Leader Election in a Synchronous Ring

Etienne Renault

18 décembre 2017

https://www.lrde.epita.fr/~renault/teaching/algorep/
1 Problem Statement

2 LCR Algorithm (comparison-based)

3 HS Algorithm (comparison-based)

4 TimeSliceh Algorithm (non-comparison-based)

5 Lower Bounds
Problem Statement

- The network digraph is a ring with \( n \) nodes
- All processes are identical
- Each process can only communicate with clockwise neighbour and counterclockwise neighbour

One process outputs “I’m the leader” while the other process outputs “I’m not the leader”
Problem Statement

- The network digraph is a ring with $n$ nodes
- All processes are identical
- Each process can only communicate with clockwise neighbour and counterclockwise neighbour

One process outputs “I’m the leader” while the other process outputs “I’m not the leader”
Problem Statement

- The network digraph is a ring with $n$ nodes
- All processes are identical
- Each process can only communicate with clockwise neighbour and counterclockwise neighbour

One process outputs “I’m the leader” while the other process output “I’m not the leader”
Impossibility Result for Identical Processes

Theorem

Let $S$ be a system of $n$ processes, $n > 1$, arranged in a bidirectional ring. If all the processes are identical then $S$ does not solve the leader-election problem.
Sketch of Proof

1. Suppose there is a system $S$ that solves this problem
Sketch of Proof

1. Suppose there is a system $S$ that solves this problem.

2. Without loss of generality, we can assume that each process of $S$ have a unique initial state.
Sketch of Proof

1. Suppose there is a system $S$ that solves this problem.

2. Without loss of generality, we can assume that each process of $S$ have a unique initial state.

3. By induction on the number $r$ of rounds, all the processes are in identical states immediately after $r$ rounds.
Sketch of Proof

1. Suppose there is a system $S$ that solves this problem.

2. Without loss of generality, we can assume that each process of $S$ have a unique initial state.

3. By induction on the number $r$ of rounds, all the processes are in identical states immediately after $r$ rounds.

4. Then if a process reaches a state where it considers to be the leader, all the other processes do so.
Sketch of Proof

1. Suppose there is a system $S$ that solves this problem.

2. Without loss of generality, we can assume that each process of $S$ have a unique initial state.

3. By induction on the number $r$ of rounds, all the processes are in identical states immediately after $r$ rounds.

4. Then if a process reaches a state where it considers to be the leader, all the other processes do so.

5. But this violates the uniqueness requirement.
Problem Statement Revisited

- The network digraph is a ring with $n$ nodes
- All processes are identical except for a UID
- Each process can only communicate with clockwise neighbour and counterclockwise neighbour
Problem Statement Revisited

- The network digraph is a ring with \( n \) nodes
- All processes are identical except for a UID
- Each process can only communicate with clockwise neighbour and counterclockwise neighbour

Two kind of algorithms solving the leader election problem exist:
- **Comparison-based**: UIDs are only used in comparisons
- **Non-Comparison-based**: UIDs may be used for computation
1 Problem Statement

2 LCR Algorithm (comparison-based)

3 HS Algorithm (comparison-based)

4 TimeSlice Algorithm (non-comparison-based)

5 Lower Bounds
LCR Algorithm

- Tribute to LeLann[1977] algorithm
  Optimized later by Chang & Roberts [1979]

- Unidirectional Ring

- The size of the ring is unknown to the processes

- Comparison-based Algorithm

- It elects the process with the maximum UID
LCC Algorithm : Informal

Each process sends its UID around the ring
LCR Algorithm : Informal

1. Each process sends its UID around the ring

2. When a process receives a UID, it compares this one to its own:
LCR Algorithm : Informal

1. Each process sends its UID around the ring

2. When a process receives a UID, it compares this one to its own:
   - If the incoming UID is greater, then it passes this UID to the next process
LCR Algorithm : Informal

1. Each process sends its UID around the ring

2. When a process receives a UID, it compares this one to its own:
   - If the incoming UID is greater, then it passes this UID to the next process
   - If the incoming UID is smaller, then it discards it
LCR Algorithm : Informal

