Lecture 18 Multi-site Atomicity **April 14, 2014**

Recap last time:

Serializability Testing

Isolation via two-phase locking

[Show slides]

Possibility for deadlocks

[Show slide]

Optimizations: Reader-writer locks

[Show slide]

Relaxed consistency (e.g., read committed)

Don't always need to enforce serializability, e.g., read committed

[show slide]

Key idea: read-locks are acquired (to prevent reading dirty data), but then released (to allow concurrent writes)

W/ serializable, T1 will wait for T2

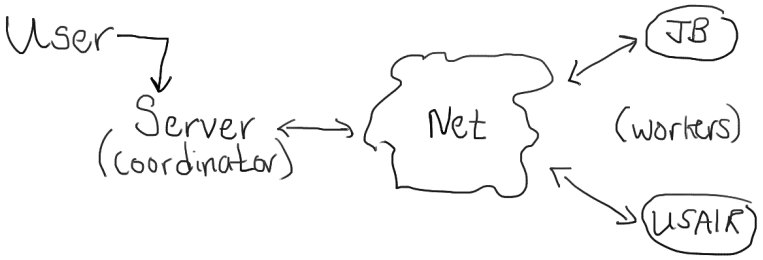
W/ read committed, T2 will release read lock after select, which will allow T1 to run; T2 will see T1's update, and get a different count at different times (but do we care)?

<u>Multisite Actions</u>

Example: travel site:

Suppose that JB and USAir are separate reservation systems, each with their own data.

There is one coordinator that the user connects to and who makes reservations on these subsystems. Diagram (with internet)



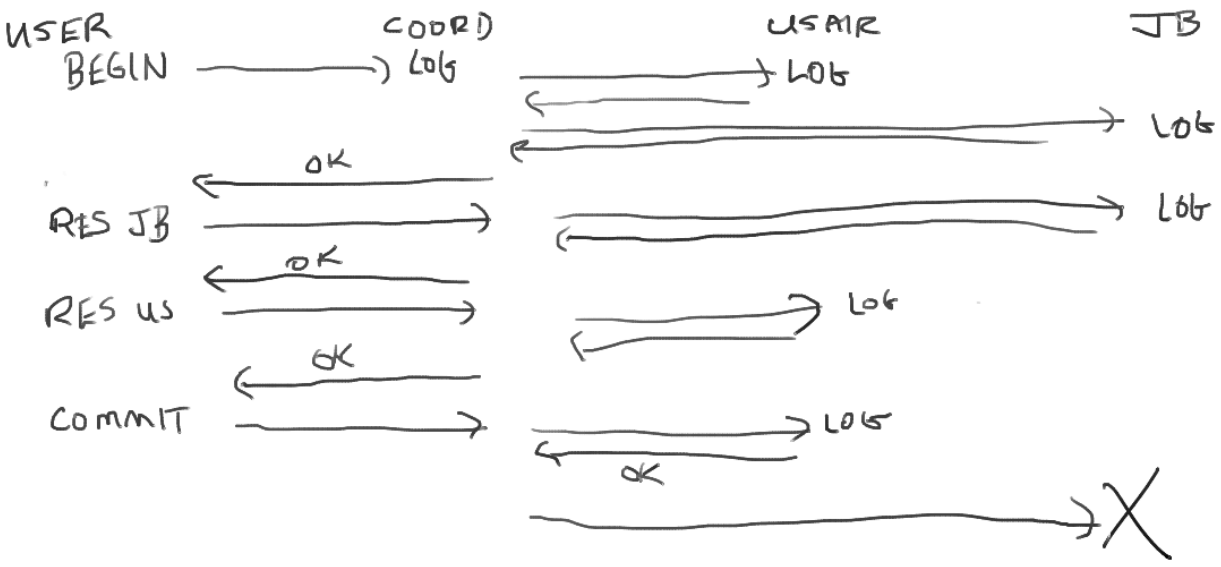
Goal:

Single transaction that spans both JB and USAir.
Should only commit if actions complete on both JB and USAir.

Example

Begin Xaction
    Reserve JB
    Reserve USair
Commit

Suppose we just treat JB and USAir as separate systems, and send them both a commit message when ready to commit.



Problem:

Don't want one to commit unless the other has also committed. One might receive the commit message, other might crash at that instant and decide to abort the transaction. (See diagram)

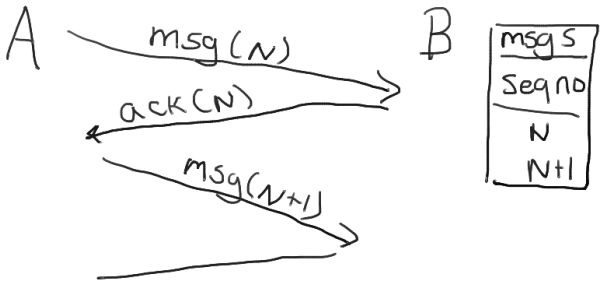
Complicated because JB / USAir might crash independently; also messages might be lost / reordered.

Soln is going to involve getting nodes to agree that they are ready to commit, even if they crash, and then actually making them commit -- "**2 Phase Commit"**.

Separate logs for each site, including coordinator.

