RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE20(O)/CSE751
Credits: 3:0:0:1/3:0:0

Term: October – February 2022

Faculty:
Sini Anna Alex

Message-Optimal Termination Detection

- The network is represented by a graph G = (V, E), where V is the set of nodes, and E $\subseteq$ V$\times$V is the set of edges or communication links.
- The communication links are bidirectional and exhibit FIFO property.
- The algorithm assumes the existence of a leader and a spanning tree in the network.
- If a leader is not available, the minimum spanning tree algorithm of Gallager et al. can be used to elect a leader and find a spanning tree using O(|E| + |V| log|V|) messages.
- Spanning-tree-based termination detection algorithm is inefficient in terms of message complexity because every message of the underlying computation can potentially cause the execution of one more round of the termination detection algorithm, resulting in significant message traffic.

Message-Optimal Termination Detection

- Consider the example shown in Figure 2.
- Suppose before node q receives message m, it has already sent a white token to its parent.
- Node p can not send a white token to its patent until node q becomes idle.
- To insure this, node p changes its color to black and sends a black token to its parent so that termination detection is performed again.

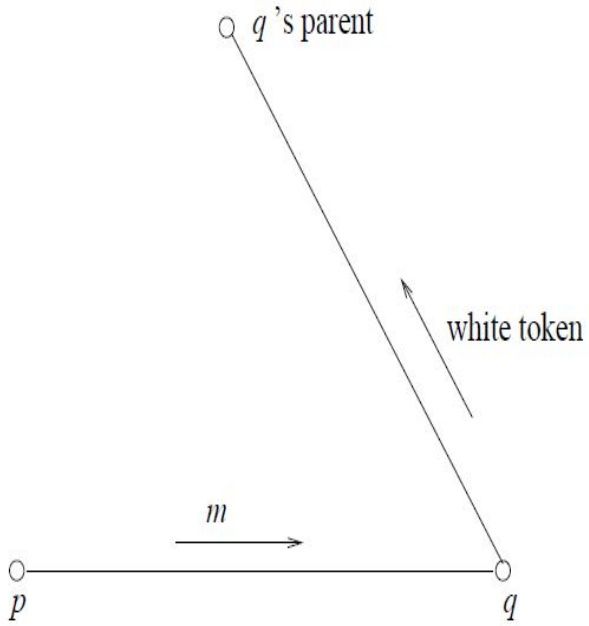

Figure 2: Node p sends a message m to node q that has already sent a white token to its parent.

Message-Optimal Termination Detection

- Initially, all nodes in the network are in state NDT (not detecting termination) and all links are uncolored.
- For termination detection,the root node changes its state to DT (detecting termination) and sends a warning message on each of its outgoing edges.
- When a node p receives a warning message from its neighbor, say q, it colors the incoming link (q, p) and if it is in state NTD, it changes its state to DT, colors each of its outgoing edges, and sends a warning message on each of its outgoing edges.
- When a node p in state DT sends a basic message to its neighbor q, it keeps track of this information by pushing the entry TO(q) on its local stack.
- When a node x receives a basic message from node y on the link (y, x) that is colored by x, node x knows that the sender node y will need an acknowledgement for this message from it.
- The receiver node x keeps track of this information by pushing the entry FROM(y) on its local stack.

Message-Optimal Termination Detection

. . . Formal Description of the Algorithm

- Procedure receive_message is given below:

Procedure receive_message(y: neighbor);
(* performed when a node x receives a message from its neighbor y on the link (y,x) that was colored by x *)

begin
receive message from y on the link (y,x)
if (link(y,x) has been colored by x) **then**
push FROM(y) on the stack
end;

- When a node p becomes idle, it calls procedure stack_cleanup, which examines its stack from the top, and for every entry of the form FROM(q), it deletes the entry and sends the *remove_entry* message to node q.

Message-Optimal Termination Detection

Formal Description of the Algorithm

- Node p repeats this until it encounters an entry of the form TO(x) on the stack.
- The idea behind this step is to inform those nodes that sent a message to p that the actions triggered by their messages to p are complete.
- **Procedure** stack_cleanup;
  begin
    while (top entry on stack is not of the form "TO()") do
      begin
        pop the entry on the top of the stack;
        let the entry be FROM(q);
        send a remove_entry message to q
      end
  end;

Message-Optimal Termination Detection

- Node x on receipt of the control message *remove_entry* from node y, examines its stack from the top and deletes the first entry of the form TO(y) from the stack.
- If node x is idle, it also performs the *stack_cleanup* operation.
- The procedure *receive_remove_entry* is defined as follows:

Procedure receive_remove_entry(y: neighbor);
(* performed when a node x receives a remove_entry message from its neighbor y *)

begin
scan the stack and delete the first entry of the form TO(y);
if idle **then**
stack_cleanup
end;

Message-Optimal Termination Detection

- A node sends a terminate message to its parent when it satisfies all the following conditions:
  1. It is idle.
  2. Each of its incoming links is colored (it has received a warning message on each of its incoming links).
  3. Its stack is empty.
  4. It has received a *terminate* message from each of its children (this rule does not apply to leaf nodes).
- When the root node satisfies all of the above conditions, it concludes that the underlying computation has terminated.

Message-Optimal Termination Detection

Performance

- In the worst case, each node in the network sends one warning message on each outgoing link. Thus, each link carries two warning messages, one in each direction.
- Since there are |E| links, the total number of warning messages generated by the algorithm is 2*|E|.
- For every message generated by the underlying computation,exactly one *remove_message* is sent on the network.
- If M is the number of messages sent by the underlying computation, then at most M *remove_entry* messages are used.
- Finally, each node sends exactly one *terminate* message to its parent and since there are only |V| nodes and |V|−1 tree edges, only |V| − 1 *terminate* messages are sent.
- Hence, the total number of messages generated by the algorithm is 2* |E| + |V| − 1 + M.
- Thus, the message complexity of the algorithm is O(|E| + M) as |E| > |V| − 1 for any connected network.
- The algorithm is asymptotically optimal in the number of messages.

Thank you