LECTURE 12

Reliable Group Communications/Transactions
Need for reliable group communications

- All processes (replicas) should agree to the same transaction.
- Reliable delivery of messages from a coordinator to replicas.
  - Easy way: Send and wait for ACK (e.g., using TCP) but results in blocking at coordinator
  - ACK implosion: All the receivers send ACKs overwhelming the sender.
Reliable Multicast

- Receivers keep track of sequence numbers of transmissions and only NACK missing messages.
- A hierarchy is possible – a group of receivers with a coordinator each.
  - Coordinator performs reliable multicast to all receivers.
- Key issue: What happens when nodes fail?
Distributed Transactions

- Partitioning of state necessary to scale

- Partitioning results in the need for transactions
  - Atomically execute operations across shards

- Examples:
  - Add meeting to calendars of two participants
  - Transfer money from one account to another
  - Looking up balance of two accounts
Concurrency + Serializability

- Execution of transactions:
  \[ T_1 \rightarrow T_2 \rightarrow T_3 \]

- Externally visible effects:
  \[ T_2 \rightarrow T_1 \rightarrow T_3 \]
Example of Serializability

- **Concurrent execution of transactions:**
  - T1: Transfer $10 from Alice to Bob
  - T2: Read balance in Alice’s and Bob’s accounts
  - Initial balance of $100 in both accounts

- **Permissible outputs for T2:**
  - (Alice: $100, Bob: $100) or (Alice: $90, Bob: $110)

- **Invalid outputs for T2:**
  - (Alice: $90, Bob: $100) or (Alice: $100, Bob: $110)
Atomic multicast

- Client contacts replica P for an update.
- P multicasts the update to all other replicas.
- If P crashes before multicast completes:
  - Some members may have received others may not!
- Requirement: Either all non-faulty members perform the update (equivalent to P crashing after) or none do (equivalent to P crashing before)
Atomic multicasting of a message “M” is uniquely associated with a group G (Group view)
- Processes to which M is to be delivered.

Key question: What happens if the group changes in between (a new process Q joins or leaves group)?
- A “view change” takes place that announces this.
- However, M needs to be either delivered to all processes in group G before the view change is executed.
Virtual Synchrony

- A message multicast to a group view $G$ is delivered to each non-faulty process in $G$.
- If the sender of the message crashes:
  - the message is either delivered to all the remaining processes; or,
  - ignored by all of them
- If these properties are satisfied, the reliable multicast is said to be virtually synchronous.
Example

- If virtual synchrony is satisfied, message not delivered to S2 and S4 after S3 crashes.
Categorization

- Unordered
- FIFO ordered
- Causally ordered

- Total ordered multicasts: messages are delivered in same order to all members.
  - The commits are consistent across all members.
Distributed commit

- Distributed commit problem is having an operation performed by all members of a process group or none at all.
  - Note: reliable multicasting is only delivery of a message.

- One phase commit: Coordinator tells participants whether or not to locally perform the operation.

- Drawback: If a candidate fails to perform the operation no way of telling the coordinator!
Two phase commit protocol (2PC)

- Coordinator sends a VOTE-REQUEST to participants
- Each participant, upon receipt, either sends VOTE-COMMIT or VOTE-ABORT.
- Coordinator collects votes; if all commit, sends a GLOBAL-COMMIT; else, sends a GLOBAL-ABORT.
- Participants who voted for commit, waits and does what the coordinator finally says.
Issues

- Failures can cause issues with the basic 2PC protocol.
- For example, a process may crash – and other processes may indefinitely wait for a message from that process.
(a) The FSM for the coordinator
(b) The FSM for the participant.
Exiting from blocking states

- A participant may be blocked in the INIT state
  - Times out and issues VOTE-ABORT if no VOTE-REQUEST is received.

- A coordinator maybe blocked in the WAIT state.
  - Times out and issues GLOBAL-ABORT if not all votes are collected within a certain time.

- A participant maybe blocked in the READY state – waiting for the results of the global vote.
  - Now the participant cannot simply time out and abort.
  - Needs to find out what was actually sent by the coordinator.
  - Either block until coordinator recovers (delays) or
  - Contact another participant to check if a COMMIT or ABORT was received.
If Q is in INIT state – it means it did not receive even the VOTE-REQUEST – thus, P should abort.

If Q is in the READY state also it has the same issue as P – so contact another participant.

- If all participants are in READY, no choice but to wait for the coordinator to recover.
Crashes

- State information stored into persistent storage for recovery upon crashes.
- When participant crashes in READY, does not know whether to abort or commit upon recovery
  - In INIT, COMMIT or ABORT states this problem doesn’t arise.
  - Needs to contact other participants
- Coordinator needs to record
  - WAIT state – retransmit VOTE-REQUEST upon recovery.
  - Decision (COMMIT or ABORT) – which is retransmitted upon recovery.
Achieving Serializability

- When is concurrent execution of transactions safe?
  - Data read/written is disjoint

- When must two transactions execute in order?
  - Intersection in data read/written

- Solution for serializability:
  - Fine-grained locks
  - Transaction coordinator first acquires locks for all data
  - Execute transaction and release locks
Two Phase Locking

- Transaction aborted since P2 did not commit to lock
Two Phase Locking

Can we reduce latency for read-only transactions?

Return data in first round if lock not held
Concurrent execution of transactions:
- T1: Transfer $10 from Alice to Bob
- T2: Read balance in Alice’s and Bob’s accounts
- Initial balance of $100 in both accounts

Problematic sequence of concurrent execution?
- TC2 reads Alice’s account balance as $100
- TC1 executes 2PL to acquire locks on both accounts, transfer $10, and release locks
- TC2 reads Bob’s account balance as $110
Lock-free Read-Only Txns

TC1

P1

P2

TC2

Read

Lock

Commit
Fault Tolerance of 2PL
Fault Tolerance of 2PL

Okay to commit after timeout?
Fault Tolerance of 2PL
When can we garbage collect transaction log?
Garbage Collection

- Lock acquisition in node log:
  - Node receives commit from TC and writes to its log

- Commit operation in TC log:
  - After all nodes say transaction committed

- Commit operation in node log:
  - Upon applying operation to local state
Fault Tolerance

Transaction cannot succeed if any partition unavailable