Leader Election in rings

All nodes are equal, but some are more equal than others

Marco Aiello, Eirini Kaldeli

University of Groningen

Distributed Systems, 2011

Outline

Looking for a unique leader

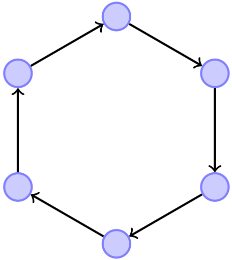
- Want to designate a unique processor as a leader, i.e. the coordinator of a task.
- The network nodes communicate in order to make a decision according to some common criterion that breaks the symmetry among them.
- Helpful in achieving fault-tolerance and saving resources. E.g. generate a new single token when a loss is detected in a token ring, or break a deadlock by removing a node from the cycle.
- There are plenty of algorithms appropriate for different network graphs, such as bi-/unidirectional rings, complete graphs, grids etc.
- E.g. given a spanning tree, leader election can be achieved by applying convergecast on it.

Leader Election algorithm

- Any process can initiate the LE algorithm (several elections can be called concurrently).
- Every p_i has two boolean variables *done* and *is_leader*. *done* is set when p_i knows that the algorithm has finished, while *is_leader* is set when p_i knows that it is the leader.
- An LE algorithm has to satisfy the following two properties:
  - *Safety*: At most one p_i is a leader:
    $\forall i, j \; i \neq j : \; \neg(is\_leader(i) \wedge is\_leader(j))$
  - *Liveness*: Eventually all p_is are either leaders or not and at least one p_i is a leader: $\forall i \; done(i) \wedge \exists j \; is\_leader(j)$

- We will consider a network of n processors circularly placed on a ring.
- Unidirectional (clockwise): each p_i sends messages to p_{i+1} and receives messages from p_{i-1} (we assume *modulo* n arithmetics).
- Bidirectional: each p_i can send and receive messages in both directions.
- Lower bounds and impossibility results for rings also apply to arbitrary topologies.

A clockwise ring



Leader election in anonymous rings

A ring is anonymous if the p_is are indistinguishable; they have no unique identifiers, and they all have identical state machines, with the same initial state.

Theorem

There is no deterministic leader election algorithm (even) for synchronous anonymous rings (and even for uniform ones).

Proof sketch

- All p_is start at the same initial state with the same outgoing messages.
- In every round each p_i sends the same messages to its neighbour, and thus all p_is receive exactly the same messages.
- Thus, because all p_is have the same state machine, they move to the same state.

Rings with identifiers

- Impossibility of leader election for asynchronous anonymous rings follows.
- Have to introduce some initial asymmetry in the network ➥ processors are assigned identifiers.
- Identifiers have to be unique and totally ordered. Each p_i knows only its own identifier.
- The algorithms that we will present suit for both synchronous and asynchronous rings.
- We will consider the asynchronous case for our analysis: assume reliable FIFO channels.
- The size of the ring n is not a priori known to the nodes: non-uniform rings.
- At the end of the algorithm the p_i with the maximal *id* is elected, while all p_is must know the *id* of the elected leader.

LeLann-Chang-Roberts (LCR) algorithm

- Assume clockwise unidirectional ring.
- One or more p_is can take the initiative and start an election, by sending an election message containing their *id* to p_{i+1}.
- When a p_i spontaneously or upon receiving a message goes in an election, it marks itself as a participant.
- If the p_i receiving an election message has a greater *id* and is not already a participant, then it sends an election message with its own *id* to p_{i+1}.
- If its own *id* is smaller, it forwards the message with the *id* it has received.
- If it receives a message with its own *id* then it declares itself as the leader.

```
boolean participant=false;
int leader_id=null;
```

To initiate an election:

```
 send(ELECTION⟨my_id⟩);
 participant:=true;
```

Upon receiving a message ELECTION$\langle j \rangle$:

```
 if (j > my_id) then send(ELECTION⟨j⟩);
 if (my_id = j) then send(LEADER⟨my_id⟩);
 if ((my_id > j) ∧ (¬participant)) then
    send(ELECTION⟨my_id⟩);
participant:=true;
```

Upon receiving a message LEADER$\langle j \rangle$:

