Distributed Systems

Termination Detection

Rik Sarkar

University of Edinburgh
Fall 2016
Termination detection

How do we know when a distributed computation has ended?

We track nodes being in state “idle” Vs “Active”

Assume: an idle node becomes active only on receiving a message from some other node.
  – (exception: the initiator: leader/server etc.)

Termination is all nodes being idle
Termination detection (weight throwing)

• We suppose that the computation is started by a process s.
  – This means, other (idle) processes start working (becomes active) after receiving message from s or some other process
  – They have no other way to know that a computation is in progress

• s wants to know when all other processes have concluded working

• S starts with weight = 1.0

• Other processes start with weight = 0
Weight throwing

• When a process sends a message, it puts part (say, half) of its weight in the message.
• When a process receives a message, it adds the message weight to its own weight.
• When a process has finished computing, (becomes idle) it sends its current weight to s
• When s has weight=1.0, it knows no other process is active
Termination detection (weight throwing)

• Works on the assumption that no message is lost
  – Methods like TCP give good guarantee for delivery
  – Many other distributed algorithms have this assumption
  – Useful for their termination detection

• Drawback:
  – What if there are many messages?
  – (Homework!)
Termation detection (Dijkstra-scholten)

• Maintains a tree of which node initiated computation at which other node
• Each node has active children counter (cc)
• When node x sends a message to y
  – x increments cc
  – If y was idle
    • y becomes active
    • y remembers x as the parent
  – If y was already active
    • y sends ack to x
• When x receives an ack
  – x decrements cc
• When y finishes all computation and is idle
  – And has cc = 0
    • y sends ack to parent
Termination detection (Dijkstra-scholten)

• How do you describe its Message complexity?