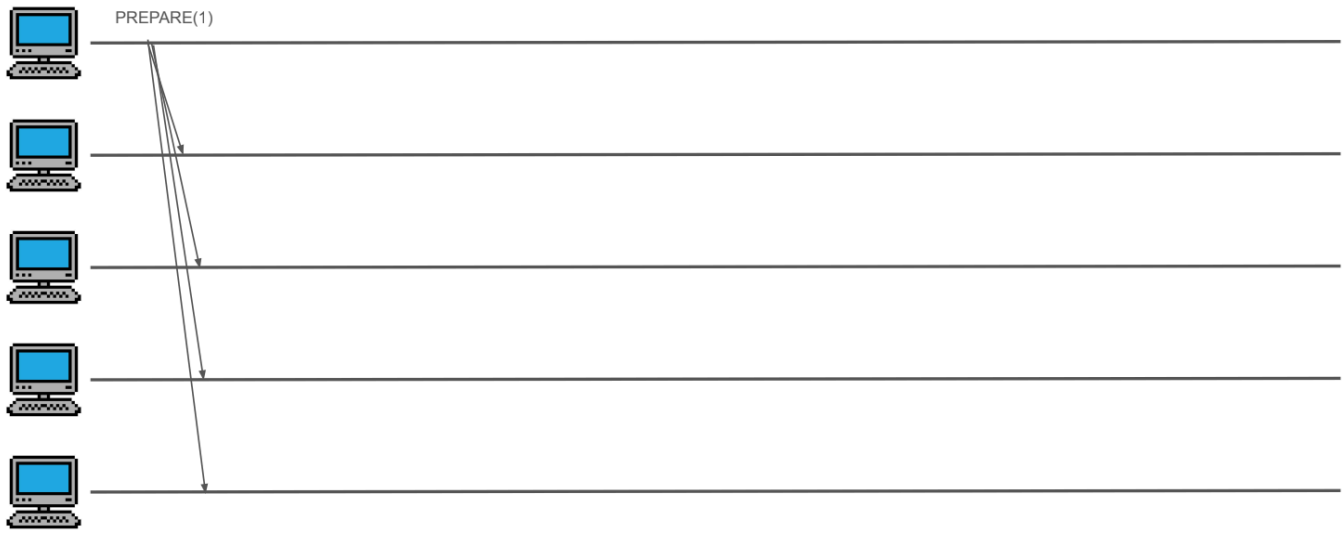
To summarize, there are three phases in each round. A round begins when a machine tries to become leader. Since different machines may try to become leaders at random times, rounds are asynchronous. A round ends with a leader's value being chosen or with the leader being ignored. There are five types of messages that may be sent in the protocol:

- PREPARE(ballot_id)
- PROMISE(ballot_id) or PROMISE(ballot_id (old_ballot_id, old_v))
- PROPOSE(ballot_id, v)

- ACCEPT(ballot_id, v)
- COMMIT(ballot_id, v)

