Wave and Traversal Algorithms

CS60002: Distributed Systems

Pallab Dasgupta
Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

Wave Algorithms

- A *wave algorithm* is a distributed algorithm that satisfies the following three requirements:
  - **Termination:** *Each computation is finite*
  - **Decision:** *Each computation contains at least one decide event*
  - **Dependence:** *In each computation each decide event is causally preceded by an event in each process*

The Echo Algorithm – a wave algorithm

var rec_p : integer init 0; // Counts no of recvd mesgs
$father_p$: process init *udef*;

For the initiator

begin forall $q \in Neigh_p$ do send ⟨ tok ⟩ to q ;
while $rec_p < \#Neigh_p$ do
begin receive ⟨ tok ⟩ ; $rec_p = rec_p + 1$ end ;
decide
end

For non-initiators

begin receive ⟨ tok ⟩ from neighbor q ; $father_p = q$; $rec_p = rec_p + 1$;
forall $q \in Neigh_p$, $q \neq father_p$ do send ⟨ tok ⟩ to q ;
while $rec_p < \#Neigh_p$ do
begin receive ⟨ tok ⟩ ; $rec_p = rec_p + 1$ end ;
send ⟨ tok ⟩ to $father_p$
end

Traversal Algorithms

- A *traversal algorithm* is an algorithm with the following three properties:
  - *In each computation there is one initiator, which starts the algorithm by sending out exactly one message*
  - *A process, upon receipt of a message, either sends out one message or decides*
  - *The algorithm terminates in the initiator and when this happens, each process has sent a message at least once*

Sequential Polling – a traversal algorithm

var rec_p : integer init 0; // For initiator only

For the initiator

begin while $rec_p < \#Neigh_p$ do
    begin send ⟨ tok ⟩ to $q_{recp + 1}$;
        receive ⟨ tok ⟩ ; $rec_p = rec_p + 1$
    end ;
    decide
end

For non-initiators

begin receive ⟨ tok ⟩ from q ; send ⟨ tok ⟩ to q ; end

Classical Depth-first Search

var $used_p[q]$: boolean init false for each $q \in Neigh_p$;
$father_p$: process init $udef$;

// For the initiator only – execute once

begin $father_p = p$; choose $q \in Neigh_p$;
$used_p[q]$ = true ; send $\langle$ tok $\rangle$ to q ;
end

Classical Depth-first Search contd..

// For each process, upon receipt of $\langle$ tok $\rangle$ from q_0:

```
begin if father_p = udef then father_p = q_0 ;
        if ∀q ∈ Neigh_p: used_p[q]
          then  decide
        else if ∃q ∈ Neigh_p: (q ≠ father_p ∧ ¬used_p[q])
          then begin if father_p ≠ q_0 ∧ ¬used_p[q_0]
                    then q = q_0
                    else choose q ∈ Neigh_p \ { father_p } with ¬used_p[q] ;
                  used_p[q] = true ; send ⟨ tok ⟩ to q
                end
          else begin used_p[ father_p ] = true ;
                    send ⟨ tok ⟩ to father_p
               end
end
```

Classical Depth-first Search Algorithm

- *The classical depth-first search algorithm computes a depth-first search spanning tree using* 2|E| *messages and* 2|E| *time units*

Awerbuch's DFS Algorithm

- Prevents the transmission of the token through a frond edge
- When process *p* is first visited by the token
  - *p* informs each neighbor *r,* except its father, of the visit by sending a ⟨vis⟩ message to *r*
  - The forwarding of the token is suspended until *p* has received an ⟨ack⟩ message from each neighbor
- When later, the token arrives at *r*, *r* will not forward the token to *p,* unless *p* is *r*'s father
- Awerbuch's algorithm computes a depth-first search tree in $4N - 2$ time units and uses $4.|E|$ messages

Cidon's DFS Algorithm

- The token is forwarded immediately
- The following situation is important:
  - Process *p* has been visited by the token and has sent a ⟨vis⟩ message to its neighbor *r*
  - The token reaches *r* before the ⟨vis⟩ message from *p*
  - In this case *r* may forward the token to *p* along a frond edge
- The situation is handled as follows:
  - Process *p* records to which neighbor it most recently sent the token – normally it expects to get it back from the same
  - If it gets it back from some other neighbor it *ignores the token*, but marks the edge *rp* as used, as if it received a ⟨vis⟩ message from *p*
  - When *r* eventually receives the ⟨vis⟩ message from *p* it behaves as if it never had sent the token to *p*
- Cidon's algorithm computes a DFS tree in $2N - 2$ time units and uses $4.|E|$ messages