Leader Election

Rings, Arbitrary Networks

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

1. Leader Election in Rings
   - $O(n^2)$ Algorithm
   - $O(n\ log(n))$ Algorithm

2. Leader Election in Trees

Outline

1. Leader Election in Rings
   - $O(n^2)$ Algorithm
   - $O(n\ log(n))$ Algorithm

2. Leader Election in Trees

Leader Election in Rings

- We assume that we are using a ring based overlay.
- We wish to choose the process with the smallest id as the leader. (NOTE: asymmetry)
- Messages can only be sent to the clockwise neighbor(left) or anti-clockwise neighbor(right).

Chang-Roberts Algorithm

```
if p is initiator then
    state ← find
    send p to next(p)
    while state ≠ leader do
        receive(q);
        if p = q then
            state ← leader
        end
        else if q < p then
            if state = find then
                state ← lost
            end
            send q to next(p)
```

Chang Roberts Algorithm - II

```
else
    while true do
        receive q
        send (q) to next(p)
        if state = sleep then
            state ← lost
        end
    end
end
```

Analysis

Message Complexity

- Assume there are $O(N)$ initiators.
- The leader's message will be sent N times.
- For other initiators, the message will be sent $N - i$ times.
- $\sum_i (N - i) = O(N^2)$.

Analysis

Message Complexity

- Assume there are $O(N)$ initiators.
- The leader's message will be sent N times.
- For other initiators, the message will be sent $N - i$ times.
- $\sum_i (N - i) = O(N^2)$.

Optimization

Global broadcast is not necessary

Outline

1. Leader Election in Rings
   - $O(n^2)$ Algorithm
   - $O(n\ log(n))$ Algorithm

2. Leader Election in Trees

Basic Idea

- We get a $O(N^2)$ complexity because each message can travel $O(N)$ hops.
- Instead of sending a message to everybody, we need to find a way to filter the set of messages (similar to Maekawa's algorithm)
- We will consider gradually larger sizes of windows in a sequence of rounds .
- Each window will allow only one of its members to participate in the next round.
- If, we are able to filter the number of participating members by a factor of 2 in each round, we will have $O(log(N))$ rounds.
- If in each round, we send only N messages, then a total of $O(N\ log(N))$ messages need to be sent.

$O(n\,log(n))$ Time Algorithm

```
initialize:
send (probe,id,0,1) to left and right

receive (probe,j,k,d) from left(right):
if j = id then
    leader ← j
    Terminate
end
if j < id and d < 2^k then
    send (probe, j, k, d+1) to right (left)
end
if j < id and d = 2^k then
    send (reply, j, k) to left (right)
end
```

$O(n\,log(n))$ Time Algorithm - II

```
receive (reply,j,k) from left(right):
if j ≠ id then
    send (reply,j,k) to right(left)
end
else
    if received (reply,j,k) from right(left) then
        send (probe,id,k+1,1) to left and right
    end
end
```

Analysis

- The maximum number of winners after k phases is:
  - To winners can at the least be 2^k entries apart.
  - Thus, the total number of winners after k phases is $n/(2^k+1)$
- The total number of messages for each initiator in phase k is 4×2^k
- Total number of messages in the k^{th} phase is:

$$4 \times 2^k \times \frac{n}{2^{k-1}+1}$$

- Total number of messages is:

$$M = \sum_{k=1}^{log(n)} 4 \times 2^k \times \frac{n}{2^{k-1}+1} = O(n\ log(n)) \tag{1}$$

Leader Election in Trees

- Let us consider arbitrary networks.
- Creating a ring based overlay is difficult (It amounts to constructing a Hamiltonian cycle – NP Hard).
- However, creating a tree based overlay is easy.
- To further optimize the process, we can choose the MST (minimum spanning tree) as the overlay.
- Assumptions:
  - Let the current node be termed as p
  - Let a neighbor be termed q
  - All the leaves (degree=1) are initiators

Initialization

```
/* Wakeup all the nodes                                   */
if p is an initiator then
    awake ← true
    foreach q ∈ neigh(p) do
        send wakeup to q
    end
end
while numWakeups <| neigh(p) | do
    receive( wakeup )
    numWakeups ← numWakeups + 1
    if awake = false then
        awake ← true
        foreach q ∈ neigh(p) do
            send wakeup to q
        end
    end
```

Send Proposal to Parent

/* Collate result from the leaves and send to parent */

received $\leftarrow$ 0

$min_p \leftarrow$ p

while *received* < *#children* **do**

  receive $< r >$ from q

  $rec_p[q] \leftarrow$ **true**

  received $\leftarrow$ received + 1

  $min_p \leftarrow min(min_p, r)$

end

send min_p to *parent*

Decide the Leader

```
/* Receive the result from the parent, and
   send to the leaves                          */
receive < r > from parent
res ← min( min_p , < r > )
if res = p then
    state ← leader
end
else
    state ← lost
end
foreach q ∈ neigh(p), q ≠ parent do
    send res to q
end
```

Analysis

Message Complexity

- On every edge, we can send at the most two wakeup messages
- We can send a proposal and its reply.
- A tree with N nodes as $(N - 1)$ edges.

Complexity

Message Complexity: 4N - 4 = $O(N)$

- Introduction to Distributed Algorithms by Gerard Tel, Cambridge University Press, 2000
- Distributed Computing, Fundamentals, Simulations and Advanced Topics by Haggit Attiya and Jennifer Welch, Wiley 2004
- Advanced Concepts in Operating Systems by Mukesh Singhal and Niranjan Shivaratri, McGrawHill, 1994
- Distributed Algorithms by Nancy Lynch, Morgan Kaufmann, 1996