Simple Paxos Example

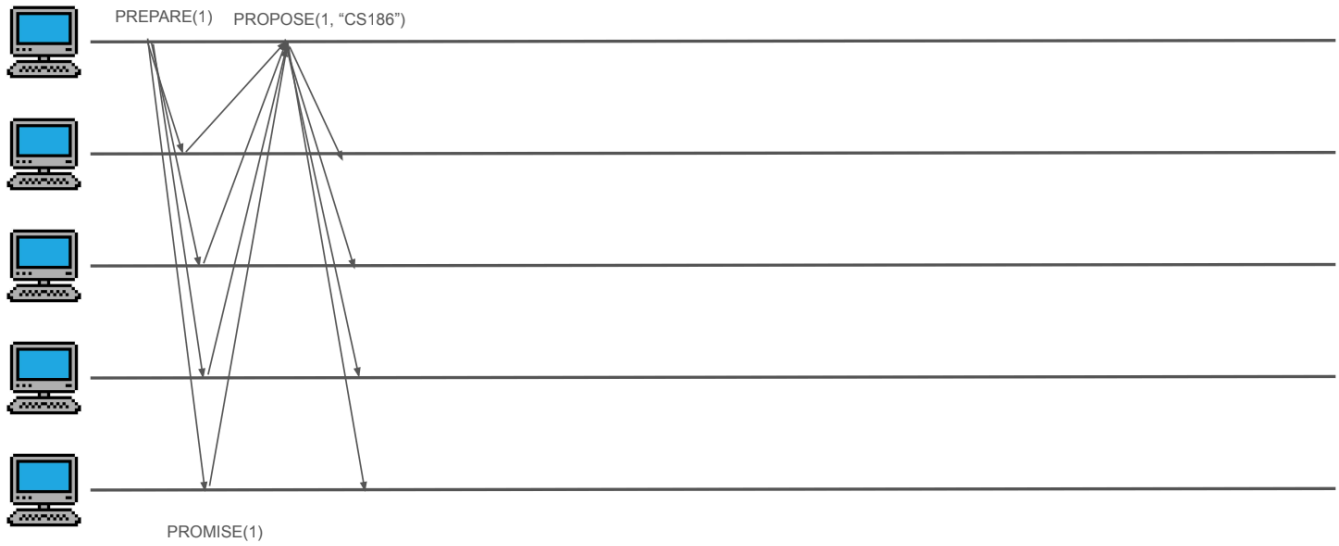
Let's walk through a simple Paxos execution. Suppose we have a Paxos group with 5 machines. Machine 1 decides to begin a Paxos round and sends and multicasts a PREPARE(1) message to a majority of participants (all participants in this case).



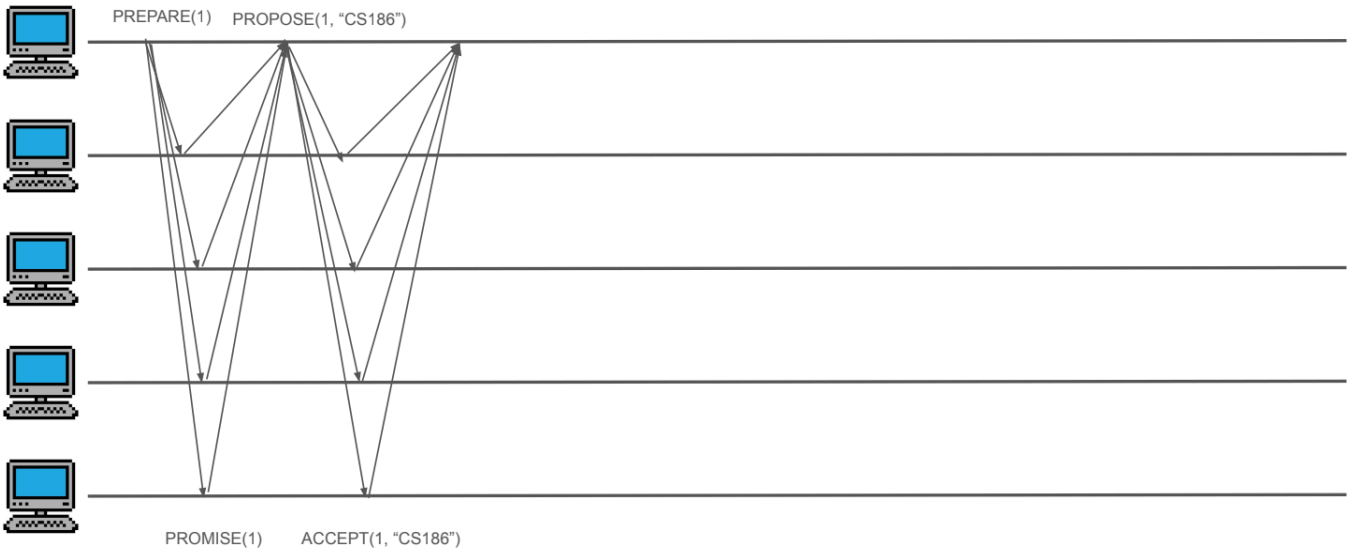
Then, each of the participants responds with a PROMISE(1) message and promises to ignore any ballots with ballot id less than 1. Before sending the PROMISE(1), each of the participants also logs the ballot id 1 and flushes it to disk.

