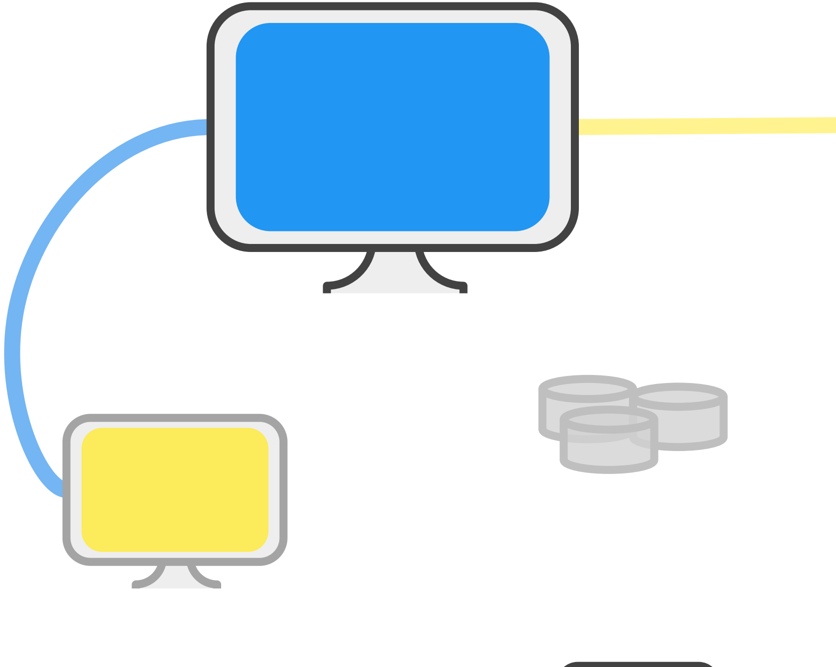
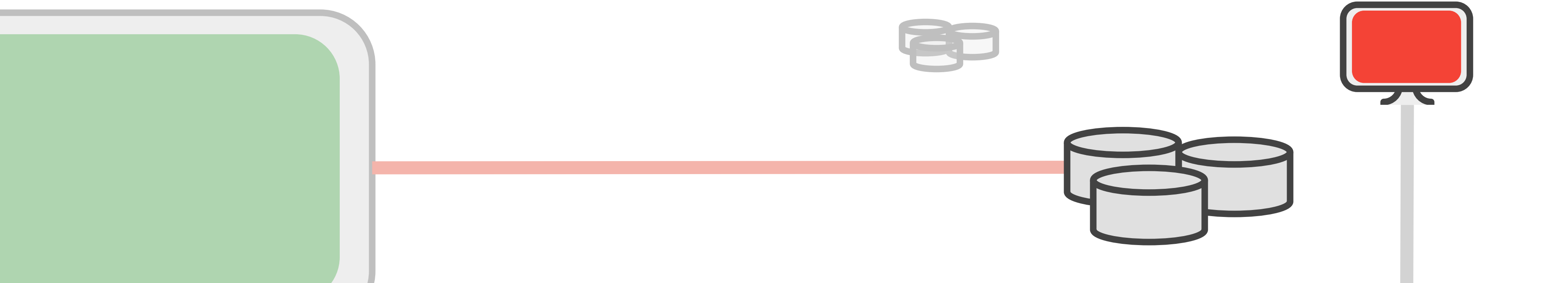
A distributed solution of the distributed termination problem

Rana, S. P. "A distributed solution of the distributed termination problem." *Information Processing Letters* 17.1 (1983): 43-46.

Presented by Fernanda Mora
February, 2016

Contents





Introduction

Motivation

- Distributed system-> Lack of knowledge on the global state.
- A process has no up-to-date knowledge on the local states of other processes.
- Distributed termination and deadlock problems arise.
- Termination: several algorithms, but rely upon designation of a process to detect global termination.
- Proposal: any of the processes may initiate a wave to detect global termination.

Rana’s algorithm

Motivation: an incorrect detection algorithm

Idea: detection & termination waves.

Basic messages are acknowledged.

A process is quiet if it is passive and all the messages are acknowledged.

1. Quiet process starts a wave tagged with its ID.
2. Only quiet processes take part of this wave.
3. If a wave completes, its initiator says “Termination”.

What is the problem?

Consider a process *P* that was not yet visited by a wave.

So *P* may take a quiet process *Q* active again by sending it a message!

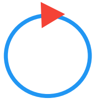
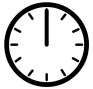
Then *P* may become quiet and take part of the wave.

Eventually this wave may complete while q is active!

Solution: use a logical clock to provide each event with a time-stamp!

Assumptions (Rana's Algorithm)

- Message-passing communication (control & basic), which can be time-stamped 

- Unique identification number 1,2,3...
- Hamiltonian cycle 
- Decentralized system 
- Symmetric processes
- Synchronized processes (logical clock) 
- A process P_i can note down the clock when B_i is satisfied
- Usual assumptions

General idea

A process is passive if B_i is satisfied, in other case it is active.

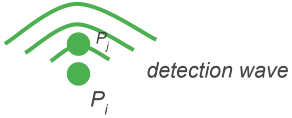
If an active process P_i satisfies B_i it sends a time-stamped detection message + counter.

If an active process receives a detection message it purges it.

If a passive process receives a detection message:

- If counter=n then termination message.

- If counter≠n
  - If the message has a greater time-stamp then it sends message and counter++.
  - Otherwise it purges it.



