Leader Election on Synchronous Rings

Janna Burman

janna.burman@lisn.upsaclay.fr

Assumptions

- **Topology** (communication graph) is in the form of a **ring**, bidirectional links by default (if unidirectional, it is then necessary an oriented ring)
- By default, **each node has a unique identifier** (id) of the set UID $\subseteq \mathbb{N}^+$, UID >> n
- **The ring is oriented**:
  - all n nodes are numbered: $p_0, p_1, \ldots, p_{n-1}$
  - they appear in the corresponding order on the ring
  - each node (process) p_i knows which outgoing link goes to process $p_{i+1 \bmod n}$ and which to $p_{i-1 \bmod n}$

Leader Election (LE) problem specification

Let us assume that each node has a variable named *status* s.t. status ∈ {leader, unknown, non_leader}

1. At termination (or from some configuration), some process has status = leader (forever)
2. During any execution, at most one process has status = leader

The variants:

At termination (or from some configuration), for any other process:

1. it learns to never become a leader (e.g., by status := non_leader)
2. it learns the id of the leader
3. it determines when the leader is alraedy elected

Algorithm (1)
by LeLann, Chang and Roberts (LCR)
for a process p_i

M (set of possible messages) = UID

$states_i$ + **$starts_i$** (possible states + starts):

my_id: UID; my_id := the id of i

msg: M∪{null}; msg := my_id

status: {leader, unknown}; status := unknown

Algorithm (2) by LeLann, Chang and Roberts (LCR)

for a process p_i

msgs$_i$ (message generation function):

generate (output/send) a message containing the value in msg for process p_{i+1}

– $msgs_i$(<state of $p_{i>}$, p_{i+1}) := msg

Algorithm (3)
by LeLann, Chang and Roberts (LCR)
for a process p_i

trans$_i$ (transition function):

msg := null
If (the received message m is in UID) then
  Case
    1. m > my_id: msg := m (to be sent in the next round)
    2. m = my_id: status := leader
  EndCase
EndIf

Proof (for basic LCR) - 1

Denote:

- **i_max** is the index of the process with maximum id
- **u_max** is its id (u_i is an id of process p_i)

Lemma 1: For all $0 \leq r \leq n\text{-}1$, after round r, $msg_{(i\_max + r) \bmod n} = u\_max$

Simple proof by induction.

Lemma 2: After n rounds, $status_{i\_max} = leader$

Proof: Using Lemma 1, show that u_max returns to process i_max in round n and it executes $status_{i\_max} := leader$

Proof (for basic LCR) - 2

Define:

- For any process i and j s.t. $i \neq j$, **[i,j)** is a set of indices {i, i+1, ..., j-1}, where the addition is modulo n
- i.e. **[i_max, i)** is a set of consecutive processes in the ring (following the ring orientation) from i_max to i-1 inclusive

Lemma 3: If $i \neq$ i_max and $j \in$[i_max, i), then always $msg_j \neq u_i$ (u_i is the id of i, and msg_j is a message generated by msgs() of j – see the pseudo-code)

Simple proof by the fact that an id smaller than u_max (any u_j) is never generated by the msgs() function of p_{i_max} process (so no p_j can receive u_j).

Lemma 4: No process j other than i_max is executing $status_j$ = leader

Proof: Using Lemma 3, show that the id of process $j \neq$ i_max never returns to j

Theorem: LCR solves the leader election problem (without variants).

Complexity Analysis (Basic LCR)
worst case default

- **In rounds** (until termination): n
- **In messages** (non-null, worst case): $\Theta(n^2)$
  - processes are oriented according to the decreasing order of their ids (e.g. from n to 1): $n + (n-1) + ... + 2 + 1 = n(n+1)/2$ messages
  - Best case: 1 to n oriented processes: $2n-1$ messages
- **In bits**: $O(n^2 \log n)$, assuming that any id in UID is of size $O(\log n)$ bits

