Termination Detection

Diverse Approaches

1

Outline

- Introduction to Termination Detection
- Weight Throwing Approach for Termination Detection

16 July 2024

2

Outline

- Introduction to Termination Detection
- Weight Throwing Approach for Termination Detection

16 July 2024

3

What is Termination Detection?

- To determine if a distributed computation has terminated.
- Why it's non-trivial?
  - No single process has complete knowledge of the global state, and global time does not exist.
- Termination is detected if every process is locally terminated and no message is in transit between nodes.

16 July 2024

4

Conditions

- *Locally terminated* state is a state in which a process has finished its computation and will not restart any action unless it receives a message.
- In the termination detection problem, a particular process (or all of the processes) must infer when the underlying computation terminates.

16 July 2024

5

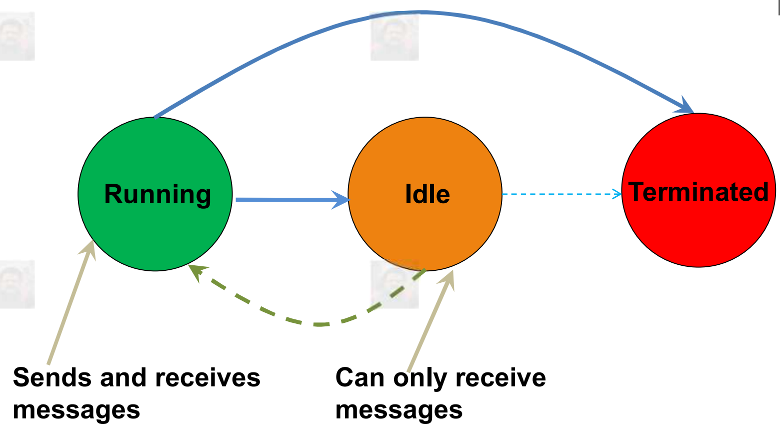
Conditions

- A termination detection (TD) algorithm must ensure the following:
  - Execution of a TD algorithm cannot indefinitely delay the underlying computation.
  - The termination detection algorithm must not require addition of new communication channels between processes.

16 July 2024

6

Process States



Definition

- A distributed computation is said to be terminated at time instant t_0 iff:
  - $(\forall i: p_i(t_0)=\text{“terminated”} \wedge (\forall\, i, j: c_{i,j}(t_0)=0)$
  - where, $p_i(t)$ denotes the state of process p_i at instant t.
  - $c_{i,j}(t)$ denotes number of messages in transit in the channel at instant from process p_i to process p_j

Outline

- Introduction to Termination Detection
- Weight Throwing Approach for Termination Detection

16 July 2024

9

Weight Throwing Algorithm

- A controlling agent monitors the computation.
- Communication channel exists between each of the processes and the controlling agent and also between every pair of processes.
- Initially, all processes are in the idle state.

16 July 2024

10

Weight Throwing Algorithm

- The weight at each process is zero and at the controlling agent it's one.
- The computation starts when the controlling agent sends a control message to one of the processes.
- A non-zero weight W (0<W<1) is assigned to each process in the active state and to each message in transit.

16 July 2024

11

Weight Throwing Algorithm

- A process sends a part of its weight in every message that it sends.
- When a process receives a message, it adds the weight received in the message to its weight.
- Thus, the sum of weights on all the processes and on all the messages in transit is always 1.

16 July 2024

12

Weight Throwing Algorithm

- When a process becomes idle, it sends its weight to the controlling agent in a control message, which the controlling agent adds to its weight.
- The controlling agent concludes termination if its weight becomes 1.

Notations

- W: weight on a process in general.
- B(DW): a basic message B sent as a part of the computation, where DW is the weight assigned to it.
- C(DW): a control message C sent from a process to the controlling agent where DW is the weight assigned to it.

The Algorithm

- Rule 1:
  - The controlling agent or an active process may send a message to one of the processes, say P, by splitting its weight W into W_1 and W_2 such that $W_1+W_2=W$, $W_1>0$, and $W_2>0$.
  - The sender then assigns its own weight as $W:=W_1$ and sends a basic message $B(DW:=W_2)$ or a control message $C(DW:=W_2)$ to process P.

16 July 2024

15

The Algorithm

- Rule 2:
  - On receipt of the message B(DW), process P adds DW to its weight W (W:=W+DW).
  - If the receiving process is in the idle state, it becomes active.
- Rule 3:
  - A process switches from the active state to the idle state at any time by sending a control message C(DW:=W) to the controlling agent and making its weight W:=0.

16 July 2024

16

The Algorithm

- Rule 4:
  - On receipt of a message C(DW), the controlling agent adds DW to its weight (W:=W+DW).
  - If W=1, then it concludes that the computation has terminated.

16 July 2024

17

Performance of the Algorithm

- A: set of weights on all active processes
- B: set of weights on all basic messages in transit
- C: set of weights on all control messages in transit
- Wc : weight on the controlling agent.
- Invariant I_1: $Wc + \sum W_{W \in (A \cup B \cup C)} = 1$
- Invariant I_2: $\forall\ W \in (A \cup B \cup C), W>0$

16 July 2024

18

Performance of the Algorithm

- Invariant I_1 states that sum of weights at all active processes including the controlling one, and on all basic and control messages in transit is always 1.
- Invariant I_2 states that weight at each active process, as well as on each basic or control messages in transit is non-zero.

Safety of the Algorithm

- According to I_1,
  - $Wc = 1 \Rightarrow \sum W_{W \in (A \cup B \cup C)} = 0$
- Applying I_2 on this,
  - $Wc = 1 \quad \Rightarrow (A \cup B \cup C) = \phi$

    $\Rightarrow (A \cup B) = \phi$
- $(A \cup B) = \phi$ implies the computation has terminated.
- ∴, it never detects a false termination.

Liveness of the Algorithm

- Again, as per I_1, $(A \cup B) = \phi$ implies
  - $Wc + \sum W_{W \in C} = 1$
- Since the message delay will be finite after the computations have terminated in all nodes, eventually Wc will be 1.
- Thus, the algorithm detects a termination in finite time.

16 July 2024

21

Thanks for your kind attention

Questions??

22