Distributed Algorithms

Solution for Project 2

Group 8

1. Alexandre Georges Rémi Martin, 406294
2. Benyamin Shafabakhsh, 406305
3. Yared Dejene Dessalk, 406228
4. Anubhav Guha, 406244
5. Jiaqiao Peng, 406038

December 14, 2018

2.1

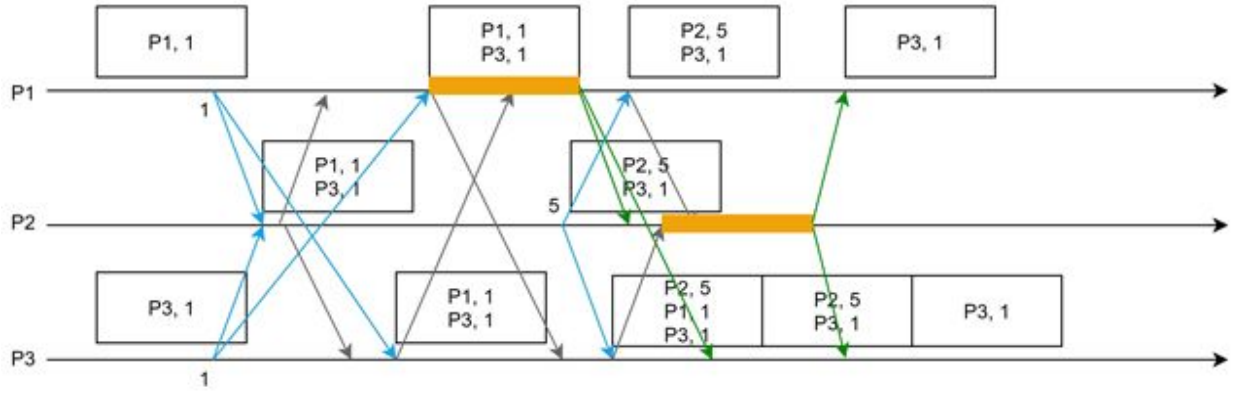
I.

If we use for example a LIFO for the Lamport algorithm, with the example following we will see that it does not work.

We have 3 processes :

1. P1 and P3 send a request
2. P1 accesses the medium as it is the first in the list
3. P2 requests an access
4. P1 releases the medium
5. P2 is on top of the pile and access the medium

The legend for the diagram above is the same as for the one course 5 slide 12.
We can see that the order of access is not fair and P3 risks to never have access to the medium.



II.

If we consider that a process can crash or that a message can be lost then the **algorithm is not deadlock free** as other processes have to wait for the previous process to be done accessing the medium.

But if we consider that processes do not crash, access the link in a reasonable period of time and messages always arrive in a reasonable period of time, then **the algorithm is deadlock free** :

There are two cases for a process to send a confirmation:
1. It does not need to access the medium or it needs to access the medium but the request has a smaller sequence number than the process, in this case, **it sends an immediate confirmation.**

2. it does need access to the medium, in this case :
   - ❖ There are other processes before it, it waits for them to confirm.

   - ❖ It is the first but not only process, it accesses the medium and when releases it sends confirmation to the requests that have been sent (aka all the processes in the list)

A process waits for n-1 confirmations to access the medium, all the processes that do not need it will confirm, and all the other ones are in the list and will confirm once they accessed the medium, so the only deadlock possibility is for previous processes to never access the medium. However each process acts the same way, waiting for previous processes before it, to confirm, until the first process in the list that gets only immediate confirmations as no other wanted to access the link first, answers with a confirmation and let it have access to the medium.

III.

It is feasible to use the maekawa algorithm with **n being a non squared number**, the result will not be a quadratic mesh but a rectangular one.
The result will be a non uniform load balancing between the nodes, the smaller side of the rectangle will be solicited more often than the longer one.
Example with n = 8 then n = 7 (which is a prime number for a better illustration)
n = 8

n = 7

On this example we can see that the algorithm is very inefficient, but still works as 2 or more processes are in common for each Pi and Pj.

2.2

I.

The Chandy-Lamport global snapshot algorithm works only for FIFO channels. It uses marker messages to separate the messages in a communication channel into those to be included in the snapshot from those not to be recorded. Using non-FIFO channels could lead to application messages overtaking the marker, thus leading to the recording of an inconsistent state of the channels and non meaningful global state.

To illustrate the above problem, let us consider the following example:

