Distributed Algorithms – 2g1513

Lecture 6 – by Ali Ghodsi
Leader Election and Anonymous Networks
Election Algorithms

- Gerard LeLann posed the Election problem in a famous paper 1977

- Many distributed systems are client server based, with one server/coordinator (leader), and multiple clients

- What happens if the leader fails?
  - Elect a new one
Problem:

- When a process crashes, several processes might detect the crash simultaneously.

Cannot assume only one process detects the crash:
- The election should get all of the processes to agree that one node is a leader.
Who to choose?

Basic idea:

- If each process has a unique identity, and the identities are ordered
  - Elect the non-crashed node with minimum/maximum identity
Formally, an algorithm is an election algorithm iff:

- Each process has the same local algorithm
- The algorithm is decentralized
  - It can be initiated by any number of processes in the system
- It reaches a terminal configuration, and in each reachable terminal configuration one process is in state *leader* and the rest are in the state *lost*
Assumptions

- Fully asynchronous system
  - No global clock, transmission times arbitrary

- Every process has a unique identity
  - Identities are totally ordered
  - Because we have finite number of processes:
    - Max identity and Min identity available (extreme values)
Remember Wave Algorithms (WA)?

WA have three properties:

- **Termination**
  - The algorithm terminates in *finite* number of steps
- **Decide**
  - At least one node executes a *decide* event
- **Dependence**
  - *Every* process executes an event which *causally precedes* the decide event
Wave Properties

Causally related chain of events

Time-space diagram

$p1$

$p2$

$p3$

time
Wave Properties

Time-space diagram

Causally related chain of events
Election Waves

- **Idea:**
  - Append processor identity in the wave
    - Whenever a node decides, an event has causally preceded in each processor where the identity has been added
    - The deciding process will know the identity of every process
    - Pick the process with least identity as leader
    - Broadcast the leader’s identity to all processes
Tree Algorithm augmented with Election

- Lets augment the Tree algorithm so that it can perform Election on trees

- Problem:
  - Tree algorithm assumes every leaf is an initiator
  - In leader election, any subset of the processes can be initiator, not necessarily every process
Tree Algorithm augmented with Election

Solution:

- Flood the whole tree with a **wakeup** token initially so that every process participates
- After that, run the tree algorithm
  - Leaf nodes add their identity to the tokens
  - Internal nodes only change the identity if they have a lower id
- After **decide**, 
- Make every node decide (on the leader) by letting nodes that have decided sending a message to their children
\begin{align*}
\text{var } w_{sp} & : \text{ boolean} \\
wr_{p} & : \text{ integer} \\
rec_{p}[q] & : \text{ boolean for each } q \in \text{ Neigh}_{p} \\
v_{p} & : \mathcal{P} \\
state_{p} & : \text{ (sleep, leader, lost)} \\
\text{init } f & \text{alse; } \\
\text{init } 0; \\
\text{init } f & \text{alse; } \\
\text{init } p; \\
\text{init } \text{sleep}; \\
\text{Not received any wakeup} & \\
\text{Wakeup Counter 0} & \\
\text{Not received from any neighbor} & \\
\text{Candidate leader is } p \text{ (me)} & \\
\text{Initial state is sleep} & \\
\text{All initiators send a wakeup to every neighbor!} & \\
\text{Everyone waits for wakeup tokens from all its neighbors} & \\
\text{We have flooded all our neighbors, and received from all our neighbors} & \\
\text{Send minimum value so far to parent} & \\
\text{Maybe deciding node has least id} & \\
\text{Am I the leader or not?} & \\
\text{Send to all children so everyone can decide} & \\
\end{align*}

