CSc72010

Leader Election

Reading

Skim 3.5
Read 3.6,
General synchronous networks
Read 4.1

Leader election algorithms

Lelann/Chang/Roberts:

- O(n) time complexity
- $O(n^2)$ message complexity

Hirschberg/Sinclair
Two reasons to look at it:

- Use of successive doubling to improve message complexity
- The best we can do for comparison-based algorithms

Hirchberg/Sinclair Details

Assumptions

- Bidirectional
- Don't know ring size
- Comparisons only for UID's

Informally: Send the UID both directions to successively greater distances (double the distance on each pass.

- Outbound: Pass on the UID if it's larger than your own, swallow it otherwise.
- Inbound: Pass everything on.
- If you get your own UID inbound, proceed to the next round

phase tracks the number of times the UID has been sent out; it is incremented when the UID returns to the initiating process

The messages are tuples over UID×{in,out}×Int, where UID is the id at the process originating the message; in or out is the direction of the message with respect to the originating process; and Int is the number of (remaining) processes that should see the message (including the originator).

states

- u:UID, initially i's UID
- send+:M ∪ { null }, initially <u, out, 1>
- send-: M ∪ { null }, initially <u, out, 1>

status: enumeration of unknown, leader, initially unknown
phase: Int, initially 0

msgs

send the current value of send+ to process i+1
send the current value of send- to process i-1

trans

```
send+ := null       % clean up state
send- := null
if incoming from i-1 is (v, out, h) then     % outbound token going clockwise
        case
        v > u and h > 1: send+ := (v, out, h-1)  % continue if bigger
        v > u and h = 1: send- := (v, in, 1)          % turn around if h=1
        v = u: status := leader
        endcase
if incoming from i+1 is (v, out, h) then   %outbound token counter-clockwise
        case
        v > u and h > 1: send- := (v, out, h-1)  % continue if bigger
        v > u and h = 1: send+ := (v, in, 1)       % turn around if h=1
        v = u: status := leader
        endcase
if incoming from i-1 is (v, in, 1) and v>u then
        send+ := (v, in, 1)
if incoming from i+1 is (v, in, 1) and v>u then
        send- := (v, in, 1)
if incoming from i-1 and i+1 are both (u, in, 1) then   % start new phase
        phase := phase+1
        send+ := (u, out, 2^phase)
        send- := (u, out, 2^phase)
```

The above is essentially the same as the book

Correctness

Invariant 1, to show that i_{max} is elected leader:
For every phase k, 0 <=k<=1+floor(log n), and

every round r with 0<=r<2^k, $send+_{i_{max}+r} = (u_{max}, out, 2^k-r)$ and

every round r with 2^k<=r<=2^(k+1), $send\text{-}_{i_{max}+2^{k+1}-r} = (u_{max}, in, 1)$

On the other side, we can change send+ to send- and send- to send+ and reverse +'s and –'s in subscripts

Can verify that u_{max} gets back to the originator and therefore survives each round.

Consider phase 1:
Phase 1, round 0:
$send+_{i_{max}} = (u_{max}, out, 2)$
$send\text{-}_{i_{max}} = (u_{max}, out, 2)$
Phase 1, round 1:

send+ $_{i_{max}+1}$ = (u_{max}, out, 1)
send- $_{i_{max}-1}$ = (u_{max}, out, 1)
Phase 1, round 2 (=2^1) when process i_{max}+2 receives it, it turns it around
send- $_{i_{max}+2}$ = (u_{max}, in, 1)
send+ $_{i_{max}-4+2}$ = (u_{max}, in, 1)

Proof is by induction, working case by case through the transition. We can use all this to verify that u_{max} gets back to the originator at the end of each phase, and therefore survives to the end.
+
Invariant 2, to show that no other process is elected:
For every phase k, 0 <= k <= floor(log n), and every round r, 0 <=r <2^{k+1}, if j∈ [i_{max}, i) then send+$_j$ ≠ i and if j∈ (i, i_{max}] send-$_j$ ≠ i.

For example, j = i_{max}-1 and i = i_{max}+1: i's UID in send- can't get past i_{max}

This is similar to the invariant for the original leader election; proof is also similar.

Complexity

Number of phases is log n (i.e., phases continue until doubling gets us to the size of the ring).
Number of messages in a phase is O(n).
Total message cost is O(n log n)

Sketch of proof:
Phase 0: All send both messages
<= 4n messages if all go out and return (this can't happen of course)

Phase k>0: Look at any block of 2^k+1 consecutive processes. At most one is still alive at the beginning of phase k.
$i_{max-2^{k-1}}$→local i_{max}←$i_{max+2^{k-1}}$

In round k-1, none of the u_i's get back except for $u_{i_{max}}$.
Phase 1: 3
Phase 2: 5
Phase 3: 9
Etc.

At most floor(n/(2^k+1)) still send tokens in phase k. The number of messages is therefore <= 4*2^k*floor(n/(2^k+1)) <= 4n
4 for the messages from each surviving UID – 2 out, in each direction, and 2 back
2^k for the distance the messages travel

Phases <= 1+ceiling(log n)

O(n log n)

Minimizing communication complexity

O(n log n) is the best we can do for communication complexity – this is the minimum complexity for breaking symmetry with UID comparisons only.

