


## Today's Topics



### Slide 1

More on Chapter 10.

- Logical Clocks, Vector Clocks.
- Global State.

Plus leader (if we have time)

- Leader election (Chapter 11.3)

## Revision



### Slide 2

- For all events it should be false that  $a \rightarrow a$ .
- If  $a$  happens before  $b$  on the same processor then  $a \rightarrow b$ .
- $a \rightarrow b$  and  $b \rightarrow c$  implies  $a \rightarrow c$ .
- If  $a$  is the event of sending a message and  $b$  is the event of receiving that message then  $a \rightarrow b$ .

This is a partial order. If there are two events  $a$  and  $b$  such that:

$$a \not\rightarrow b \wedge b \not\rightarrow a$$

Then  $a$  and  $b$  are concurrent, often written  $a \parallel b$

## Concurrency

### Slide 3

- Concurrency is not about things happening at the same time.
  - With the problem of clock synchronisation it is hard to say things happen at the same time.
- Concurrency is about things being independent.
  - If  $a \parallel b$  then  $a$  could happen at the same time as  $b$ , or  $a$  could happen before  $b$ , or  $b$  could happen before  $a$ .

## Vector Timestamps

### Slide 4

Problem with Lamport timestamps.

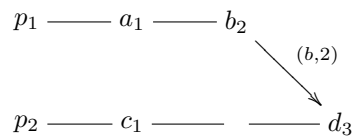
- $a \rightarrow b$  implies  $L(a) < L(b)$

But

- $L(a) < L(b)$  does not imply  $a \rightarrow b$ .

## Vector Timestamps $L(a) < L(b) \not\Rightarrow a \rightarrow b$

Slide 5



- $L(c) < L(b)$  but  $c \parallel b$ .

The trick is to extend Lamport timestamps so that we can somehow capture causality.

## Vectors

If we have  $n$  processes we will be interested in vectors of length  $n$  one entry for each processor.

Slide 6

- Given a vector,  $V$ , let  $V[i]$  be the value of the  $i$ 'th component corresponding to process  $i$ .
- We say that  $V < W$  if for all  $i$

$$V[i] < W[i]$$

- For example
  - $(1, 2, 3) < (2, 4, 5)$
  - $(1, 2, 1) \not< (2, 1, 2)$  and  $(2, 1, 2) \not< (1, 2, 1)$

## Vector Clocks

### Slide 7

- Each processor has its own vector clock. The  $i$ 'th entry represents the information it has about the events that have occurred on process  $i$ .
- When a process receives a message it uses the information it gets to update what it knows about other processes.

## Vector Clocks

Putting all this together we have the following rules (where  $V_i$  is the vector clock for process  $i$ ):

**VC1** Initially  $V_i[j] = 0$  for all  $i$  and  $j$ ;

**Slide 8** **VC2** Given an internal event,  $p$  in a process  $V_i[i] = V_i[i] + 1$  then  $p$  is timestamped with  $V_i$ .

**VC3** On sending a message include the timestamp.

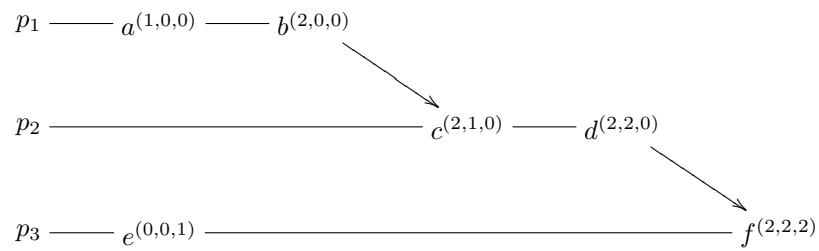
**VC4** When a process  $i$  receives a message with time stamp  $t$ . sets

$$V_i[j] = \max(V_i[j], t[j])$$

for all  $j$  and then  $V_i[i] = V_i[i] + 1$  to signify that the process has received the message.

### Example: Vector Timestamps

Slide 9



### Why do we use Timestamps?

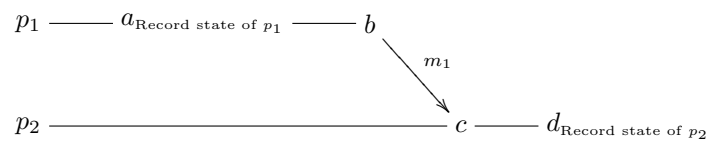
Slide 10

- Timestamps give us information what has happened before.
- It gives us a way of tracking the partial order  $\rightarrow$ .
- Application include: checkpoints and recovery, garbage collection, causal memory, detecting global properties of the system, distributed event simulations (making real and logical time correspond), group communications algorithms, ...
- Practical problems include trying to minimise the message payload (reducing the amount of information that needs to be sent in a timestamp).

## Global State

Slide 11

- What is the global state of a distributed system?
- Suppose you have two processors  $p_1$  and  $p_2$
- $p_1$  records its state and sends a message to  $p_2$  after  $p_2$  receives the message  $m_1$  it records its state.



- This is not necessarily a good thing. The message was saved as being sent but in  $p_1$ 's recorded state the message was not sent.