To deal with message losses, going to use a protocol like exactly once RPC.

Diagram:



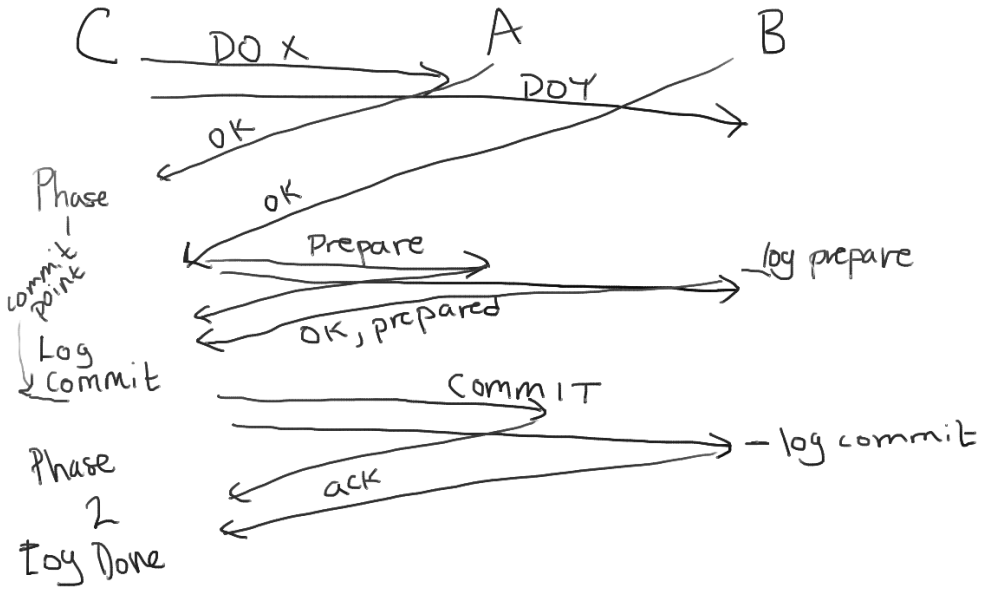
Basic protocol is as follows:

(1) Coordinator sends tasks to workers; workers log xaction begin and updates just like normal.

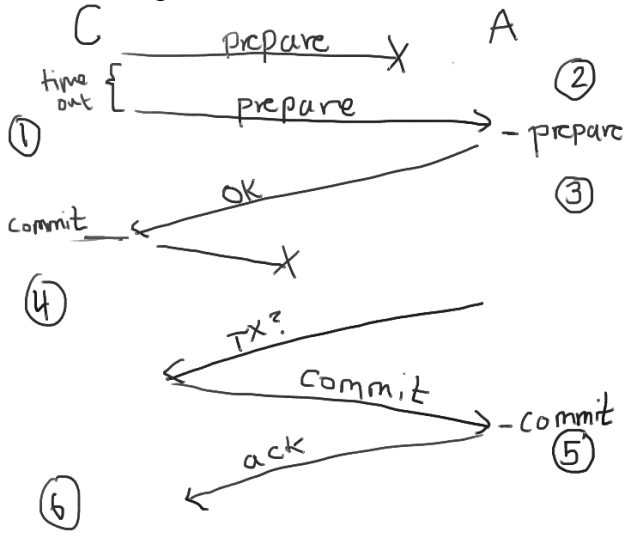
(2) Once all tasks are done, coordinator needs to get workers to enter prepared (tentatively committed) state

(3) Tentatively committed here means workers will definitely commit if coordinator tells them to do so; coordinator can **unilaterally commit**

Protocol (with no loss): (prepare, vote, commit, ack) (show logs)



Suppose messages are lost? Use timeouts to resend prepare, commit messages



	outcome	action
1	ABORT	Send abort to A
2	ABORT	reply "No" to prepare
3	UNKNOWN	request outcome
4	COMMIT	resend commits, remember xaction
5	COMMIT	do nothing
6	COMMIT	forget xaction

Crashes? Need to make sure that logs on workers ensure that they can recover into the tentatively committed state.

Log records:

Write prepare record on workers before "Yes" vote.

Commit record still written on coordinator -- that is commit point of the whole transaction

Coordinator also writes a "done" message

Suppose worker crashes:

Before prepare (2)

After prepare (3)

After receiving commit (5)

Suppose coordinator crashes:

Before prepare (1)

After sending some prepares -- Aborts

After writing commit (4)

After writing done (6)

How does coordinator respond to "TX?" inquiry? Does it keep state of all xactions forever? (No -- once it has received acks from all workers, it knows they have received outcome.)

Notice that workers *cannot forget state of transaction* until after they hear commit / abort from coordinator, even if they crash. This makes protocol somewhat impractical in cross-organizational settings.

What to do instead?
Use compensating actions (e.g., airlines will allow you to cancel a purchase free of charge within a few hours of making a reservation.)

2PC provides a way for a set of distributed nodes to reach agreement (e.g., commit or abort.) Note, however, that it only guarantees that all nodes eventually learn about outcome, not that they agree at the same instant.

"2 Generals Paradox" (slides)
Can never ensure that agreement happens in bounded time (though it will eventually happen with high probability.)