Sheet 3, SS 2015
Verteilte Systeme/ Distributed Systems
Published 28.05.2015

Katinka Wolter
Simon Schmitt
Inst. für Informatik
Freie Universität Berlin

Distributed Systems Lecture - Discussion 04.06.2015

Assignment 1. Clock algorithms

1. Please explain the difference between a logical clock and a vector clock.
2. Please add the values of the vector clocks to the following graph using the algorithm VC2:

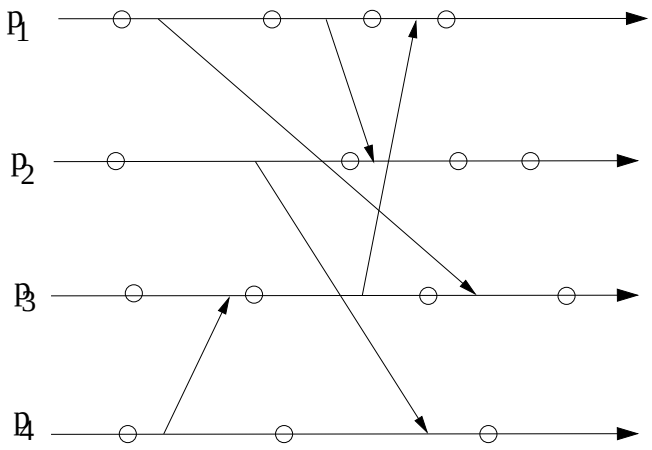

Figure 1: Processes with vector clocks

3. What property does the algorithm satisfy?

Assignment 2. Clock algorithms

Please consider the following clock algorithm.

1. Please use the above figure to explain how the algorithm works.
2. Please specify a property using clock values and relations which is satisfied by the algorithm.

P ::
var
    v: **array**[1...N] of integer
for any initial state s :
    $(\forall i, i \neq s.p : s.v[i] = 0) \wedge (s.v[s.p] = 1)$;

send event (s, send, t);
    $t.v[t.p] := s.v[t.p] + 1$;

receive event(s, receive(u), t);
    $t.v := s.v$;
    $t.v[u.p] := \max(s.v[u.p], u.v[u.p])$;

internal event(s, internal, t);
    $t.v[t.p] := t.v[t.p] + 1$;

Figure 2: A clock algorithm

Distributed Systems Seminar - Discussion 11.06.2015

Assignment 1. FUcoin - The Implementation Phase I

During the discussion of Bitcoin we have seen that bitcoin is an event-based mechanism, where each node needs to have global knowledge of all events. We have discussed scalability issues and it became clear that a bitcoin system can only run on powerful machines.

Our system will be different in that it shall be truly distributed, scalable and that it can run on small, maybe also mobile devices. The nodes in our system have only partial knowledge of the system. Therefore, we propose a state-based system - you should discuss whether a system based on events would work too.

We want to implement FUcoin using the akka framework. There are several steps necessary to implement a working and distributed banking system. We do not focus on security or trust in any step, since that topic is handled in other courses. We explicitly allow negative balance on an account and also negative transactions.

The specified interface between participants is as follows:

Listing 1: FuCoin Interface Specification

```
class Wallet {
        public:
                vector<WalletPointer> join();
                void storeOrUpdateWallet(Wallet w);
                void invalidateWallet(Wallet w);
                void receiveTransaction(int amount);
                vector<WalletPointer> searchWallet(String address);
```

```
        private:
                void leave();
                void performTransaction(WalletPointer w, int amount);

                String address;
                int moneyAmount;
                vector<WalletPointer> allKnownNeighbors;
                vector<Wallet> synchronizedNeighbors;
}

class WalletPointer {
        public:
                String address;
}
```

A participant (for simplicity a.k.a. Wallet) is identified by its address and holds its savings as an attribute. A wallet can be a thread on a machine or a process in a network.

A participant can join the network using a pre-known participant as an entry point. When that happens, the entry point returns a set of N pointers to other known neighbors (function *join*).

A participant has to store itself (the wallet object) on these N neighbors and update them if a transaction occurs in order to ensure consistency in the network (function *storeOrUpdate-Wallet*).

After a participant joins, he has to find its Wallet in the network (function *searchWallet*). This could be achieved by asking the N previous neighbors. However, they might have left the network in the meantime. If that is the case, the joining participant shall create a gossip wave through the network in order to find at least one participant holding the wallet. After the wallet is found, old wallet versions in the network have to be invalidated.

If a participant wants to store its wallet the requesting participant shall be added to the known-neighbors list (function *storeOrUpdateWallet*).

If a node leaves the network, it has to ensure that the wallet objects it might hold remain accessible in the network. Therefore it shall push its objects to an arbitrary neighbor.

Please implement and test the specified features. Please develop your solution in this repository which should be accessible to you: `https://git.imp.fu-berlin.de/2015SSDistributedSystems/DS15`. Create a subfolder for your project/group.