1. Each process sends its UID around the ring

2. When a process receives a UID, it compares this one to its own:
   - If the incoming UID is greater, then it passes this UID to the next process
   - If the incoming UID is smaller, then it discards it
   - If it is equal, then the process declares itself the leader
Example

Initial State

1 - 51 - 15 - 42 - 13 - 21 - 1
Example

Round 1
Example

Round 2
Example

Round 3
Example

Round 4
Example

Round 5
Example

Round 7
Example

Election Successful
Complexity

- **Best Case**: *UIDs are sorted by increasing order*
  - $n$ rounds
  - $O(n)$ messages

- **Worst Case**: *UIDs are sorted by decreasing order*
  - $n$ rounds
  - $O(n^2)$ messages

When a node has been elected, $n$ rounds and $n$ messages are required to ensure the halting of the system.
Problem Statement

LCR Algorithm (comparison-based)

HS Algorithm (comparison-based)

TimeSlice Algorithm (non-comparison-based)

Lower Bounds
The communication complexity of LCR algorithm is high!

We want to minimize the number of messages to avoid network congestion.
HS Algorithm

- Tribute to Hirshberg & Sainclair [1980] algorithm
- Bidirectional Ring
- The size of the ring is unknown to the processes
- Comparison-based Algorithm
- It elects the process with the maximum UID
Each process $i$ operates in phases

1. In each phase $\ell$:
   - Each process $i$ sends out tokens containing its UID $i$ in both directions.
   - Tokens travel a distance $2 \times \ell$ and return to their origin.
   - When a process $i$ receives a token $t$ containing UID $t_{uid}$:
     - If $t_{uid} < UID_i$, then the token is discarded.
     - If $t_{uid} > UID_i$, then the process $i$ relays the token.
     - If $t_{uid} = UID_i$, then the process is the leader.
   - If both tokens come back safely, process $i$ starts a new phase.
   - Otherwise, the process considers itself as a non-leader.
HS Algorithm : Informal

1. Each process \( i \) operates in phases

2. In each phase \( \ell \):

- \( \text{Process} \ i \) sends out tokens containing its UID \( i \)
- Tokens travel distance \( 2 \times \ell \) and return to their origin \( i \)
- When a process \( i \) receives a token \( t \) containing UID \( t_{\text{uid}} \):
  - if \( t_{\text{uid}} < \text{UID} \) then the token is discarded
  - if \( t_{\text{uid}} > \text{UID} \) then the process \( i \) relays the token
  - if \( t_{\text{uid}} = \text{UID} \) then the process is the leader
- If both tokens come back safely, process \( i \) starts a new phase
- Otherwise the process considers itself as a non-leader
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$:
   - Process $i$ send out tokens containing its UID; in both directions

   - If both tokens come back safely, process $i$ starts a new phase
   - Otherwise the process considers itself as a non-leader
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$:
   - Process $i$ send out tokens containing its UID$_i$ in both directions
   - Tokens travel distance $2 \times \ell$ and return to their origin $i$
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$ :
   - Process $i$ send out tokens containing its UID$_i$ in both directions
   - Tokens travel distance $2 \times \ell$ and return to their origin $i$
   - When a process $i$ receive a token $t$ containing UID $t_{uid}$ :
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$ :
   - Process $i$ send out tokens containing its UID in both directions
   - Tokens travel distance $2 \times \ell$ and return to their origin $i$
   - When a process $i$ receive a token $t$ containing UID $t_{uid}$ :
     - if $t_{uid} < UID_i$ then the token is discarded
     - if $t_{uid} > UID_i$ then the process $i$ relays the token
     - if $t_{uid} = UID_i$ then the process is the leader
   - If both tokens come back safely, process $i$ starts a new phase
   - Otherwise the process considers itself as a non-leader
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$:
   - Process $i$ sends out tokens containing its UID$_i$ in both directions
   - Tokens travel distance $2 \times \ell$ and return to their origin $i$
   - When a process $i$ receives a token $t$ containing UID $t_{uid}$:
     - if $t_{uid} < UID_i$ then the token is discarded
     - if $t_{uid} > UID_i$ then the process $i$ relays the token
     - if $t_{uid} = UID_i$ then the process is the leader

