Two Phase Commit

Two Phase Commit

An algorithm that achieves consensus - also helps solve atomic commit!

Atomic commit: If we want distributed transactions that touch multiple partitions, we need each write to either succeed or fail. This is easy enough to do on a single machine, but when the transaction spans multiple nodes connected via a network, they need to all agree on whether the transaction is committed or aborted (consensus).

Why do we need atomic commit?

To keep things from getting out of sync due to partially completed transactions:

- Cross partition transactions
- Global secondary indexes
- Keeping other derived data consistent like data warehouses or caches

Two Phase Commit

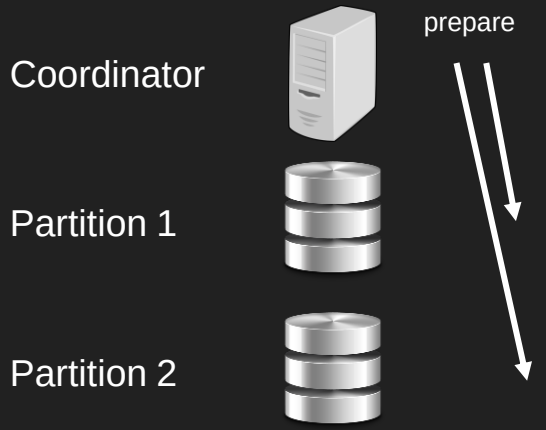
Coordinator

Partition 1

Partition 2

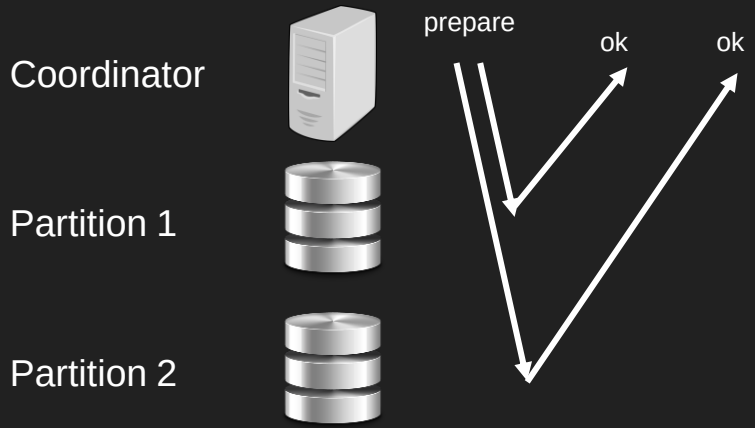
1) The database partitions have received the request to write, 2PC will now begin.

Two Phase Commit



1) The database partitions have received the request to write, 2PC will now begin.
2) The coordinator sends out a prepare request to all nodes:

Two Phase Commit



1) The database partitions have received the request to write, 2PC will now begin.
2) The coordinator sends out a prepare request to all nodes:
    a) Waits for a response from each node

Two Phase Commit

1) The database partitions have received the request to write, 2PC will now begin.
2) The coordinator sends out a prepare request to all nodes:
    a) Waits for a response from each node
    b) If not every partition responds with an “OK”, send result of “abort” to all nodes
    c) If all partitions respond with “OK”, send result of “commit” to all nodes

Why Two Phase Commit Works

- Each transaction has a unique ID from the single coordinator node
- Each node executes its own local transaction per distributed transaction
  - This gives ACID properties like local atomicity and serializability
- When a node responds OK to the prepare phase, it says that it will guaranteed be able to commit said transaction under any circumstances once it receives word to do so
- Coordinator node has internal log of whether each transaction should be committed or aborted in the event that it fails before sending result
  - Known as the commit point, after receiving prepare responses, coordinator must retry forever to commit or abort transaction on all nodes until they all respond with OK

Issues with Two Phase Commit

2PC is not fault tolerant:

- If a partition fails after the prepare phase, the coordinator must try forever to reach it
- If the single coordinator node fails, everything breaks down
  - System is completely blocked, nodes cannot abort a transaction as other ones may have committed it, there is no way to know
  - “In doubt” nodes holding onto locks for relevant rows, prevents reading them (recall two phase locking)

Heterogenous vs. Database Internal Transactions

Heterogenous transactions: 2PC running on different types of software (using an API called XA) - since XA needs to work on many different types of storage systems it cannot be very optimized for performance.

Database internal transactions: 2PC running on the same type of software, can be significantly optimized to do things like run serializable snapshot isolation or detect deadlocks amongst nodes, still it is the case that all nodes need to respond for transactions to be committed or aborted.

Two Phase Commit Summary

Used for distributed atomic transaction, by using a single coordinator node to:

- Ask all nodes if they are able to commit a transaction
- If so, inform all nodes that they should go ahead and commit it

This is nice, but 2PC lacks fault tolerance! If the coordinator node is down, the whole system cannot proceed - if a participant node is down, the transaction cannot be committed or aborted until the coordinator node can reach it.

Next, we will examine some consensus algorithms that actually display a degrees of fault tolerance!