LCR variants (1)
for a process p_i

M (set of possible messages) = UID ∪ { "terminated"}

states$_i$ + **starts$_i$** (possible states + starts):

my_id: UID; my_id: = id of i

msg: M∪{null}; msg: = my_id

status: {leader, unknown, non_leader}; status := unknown

leader_id: UID; leader_id: = my_id

terminated: boolean; terminated:= *false*

LCR variants (2)
for a process p_i

msgs$_i$ (message generation function):

If (status = unknown) then

generate (output/send) a message containing the value in msg for process p_{i+1}

- $msgs_i(<\text{state of } p_{i>}, p_{i+1}): = msg$

Else If (¬ terminated) then

$msgs_i(<\text{state of } p_i>, p_{i+1}) :=$ “terminated”

terminated: = *true*

Else

$msgs_i(<\text{state of } p_{i>}, p_{i+1}): = null$

EndIf

LCR variants (3)
for a process p_i

trans$_i$ (transition function):

```
msg := null
If (the received message m is in UID) then
    Case
      1. m > my_id:      msg: = m;
                         leader_id: = max (leader_id, m)
      2. m = my_id:      status: = leader
    EndCase
Else If (the received message m is "terminated" and status ≠ leader) then
      status: = non_leader
EndIf
```

Hirschberg and Sinclair algorithm (HS)

Each process p_i operates in ***phases*** 0, 1, 2, ...

- At each phase k, the process p_i sends two tokens containing its identifier id_i in two directions.
  - They are assumed to travel to distance 2^k and then return to their origin p_i.
  - If both tokens are returned to it, the p_i process moves on to the next k+1 phase, i.e., ***remains active***. However, the tokens may not be returned. In this case, the process is said to be ***eliminated***.
- As long as a token is in its exploration stage, each process p_j on its path compares id_i to its identifier id_j.
  - If $id_i < id_j$ the process eliminates the token and
  - If $id_i > id_j$ it relays it.
  - If $id_i = id_j$, the process declares itself leader.

All processes relay a returning token (back to the origin p_i).

HS: pseudo-code for p_i (1)

M (set of possible messages) =

$\text{UID} \times \{\text{in, out}\} \times \mathbb{N}^+$

states$_i$ + **starts$_i$** (possible states + starts):

- **my_id**: UID; my_id: = the id of i
- **msg+**: M∪{null}; msg+: = (my_id, out, 1)
- **msg-** : M∪{null}; msg- := (my_id, out, 1)
- **status**: {leader, unknown} ; status := unknown
- **phase**: $\mathbb{N}^0$; phase := 0

HS: pseudo-code for p_i (2)

msgs$_i$ (message generation function):

- generate (output/send) a message containing the value in **msg+ for process p_{i+1}**
  - $\text{msgs}_i(<\text{state of } p_i>, p_{i+1}) := \text{msg+}$
- and the value in **msg- for process p_{i-1}**
  - $\text{msgs}_i(<\text{state of } p_i>, p_{i-1}) := \text{msg-}$

HS: pseudo-code for p_i (3)

trans$_i$ (transition function):

```
msg+: = msg-: = null
If (a message received from p_{i-1} is (id, out, h)) then
    Case
      1. id > my_id and h > 1 : msg+ := (id, out, h-1)
      2. id > my_id and h = 1 : msg- := (id, in, 1)
      3. id = my_id :           status: = leader
    EndCase
If (a message received from p_{i+1} is (id, out, h)) then
    Case
      1. id > my_id and h > 1 : msg- := (id, out, h-1)
      2. id > my_id and h = 1 : msg+ := (id, in, 1)
      3. id = my_id :           status: = leader
    EndCase
If (a message received from p_{i-1} is (id, in, 1) and id > my_id) then
     msg+ := (id, in, 1)
If (a message received from p_{i+1} is (id, in, 1) and id > my_id) then
     msg- := (id, in, 1)
If (two messages received from p_{i-1} and from p_{i+1}, each is (my_id, in, 1)) then
     phase := phase + 1
     msg- := (my_id, out, 2^phase)
     msg+ := (my_id, out, 2^phase)
```

Message complexity of HS (1)

In each **phase k** :