\begin{align*}
\text{begin if } p \text{ is initiator then} \\
\text{begin } w_{sp} & := \text{true} ; \\
\text{forall } q \in \text{ Neigh}_{p} \text{ do send } \langle \text{wakeup} \rangle \text{ to } q \\
\text{end ;} \\
\text{while } wr_{p} < \# \text{Neigh}_{p} \text{ do} \\
\text{begin receive } \langle \text{wakeup} \rangle ; \text{wr}_{p} := \text{wr}_{p} + 1 ; \\
\text{if not } w_{sp} \text{ then} \\
\text{begin } w_{sp} := \text{true} ; \\
\text{forall } q \in \text{ Neigh}_{p} \text{ do send } \langle \text{wakeup} \rangle \text{ to } q \\
\text{end} \\
\text{end :} \\
\text{(* Now start the tree algorithm *)} \\
\text{while } \#\{ q : !\text{rec}_{p}[q] \} > 1 \text{ do} \\
\text{begin receive } \langle \text{tok}, r \rangle \text{ from } q ; \text{rec}_{p}[q] := \text{true} ; \\
v_{p} := \text{min}(v_{p}, r) \quad v_{p} \text{ is the minimum identity seen so far} \\
\text{end ;} \\
\text{send } \langle \text{tok}, v_{p} \rangle \text{ to } q_{0} \text{ with !}\text{rec}_{p}[q_{0}] ; \\
\text{receive } \langle \text{tok}, r \rangle \text{ from } q_{0} ; \\
v_{p} := \text{min}(v_{p}, r) ; \quad (* \text{decide with answer } v_{p} \text{ *)} \\
\text{if } v_{p} = p \text{ then state}_{p} := \text{leader} \text{ else state}_{p} := \text{lost} ; \\
\text{forall } q \in \text{ Neigh}_{p}, q \neq q_{0} \text{ do send } \langle \text{tok}, v_{p} \rangle \text{ to } q \\
\text{end}
\end{align*}
What is the message complexity?

- The wakeup is flooded to all neighbors by every node
  - Since it is a tree of $N$ nodes, we have $N-1$ edges, we will have $2(N-1)$ wake messages

- Tokens are passed around by tree algorithm
  - All leaves send tokens all the way up to the first deciders ($N-1$, messages)
  - Then $N-1$ tokens are sent all the way down to the leaves

- Leaves us with $4N-4$ message complexity
What is the time complexity?

- Assume that the tree has diameter $D$
  - Diameter is the longest of the shortest paths between any pair of nodes

- Time complexity:
  - $D$ messages to flood wakeups, at $D+1$ everyone has started the tree alg.
  - $D$ messages for the tree alg to reach the first decision
  - $D$ messages to make everyone decide

- Gives $3D+1$ message complexity (not tight)
What about elections on rings?

- The first ever election algorithm was done for rings by LeLann

- Simple idea:
  - Let every initiator send a token with their identity around the whole ring
  - Nodes are *not* allowed to initiate after they receive a token (example…)
Example

- Nodes are **not** allowed to initiate a message after they receive a token (example…)

![Diagram](image-url)
Example

- Nodes are **not** allowed to initiate a message after they receive a token (example…)

![Diagram showing nodes and an initiator](image_url)
Example

- Nodes are not allowed to initiate a message after they receive a token (example…)

![Diagram showing nodes in a circular network, with one node highlighted as the initiator, and the note that the initiator is not allowed to initiate.](image-url)
Example

- Nodes are *not* allowed to initiate a message after they receive a token (example…)

![Diagram showing nodes and token flow]

- Initiator: not allowed to initiate
Example

- Nodes are **not** allowed to initiate a message after they receive a token (example…)

![Diagram showing nodes and their restrictions](image-url)
Example

- Nodes are **not** allowed to initiate a message after they receive a token (example…)

```
initiator
not allowed to initiate
```

```
initiator
not allowed to initiate
```

```
initiator
not allowed to initiate
```
Example

- Nodes are **not** allowed to initiate a message after they receive a token (example…)

```
initiator
not allowed to initiate
```
Example

- Nodes are **not** allowed to initiate a message after they receive a token (example…)

![Diagram showing nodes not allowed to initiate](image)
LeLann’s ring election

- Whenever a node receives back its id, it has seen every other initiators id
  - Assuming FIFO channels

