ECE 428 Distributed Systems SP17 Srujun Thanmay Gupta (sgupta80)
MP3 Design Document Anirban Banerjee (banerj10)

The Distributed Transactions protocol:

Optimistic Concurrency

For our distributed transaction protocol, we are using an optimistic concurrency control model to ensure that all successfully committed transactions are serially equivalent. The model works as follows:

- Each transaction carried out over the system has a unique transaction ID attached to it. Currently, we use a single coordinator node to obtain a new transaction ID (that is higher than all previous ones used so far) each time a client begins a new transaction.
- To guarantee the **Atomicity** property, an ongoing transaction does not modify the actual data. It instead writes to a temporary buffer created for that transaction's client. The original data is only updated when a transaction is committed by a user, and serial equivalency is not violated. This helps avoid the lost update and inconsistent retrieval problems, and thus also ensuring atomicity.
- To ensure **Isolation**, for each key-value pair stored at each server, two records are kept: (1) the last transaction ID that read from that pair, and (2) the last transaction ID that wrote to that pair. This approach is based on Timestamp Ordering.
  - Read operation: transaction with ID *T* will be successful only if the key was last written to in a transaction with ID lower than *T*.
  - Write operation: transaction with ID *T* will be successful only if the key was last read by and written to in transactions with IDs lower than *T*.
- However, if the recorded transaction ID is higher than that of the current transaction, it means that a later transaction has also read from/written to the same value. In this scenario, the current transaction is aborted completely and its buffer is deleted.

Coordinator - Client - Server Interaction

The system consists of one coordinator, three clients and five servers. The coordinator is responsible for providing unique transaction IDs to clients that have begun a new transaction. Centralizing the assignment of transaction IDs ensures that IDs across different clients are unique without requiring communication between clients.

The clients are the interface by which users can access and modify the key-values stored in the servers. The clients are the only nodes that actually send messages to other nodes to modify the state of the system - the coordinator and the servers passively listen for messages from the clients and do not send any messages except for replies to clients. Each server stores a set of key-value pairs as well as N different buffers, where N is the number of clients. Each buffer stores tentative updates from ongoing transactions for that client.

Two-Phase Commits at Servers

We treat a client's COMMIT request as a Write operation on every server that is involved in that transaction. To ensure that the serial equivalence property is maintained, all servers need to be able to successfully commit a transaction before any actually commit it. This is to ensure consistency across the servers. Thus we use two-phase commits to accomplish this. The client first asks each associated server if it is able to commit the transaction with the Write operation requirements (as mentioned in the Optimistic Concurrency section above). Once every server responds with a 'successful' message, another request is sent to actually carry out the commit. Even a single server returning a 'failed' message causes the transaction to be aborted.

Avoiding Deadlocks

Under the optimistic concurrency control model described above, locks are not required and hence there is no need for detecting and avoiding deadlocks. When a transaction operation finds that a value on a server has been modified by a more recent transaction (i.e. the last recorded read/write was by a transaction with a higher value), the transaction is aborted. In this manner, serial equivalence is maintained without the need for locks.

Future Work

Due to the small size of our transactions system, we delegate a coordinator node to assign transaction IDs. Because it is a central coordinator, it creates a single point of failure and hinders the scalability of this system. To address this, we can potentially implement a multicast system similar to what we find in ISIS Total Ordering. Or an even better solution would be to implement a consensus protocol between all the coordinators in the system.