The algorithm (CSP notation)

$P :: [P_1 \parallel \cdots \parallel P_n]$

where, for each $1 \leqslant i \leqslant n$, $P_i :: *[S_i]$.

// Definimos el proceso concurrente *P*

$P_i :: B_i := \text{false};$

$\quad *[\ S_i'$

$\quad\ \square B_i \rightarrow \text{BTIME}_i := \text{CLOCK-TIME}$

// S_i'= P_i waits for messages (if it receives a message then B_i becomes false)

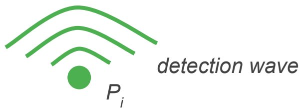
BTIME$_i$ = Time when P_i last satisfied B_i

CLOCK-TIME = current clock-time of process P_i

The algorithm (CSP notation)

```
Pi :: Bi := false;
  * [ Si
    □Bi → BTIME_i := CLOCK-TIME;
          TIME := BTIME_i;     COUNT:=1
          P_{i+1} !detection-message (TIME, COUNT)
    □P_{i-1} ?detection-message (TIME, COUNT) →
      [¬Bi → purge the message
      □Bi →
        [TIME < BTIME_i → purge the message
        □TIME ≥ BTIME_i →
          COUNT := COUNT + 1;
          P_{i+1} !detection-message
             (TIME, COUNT)
          ]
      ]
```

If B_i becomes passive then stamps and sends detection message



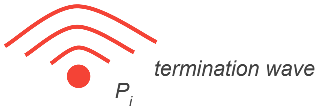
P_i receives detection message:
If active it purges it.
If passive then:

If finished after TIME it purges message
If finished before TIME it joins detection wave and sends det. message

The algorithm (CSP notation)

```
P_i :: B_i := false;
  * [S_i'
  □B_i → BTIME_i := CLOCK-TIME];
          TIME := BTIME_i;
          COUNT := 1;
          P_{i+1} ! detection-message (TIME, COUNT)
  □P_{i-1} ? detection-message (TIME, COUNT) →
     [COUNT = n →
        P_{i+1} ! terminate-message;
         TERMINATE
     □COUNT ≠ n →
        [¬B_i → purge the message
        □B_i →
           [TIME < BTIME_i → purge the message
           □TIME ≥ BTIME_i →
              COUNT := COUNT + 1;
              P_{i+1} ! detection-message
                 (TIME, COUNT)
           ]]]
   P_{i-1} ? termination-message →
     P_{i+1} ! termination-message;
      TERMINATE
  ]
```

If *COUNT=n*, P_i sends termination message and terminates

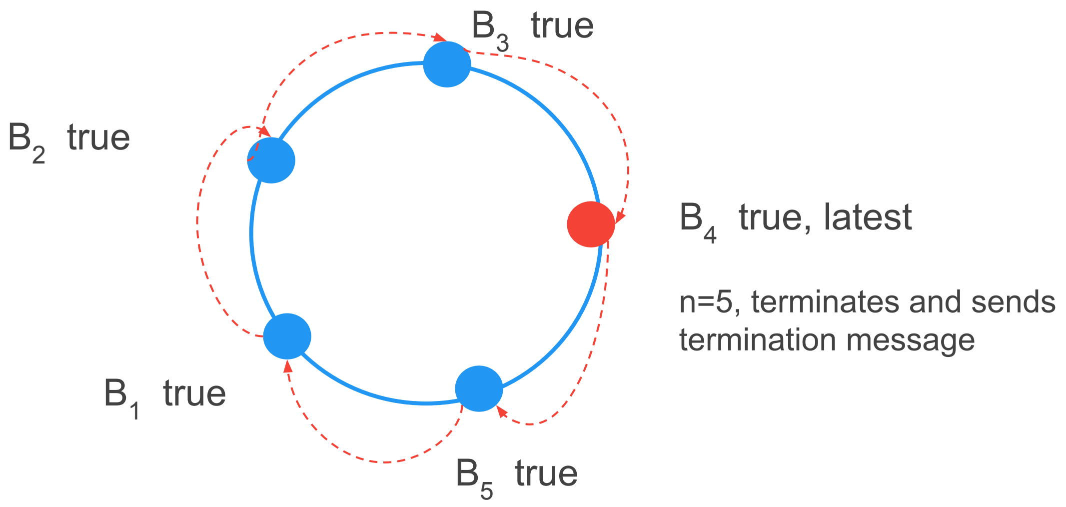


If P_i receives termination message then sends message and terminates itself

Correctness

Correctness: assertion 1

- *If the global termination condition is satisfied, termination will be eventually detected*



- *There is no possibility of detecting false termination*

Discussion

Remarks

- Difficult to estimate # of messages passed in the detection phase.
- Exactly *n* messages are passed in the termination phase.
- More than one process can detect termination: no problem (other algorithms do have problems on this!)
- Other processes require to generate spanning tree.
- Limitation: synchronous communication (CSP), messages on transit. Approaches to deal with asynchronicity:
  - Modify global condition: (B_i is true $\forall i$) & (no message on transit)
  - Modify local predicates B_i: quiet = passive & all messages are acknowledged.

References

- Fokkink, Wan. *Distributed Algorithms: An Intuitive Approach*. MIT Press, 2013.
- Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." *Communications of the ACM* 21.7 (1978): 558-565.
- Hoare, Charles Antony Richard. *Communicating sequential processes*. Springer New York, 1978.
- Van Wezel, Michiel C., and Gerard Tel. "An assertional proof of Rana's algorithm." *Information processing letters* 49.5 (1994): 227-233.

Thanks