- Let every node keep a list of every identifier seen \( (list_p) \)
  - If non-initiator, \( state=lost \) immediately
  - If initiator, when own id received:
    - \( state=leader \) if \( \min\{list_p\}=p \)
    - \( state=lost \) otherwise
LeLann’s Algorithm

\[
\begin{align*}
\text{var } & \text{ List}_p : \text{ set of } \mathcal{P} & \text{init } \{p\} ; & \color{red}{\text{Initially only know myself}} \\
& \text{state}_p ;
\end{align*}
\]

\begin{align*}
\text{begin if } p \text{ is initiator then} & \\
& \text{begin state}_p := \text{cand} ; & \text{send } \langle \text{tok}, p \rangle \text{ to } \text{Next}_p ; & \text{receive } \langle \text{tok}, q \rangle ; & \\
& \text{while } q \neq p \text{ do} & \\
& & \text{begin List}_p := \text{List}_p \cup \{q\} ; & \text{send } \langle \text{tok}, q \rangle \text{ to } \text{Next}_p \text{ ; receive } \langle \text{tok}, q \rangle & \\
& & \text{end} ; & \\
& & \text{if } p = \min(\text{List}_p) \text{ then state}_p := \text{leader} & \\
& & \text{else } \text{state}_p := \text{lost} & \\
& \text{end} ; & \text{else while } \text{true do} & \\
& \text{begin receive } \langle \text{tok}, q \rangle ; & \text{send } \langle \text{tok}, q \rangle \text{ to } \text{Next}_p ; & \\
& \text{if } \text{state}_p = \text{sleep} \text{ then state}_p := \text{lost} & \\
& \text{end} & \text{end}
\end{align*}

Send my id, and wait
Repeat forwarding and collecting ids until we receive our id
Termination: did we win or lose
Non-initiators just forward and lose

end
Message Complexity

- Worst case is every node is initiator ($N$)
  - Every initiator sends $N$ messages

- Gives a total of $N^2$ messages
Time Complexity

- Assume last initiator $f$ starts at time $N-1$
  - $f$ terminates after its token has circulated the ring, $N$ steps

- Time complexity $2N-1$
Chang-Roberts – An improvement

Chang and Roberts came up with a small improvement

Idea:
- When a node receives a token with smaller id than itself, why should it keep forwarding it?
- It is a waste, we know that that id will never win!
- Lets drop tokens with smaller ids than ourselves!
Discussion about election

- Are election algorithms of this “type” really useful?
  - If a node in the ring breaks down, the circle is broken, election will not work (same for trees)
  - The authors assume some external “connector” will fix the ring, and everything will proceed
    - Valid assumption?
    - In the case of a tree, if we anyway have to construct a new tree, the we could already embed leader information in that process
  - Is it reasonable to assume that nodes have ordered finite set of ids?
Surprise: DKS & Chord 😊

- They are scalable, keep ordered ids, and have magic “connectors”
- If node 2 fails, node 0 uses its successor list to reconnect the ring
- Same idea can be applied in trees
How to make it work

- But if we drop a token with id \( s \), node \( s \) will be waiting forever on its token to come back
- How do we solve that?
How to make it work

- If node $s$ token token is dropped, it will anyway receive another token with lower id
  - In that case, $s$ should set $state=lost$
Chang and Roberts Algo

\[
\text{var } state_p ;
\]

\begin{align*}
\text{begin if } p \text{ is initiator then} & \\
& \text{begin } state_p := \text{cand} ; \text{ send } \langle \text{tok}, p \rangle \text{ to } Next_p ; \\
& \text{while } state_p \neq \text{leader} \text{ do} \\
& \quad \text{begin receive } \langle \text{tok}, q \rangle ; \\
& \quad \quad \text{if } q = p \text{ then } state_p := \text{leader} \\
& \quad \quad \text{else if } q < p \text{ then} \\
& \quad \quad \quad \text{begin if } state_p = \text{cand} \text{ then } state_p := \text{lost} ; \\
& \quad \quad \quad \quad \text{send } \langle \text{tok}, q \rangle \text{ to } Next_p \end{align*}

\text{end}

\text{end}

\text{else while true do}

\begin{align*}
& \text{begin receive } \langle \text{tok}, q \rangle ; \text{ send } \langle \text{tok}, q \rangle \text{ to } Next_p ; \\
& \quad \text{if } state_p = \text{sleep} \text{ then } state_p := \text{lost}
\end{align*}

\text{end}

(* Only the leader terminates the program. It floods a message to all processes to inform they of the leader’s identity and to terminate *)
Nodes 1, 3, 6 are initiators

```java
var state_p;
begin if p is initiator then
    begin state_p := cand; send (tok, p) to Next_p;
        while state_p ≠ leader do
            begin receive (tok, q);
                if q = p then state_p := leader
                else if q < p then
                    begin if state_p = cand then state_p := lost;
                        send (tok, q) to Next_p
                    end
                end
            end
    end
else while true do
    begin receive (tok, q); send (tok, q) to Next_p;
        if state_p = sleep then state_p := lost
    end
end
```
Chang Roberts Analysis

