DISTRIBUTED ALGORITHMS 2013

SEMINAR 1

Archana Nottamkandath & István Haller

Vrije Universiteit Amsterdam

October 30, 2013
Welcome to the DA seminar!

OUR MISSION

Understand and solve DA problems.

Seminars twice a week:

- Wednesday 11.00-12.45 in room M607
  lecturer Archana Nottamkandath, a.nottamkandath@vu.nl

- Thursday 11.00-12.45 in room S607
  lecturer István Haller, ihr210@few.vu.nl
Seminars contain exercises with algorithms from the previous course

Exercises available on the course website, in the textbook http://www.cs.vu.nl/~tcs/da/

Presence at seminars is optional but advisable
Motivational slide

- Exercises *very similar* to the ones at the exam
- Keep up to date with the contents of the course
- Not enough time at the end to go through all the algorithms
SEMINAR PREPARATION

- Take a look at the exercises before the seminar - very useful
- Participate with ideas on how to solve an exercise before writing the solution on the blackboard
- Each student should solve at least one exercise at the blackboard
And when you have questions, do ask them!
EXERCISE 2.3

Define the union of $S_1 = (C, \xrightarrow{1}, I)$ and $S_2 = (C, \xrightarrow{2}, I)$ as $S = (C, \rightarrow, I)$ with $\rightarrow = (\xrightarrow{1} \cup \xrightarrow{2})$. Prove that if $P$ is an invariant of $S_1$ and $S_2$, then $P$ is an invariant of $S$.

INVARIANT

Conditions of Invariant:

- Invariant is true in all Initial Conditions
- Invariant is preserved in all transitions
Exercise 2.4

Consider the execution below:

Use Lamport’s logical clock to assign clock values to these events. Do the same for the vector clock.
Lamport’s clock $LC$ assigns to each event $a$ the length $k$ of a longest causality chain $a_1 \prec \ldots \prec a_k = a$.

To compute $LC$:

- if $a$ is internal or send event, and $k$ is the previous clock value at that process, then $\Theta(a) = k + 1$
- if $a$ is receive event, $k$ is the previous clock value at receiving process, and $b$ is the send event corresponding to $a$, then $\Theta(a) = \max(k, \Theta(b)) + 1$
**Reminder: Vector Clock**

- VC maps occurrences of events in a computation s.t. $a \prec b \iff VC(a) \prec VC(b)$.

- $VC(a) = (k_0; ..; k_{N-1})$ where each $k_i$ is the length of a longest causality chain $a^i_1 \prec .. \prec a^i_{k_i}$ of events at process $p_i$ with $a^i_{k_i} \prec a$
Define the causal order for the transitions of a system with **synchronous** communication.

**Observations**

- Synchronous communication: sending and receiving of the same message are one atomic event
- For synchronous communication, the **sending party** must wait with sending a message until the **receiving party** is ready to receive. Therefore, the sending party can read the state (i.e. **clock value**) of the receiving party.
Adapt Lamport’s logical clock for synchronous systems, and give a distributed algorithm for computing the clock at run-time.

**Lamport’s clock for Asynchronous Communication**

Lamport’s clock $LC$ assigns to each event $a$ the length $k$ of a longest causality chain $a_1 \prec \ldots \prec a_k = a$.

To compute $LC$:

- if $a$ is *internal* or *send* event, and $k$ is the previous clock value at that process, then $\Theta(a) = k + 1$
- if $a$ is *receive* event, $k$ is the previous clock value at receiving process, and $b$ is the send event corresponding to $a$, then $\Theta(a) = \max(k, \Theta(b)) + 1$
Exercise 2.6

Give an example where $LC(a) < LC(b)$ while $a$ and $b$ are concurrent events.

Reminder: Logical clock

A logical clock $C$ maps occurrences of events in a computation to a partially ordered set such that $a \prec b \Rightarrow C(a) < C(b)$
**Exercise 2.7**

Give an algorithm to compute the vector clock at run-time.

**Vector Clock**

- VC maps occurrences of events in a computation s.t. $a \prec b \iff VC(a) \prec VC(b)$.
- $VC(a) = (k_0; ..; k_{N-1})$ where each $k_i$ is the length of a longest causality chain $a_i^1 \prec .. \prec a_i^k$ of events at process $p_i$ with $a_i^k \prec a$. 
EXERCISE 3.1

Give an example to show that the Chandy-Lamport algorithm is flawed if channels aren’t FIFO.
REMINDER: CHANDY-LAMPORT ALGORITHM

- initiators - take local snapshot and send \(<mkr>\) messages to all their neighbors
- non-initiators - when they receive the \(<mkr>\) message (for the first time), they take local snapshot and send \(<mkr>\) to all their neighbors
- a node \(q\) computes as the channel state for \(pq\) the messages it receives via \(pq\) after taking its local snapshot and before receiving \(<mkr>\) from \(q\)
EXERCISE 3.2

Propose an adaptation of the Chandy-Lamport algorithm, in which basic messages may be buffered at the receiving process, and the channel states of the snapshot are always empty.
Give an example in which the Lai-Yang algorithm computes a snapshot that is not a configuration of the on-going execution.
Reminder: Lai-Yang Algorithm

- Initiators - take local snapshot and append true to each outgoing basic message sent after the snapshot.
- Non-initiators - when they receive a message with true or a control message, they take their local snapshot of the state before the reception of this message.
- A node $q$ computes as the channel state for $pq$ the basic messages without the true tag it receives via $pq$ after taking its local snapshot.
- A node $p$ sends a control message to $q$, informing the latter about how many basic messages without the true tag $p$ has sent into $pq$. 
Adapt the Lai-Yang algorithm such that it supports multiple subsequent snapshots.
**Reminder: Lai-Yang Algorithm**

- **Initiators** - take local snapshot and append `true` to each outgoing basic message sent after the snapshot.

- **Non-initiators** - when they receive a message with `true` or a control message, they take their local snapshot of the state before the reception of this message.

- A node `q` computes as the channel state for `pq` the basic messages without the `true` tag it receives via `pq` after taking its local snapshot.

- A node `p` sends a control message to `q`, informing the latter about how many basic messages without the `true` tag `p` has sent into `pq`. 
Exercise 3.5

Give a snapshot algorithm for undirected networks with non-FIFO channels which uses:

- marker messages, tagged with the number of basic messages sent into a channel before the marker message
- acknowledgements, and
- temporary (local) freezing of the basic computation