ACS Theory Assignment 3

Kai Arne S. Myklebust, Silvan Adrian

Handed in: January 4, 2019

Contents

1 Question 1: Reliability **1**
1.1 1 . . . 1
1.2 2 . . . 2
1.3 3 . . . 2

2 Question 2: Vector Clock **2**

3 Question 3: Distributed Coordination **3**

4 Question 4: Distributed Transactions **4**
4.1 1 . . . 4
4.1.1 Node 1 . . . 4
4.1.2 Node 2 . . . 4
4.1.3 Node 3 . . . 5
4.2 2 . . . 5
4.3 3 . . . 5

1 Question 1: Reliability

1.1 1

A daisy-chain network has a graph consisting of links:

$$l = n - 1 \tag{1}$$

The probability of a failure is p, the probability that there is no error is therefore $1 - p$. If all links are connected it means no link failure and since we assume that

a link failure is independent. we then get:

$$(1-p)^{n-1} \tag{2}$$

1.2 2

In the fully connected network there are 3 links that can fail and as long as 2 links are still fully functioning then also all the building are still connected. That 1 link fails we have the probability $p(1-p)^2$ and when 0 links fail: $(1-p)^3$. So we get that a fully-connected network is working is then:

$$3p(1-p)^2 + (1-p)^3 \tag{3}$$

In this example we didn't use a general approach and rather directly used the amount of links (3 and 2).

1.3 3

For above we have now the 2 probabilities from which we can calculate which is the more reliable solution:

For Daisy Chain we get: $p_d = (1-0.000001)^{3-1} \approx 0.999998$

For fully connected we get: $p_f = 3 * 0.0001(1-0.0001)^2 + (1-0.0001)^3 \approx 0.99999997$

So we get that a fully connected with the less reliable links would offer a better solution for the town

2 Question 2: Vector Clock

- **A1:** If clock before and Message clock have the same value, Action **A1** will be performed.
- **A2:** If clock before is on (0,0,0) then we run action **A2** for incrementing the clock and incorporate the update.
- **A3:** If clock before has a higher value then (0,0,0) we will perform the action **A3** to merge clock before and message clock.

Event	Clock Before	Message Clock	Action	Clock After
A	(0,0,0)	(1,0,0)	A2	(1,0,0)
B	(0,0,0)	(0,1,0)	A2	(0,1,0)

C	(0,1,0)	(1,0,0)	A3	(1,1,0)
D	(0,0,0)	(0,0,1)	A2	(0,0,1)
E	(1,1,0)	(1,1,0)	A1	(1,1,0)
F	(0,0,1)	(1,1,0)	A3	(1,1,1)
G	(1,0,0)	(0,1,0)	A3	(1,1,0)
H	(1,1,0)	(0,0,1)	A3	(1,1,1)
I	(1,1,1)	(1,1,1)	A1	(1,1,1)
J	(1,1,0)	(0,0,1)	A3	(1,1,1)
K	(1,1,1)	(1,1,1)	A1	(1,1,1)

3 Question 3: Distributed Coordination

While the 2-Phase commit protocol can't recover on a failure of both the coordinator and a participant the 3-phase commit protocol can, this because of it's asynchronous nature. The 3-phase protocol introdruces for that the prepared for commit state. If a coordinator fails in 3-phase protocol before sending a `preCommit`, the participants will unanimously agree that the operation has been aborted. The coordinator will not send out `doCommit` messages until all prepared to commit haven been `acked`, this eliminates the possability that any of the participants completed the transactions before all participants were aware of it (which is one of the reasons that there can appear infinite blocking).

4 Question 4: Distributed Transactions

4.1 1

4.1.1 Node 1

Since Transaction 2 wants to get an exclusive lock on `X` it needs to wait until T1 is finished with reading `X`.

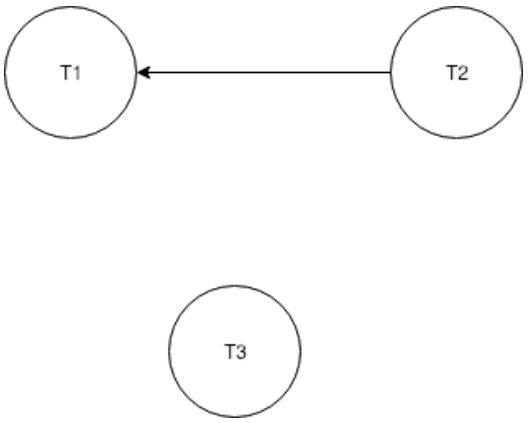

Figure 1: Waits-for-graph Node 1

4.1.2 Node 2

Transaction 3 wants to write on `B` while T2 reads `B`, so has to wait until T2 is finished.

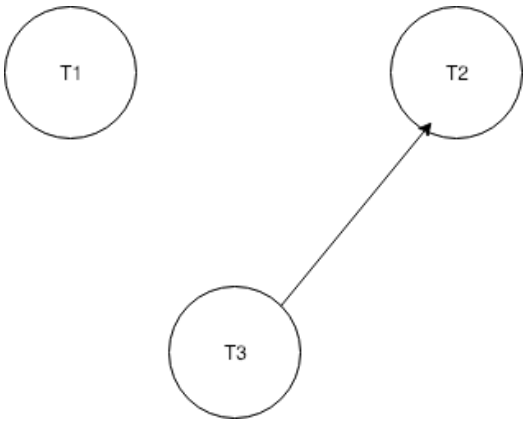

Figure 2: Waits-for-graph Node 2

4.1.3 Node 3

Transaction 1 wants to write `C` and T3 reads `C`, so T1 has to wait for.

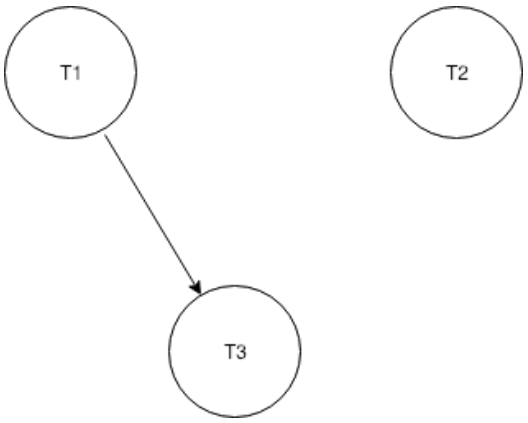

Figure 3: Waits-for-graph Node 3

4.2 2

After putting all the wait for graphs together, we get the following graph.

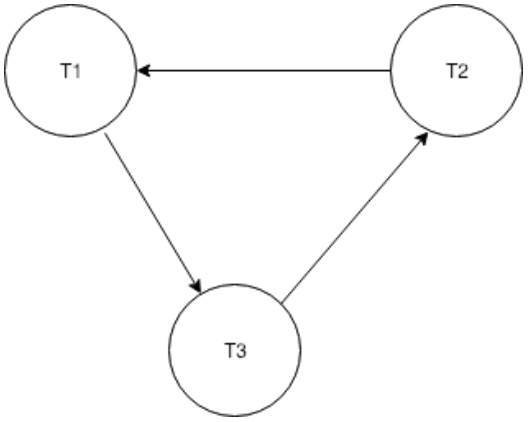

Figure 4: Global Waits-for-graph

4.3 3

-