EE 382V: Parallel Algorithms **Fall 2017**

Lecture 3: August 19

Lecturer: Vijay Garg *Scribe: Howie Benefiel*

3.1 Lamport Clocks

The motivation for this is that we need some mechanism to tell us $e \to f$.

Definition 3.1 *A map C is a logical clock if $c : E \to \mathbb{N}$ s.t. $e \to f \Rightarrow c(e) < c(f)$*

Notice in Definition 3.1 that C is a unidirectional material conditional, not bidirectional. Numbers should monotonically increase with a process Every process should send a number when it receives an event - take $max(P_1, P_n)$ then add 1.

3.1.1 Lamport Clock Algorithm

Algorithm 1 Calculate a logical clock for a distributed system

```
c ← 0                          ▷ Initialization
c ← c + 1                      ▷ on internal event
c ← c + 1                      ▷ on send event
Send c with message
c ← max(c, d) + 1              ▷ on receive of message with d as clock
```

This algorithm will always return a logical clock which will monotonically increase with events in the system.

3.2 Vector Clocks

3.2.1 Motivation

Using the simple Lamport Clock, we cannot infer that e happened before f. That is because of the unidirectional implication in the definition. To get a complete ordering, can we come up with a map where

Definition 3.2 *$V : E \to \mathbb{N}^n$ s.t. $e \to f \Leftrightarrow V(e) < V(f)$*

Notice in Definition 3.3 that we have if and only if. This means that if we have the values $V(e)$ and $V(f)$, we can determine whether e happened before f.

Definition 3.3 *x, y are vectors in n-dimension*

$$\forall i : x[i] \leq y[i] \wedge \exists j : x[j] < y[j]$$

As an example of how this comparison works, $(2,7,9) < (3,7,9)$ and $(2,7,9)||(3,5,2)$. To map natural numbers to poset, you need vectors. You cannot do it with a scalars. As an example of how this works,

3.2.2 Algorithm for Vector Clock

Algorithm 2 An algorithm to track vector clocks

P_i:: V: array[1 ... n] of int
init: all 0's except V[i] = 1;
$V[i] \leftarrow V[i] + 1$ ▷ for internal event
$V[i] \leftarrow V[i] + 1$ ▷ for send event
Send V with message
$V \leftarrow max(V, W)$ ▷ for receive event with vector W
$V[i] \leftarrow V[i] + 1$

Proof: The proof that this algorithm returns a consistent clock which gives a total order is taken in two parts.

1st Part: $e \rightarrow f \Rightarrow V(e) < V(f)$
Proof
This is proven by the vector increasing along each edge in the system diagram.

2nd Part: $e \not\rightarrow f \Rightarrow V(e) \not< V(f)$
Proof
$e \not\rightarrow f \Rightarrow$ there is no path from e to f $\Rightarrow V(f)[i] < V(e)[i]$
e is on $P_i \wedge$ f is on $P_j \Rightarrow V(e) \not< V(f)$ ■ Explanation P_2 does not know about P_1, so P_1's values for P_1 must be larger.

3.2.3 Directly Precedes

If we only care about direct messages, i.e., no transitivity, we can get away with only sending the component for the sending process. Each process still stores a full vector with an element for each process.