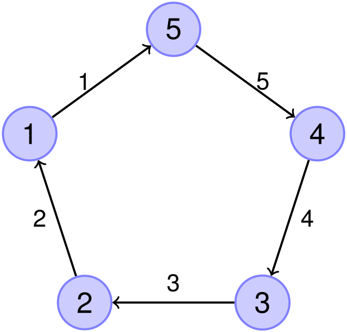
```
 leader_id:=j;
 if (my_id ≠ j) then send(LEADER⟨j⟩);
```

The LCR LE algorithm:remarks

- Only the message with the largest identity completes the round trip and returns to its originator, which becomes the leader.
- Time complexity: $O(n)$
- The leader has to announce itself to all p_is through the leader messages, so that termination is guaranteed and everybody knows who the leader is.
- The algorithm verifies the safety and liveness conditions with:
  - $done(i) \equiv (leader\_id(i) \neq \texttt{null})$
  - $is\_leader(i) \equiv (leader\_id(i) = i)$

An example run of the LCR algorithm

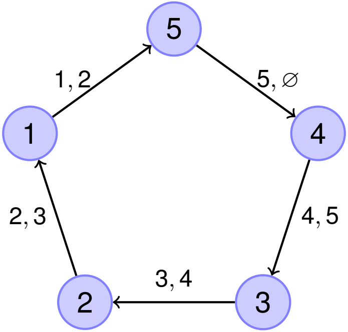
Assume all p_is are initiators.



Messages transmitted:

$\langle 1 \rangle$	1 times
$\langle 2 \rangle$	1 times
$\langle 3 \rangle$	1 times
$\langle 4 \rangle$	1 times
$\langle 5 \rangle$	1 times
total	5 times

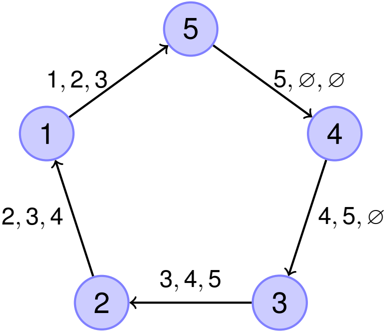
An example run of the LCR algorithm



Messages transmitted:

$\langle 1 \rangle$	1 times
$\langle 2 \rangle$	2 times
$\langle 3 \rangle$	2 times
$\langle 4 \rangle$	2 times
$\langle 5 \rangle$	2 times
total	9 times

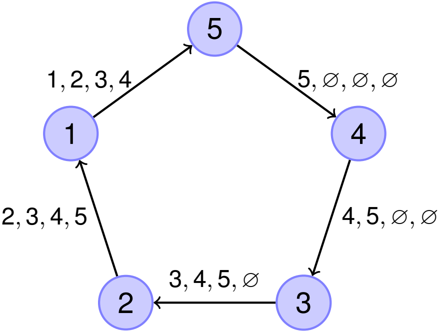
An example run of the LCR algorithm



Messages transmitted:

$\langle 1 \rangle$	1 times
$\langle 2 \rangle$	2 times
$\langle 3 \rangle$	3 times
$\langle 4 \rangle$	3 times
$\langle 5 \rangle$	3 times
total	12 times

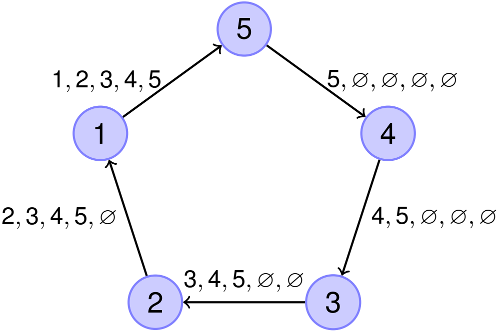
An example run of the LCR algorithm



Messages transmitted:

$\langle 1 \rangle$	1 times
$\langle 2 \rangle$	2 times
$\langle 3 \rangle$	3 times
$\langle 4 \rangle$	4 times
$\langle 5 \rangle$	4 times
total	14 times

An example run of the LCR algorithm



Messages transmitted:

$\langle 1 \rangle$	1 times
$\langle 2 \rangle$	2 times
$\langle 3 \rangle$	3 times
$\langle 4 \rangle$	4 times
$\langle 5 \rangle$	5 times
total	15 times

Now, the leader $id = \langle 5 \rangle$ has to be announced to all nodes with 5 more messages. So, in total 15+5=20 messages are transmitted.

