Name: Qing Peng

1. What are distributed transactions in Oracle DB?

   Distributed transactions is a set of transactions that update data on over two distinct nodes of a distributed database.

   How are they different from Remote Transactions? Give examples.

   All statements in Remote Transactions refer to a single remote node. But in Distributed transactions, statements refer to two or more nodes of a distributed database.
   For example, this is a distributed transaction:
   UPDATE test1@testcase1.com
   SET attribute1 = 'test'
   WHERE id = 1;
   UPDATE test2@testcase2.com
   SET attribute2 = 'test'
   WHERE id = 5;
   COMMIT;

   And this is a remote transaction:
   UPDATE test@testcase.com
   SET attribute1 = 'test'
   WHERE id = 1;
   UPDATE test@testcase.com
   SET attribute2 = 'test'
   WHERE id = 5;
   COMMIT;

   How does Oracle DB use Naming service and 2-Phase Atomic COmit Protocols to manage distributed transactions?

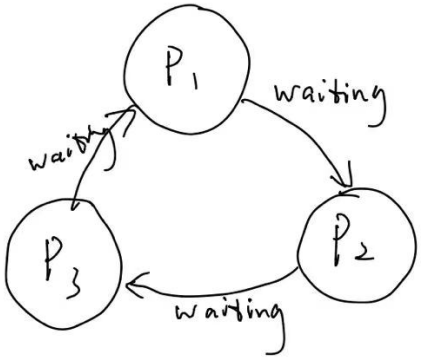
   Name of a global object name contains three parts: Object name, Database name and Domain. We can access a global object with Object name@Database name.Domain.

   Oracle DB uses two-phase commit mechanism to include all statements in a transaction, to guarantee that all database servers in a distributed transaction either all commit or all roll back in the transaction. This mechanism also protects implicit DML operations.

2. What is Deadlock? How does it cause problems in a Distributed Systems? Clearly explain using an example.

   Deadlock occurs when each process is waiting for another process to proceed, and there is a cycle in the graph of this relationship. Deadlock can cause starvation, and make the processes in the deadlock stop making progress.

For example, in the below case, P1 is waiting for p2, p2 is waiting for p3, and p3 is waiting for p1. This forms a waiting cycle, and neither of the three processes can make progress.



3. How would you use the PAXOS algorithm for Clock Synchronization? Please explain what the consensus use case is, and then how PAXOS works to achieve this, using the references provided below.

   The consensus use case is we want to make all machines in the distributed system synchronized and agree on the same time.

   The messages passing can be delayed, lost out of order, duplicated but not corrupted. And the processes may fail by stopping, restart, but not present Byzantine faults.

   Paxos guarantees that nodes will choose a single value, but does not guarantee that a value will be chosen if most of the nodes are unavailable. A Paxos node can act in three roles: Proposer, Acceptor and Learner. Proposers propose values to reach consensus on; Acceptors contribute to reaching the consensus itself; Learners learn the agreed upon value. Paxos nodes can take multiple roles, and they know the number of acceptors.

   1. Proposer sends a unique prepare id and a timestamp to all acceptors
   2. Acceptor receives a prepare message, and check if it has promised the id, if yes, then ignore it, otherwise, reply with promise id, and promise to ignore any request lower than this promise id.
   3. If a majority of acceptors promise, no id lower than this id can make it through.
   4. After the proposer gets the majority of promise messages for this id, it sends accept-request id and the value to all acceptors.
   5. Acceptors receives the accept-request message, and check if it promises to ignore this id, if yes, then ignore it, otherwise, reply to the proposer with accept id and value, and send it to all learners.
   6. If a majority of acceptors accept this proposal, the consensus is reached.

4. Study Alibaba Fescar note. How does Fescar manage distributed transactions? What protocols are used? describe in detail. How does it avoid Deadlock?

   There are three components negotiating the process of distributed transactions: the Transaction coordinator(TC) maintains the running state of global transactions, coordinate and drive the commit or rollback; the transaction manager(TM) controls the boundaries of global transactions and initiates the global commit or rollback; the resource manager(RM) controls branch transactions which registers branch, report status and receives instructions from TC.

   Fescar deployed RM on the application side as a middleware layer, so it does not depend on the support for the XA protocol provided by the database itself.

