Hand in two: Exercise 4.3 and 4.6
Group: IT03

Kasper Iverslien Borgbjerg (201808262) – 201808262@post.au.dk
Oliver Christensen (201808502) – 201808502@post.au.dk
Frederik Bay (201800246) – 201800246@post.au.dk

September 30, 2020

Kasper Iverslien Borgbjerg – 201808262 Oliver Christensen – 201808502
Frederik Bay Nørgaard – 201800246

Peer-to-peer network

The code attached to this report should cover the requirements for a peer-to-peer network, where same cannot be send or recieved more than once per client.

Exercise 4.3

Assume players use the above protocol based on vector clocks to communicate. For each send-event, specify the vector clock that accompanies the message that is sent. There are 4 messages sent, and hence 6 pairs of messages. For each such pair, use the vector clocks associated to the messages to determine if the messages in the pair are concurrent, or the pair is in the causal past relation.

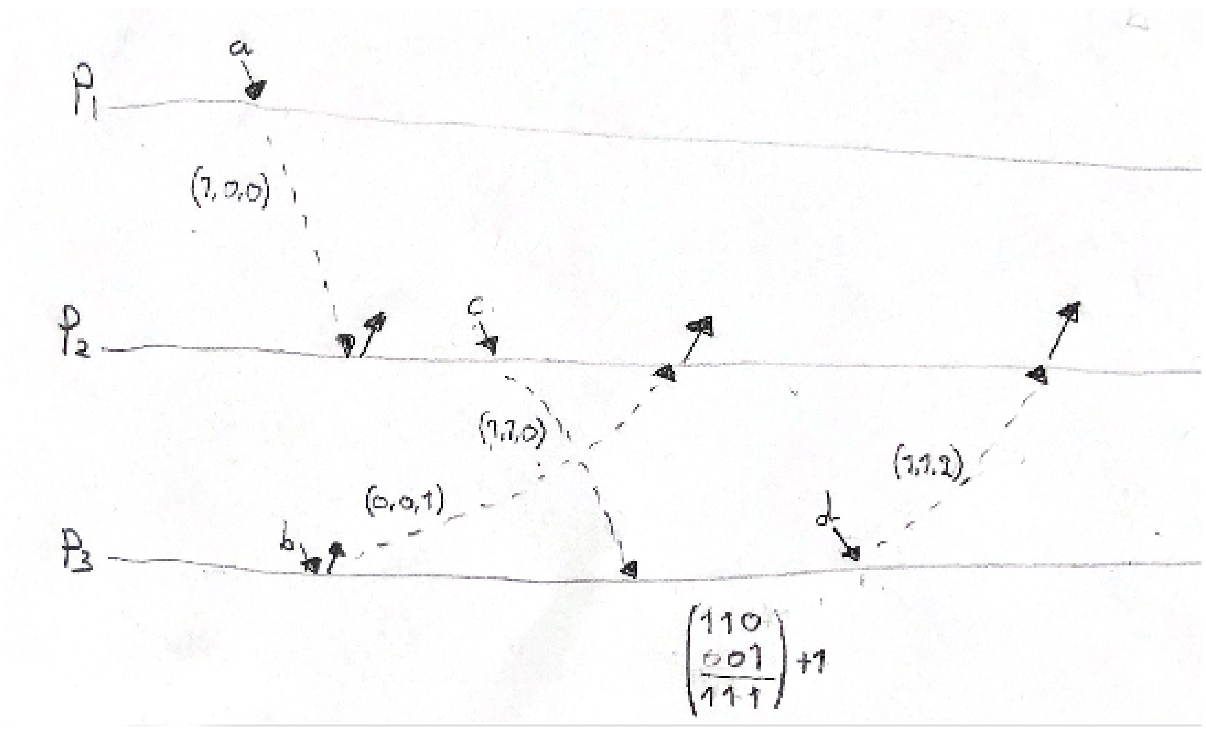

Figure 1

The following pairs from the four send events are as follows:

- a –> b = (1,0,0) (0,0,1) CONCURRENT
- a –> c = (1,0,0) (1,1,0) CASUAL PAST
- a –> d = (1,0,0) (1,1,2) CASUAL PAST
- c –> d = (1,1,0) (1,1,2) CASUAL PAST
- b –> c = (0,0,1) (1,1,0) CONCURRENT
- b –> d = (0,0,1) (1,1,2) CASUAL PAST

The argumentation of whether the events are concurrent or in CP of each other is determined by looking at the vector clock associated with the messages sent. Looking at the first pair we compare VC(1,0,0) with VC(0,0,1) and see that the decimals in the latter, are not bigger or related to the first VC, thus we can argue that mb is not in the casual past of ma and therefor concurrent. The argumentation for the other pairs, follows the same logic, where vector clocks that are concurrent are sorted in a lexicographically order, because in lexicographically order det vector clock (1,0,0) is regarded larger than (0,0,1).

Kasper Iverslien Borgbjerg – 201808262
Frederik Bay Nørgaard – 201800246
Oliver Christensen – 201808502

Exercise 4.6

Test you system and describe how you tested it.

The following system was tested through the terminal by connecting the peers to each other, and see how they would react in different situations. By connection two peer and see how they would react, and then connect a third to one of them. This would cause panic in some situations, which explains a lot why our code did not work as expected in the end. The peers are acting as follows. When the peer is executed, it will standby and wait for connecting in 20000 milliseconds, where then it will request in terminal for a transaction. It will request a "from", "to" and the amount to be transferred. When a transaction has been made, it will be send to its connections, and then it will either create or use the old account if it already initiated.

Discuss whether connection to the next ten peers is a good strategy with respect to connectivity. In particular, if the network has 1000 peers, how many connections need to break to partition the network?

When every peer connects to the next ten peers, it will uphold 100 connections, which then will partition the network, if those are removed. This secures high connectivity because, the whole network is connected, but still can be partitioned quite easy. This means that it upholds the 100 connections through ten peers. If one peer is connected to 10 peers and they are removed in a row, this peer will be separated from the network.

Argue that your system has eventual consistency if all processes are correct and the system is run in two-phase mode.

In the two phase model it secures eventual consistency in terms of all transactions being sent, even when there a new peers connecting the network. This is secured by storing a list of all transactions, which can be sent when a peer connects to the network. Though, in our the transactions are sent right away, and is only stored with a boolean determining if they are exuted or not.

Assume we made the following change to the system: When a transaction arrives, it is rejected if the sending account goes below 0. Does your system still have eventual consistency? Why or why not?

if a transaction is rejected, when the sending account goes below zero, the system will no longer uphold eventual consistency, if the transaction is sent. This means that the ten peers connected to peer, will not update their local ledger and not send this to their ten peers.