Summary of Distributed Transactions

By S. Xiao Fernández Marín

1 Summary

A distributed transaction (DT) is a transaction that accesses several different servers. These transactions can be flat or nested, depending on how are structured.

1. **Flat transaction:** The request of the client is made to more than one server and it is sequential, the server completes each request before moving to the next server using locking.
2. **Nested transaction:** The client send requests to revers, those servers can open subtransactions and those subtransactions open new subtransactions etc. The transactions at the same level run in parallel (be concurrent).

These transactions could not be possible without the help of the **coordinator of a DT**. It helps the servers to communicate with other servers to coordinate their actions. It has to decide if commit or abort the DT.

The servers will provide an object called *participant*, that will take track of their recoverable objects used on the transaction. The participant will also keep which one is its coordinator and the coordinator will keep a list of its participant. In this way, it will be easier to collect the information needed when committing.
In Figure 1 we can see the graphic of a DT and how the coordinator is the one in charge of everything.

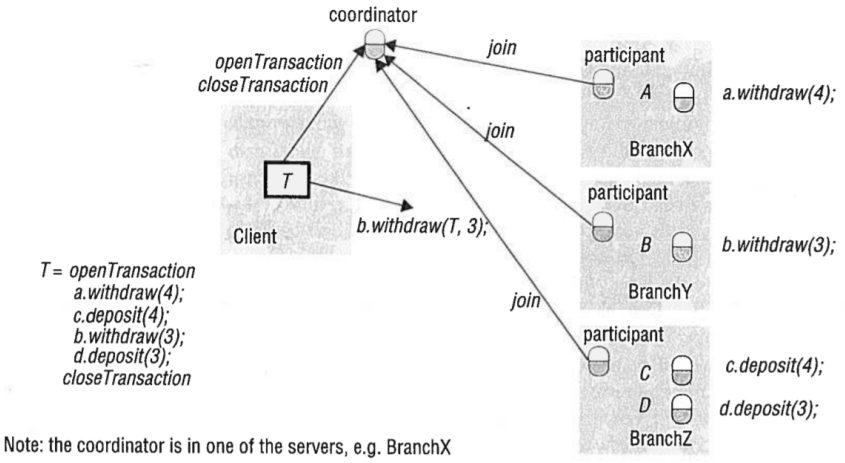

Figure 1: Distributed transaction

When a DT ends, the **atomic commit protocols** check that all of the services involved commit or abort the transaction. There are several ways of doing this:

- *One-phase atomic commit protocol:* Let the coordinator communicate the commits or abort request to all the participants. This is an inadequate protocol as if for example a server crash down, it will need to abort the transaction, but it would not be possible as the server is down.
- *Two-phase commit protocol:* The servers can communicate to vote whether commit or abort the transaction. The coordinator sends a "canCommit" request. When a participant votes "yes" (commit), it can not change the vote so it has to store the objects that have altered in

the transaction. If it votes "no" it aborts.
After that, the coordinator takes into account all the votes of the servers involved in the transaction and if there is one "abort", it aborts ("doAbort"). If not, the commit is done ("doCommit") and sends a "haveCommited" message to all the participants.

2 Questions not answered by this text

It is a blocking protocol, if a participant needs blocks of information another participant is using, it will need to wait until the one using them frees the blocks.
There is also a problem if the coordinator fails before sending a response to the participants, they will enter on block state. This problem can be overcome using the Saga pattern.

3 What has changed since this was written

There are several problems with the two-phase commit protocol as seen before, so exists a pattern called "Saga" that works with local transactions. If one fails, it undoes the changes.