If both tokens come back safely, process $i$ starts a new phase. Otherwise, the process considers itself as a non-leader.
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$ :
   - Process $i$ send out tokens containing its UID$_i$ in both directions
   - Tokens travel distance $2 \times \ell$ and return to their origin $i$
   - When a process $i$ receive a token $t$ containing UID $t_{uid}$ :
     - if $t_{uid} < UID_i$ then the token is discarded
     - if $t_{uid} > UID_i$ then the process $i$ relays the token
     - if $t_{uid} = UID_i$ then the process is the leader
HS Algorithm: Informal

1. Each process \( i \) operates in phases.

2. In each phase \( \ell \):
   - Process \( i \) sends out tokens containing its UID \( i \) in both directions.
   - Tokens travel distance \( 2 \times \ell \) and return to their origin \( i \).
   - When a process \( i \) receives a token \( t \) containing UID \( t_{uid} \):
     - If \( t_{uid} < UID_i \), then the token is discarded.
     - If \( t_{uid} > UID_i \), then the process \( i \) relays the token.
     - If \( t_{uid} = UID_i \), then the process is the leader.
   - If both tokens come back safely, process \( i \) starts a new phase.
HS Algorithm : Informal

1. Each process $i$ operates in phases

2. In each phase $\ell$:
   - Process $i$ send out tokens containings its UID $i$ in both directions
   - Tokens travel distance $2 \times \ell$ and return to their origin $i$
   - When a process $i$ receive a token $t$ containing UID $t_{uid}$:
     - if $t_{uid} < UID_i$ then the token is discarded
     - if $t_{uid} > UID_i$ then the process $i$ relays the token
     - if $t_{uid} = UID_i$ then the process is the leader
   - If both tokens come back safely, process $i$ starts a new phase
   - Otherwise the process considers itself as a non-leader
Example

Initial State
Round 1, phase 1, distance = 1

Example
Example

Round 1, distance = 1, Discarded messages
Example

Round 2, phase 1, distance = 1

Round 2, phase 2, distance = 2

Messages then go back to 51, and a last phase is started so that 51 can detect it is the leader.
Example

Round 2, phase 2, distance = 2
Example

Round 2, phase 2, distance = 2
Messages then go back to 51, and a last phase is started so that 51 can detect it is the leader.
Communication Complexity 1/2

1. **Phase 0**: every process sends a message in both directions
   
   $4 \times n$ messages

2. **Phase $\ell$**: for $\ell > 0$ a process sends a token if it receives exactly two tokens in phase $\ell - 1$, i.e., it has not been defeated in phase $2^{\ell-1}$. This implies that within any group of $2^{\ell-1} + 1$ consecutive processes at most one will initiate tokens in phase $\ell$. There is $\left\lfloor \frac{n}{2^{\ell-1}+1} \right\rfloor$ process that initiates tokens at phase $\ell$.

   At phase $\ell$ the number of messages is $4(2^\ell(\left\lfloor \frac{n}{2^{\ell-1}+1} \right\rfloor)) \leq 8n$
How many phase are executed before a leader is elected?
Communication Complexity 2/2

How many phase are executed before a leader is elected?

\[1 + \lceil \log n \rceil\]
How many phases are executed before a leader is elected?

$$1 + \lceil \log n \rceil$$

The number of messages is at most $8n(1 + \lfloor \log n \rfloor)$. 
Time Complexity

- The time complexity for phase $\ell$ is $2^{\ell+1}$
- The complexity of all but the final phase is $2 \times 2^{\log n}$

- In the final phase takes $n$ since tokens only travel outbound
- The final complexity is at most $3n$ (if $n$ is power of 2) $5n$ otherwise.
1 Problem Statement

2 LCR Algorithm (comparison-based)

3 HS Algorithm (comparison-based)

4 TimeSlice Algorithm (non-comparison-based)

5 Lower Bounds
TimeSlice Algorithm

- Unidirectionnal Ring
- UIDs are positive integer
- Deeper use of synchrony (especially non-arrival of a message) than HS or LCR
- Non-comparison-based Algorithm
- \( n \) is known in advance
- It elects the process with the minimum UID
TimeSlice Algorithm : Informal

1. Computation proceeds in phases where each phase consists in $n$ consecutive rounds.
TimeSlice Algorithm : Informal

1. Computation proceeds in phases where each phase consists in \( n \) consecutive rounds.
2. Each phase is devoted to the possible circulation of a token carrying a particular token
Computation proceeds in phases where each phase consists in $n$ consecutive rounds.

