

# Distributed Systems Module

## Concurrency Parallelism & Distributed Systems

Carlos Segarra

December 17, 2019

### Contents

<b>1</b>	<b>Concepts of Distributed Systems</b>	<b>2</b>
1.1	Definition of a Distributed System . . . . .	2
1.2	Challenges of Distributed Systems . . . . .	2
<b>2</b>	<b>Distributed Algorithms</b>	<b>2</b>
2.1	Time & Global States . . . . .	2
2.2	Coordination and Agreement . . . . .	3
<b>3</b>	<b>Distributed Shared Data</b>	<b>3</b>
3.1	Distributed Transactions . . . . .	3
3.2	Consistency & Replication . . . . .	3

# 1 Concepts of Distributed Systems

## 1.1 Definition of a Distributed System

## 1.2 Challenges of Distributed Systems

# 2 Distributed Algorithms

## 2.1 Time & Global States

### Time

**Logical Clocks** Implemented to capture the happened-before relation. They satisfy:

1. If  $a$  and  $b$  are two events in the same process,  $a \rightarrow b \Rightarrow C(a) < C(b)$
2. If  $a$  sends a message to  $b \Rightarrow C(a) < C(b)$

### Lamport's logical clocks:

- Each process  $P_i$  has a counter  $C_i$
- $C_i$  is updated using the following rules:
  1. When an event happens at  $P_i$  increment  $C_i$  by one.
  2. When  $P_i$  sends a message, set  $ts(m) = C_i$
  3. When  $P_i$  receives a message, set  $C_i = \max(C_i, ts(m))$  and then increase  $C_i$  by one.

Lamport's clocks do not guarantee that if  $C(a) < C(b) \Rightarrow a \rightarrow b$ .

### Vector clocks:

- Each process  $P_i$  has an array  $VC_i[1, \dots, n]$
- It is updated as follows:
  1. When  $P_i$  sends a message  $m$ , it adds 1 to  $VC_i[i]$  and sends  $m$  with  $ts(m) = VC_i$ .
  2. When  $P_j$  receives a message from  $P_i$ , it updates each  $VC_j[k]$  to  $\max(VC_j[k], ts(m)[k])$  and increments  $VC_j[j]$  by one.

### Global States

A global state of the system is necessary for:

- Failure Recovery
- Detection of Properties: deadlocks, termination
- Debugging

We define some concepts:

1. The **history** of a process  $p$  is the sequence of events occurred at that process:  $h(p) = \langle p_0, p_1, \dots \rangle$  (either internal or message sending).
2. The **state**  $i$  of process  $p$  is  $p$ 's history until event  $i$ :  $s_i(p) = \langle p_0, \dots, p_i \rangle$ .
3. The **global history** is the union of all the individual histories.
4. A **cut** is the global history up to a specific event in each process history.
5. A cut is **consistent** if it contains all the *happened-before* events. A consistent cut corresponds to a **consistent global state**.
6. A **run** is a total ordering of all events in a global history consistent with each local history.

7. A **linearization** or **consistent run** is a run consistent with the *happened-before* relation.

8. We say state  $S'$  is reachable from  $S$  if there is a linearization such that  $S$  precedes  $S'$ .

### Distributed Snapshot

#### Global Predicates

Consistent global states form a lattice with reachability relation between sets. A **global state predicate**,  $\varphi$  is a property that is either true or false for a global state.

- A predicate is **stable** if once it becomes true, it remains true for all reachable states.
- A predicate is **non-stable** if it can become true and then false.
- A predicate  $\varphi$  possibly happened: if it is true for any of the consistent states in the lattice.
- A predicate  $\varphi$  definitely happened: if all paths from origin to end contain a consistent global state for which the predicate is true.

## 2.2 Coordination and Agreement

Mutual Exclusion

Election Algorithms

Multicast Communications

Consensus

## 3 Distributed Shared Data

### 3.1 Distributed Transactions

Introduction

Problems with Concurrent Transactions

Concurrency Control

Distributed Transactions

### 3.2 Consistency & Replication

Introduction

Data-Centric Consistency Models

Client-Centric Consistency Models

Consistency Protocols