Coordination Services

Background

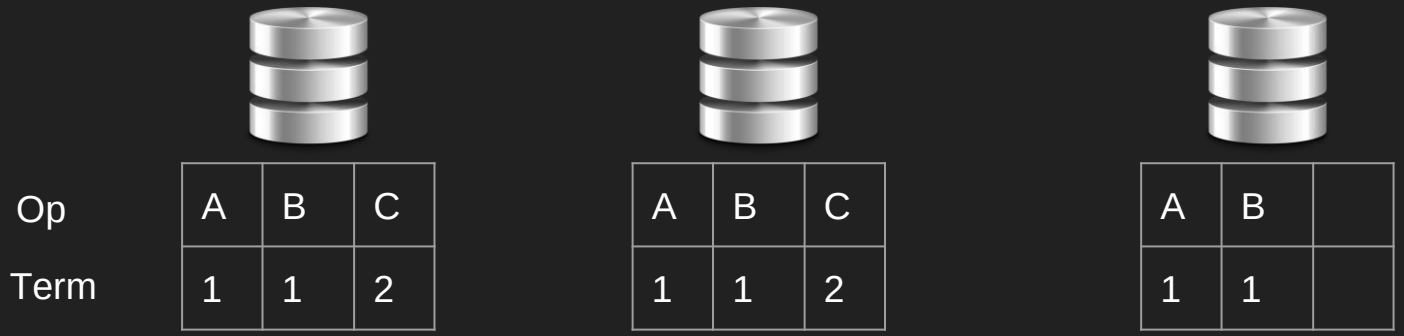
In clusters of nodes, there is often some shared state or configuration between them that all of the nodes should be aware of. This includes the IP address of the nodes in the cluster, partitions residing on each node, which nodes are alive and which are not, distributed locks, etc.

To solve this problem, and make it both reliable and consistent, many more complicated technologies have chosen to use existing coordination services within their tech stack (the most popular ones are ZooKeeper and etcd).

Meant for read heavy workloads.

Coordination Services

Highly available key-value stores built on top of a consensus layer. As a result, writes are slower than if there were no broadcasting mechanism (do not use as a general purpose database).



Reads in Coordination Services

Thus far, we have mentioned that coordination services use consensus mechanisms for replication. However, in order to achieve high read throughput, coordination services by default are not actually strongly consistent (recall that having a replicated log itself is not enough to be strongly consistent).

So what guarantees do coordination services make about reads?

No Monotonic Reads

Recall: A monotonic read is when a client makes a read, then reads from a less up to date replica and it appears as if time is moving backwards.

Coordination services can avoid this by using their replicated log:

- Every time a client makes a read or write, it stores the ID of the last point in the replicated log that it has seen
- Therefore if it reads from a replica with a less up to date log than that ID, it knows to either wait until the replica gets more updates, or try a different replica to read from

Ensuring Predicate Validity

Recall: A predicate is information that you read from a database before making some other read or write.

In coordination services, you can attach a “watch” to any key that you read, making sure that a client will receive a notification informing it of the fact that the read was outdated (kind of similar approach to serializable snapshot isolation).

Strong Consistency

While by default coordination services do not provide strong consistency, you can modify reads so that they become strongly consistent.

- Read from the leader
- Sync
- Quorum Reads?
  - Interesting area of research but not completely perfect

Sync

- From a client node, write the command “sync” into the replicated log in order to get the current up to date ID of the last position in the log
- Make a read from any replica
- The replica will only be allowed to return data to the client if the sync has been propagated into its log

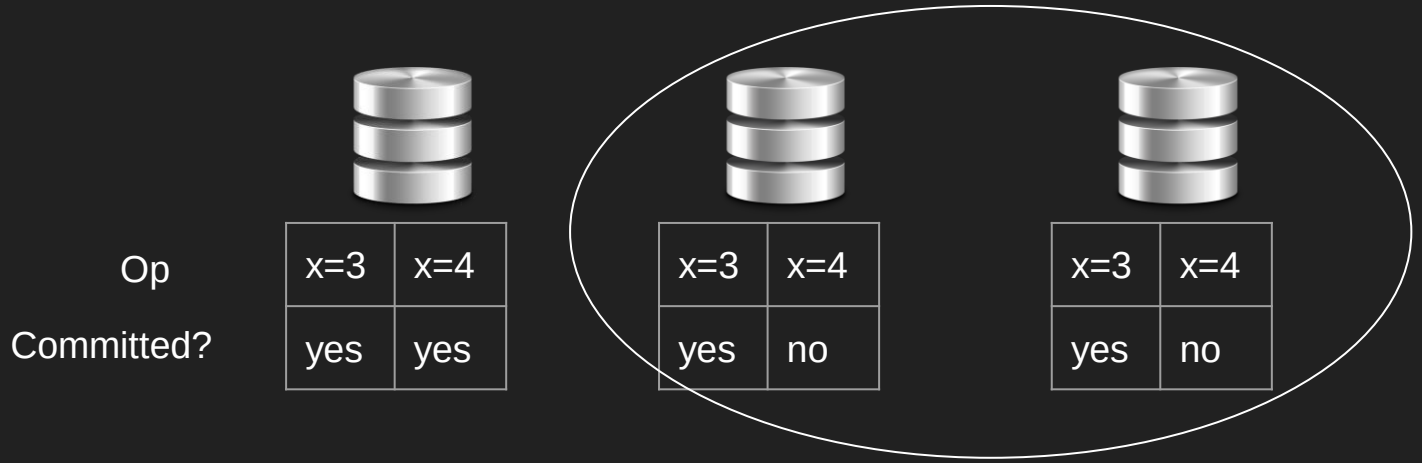
Quorum Reads

Note that in both of these scenarios, we are putting excess load on the leader, what if we just want to be able to get up to date reads from the replicas?

Recall: Replicated consensus algorithms require a majority of nodes to accept a write, so reading from a majority of nodes should have at least one node with the most up to date write.

Quorum Reads Continued

However, even these are not perfect - see the following race condition.



In this case, the leader (R1) sees that replicas 2 and 3 have accepted the write for x=4, so it goes and commits x=4 locally and tells R2 and R3 to commit x=4. However, our quorum read comes in before they commit x=4, and so we get both replicas saying x=3. We can see that there is a newer uncommitted entry on these replicas that will overwrite the value and try the read again, but there is no guarantee x=4 will be committed by then next time.

Coordination Services Conclusion

Coordination services act as a very important subcomponent of many modern day data storage systems. Unlike gossip protocols which pass information directly from node to node, they use a centralized, replicated key value store built on top of a consensus algorithm.

While coordination services are not built to handle strong consistency out of the box, there are some ways of achieving strongly consistent reads, at the cost of significantly reduced performance.