CSc72010

General Synchronous Networks

Reading

Sections 4.2-4.4

Introduction

Basic tasks:

Broadcast
Convergecast
Loop-free communication

In this section of the course, we look at important proof techniques on simple algorithms. Also, we introduce a synchronous version of some important asynchronous algorithms.

Assume a general directed graph for the network. Processes know who their neighbors are and can tell whether an incoming and outgoing edge go to the same neighbor. (This could also be done with an initial round).

Leader Election

Assume a *strongly connected* digraph and UID's with comparisons only. Both leaders and non-leaders will output status. Processes know an upper bound on the diameter of the network graph (d).

Requirements for leader election algorithm:

- Elect the process with the max UID by flooding the max UID throughout the network.
- Each process maintains the max UID seen so far and sends over all outgoing edges on each round.
- After d rounds, compare max-uid to own uid. If equal, declare self leader; otherwise, declare self non-leader.

Correctness Proof

Suppose that
u is a process's UID;
max-uid is the maximum known uid;
status $\in$ {unknown, leader, non-leader};
rounds is the number of rounds

Assertion: After d rounds,

$status_{i_{max}}$ = leader
$status_j$ = non-leader for $j \neq i_{max}$

Invariant assertion: For 0 <= r <= d and for every j, if distance(i_{mzx}, j) <= r then max-uid$_j$ = u_{max}.
That is, u_{max} propagates to all nodes within distance r by the end of r rounds.

Complexity

Time: d
Messages: O(d |E|)

Optimization

Don't send the same value more than once – send each value in the round after you first hear it. Use a flag to say whether the value is new.

Correctness is proved by a *simulation relationship* between the algorithms. The idea of simulation relationships:

- Run the algorithms side-by-side (call the original the "specification," the optimized algorithm the "implementation").
- The specification is known (or at least assumed) to be correct; the implementation is not known to be correct, but must be proved correct.
- Prove an invariant relating states of the processes (this is called the *simulation relation*).
- Use the simulation relation to prove correctness of the implementation.

For this example, the invariant says that the state variables u, max-uid, status, rounds are the same in both algorithms after r rounds.

Prove by induction on the length of the execution; in this case, the key thing is to show that the max-uid's stay the same.

Induction base: Algorithms have the same initialization.

Induction hypothesis: For r < r' rounds, the states are the same.

Induction step: After round r', we must show that max-uid$_j$ is the same in both executions, for all j.

By hypothesis, max-uid$_j$ is the same before r', and so is max-uid$_i$ for each in-nbr i of j.

Helper invariant: For any round r and any i,j where j is an out-nbr of i, then if max-uid$_j$ < max-uid$_i$ after r rounds, then new$_i$ = true.
Intuition: When max-uid$_i$ changes (gets bigger), i will tell j in the next round – therefore if max-uid$_j$ < max-uid$_i$ then max-uid$_i$ just now changed.

Consider any particular in-nbr i of j.
Case 1. new$_i$ = true: then i sends a message in both executions

Case 2. new_i = false: i sends a message in FloodMax but not OptFloodMax (and, by the helper invariant, $max\text{-}uid_j >= max\text{-}uid_i$.

If the set of messages sent by neighbors of j in FloodMax contains a UID larger than $max\text{-}uid_j$, then that UID is a member of the set of UID's sent by neighbors of j in OptFloodMax. Hence the new setting of $max\text{-}uid_j$ is the same in both algorithms. [Picture with neighbors, some with larger ID, some with smaller.]

Broadcast (BFS)

This translates familiar sequential BFS to a distributed version.

Algorithm idea

Initially, a start node is marked.
At each round, all newly marked nodes send search messages to all outgoing neighbors.
If an unmarked node receives a search message, it marks itself and sends a search message to outgoing neighbors (out-nbrs).
The node from which the first message arrives is the parent of a process.

```
State:
newMark:Boolean := true for start node, false elsewhere
parent:Int := -1

msgs:
if newMark then send search to all out-nbrs

transitions:
for each in-nbr do
        if incoming message is non-null /\ parent ~= -1 then
          newMark := true
          parent := in-nbr from whom message was received
        else
          newMark := false
        fi
od
```

Invariant assertion: After d rounds, every node within d of that start node has a parent node.
[Prove as exercise?]

Spanning tree example: powerpoint
Note that each node at distance d appears at depth d in the BFS tree.

Complexity:
Time: diam

Messages: |E|

Applications

Broadcast: Piggyback the actual message on the search messages
Child pointers: When receiving a search message, respond with either a parent or non-parent message.
This can be used for return messages.
If communication is not always bi-directional, may need to run another instance of SynchBFS to reply.
Broadcast/convergecast: When leaves receive a message, send to parents – eventually, start node gets all replies.
Leader election: Use broadcast/convergecast to determine max (or min) UID in network.
Computing the diameter: All processes do BFS, determine max-dist$_i$ from i to any other process by piggybacking distance information. Then broadcast/convergecast to get the largest distance in the graph.

Spanning Tree

Note that BFS computes a spanning tree (the parent pointers identify the edges). How many edges are there in a spanning tree? The Cisco STP uses this.

Cisco Version

All processes send a BPDU (Basic Protocol Data Unit) at each round (actually, default is every 2 seconds – but we will describe this as a synchronous algorithm, running in rounds). The BPDU contains the id (MAC address) of the sending process, the id of the process it thinks is the root, and the distance from the sending process to its presumed root.

myID:macaddr
myRoot:macaddr
myCost:Int

When a process receives a BPDU, it compares the ID of the root (designated by the neighbor that sent the BPDU) to the local value for the ID of the root. If the new root ID is lower, it replaces its local root ID with the new one and adds one to the distance in the incoming BPDU and makes that its distance to the root. If it receives different BPDU's having different roots, it uses the "best," i.e., the one with the lowest root ID and the lowest distance (if root Ids are the same).
Finally it designates the port to which the sending neighbor is connected as the root port. (Effectively choosing its parent)

Note that this is essentially BFS

Shortest Paths

Assume now that we have directed weighted links. Each node knows the weights of all incident edges and the number of nodes in the graph.
Motivation is min-cost communication

Problem: Determine the shortest path from i_0 to each other node.

This algorithm is the basis of Bellman-Ford, aka RIP (Route Information Protocol). In the presence of failures this is an unstable routing protocol and isn't much used.

Each node keeps track of the shortest known distance from i_0 and updates it as new information arrives; also communicates changed distances.

states_i
dist:Int := infinity for i_0 ~= i, 0 for i_0
parent:Int := undefined

messages_i
send dist_i to all out-nbrs

transitions_i
if $\text{weight}_{ji} + \text{dist}_j < \text{dist}_i$ for some in-nbr j of I, then update dist_i to $\text{weight}_{ji} + \text{dist}_i$ and update parent to j

Terminates after n-1 rounds

Invariant Assertion:
At round r, 0 <= r <= n-1, very process i has its dist and parent components set to the shortest path values among those paths consisting of at most r edges.

Complexity:
Time = n-1
Messages: (n-1)*|E|

Minimum Weight Spanning Tree