

# Distributed Systems

Eddy Caron\*

ENS de Lyon

## Contents

<b>1</b>	<b>Algorithm for Distributed System</b>	<b>2</b>
1.1	Modelization . . . . .	2
1.2	Fairness . . . . .	2
1.3	Network topology . . . . .	2
1.4	How to write a distributed algorithm? . . . . .	3
<b>2</b>	<b>Communication protocols</b>	<b>3</b>
2.1	Sliding window communication protocol . . . . .	3
2.2	Timer-based protocol . . . . .	5
<b>3</b>	<b>Wave algorithm</b>	<b>5</b>
3.1	Wave algorithm . . . . .	5
3.2	The tree algorithm . . . . .	6
3.3	The echo algorithm . . . . .	7
3.4	The Polling Algorithm . . . . .	8
3.5	The Traversal algorithm . . . . .	8
3.6	Tarry's Graph Traversal Algorithm . . . . .	8

---

\*hadriencroubois.com

**Final grade** 2/3 Final exam + 1/3 mid-term exam + 1/3 (project + partial)

We will use the programming language *Erlang* (1987  $\rightsquigarrow$  1998)

- Concurrent, real time, distributed
- $\Rightarrow$  Use BEAM

Variables can be integers, float, PID, functions, tuples, maps, ... and atoms (specific to Erlang).

There are built-in functions for message passing

Lists are not strongly typed, tuples are like python's

In Erlang, there is no overwriting of the variables (cannot affect them a new value).

## 1 Algorithm for Distributed System

### 1.1 Modelization

**Definition 1** (Transition relation). Let  $S = (\mathcal{C}, \rightarrow, \mathcal{I})$  be a transition system. An execution of  $S$  is a maximal sequence  $E = (\gamma_0, \gamma_1, \dots, \gamma_n)$

**Definition 2** (Terminal configuration). A terminal configuration is a configuration  $\gamma$  for which there is no  $\delta$  such that  $\gamma \rightarrow \delta$ . Note that a sequence  $E = (\gamma_0, \gamma_1, \gamma_2, \dots)$

$\rightarrow$  this notion is very tricky in parallel programming, as it is difficult to know whether all the executions are finished or not.

**Definition 3.** A configuration  $\delta$  is reachable from  $\gamma$  (notation  $\delta \rightsquigarrow \gamma$ ), if there exist a sequence  $\gamma = \gamma_0, \gamma_1, \gamma_2, \dots, \gamma_k$  with  $\gamma_i \rightarrow \gamma_{i+1}$  for all  $0 \leq i < k$ . Configuration  $\delta$  is reachable if it is reachable from an initial configuration.

**Definition 4** (System with Asynchronous Message Passing). Cf APPD course

### 1.2 Fairness

**Definition 5.** An execution is weakly fair if no event is applicable in infinitely many consecutive configurations without occurring in the execution

**Definition 6.** An execution is strongly fair if no event is applicable in infinitely many configuration without occurring in the execution.

### 1.3 Network topology

Many topology exists, including :

- Ring
- Tree
- Hypercube
- Star
- Clique

You can "change the topology" of your network to adapt and algorithm, for example taking a star sub-graph of a clique.

## 1.4 How to write a distributed algorithm?

- A specific code for a specific node (or family of node)
  - The sender code and the receiver code
  - The initial code and the non-initial code
- One code for all
  - The same code is executed on each node
  - A requirement can switch to the right code for a node

Use label to separated the code between initiators, non-initiator or reception of a specific message (for example stopping the execution).

**Warning** The execution flow is not clear: there is no assumption about the label selected for the execution (it is randomly chosen), so the algorithm must run for all the chosen labels for al the processors in the current state.

It is useful to try to find an incorrect workflow to test whether the algorithm is correct.

No assumptions must be made on the execution time (especially linking synchronization with the size of the code)!

## 2 Communication protocols

### 2.1 Sliding window communication protocol

**Example** Write an algorithm to exchange informations between both process

```
1 Var:  
2  $data_{in}=N$ ;  
3  $data_{recv} = -1$ ;  
4  $is\_sent = -1$ ;  
5 label 1{ $is\_sent==1$ }  
6 | send  $\langle msg, data_{in} \rangle$   $is\_sent=0$ ;  
7 end  
8 label 2{receive  $\langle msg, i \rangle$ }  
9 |  $data_{recv}=i$ ;  
10 end
```

The number of exchanged message can be a good measure of the performance of the algorithm.

```

    // Now, we want to send an array of data:
1  Var:
2  datain=N;
3  datarecv = -1;
4  j=0;
5  i=0;
6  label 1{is_sent[i]==1}
7    | send ⟨msg,datain[i]⟩;
8    | is_sent[i++]=0;
9  end
10 label 2{receive ⟨msg,v⟩}
11 | datarecv[j++] = v;
12 end

