Lec 19 : Nested atomic actions, and multisite atomicity 4/15/09

Sam Madden

So far, we've focused on the case where transactions are running on just one system, and where all of transaction either commits or aborts.

Today -- several additional goals

- Nested atomic actions -- the ability to have sub-actions which may or may not succeed, that don't affect the outcome of the whole action (nested atomic actions)

- Multisite actions -- transactions that span multiple sites (Distributed transactions); why?

  - Performance (2 machines store a database, both machines participate in querying or updating of their part of the DB.)

  - Administrative -- two databases owned by two different organizations, want to run commands that span both

Example: travel site:

Single System:

Wanted nested actions to allow subtransactions to run to completion without committing, and then commit subtransactions

```
Begin  (Parent P)
        Begin  (A)
                Reserve JB
        End
        Begin (B)
                Reserve USAir  ---> if this aborts, JB reservation isn't lost
        End                          ---> if this aborts, P can try something else without
                                                losing JB reservation
End
        Parent P
                Sub xaction A    |  Commit iff
```

Sub xaction B | P commits

Goals:

A + B's effects aren't visible externally unless P commits.
(Once P commits, A + B's effects both visible)

A + B can independently abort

B shouldn't see A's effects unless A has finished.

Once A has reached and, it shouldn't abort unless A aborts (since B may use its results)

We are going say that the a nested transaction that has reached its "End" has "tentatively committed" -- it is ready to commit, but is waiting for outcome of its parent to be decided.

Changes to protocol:
Locking:

When to release A/B's locks? Before P commits? No -- because P might abort!
Only after P commits.

But B should be able to read A's data after it has reached "End".

Change lock acquire protocol to check to see if waiting for a lock from a tentatively committed transactions w/ same parent

Logging:
Need to keep separate begin / end for subactions, in case one of them aborts and we crash, so we are sure its effects are undone.

During log processing, when we encounter a sub-transaction, only want to commit it if its parent commits

Begin P
Begin A
UP seatmap1
Tentative Commit A (Parent P)
Begin B
UP seatmap 2
Tentative Commit B (Parent P)
End P

Recovery: scan backwards, determining winners, undoing losers

(e.g., sub actions that abort or sub actions whose parent aborted)
scan forwards, redoing winners

Multisite Actions

Similar protocol and rules -- suppose now 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)

Like before, want JB to commit only if USAir commits, and vice versa.

Harder than before, because now JB / USAir might crash independently, and messages might be lost / reordered.

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:

Coordinator sends tasks to workers

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

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)

Why not just send COMMIT messages to all sites once they've finished their share of the work?

One of them might fail during COMMIT processing, which would require us to be able to ABORT suboords that have already committed.

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

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

Log records:

Write TC 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?

After prepare?

Suppose coordinator crashes:

Before prepare

After sending some prepares

After writing commit?

After writing done?

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.)