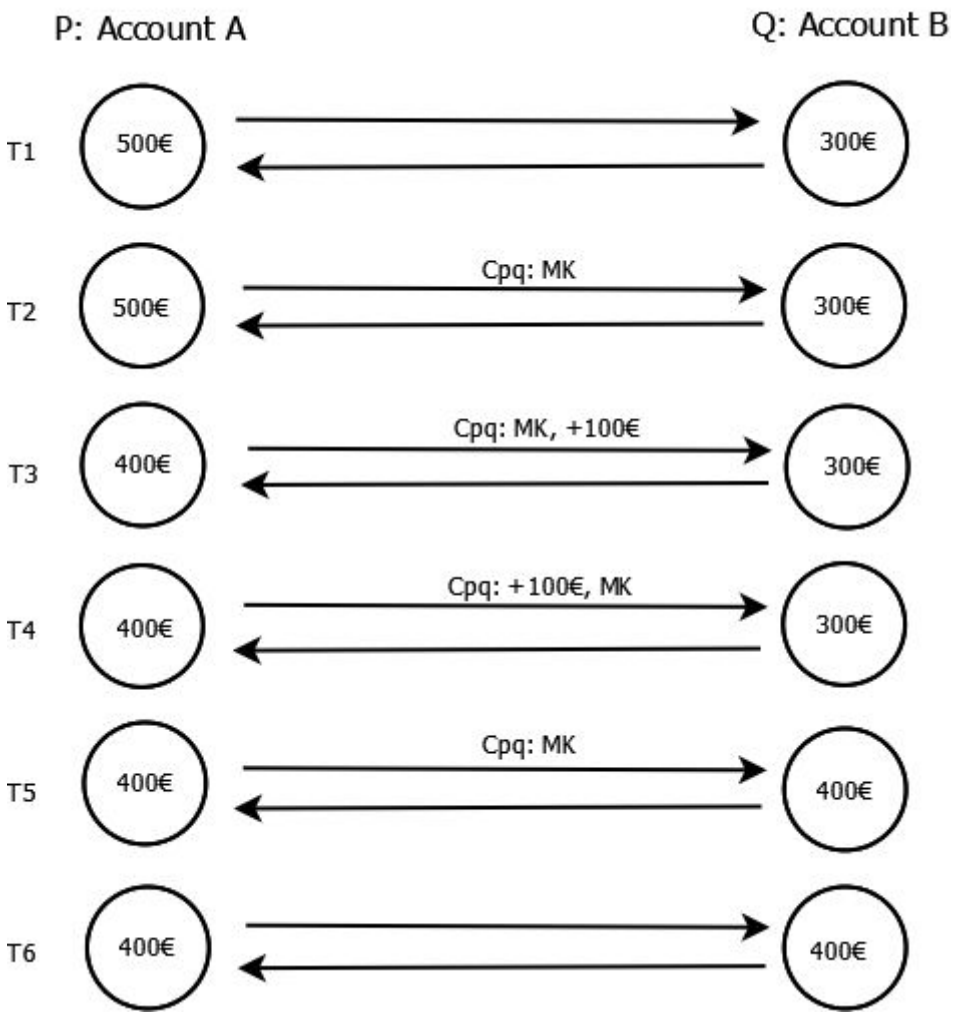
Let P and Q be two distinct branches of a distributed system which maintain bank accounts A and B respectively. Let us also consider that a branch refers to a process. Messages have the following formats:

MK : Marker message

+/- Amount: Credit/debit message where Amount shows the amount to be credited/debited

C_{pq}: Channel state from process P to Q

C_{qp}: Channel state from process Q to P



Implementing Chandy-Lamport algorithm, we will have the following sequences of actions, which are also in the above figure.

T1 => Initially, account A = 500€, account B = 300€
T2 => Branch P initiates its local snapshot and send maker message MK to process Q.
account A = 500€, account B = 300€, C_{pq} = MK
T3 => Branch P initiates a transfer of 100€ from account A to account B. account A is decremented by 100€ to 400€ and a request for +100€ credit message to account B is sent.
account A = 400€, account B = 300€, C_{pq} = MK, +100€
T4 => +100€ credit message overtakes marker message MK; this can happen since the channel is not FIFO
account A = 400€, account B = 300€, C_{pq} = +100€, MK
T5 => Branch P receives +100€ message and update its current balance.

account A = 400€, account B = 400€, C_{pq} = MK

T6 => Branch Q receives marker message MK and takes snapshot of its local state and send marker message to P

account A = 400€, account B = 400€

From the above sequence of actions, at T6 we will have a global snapshot collected from the two branches(account A = 500€, account B = 300€). Since the +100€ credit message from branch P overtakes the ongoing maker message in the non-FIFO communication channel, the credit message, which is not part of the snapshot taken by P, is included when process Q takes its snapshot. This result incorrect balances in the two accounts and the overall global state will not be consistent. So, the Chandy-Lamport algorithm can not be implemented in non-FIFO communication channels.

II.

If the channel states of the snapshot are always empty and basic messages may be buffered at the receiving processes, we can use Lamport timestamps being tagged in all basic and marker messages. Using Lamport timestamps, we can measure time and causality. The idea is that each message carries a timestamp, and that we can compare these timestamps as a way to determine whether one message happened before or after the marker message. Doing this for all basic messages in the buffer of the receiving process, the receiving process can distinguish all message that needs to be executed before taking the snapshot.

III.

We can modify the Chandy-Lamport algorithm for a non-FIFO communication channel by using message marker which is tagged with the number of basic messages sent into a channel before the marker message. It also uses freezing of the neighboring processes until they acknowledge taking their own snapshot.

The pseudocode for this algorithm is given below:

Marker receiving rule for process p_i (from p_j)

On receipt of a marker message with TAG at p_i over channel *c*:

if (p_i has not yet recorded its state) it

executes all pre-snapshot basic messages(by comparing its id with TAG number from the marker message);

records its process state;

Sends an acknowledgment to p_j;

end if

Marker sending rule for process p_i

After p_i has recorded its state, for each outgoing channel *c*:

TAG = the number of messages sent over *c* after the snapshot is taken;
p_i sends one marker message having tag number of TAG over *c*
(before it sends any other message over *c*);
BLOCK = all neighboring process

Acknowledgment receiving rule for process p_i (from p_j)

Remove process p_j from BLOCK;

IV.

It is known that node running the Lai-Yang algorithm takes local snapshot and keeps all historical states of the attached communication channels. If a single node fails and is not in synchronization with global execution, it keeps taking its local snapshot. But when the node returns to the normal state, its local snapshots will not be consistent with the global snapshot state.