- Worst case complexity same as LeLann’s
  - Time Complexity: $2N-1$
  - Message Complexity: $O(N^2)$
  - Considered a sorted ring with $N$ initiators

$$\sum_{i=0}^{N-1} (N - i) = N - \sum_{i=0}^{N-1} i = N - \frac{(N-1)N}{2} = \frac{(N+1)N}{2}$$
Further improvements?

var $state_p$ :

begin if $p$ is initiator then
    begin $state_p := \text{cand}$ ;
    send $\langle tok,p \rangle$ to $Next_p$ ;
    while $state_p \neq \text{leader}$ do
        begin receive $\langle tok,q \rangle$ ;
        if $q = p$ then $state_p := \text{leader}$
        else if $q < p$ then
            begin if $state_p = \text{cand}$ then $state_p := \text{lost}$ ;
            send $\langle tok,q \rangle$ to $Next_p$
        end
    end
end

else while true do
    begin receive $\langle tok,q \rangle$ ;
    send $\langle tok,q \rangle$ to $Next_p$ ;
    if $state_p = \text{sleep}$ then $state_p := \text{lost}$
    end

end

(* Only the leader terminates the program. It floods a message to all processes to inform them of the leader’s identity and to terminate *)
Extinction applied to the Echo algo

- Remember the Echo algorithm?
  - Centralized algorithm
  - Constructs a spanning tree rooted at the initiator
  - Works on arbitrary connected undirected graphs

- Do election by running echo with the initiators id
  - If another node is initiator, it will also start an echo with its id
  - Since echo covers all nodes in the graph, some nodes will receive waves from both instances
    - The node only participates in the wave with the lower id (caw)
    - The wave with larger id becomes extinct
    - Eventually the wave with smallest id reaches every node and wins the election
    - Use the constructed spanning tree to inform everyone about the result
Anonymous Networks

- So far, we have assumed every process has a unique identifier

- What if they do not?
  - Is this an interesting question?
  - In some cases, processes do not have ids
    - Sensor networks, mass-produced without ids, bought in bulk
    - Lego Mindstorm, the CPUs do not have ids
    - Some multiprocessors, the processors are arranged in grids, only know their neighbors (left, right, up, down)
  - Mostly of theoretical interest though
No names, no size

- We have been using ids to break symmetries in all election algorithms

- Furthermore, we might not even know the network size
Compute Network size with a Leader

- If leader exists, the network size can be computed
  - Use the Echo algorithm, start at the leader, it constructs a tree
    - All leaves send 1 to their parents
    - Internal nodes send the sum the value of their children to their parent
    - The root of the tree will know the network size

- If leader knows size is N, pick first id in \{1, 2, \ldots, N\}
  - Divide the rest of the interval into k parts for k children
  - Every child repeats this
  - When all leaves have been reached, everyone has a unique identity
**Impossibility of Election**

- Even if network size is known, but no leader is known
  - Election is impossible in anonymous networks

**Solution:**
- Probabilistic algorithms
Probabilistic Election in Anonymous Trees

- Run the tree algorithm
  - Eventually a decide happens at two neighbors
  - Each one picks a random number and sends to other one
  - If both pick the same number (low probability)
    - Restart again generating random numbers
  - Let the one with the lowest value win
Possible or Impossible?

- How come this is possible, we proved it cannot be done

- In our model, there is no way to generate a random number, in fact external IO is not in the model (keyboard, rnd gen, clock etc)

- Extend the model to make it probabilistic
  - Deterministic algorithms do not use random numbers
  - Probabilistic algorithms can pick random numbers in \{1,\ldots,N\}
Summary

- **Deterministic election algorithms (chap 7)**
  - For trees, using waves
  - For rings, LeLanns and Chang/Roberts
  - For graphs, Extinction of Echo

- **Anonymous networks**
  - Network size can be computed if leader exists
  - Unique names can be distributed if leader exists
  - Impossible to construct a deterministic election algorithm without a leader
  - Probabilistic algorithms using random numbers
    - Using the tree algorithm to elect a leader