## State Recording: Applications

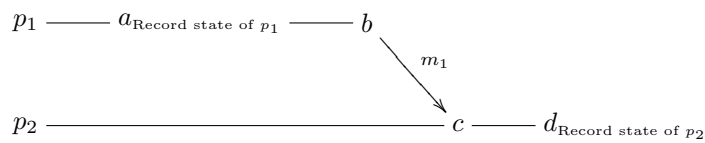
Slide 12

- Distributed Garbage Collection. A collection of objects distributed across processors. We want to know if an object can be safely deleted. We need to know if there are no other processes in the network referencing the object or that there are not other messages in transit referencing that object.
- Deadlock detection. Is there a cycle of nodes all waiting for messages from each other before they can proceed.
- Termination. Has some computation stopped?

## Cuts

Back to the example:

Slide 13



The reason why we have a problem is that we have event  $c$  but not event  $b$  that preceded it and  $b \rightarrow c$ . This is called an inconsistent state.

We want to record states that respect the causal relations  $\rightarrow$

## Cuts

Slide 14

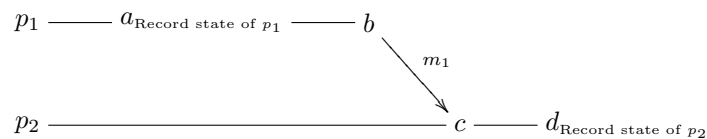
- Given a process  $p_i$  its history is the set of events that occur. For example on the previous slide  $\text{history}(p_1) = \langle a, b \rangle$  and  $\text{history}(p_2) = \langle c, d \rangle$ . Note the history must include send and receive operations.
- A cut for every process  $p_i$  in the system some prefix of its history. For example the cut in the previous case would be:

$$\langle a \rangle, \langle c, d \rangle$$

- A cut  $C$  is called consistent if for each event it contains, it also contains all the events that happened-before that event. That is for all  $e \in C$  if  $f \rightarrow e$  then  $f \in C$ .

## Consistent Cuts

Back to the example:



Slide 15

The Consistent cuts are as follows:

- $\langle a \rangle, \langle \rangle$  where  $\langle \rangle$  is the empty sequence.
- $\langle a, b \rangle, \langle \rangle$
- $\langle a, b \rangle, \langle c \rangle$
- $\langle a, b \rangle, \langle c, d \rangle$

## The Snapshot Algorithm

The snapshot algorithm is a distributed algorithm which records the global state of a set of processes, which also records messages in transit. The algorithm assume that:

Slide 16

- neither channels nor processes fail: communication is reliable;
- Channels are unidirectional deliver messages in the order that they are sent;
- The graph of processes and channels is strongly connected (there is a path between any two processes);
- any process may initiate a global snapshot at any time and the algorithm does not interfere with what is running at the moment.

## The Snapshot algorithm



The essential idea of the algorithm is as follows.

### Slide 17

- Each process records its state and also for each incoming channel a set of messages sent to it.
- The process records, for each channel, any message that arrived after it recorded its state and before the sender recorded its own state.

This scheme allows us to record the states of processes at different times but to account for the differences between process states in terms of messages transmitted but not yet received.

## The Snapshot Algorithm



### Slide 18

The algorithm proceeds through a use of special marker messages, which are distinct from any other messages the process sends or receives. The marker has a dual role as a prompt for the receiver to save its own state, if it has not already done so; and as a means of determining which messages to include in the channel state.

## The Snapshot Algorithm

The algorithm has two transitions for each process:

Slide 19

**Marker Receiving rule** If the process has not yet recorded its state then record its state and start to record any message that arrive to it on incoming channels. If it has already recorded its state it saves the state as the initial recorded state plus all messages that where sent.

**Marker Sending Rule** If the process receives a marker it sends the marker out to every other processes in the network that it is connected to via outgoing channels..

Examples done on the board.

## Leader Election Algorithms

Slide 20

Some of the algorithms for clock synchronisation required a leader process. We want an algorithm the always gives one leader to a group of processes. There are two on the market:

- The Bully Algorithm
- The Ring Algorithm

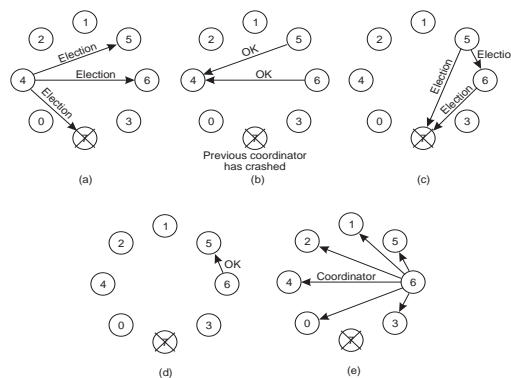
## The Bully Algorithm

Slide 21

- Each process has a number.
- If a process,  $P$  notices that there is no leader or it receives an ELECTION message then it sends an ELECTION to all higher number processes.
- If one of the higher-up answers, it takes over then  $P$  can stop otherwise  $P$  takes over.

## The Bully Algorithm

Slide 22



## The Ring Algorithm

Same idea as Bully algorithm but send a message only to the next higher process alive process.

### Slide 23