☞ Note that each identifier i is sent i times.

LCR algorithm: message complexity

- We are interested in message complexity: Depends on how the ids are arranged.
  - The largest *id* always travels all around the ring (n msgs).
  - 2nd largest *id* travels until reaching the largest.
  - 3rd largest *id* travels until reaching largest or second largest.
  - ... etc.
- Worst way to arrange the ids is in decreasing order (and all p_is are initiators): 2nd largest causes $n-1$ messages, 3rd largest $n-2$ messages etc.
- Number of msgs = $(n + (n-1) + ... + 1) + n = \frac{n(n+1)}{2} + n$ (including the n leader messages at the end).

➤ Worst case complexity = $O(n^2)$

LCR algorithm: average message complexity

Theorem

The average message complexity of the LCR algorithm is $O(n \log n)$.

Proof.

- Consider all $n!$ rings (all possible permutations).
- Each *id* makes 1 step $\rightarrow$ $n!$ times.
- Each *id* takes a kth step if it is the largest among all its neighbours from p_{i+1} to p_{i+k-1}: $\Pr\{max\_among\_k\} = \frac{1}{k}$.
- Add $n!n$ times for the leader announcement phase.
- So, average number of messages = $\frac{1}{n!}n((n! + \frac{n!}{2} + \ldots + \frac{n!}{n}) + n!) = n(1 + \frac{1}{2} + \ldots + \frac{1}{n}) + n = O(n)O(\log n) = O(n \log n)$. $\square$

Can do better: an $O(n \log n)$ algorithm

- Can we improve message complexity?
- There are several algorithms that solve the problem of leader election in asynchronous rings with $O(n \log n)$ message complexity.
- Try to have messages containing smaller *ids* travel smaller distances across the ring.
- Hirschberg and Sinclair (HS) algorithm: carry out elections on increasingly larger sets of p_is.
- Assume that links allow *bidirectional* communication, again n is not known in advance.

The HS algorithm: elections in neighbourhoods

- Elections are performed in neighbourhoods: the k-neighbourhood of a p_r is the set of processors that are at distance at most k from p_r (k left plus k right neighbours).
- Operate in (asynchronous) phases: p_i tries to become a leader in phase k among its 2^k neighbourhood; only if p_i is the winner, i.e. it has the highest *id* in its $2^k - \mathit{neighbourhood}$, it can proceed to phase $k+1$.
- The size of the neighbourhood doubles in each phase.
- Fewer p_is proceed to higher phases, until a single winner gets elected in the whole ring.

The HS algorithm: sending messages

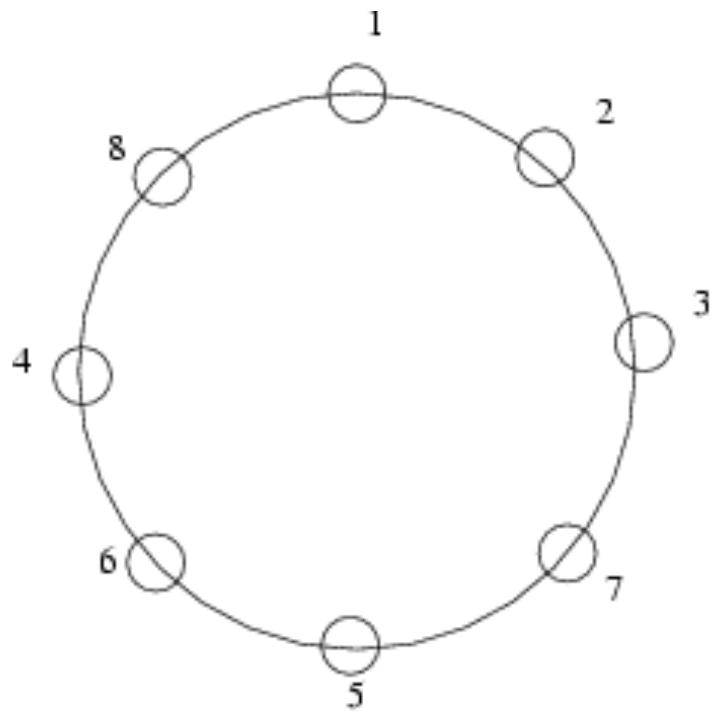
- Initially, all p_is initiate a candidancy (phase 0),e.g. after having received a broadcasted request for electing a leader.
- The ELECTION messages sent by candidates contain three fields:
  - The *id* of the candidate.
  - The current phase number k.
  - A hop counter d, which is initially 0 and is incremented by 1 whenever the message is forwarded to the next p_i.
- If a p_j receives a ELECTION$\langle r, k, d\rangle$ where $d = 2^k$ then it is the last processor in the 2^k-neighbourhood of p_r with $id = r$.

