6.033 Lecture 17
Fault Tolerance IV

Americo De Filippo

July 4, 2023

References

[1] Saltzer, Jerome H. and M. Frans Kaashoek. Principles of Computer System Design: An Introduction (2009): **Section: 9.4 / 9.5**

The topic that we are gonna tackle today is isolation. The idea behind isolation is to run multiple actions at the same time, and have a result which you would have if runned one after the other.

1 Starting with slow and correct

Let's say you have 2 actions T_1, T_2. The first read x and write y, the second on the other hand write x and write y. So we can see this steps runned in a lot of number of ways. So if you look at the atomic actions, there are some that conflict with each other and others that don't.

1.1 Search for the conflicts

Some usually confictual actions are r(z), w(z) and combination of them. So we are gonna search for the conflictual actions and we are going to serialize. What we mean is to run the actions in a different orders that will make the result look just the same as the action would've runned sequentially.

1.2 Serializability

Trace's confict arrow should be in the same order as some serial order of actions.

1.2.1 Actions graphs

Given one traces we are gonna produce a graph and from there see if some proprety holds. The nodes are the actions, the edges are the trace between the actions. If two action share a conflict operation you are going to draw an edge between the actions. When you draw the graph you can find a path for going

all through the path with the shorter path (in case of multiple serial orders). And that will be the equivalent order of actions on the one in which they run sequentially.

Result 1: If the graph has not cycle the order is serializable and viceversa Notice: If it would happen that for run some operations we would have some conflicts that could be resolved. To prove this remember that if you have no cycle you can run a topological sort order. A topological sort is described as follows:

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge

while S is not empty do
      remove a node n from S
      add n to L
      for each node m with an edge e from n to m do
          remove edge e from the graph
          if m has no other incoming edges then
              insert m into S

  if graph has edges then
      return error   (graph has at least one cycle)
  else
      return L   (a topologically sorted order)
```

1.2.2 Locks

Recalling if you have a variable you can do 2 actions onto this, acq(lock of x), rel(""). With lock you can really slow down the system, you would probabily remove the concurrency and have a slower system. Futhermore, you could get in trouble creating cycle in the graph.

1.2.3 Simple locking

The idea is that every action before doing every reads or writes would acquire the locks, in this way no other conflicting action can be in the same state. So this protocol will provide isolation but it would be slow, another problem would be that every action has to know all the data that it needs (there could be things like if statements that can make the situation complicated).

1.2.4 Two phase locking

The idea is that there should be no release before all the acquire are done. It does provide isolation. How we are gonna prove this is by constructing the graph after using 2PL, if that graph has no cycle the approch will be correct.