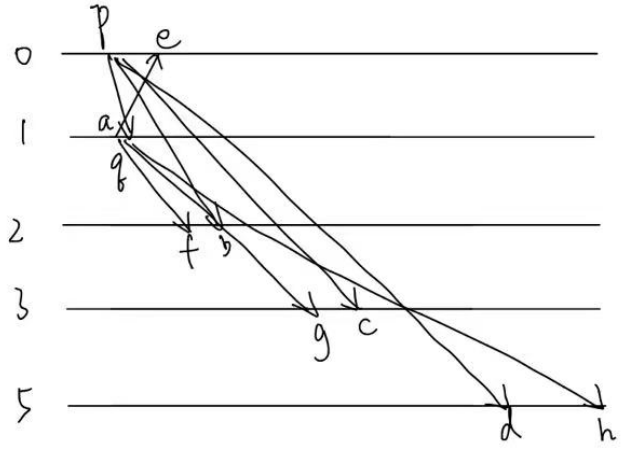
   Fescar uses a different version of two-phase commit than XA's 2PC.

   Phase 1: Fescar's JDBC data source proxy organizes the data images of business data before and after the SQL execution into an undo log, and writes update commands and undo log in the local transaction to commit by ACID feature. Therefore, it can guarantee any modifying commands has an undo log.

   Phase 2: If the resolution is a global commit, branch transactions can complete the synchronization directly. If the resolution is global rollback, RM finds the corresponding undo log, and generates the reverse update SQL, which it executes by rolling back the record to complete the branch rollback.

   In XA mode, the branch transaction is blocked, and waits for XA commit or XA rollback, which can cause the problem of deadlock. But in Fescar mode, since the branch transaction commit directly, lock is not required which solve the problem of deadlock.

5. In a group of four machines {0, 1, 2, 3, 5} machines 0 and 1 send messages simultaneously via multiple unicasts to the group. In this context, explain Global ordering, causal ordering and partial ordering approaches for time ordering, and compare them.



Total ordering, if a message is delivered before another message at one process, then the same order will be preserved at all processes. Since machine 0 sends messages to machines 1, 2, 3, 5, sending happens before receiving, events that machines 1, 2, 3, 5 receive messages must be after machine 0 sends messages. Likewise, machines 0, 2, 3, 5 receiving messages must be after machine 1 sends messages. Every event has its own timestamp, we can order all the events by their timestamps. In the graph above, the total ordering is: a e f b g c d h.

Causal ordering: if a message happens before another message in the distributed system, then the order will be preserved in the delivery of the messages. Since machine 0 and 1 send messages to all the other machines simultaneously, other machines receiving messages must be after machine 0 and 1 send messages. We do not focus on the total ordering, only care about the ordering made by cause and result.

Partial ordering: reflect a flow of information between processes, only containing part of the order of processes. Causal ordering is one of the partial ordering. The causal ordering mentioned is a kind of partial ordering.

6. Transaction API (JTA) allows applications to perform distributed transactions. For the Musicstore in #1, explain a design for implementation using JTA. Show the key Classes and how you use them to resolve concurrency issues and deadlock.

   There should be a Server class to contain some operation apis; a Coordinator class as a transaction manager to support ACID properties of distributed transactions; a Client class to send requests; a Participant interface to support two-phase commit.

   There are some methods to send requests to operate data in the class Client, like reading data, deleting data, updating data, inserting data.

   The Server class contains some methods to operate the database, including CRUD operations. The server class also implements the interface Participant, which contains some methods related to two-phase commit:
   canCommit(): send a request to coordinator to ask if it can commit.
   vote(): vote for “commit” or “abort”, which indicates whether it is ready to commit.

   The Coordinator class contains methods to support ACID for transactions:
   prepare(): sends commands to all participants and waits for the responses.
   commit(): sends command to participants to commit the operation.
   abort(): sends command to participants to abort the operation.

   When a server receives request from a client, the server sends canCommit() request to the coordinator. Then the coordinator sends commands to each participant and waits for response from each participant. If each participant votes for “commit”, then the coordinator sends commit commands to all the participants. Otherwise, if any participant votes for “abort”, the coordinator sends abort commands to all the participants.

To resolve the concurrency issues and prevent deadlock, we can use timestamps to synchronize all the machines in the system. Use PAXOS algorithm for clock synchronization, always abort the older transaction and update the data with the latest timestamp. Also, make sure the synchronization request is sent before the last request ends. Then it can prevent deadlock problems.