The HS algorithm: sending messages

- If the p_i receiving the election message has a greater *id*, then it swallows the message, otherwise it relays it to the same direction, after incrementing d by 1.
- If the message makes it till the 2^k-distance p_i, then p_i sends back a REPLY message, which is forwarded till it reaches the candidate p_r.
- If the candidate receives replies from both directions, then it is the winner of its 2^k neigbourhood.
- A p_i that receives an election message with its own *id* is the leader of the ring.
- The leader should also announce itself to all other nodes (like in LHR).

An HS example

- Initially:
  - All processes are leaders
- Round 0:
  - 6 , 7 and 8 are leaders
- Round 1:
  - 7, 8 are leaders
- Round 2:
  - 8 is the only leader



(b)

To initiate an election (phase 0):

```
send(ELECTION⟨my_id, 0, 0⟩) to left and right;
```

Upon receiving a message ELECTION$\langle j, k, d \rangle$ from left (right):

if $((j > my\_id) \wedge (d \leq 2^k))$ then
 send(ELECTION$\langle j, k, d+1 \rangle$) to right (left);
if $((j > my\_id) \wedge (d = 2^k))$ then
 send(REPLY$\langle j, k \rangle$) to left (right);
if $(my\_id = j)$ then announce itself as leader;

Upon receiving a message REPLY$\langle j, k \rangle$ from left (right):

if $(my\_id \neq j)$ then
 send(REPLY$\langle j, k \rangle$) to right (left);
else
  if (already received REPLY$\langle j, k \rangle$)
   send(ELECTION$\langle j, k+1, 1 \rangle$) to left and right;

HS algorithm: communication complexity

- At phase k at most $4 \cdot 2^k$ messages are circulated on behalf of a particular candidate (elections and replies).
- How many candidates compete in phase k, in worst case?
- At phase $k = 0$ there are n candidates.

Lemma

For every $k \geq 1$ the number of processors that will continue to phase k is at most $\lfloor \frac{n}{2^{k-1}+1} \rfloor$.

- Proof: the minimun distance between two winners at phase $k-1$ is $2^{k-1}+1$.
- The total number of messages sent at phase k that is not the last phase is $4(2^k \lfloor \frac{n}{2^{k-1}+1} \rfloor) = 8n \lfloor \frac{2^{k-1}}{2^{k-1}+1} \rfloor < 8n$

HS algorithm: communication complexity

- The total number of phases till the leader is elected is $\lceil \log n \rceil + 1$ (including phase 0).
- In last phase $2n$ msgs are sent (no replies).
- So, the total number of messages in worst case is $4n + \sum_{k=1}^{\lceil \log n \rceil - 1} (4 \cdot 2^k \frac{n}{2^{k-1}+1}) + 2n \leq 6n + 8n(\lceil \log n \rceil - 1)$.
- Message complexity: $O(n \log n)$

HS algorithm: time complexity

- The max time for each phase k that is not the final is 22^k.
- The max total time required by phases 0 to k is $2(2^0 + 2^1 + \ldots 2^k) = 2(2^{k+1} - 1)$.
- The max total time required by all phases till the penultimate one is thus $2(2^{\lceil \log n \rceil + 1} - 1)$.
- Time for the final phase is n.
- Time complexity: $O(n)$

Lower bound for LE algorithms

But, can we do better than $O(n \log n)$?

Theorem

Any leader election algorithm for asynchronous rings whose size is not known a priori has $\Omega(n \log n)$ *message complexity (holds also for unidirectional rings).*

- Both LHR and HS are *comparison-based* algorithms, i.e. they use the identifiers only for comparisons $(<, >, =)$.
- In synchronous networks, $O(n)$ message complexity can be achieved if general arithmetic operations are permitted (non-comparison based) and if time complexity is unbounded.