After receiving PROMISE messages from a majority of participants, machine 1 determines that it is the leader. Since none of the participants had accepted any ballot before,

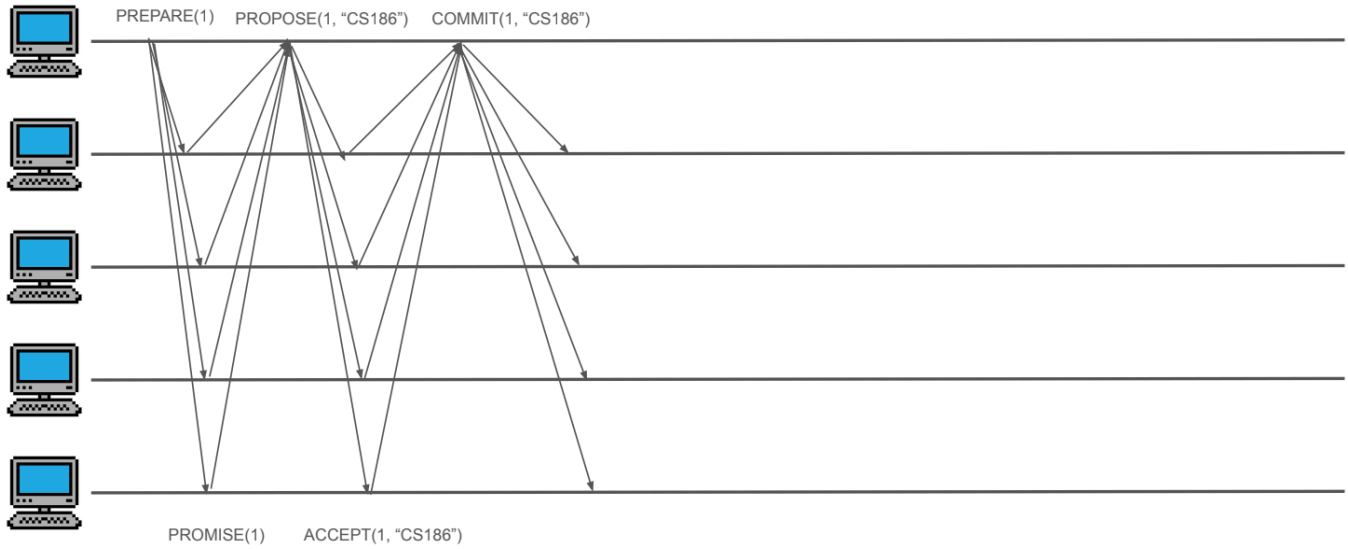
machine 1 is free to propose any value it wants. Since it's cool, it proposes "CS186" as its value. It then broadcasts a PROPOSE(1, "CS186") message to the participants.



Once a participant receives the PROPOSE(1, "CS186") message, it sends an ACCEPT(1, "CS186") back to the proposer since it has not promised to ignore ballot id 1. Before sending the ACCEPT message, the participant logs (1, "CS186") and flushes it to disk.



After machine 1 recieves ACCEPT(1, "CS186") from a majority of participants, it knows that "CS186" has become the chosen value. It then logs (1, "CS186") and flushes it to disk. Then it broadcasts a COMMIT(1, "CS186") message to all participants to let them know that a value has been chosen.



More Consensus Content

As consensus and Paxos are new material this semester (Fall 2023), there are not any problems from the past to pull from. As a result, your best bet to study for exams is to review lecture slides, discussions slides, and the Vitamin.