Theorem. The communication complexity of leader election is $\Omega(n \log n)$ if we have UID comparisons only, even with network size known and bi-directional links.

Notes:

We can do it in n rounds with n messages if we know n and have a minimum in the data type and can count (Each process counts rounds and if its UID is k, it sends its UID at round k; the first UID to circulate is the leader).
Time complexity is bad; it is $u_{min}*n$

Even if n is unknown we can get O(n) messages, but the price in time complexity is terrible: processes forward UID's that are bigger than their own, but for the kth UID they wait 2^k rounds before forwarding it. The leader is the first UID that circulates.

Back to the theorem with UID comparisons only
Assumptions:
All processes start in the same state, except for UID
They can manipulate UID's only by copying, sending, receiving, and comparing
They can store, resend, use results of comparisons, etc.

Key ideas in proof:

1) If two neighborhoods "look alike," the middle processes of each neighborhood will make the same decision. The consequence is that information about the differences between the neighborhoods must have time to get to the middle processes from outside.
1) There are rings with lots of big similar neighborhoods.

Order equivalence:
$(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_k)$ are order-equivalent provided that $u_i <= u_j$ if and only if $v_i <= v_j$.
Example: (1324) is order-equivalent to (7 12 10 25)

Active round:
A round is active in the execution if at least one message is sent.

k-neighborhood: A process and the k processes to either side of it (there are 2k+1 processes in a k-neighborhood).

Corresponding states: Let $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_k)$ be sequences of UID's. s and t are corresponding states of process j with respect to $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_k)$ if they are identical except that u_i appears in s exactly where v_i appears in t.

Lemma 3.5 (page 40): A is a comparison-based algorithm. Assume i and j have order-equivalent k-neighborhoods. At any point after at most k active rounds, i and j are in corresponding states with respect to the sequences of UID's in their respective neighborhoods.

Interpretation:
Because processes with order-equivalent k-neighborhoods are indistinguishable until enough active rounds have happened – i.e., until information outside the order-equivalent neighborhoods has to reach the processes before they can be told apart.

Example:
(1 3 6 5 2 7 9) versus (2 7 9 8 4 10 11) – these are order-equivalent – it takes more than 3 rounds for 5 and 8 to be distinguishable from the executions.

Proof of Lemma:
Assume without loss of generality that $i \neq j$. We use induction on the number of rounds r in the execution.

Base: All start in the same state (up to UID's), therefore all are in corresponding states.

Hypothesis: Assume that the lemma holds for all k and for all $r < r'$ rounds.

Step: Pick k, i, and j, and suppose processes i and j have order-equivalent k-neighborhoods. Also, assume that in the first r' rounds there have been no more than k active rounds (otherwise, the lemma is trivially true).

Cases:
1) Neither i nor j receives a message at round r'
1) At least one of i and j receives a message at round r'

In case 1) the transition is from the current state using the "null" message, which is the same for both i and j, because they can only copy, send, receive, or compare UID's, not process them.

In case 2), assume that i receives messages
a) From both sides
a) From one side (by symmetry, we only need to prove it for one side)

i and j were in corresponding states after r'-1 rounds by induction hypothesis. The active rounds up to round r'-1 must have been <=k-1 since <=k at round r' and a message arrives in round r'.

Since processes i-1 and j-1 have order-equivalent k-1 neighborhoods, we know that they must have been in corresponding states after r'-1 rounds (using the induction hypothesis) and therefore they send the equivalent messages (with respect to UID's) in round r'. Similarly for i+1 and j+1. So j-1 and j+1 send corresponding messages to j in round r'. To summarize, i and j are in corresponding states at the beginning of round r' and receive corresponding messages during round r'. Thus they must be in corresponding states at the end of round r'.

Similar argument for one-sided message.

The second fact is that there are many rings with large order-equivalent neighborhoods. An important example is the bit-symmetric ring, aka the dyslexic counting ring.

These are rings whose size is a power of 2. The UID's are assigned by numbering the processes clockwise, then reversing the bit representation of the process number.

Property:
Every length n/2 sequence has 2 order-equivalent sequences
Every length n/4 sequence has 4 order-equivalent sequences
Etc.

Leader Election in General Synchronous Networks

Flooding: requires knowledge of diam.
Cisco algorithm:
Message is BPDU – contains (UID, root UID, distance to root)
Send BPDU every "round"
If you receive a better root UID than your own, change your root UID and distance to root
Continue for k rounds (configurable k)

Correctness depends on telling the switches the diameter of the network, in the form of k

It's actually a little more complicated because the network is asynchronous and messages can be lost.

Assignment

1. Look at connectivity and BPDU's in switches in lab. Determine which switch is the root. Turn in a diagram with the switches & connections and designate the root. The goal of this exercise is to get used to looking at the switches so you will be able to observe the spanning tree protocol, which I will introduce shortly.
2. Problem 3.18