1. **for each active process** i, there are at most $\mathbf{4 \cdot 2^k}$ **tokens** with the id of i that are transmitted on the ring
   - For k = 0, there are n active processes
   - For $k \geq 1$, there are at most $\lfloor n/(2^{k-1}+1) \rfloor$ active processes:
     - Since, the minimum distance between 2 active processes is $2^{k-1} + 1$. Otherwise, at least one of these two processes would not have passed to phase k (it would have been eliminated at phase k-1)
2. then, at most $4 \cdot 2^k \cdot \lfloor n/(2^{k-1} + 1) \rfloor \leq 8n$ messages sent in phase $k \geq 1$; and 4n for k = 0

Message complexity of HS (2)

- The maximum number of phases until the election is $\lceil \log n \rceil + 1$ (including phase 0)

➔ At most $8n\lceil \log n \rceil + 4n$ messages in total

➔ **O(n log n)**

Time complexity of HS

- The last phase $\lceil \log n \rceil$ takes n rounds
- Any other phase k lasts at most 2·2k rounds

➔Complexity in rounds:
$2(2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{\lceil \log n \rceil - 1}) + n =$
$2(2^{\lceil \log n \rceil} - 1) + n$

➔ n is a power of 2: at most 3n – 2 rounds

➔ n is not a power of 2: at most 5n – 2 rounds

➔ **O(n) rounds**

Time Slice (TS) algorithm

- **The processes know the size n of the given oriented ring** and count the rounds.
- **The elected process is the one with the smallest identifier**.
- An execution consists of a sequence of phases, each phase consisting of n rounds:
  - Phase 1 contains rounds 1,2,3,..., n
  - Phase 2, rounds n+1, n+2, ..., 2n
  - etc.
  - Phase v, contains the rounds (v-1)n+1, (v-1)n+2, ..., v·n
- Each phase is associated with the possible circulation of a token carrying a unique id (of an existing process), all around the ring. Only id v is allowed to circulate in phase v.
- If the process with id v reaches phase v (round (v-1)-n+1) without having received a non-null message, it assigns itself the leader and informs the other processes (sends the token with its id).

TS: pseudo code for p_i (1)

M (set of possible messages) = UID

states$_i$ + **starts$_i$** (possible states + starts):

my_id: UID; my_id: = the id of i

round: $\mathbb{N}^+$; round := 1

leader_id: UID ; leader_id := my_id

msg: M∪{null};
    If (my_id = 1) then msg := my_id
    Else msg := null

status: {leader, non_leader, unknown} ;
    If (my_id = 1) then status := leader
    Else status := unknown

TS: pseudo code for p_i (2)

msgs$_i$ (message generation function):

generate (output/send) a message containing the value in msg for process p_{i+1}

– $msgs_i(\langle state\ of\ p_i \rangle, p_{i+1})$: = msg

TS: pseudo code for p_i (3)

trans$_i$ (transition function):

```
msg := null
If (status = unknown) then
    If (the received message m is non-null, in UID) then
        status := non_leader
        leader_id := msg := m
    Else If (round = (my_id - 1)·n) then
        status := leader
        leader_id := msg := my_id
    EndIf
EndIf
round: = round + 1
```

Time-Slice complexities

- **In time:** u_min·n rounds, where u_min is the minimum id in the ring
- **In messages**: only n messages!

Impossibility of LE without identifiers

Theorem: In a ring of size n > 1, **if all processes are identical,** then **there is no *deterministic* leader election algorithm** even if the ring is bidirectional, oriented or not (synchronous or not) and even if the size n is known to each process.

Proof: Assume by contradiction that such an algorithm exists

- Let us consider an execution that starts in an initial configuration C_0, in which every process is in the same state s_0
- Let us assume a perfectly synchronous execution such that **for each round r**:
  - each process sends the same messages to the same neighbours (i.e. any p_i sends message m_{r+} to p_{i+1} and m_{r-} to p_{i-1})
  - then, any process changes its state from s_r to s_{r+1}
- Prove by induction on each round r that each process is in the same state s_r as any other process, in each r.
- Let us assume that in a round r^*, a leader is elected. But since every process is in the same state, then every process is elected.

→ contradicts the specification of the LE problem