Each phase is devoted to the possible circulation of a token carrying a particular token

- In phase $\ell$, only a token carrying $\ell$ circulates

If $(\ell - 1)n + 1$ is reached without having received non-null message and a process with UID $\ell$ exist then it sends its token on the ring.

When a process receive a non-null token

- If token UID is equal to process' UID then the process declares itself as the leader
- Otherwise, it declares itself as a non-leader and relay the token
Computation proceeds in phases where each phase consists in \( n \) consecutive rounds.

Each phase is devoted to the possible circulation of a token carrying a particular token

- In phase \( \ell \) only a token carrying \( \ell \) circulates
- Phase \( \ell \) consist of rounds \((\ell - 1)n + 1\) to \( \ell n \)
TimeSlice Algorithm: Informal

1. Computation proceeds in phases where each phase consists in $n$ consecutive rounds.
2. Each phase is devoted to the possible circulation of a token carrying a particular token
   - In phase $\ell$ only a token carrying $\ell$ circulates
   - Phase $\ell$ consist of rounds $(\ell - 1)n + 1$ to $\ell n$
3. If $(\ell - 1)n + 1$ is reached without having received non-null message and a process with UID $\ell$ exist then it sends its token on the ring.
TimeSlice Algorithm : Informal

1. Computation proceeds in phases where each phase consists in $n$ consecutive rounds.

2. Each phase is devoted to the possible circulation of a token carrying a particular token
   - In phase $\ell$ only a token carrying $\ell$ circulates
   - Phase $\ell$ consist of rounds $(\ell - 1)n + 1$ to $\ell n$

3. If $(\ell - 1)n + 1$ is reached without having received non-null message and a process with UID $\ell$ exist then it sends its token on the ring.

4. When a process receive a non-null token
TimeSlice Algorithm : Informal

1. Computation proceeds in phases where each phase consists in $n$ consecutive rounds.

2. Each phase is devoted to the possible circulation of a token carrying a particular token
   - In phase $\ell$ only a token carrying $\ell$ circulates
   - Phase $\ell$ consist of rounds $(\ell - 1)n + 1$ to $\ell n$

3. If $(\ell - 1)n + 1$ is reached without having received non-null message and a process with UID $\ell$ exist then it sends its token on the ring.

4. When a process receive a non-null token
   - if $\text{token}_{\text{uid}}$ is equal to process’UID then the process declares itself as the leader
TimeSlice Algorithm : Informal

1. Computation proceeds in phases where each phase consists in \( n \) consecutive rounds.
2. Each phase is devoted to the possible circulation of a token carrying a particular token
   - In phase \( \ell \) only a token carrying \( \ell \) circulates
   - Phase \( \ell \) consist of rounds \((\ell - 1)n + 1\) to \(\ell n\)
3. If \((\ell - 1)n + 1\) is reached without having received non-null message and a process with UID \(\ell\) exist then it sends its token on the ring.
4. When a process receive a non-null token
   - if \(\text{token}_{uid}\) is equal to process’UID then the process declares itself as the leader
   - Otherwise, it declares itself as a non-leader and relay the token
Example

Round 1, phase 1
Round 2, phase 1
Round 3, phase 1
Round 4, phase 1
Round 5, phase 1
Round 6, phase 1

Etienne Renault
Example

Round 1, phase 1
Round 2, phase 1
Example

Round 3, phase 1
Example

Round 4, phase 1
Example

Round 5, phase 1
Example

Round 6, phase 1
Example
Complexity

- Communication: $O(n)$
- Time: $n \times UID_{min}$

Limitations

- Small ring networks
- UIDs from small positive integers
- Huge running time
1 Problem Statement

2 LCR Algorithm (comparison-based)

3 HS Algorithm (comparison-based)

4 TimeSlice Algorithm (non-comparison-based)

5 Lower Bounds
Lower Bounds

Comparison-based

The best case is $\Omega(n \log n)$ messages.

Non-Comparison-based

$O(n)$ messages can be reached but only at the cost of large time complexity (Ramsey Theorem).