# 6.033 Lecture 19
# Fault Tolerance VI

Americo De Filippo

July 6, 2023

## References

[1] Saltzer, Jerome H. and M. Frans Kaashoek. Principles of Computer System Design: An Introduction (2009): **Section: 10.1 - .4**

This time we are gonna look at multi site atomicity. Rembering when we said that an atomic action is both recoverable and isolated. The topic for today is gonna be applied when wanting to have atomicity on different computers.

## 1 Nested atomic actions

Here we are gonna see how this behevies on a single computer case. Let's say I want to buy a ticket from a company A and at the same time from B. I want that the action of buying will be atomic with respect to each other, So the notion of nested atomic action is having a big action atomic action and nested in it 2 other atomic action with respect to one another.

### 1.1 Tentative commit

Let's say that at the same time A and B want to commit, in this case we need some kind of supervisor S that will handle the problems with the simultanesy commit. So want we actually want is that A and B do everything that has to do exept commit.

### 1.2 Recovery via logging

Let's say we have a log starting with the supervisor action S, and starting calling the sub-atomic actions. Now going backwords he they are going to write the commit records when each action commit (the last to commit will be S). Of course the locks will be released only after the commit points. As we may predict A/B are dependent from S, but S is not completely dependent from A/B, this means that if A/B fail S could not fail, on the other hand if S fail A/B fail as well.

**what if A,B conflict?** We may have a situation where the 2 sub action could conflict with each other, in this case we have to modify a bit the previus statement. If A commits, the nested action within A can see the changes made by A. This doesn't mean that the actions outside can see the changes just the ones nested within A.

# 2 2 phase commit protocol

Suppose we have our nodes S, A, B. S start executing t, and is gonna request that A and B start processing. Now what is gonna happen is that A and B are gonna write their own log record, and eventualy arrive to the tentative commit point. At this point S is going to look and decide if A or B is going to commit, if allowes the commit he write the commit record (the S commit record is the one that really matters for outside output).

## 2.1 Exacly one RPC into this

Recalling the RPC protocol is developed in order to run only once a procedure call between the client and a server. How we did this is by storing a table at both the end point. In this way we are gonna store the request that have been executed in a way that the server does not do an action multiple times.

**How to use this notion to our 2PC?** What is gonna happen now is that we are going to keep a list of pending action both at S and at A/B. If there is some lost and the action is going to be retried again just as we did in the networking case. When A/B finishes doing some action is going to mark in its pending table the result of that action makind that tentative commits (and is gonna communicate to S the result). This can be applied for all the messages into a communication, like commmit messages.