```

```

    // Now, we had a procedure lost that erase messages:
1  label Lostn=⟨msg⟩
2  | kill(msg)
3  end

```

```

    // We can do it with using 0 acknowledgements, by using the data you need to send as
    an acknowledgement:
1  Var:
2  sp is the number of words received by p from q (without missing ones);
3  lp, lq is the number of ahead values → size of the sliding window;
4  label Sp:{ap ≤ i < sp + lp}
5  | send ⟨pack,inp[i],i⟩ to q;
6  end
7  label Rp:{⟨pack,w,i⟩ ∈ Qp}
8  | receive ⟨pack, w, i⟩;
9  | if outp[i] == -1 then
10 | | outp[i] = w;
11 | | ap = max{ap, i - lq + 1};
12 | | sp = minj{outp[j] = -1};
13 | else
14 | | ignore;
15 | end
16 end
17 label Lp:{⟨pack, w, i⟩ ∈ Qp}
18 | Qp = Qp \ ⟨pack, w, i⟩;
19 end

```

## 2.2 Timer-based protocol

Create channel, and see whether it is opened or not (See slides for details).

## 3 Wave algorithm

### 3.1 Wave algorithm

We need to solve broadcast, global synchronisation, trigger events, parallel computing, data-parallelism.

For that, we use message passing.

Requirements:

- Termination
- Decision
- Dependence

List of aspects:

- Centralized: one initiator
- Decentralized:  $n$  initiators
- Network: topology, undirected links, connected, *Asynchronous* on *Synchronous*

Initial knowledge:

- Name
- Name of its neighbour

Most of these algorithm send “empty messages” (simple token).

The ring algorithm is a wave algorithm.

### 3.2 The tree algorithm

On a tree:

- All leaves initiate the algorithm
- Each process sends exactly one message

```
1  $rec_p \leftarrow 0$  // # of received message
2  $father_p \leftarrow \text{undefined}$ ;
3 begin Initiator
4   for all  $q \in \text{Neighbourhood}(p)$  do
5     | send  $\langle \text{tok} \rangle$  to  $q$ ;
6   end
7   while  $rec_p < \# \text{Neighbourhood}(p)$  do
8     | receive  $\langle \text{tok} \rangle$ ;
9     |  $rec_p \leftarrow rec_p + 1$ ;
10  end
11  decide;
12 end
13 begin Non-initiators
14   receive  $\langle \text{tok} \rangle$  from some neighbour  $q$ ;
15    $father_p \leftarrow q$ ;
16    $rec_p \leftarrow rec_p + 1$ ;
17   for all  $q \in \text{Neighbourhood}(p) \setminus \{father_p\}$  do
18     | send  $\langle \text{tok} \rangle$  to  $q$ ;
19   end
20   while  $rec_p < \# \text{Neighbourhood}(p)$  do
21     | receive  $\langle \text{tok} \rangle$ ;
22     |  $rec_p \leftarrow rec_p + 1$ ;
23   end
24   send  $\langle \text{tok} \rangle$  to  $father_p$ 
25 end
```

### 3.3 The echo algorithm

```
1  $rec_p \leftarrow 0$  // # of received message
2  $father_p \leftarrow \text{undefined}$ ;
3 begin Initiator
4   for all  $q \in \text{Neighbourhood}(p)$  do
5     send  $\langle \text{tok} \rangle$  to  $q$ ;
6   end
7   while  $rec_p < \# \text{Neighbourhood}(p)$  do
8     receive  $\langle \text{tok} \rangle$ ;
9      $rec_p \leftarrow rec_p + 1$ ;
10  end
11  decide;
12 end
13 begin Non-initiators
14   receive  $\langle \text{tok} \rangle$  from some neighbour  $q$ ;
15    $father_p \leftarrow q$ ;
16    $rec_p \leftarrow rec_p + 1$ ;
17   for all  $q \in \text{Neighbourhood}(p) \setminus \{father_p\}$  do
18     send  $\langle \text{tok} \rangle$  to  $q$ ;
19   end
20   while  $rec_p < \# \text{Neighbourhood}(p)$  do
21     receive  $\langle \text{tok} \rangle$ ;
22      $rec_p \leftarrow rec_p + 1$ ;
23   end
24   send  $\langle \text{tok} \rangle$  to  $father_p$ ;
25 end
```

### 3.4 The Polling Algorithm

Cf slides

### 3.5 The Traversal algorithm

They are a subset of wave algorithms. All events are totally ordered by the causality relation, and the last event occurs in the same process as the first event.

**Definition 7** (f-traversal algorithm). *An algorithm is a f-traversal algorithm for a class of network if:*

- *It is a traversal algorithm*
- *In each computation,*

**Example** The ring algorithm is a traversal algorithm.

### 3.6 Tarry's Graph Traversal Algorithm

We start from a connected graph, and apply two rules:

- A process never use the same communication channel twice
- A process will send to its father a message only if there is no other choice in respect to the first rule

// TODO
---------