CS5320: Distributed Computing, Spring 2020

Report: Efficient Anonymous Leader Election Algorithm for Connected Topologies

SAI HARSHA KOTTAPALLI, CS17BTECH11036

Introduction

Leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes).

Existence of a leader is very important in many of the distributed applications. System-wide management and can be a coordinator, initiator, etc. Choosing an optimal leader would result in less communication overhead, ideally being the central node. Example usage: broadcasting something to the network. The proposed algorithm can then be used for Mutual Exclusion, Termination Detection and Global Snapshots.

Our method is better than other methods for leader election because it does not need the processes to be in a fixed shape topology (like Frankfurt and Peterson's algorithms). It treats all the processes equally instead of giving inherent advantage to some processes without a reason (like giving priority to the node with lowest/highest process id in case of the normal bully algorithm). It also handles dynamic network topology where new nodes are added and some removed.

We use conditional pthreads for the simulation so that all the receiver threads are ready before the sender threads start sending data so that simulation only accounts for the time taken in electing the leader.

Algorithm

High level details

1. Distance-vector (DV) algorithm is iterative, asynchronous, and distributed. Use the DV algorithm to iteratively compute the shortest path from each node to every other node and store it within the node. Will happen periodically detecting any changes to network topology(addition of node or deletion of node) and communicating it to the other network nodes.
2. For the messages involved in DV and bully usage of appropriate encoding for exchanging messages and parsing at the receiving end.
3. Compute the mean of the said distance vector, ignore non-reachable nodes.
4. Use the above attained metric in the modified Bully algorithm with the value 1/(mean) instead of lowest PID , to elect the leader periodically. This makes sure that the leader is alive.
5. This ensures that the process that should encounter least communication costs will be the leader.

Distance Vector Protocol

The DVP implementation in our simulation supports addition of nodes as well as removal of nodes from the network topology. This is handled by the way our data structures for DV is made for the DV Table.

Below is the algorithm which explains how DV calculated optimal paths for a given topology.

Node x:

Initialization

```
// Let N contain list of all nodes in network topology
// Let D store distance vectors of nodes according to x

For all y in N:
    if y is neighbour of x:
        $D_x(y)$ = cost(x, y)
    else:      $D_x(y)$ = INF

For all z in N:
    if z != x:
        $\forall$ y $\in$ N, $D_z(y)$ = INF   // node x doesnt know about other
                                               nodes

For all k in Neighbours:
    Send($D_x$) // $D_x$ is the current forwarding table for x.
```

Sending Loop

```
// will let us know if any node has disconnected or a new node has
been attached.

Every x interval:
    For all k in Neighbours:
        Send($D_x$) // $D_x$ is the current costs for x to other
                        nodes.
```

Receiver Loop(D_z)

```
// z is sender node
// If any changes are present in the cost vector sent by z
if isRecalculateRequired($D_z$) :
    Save $D_z$ into local D.
    For all y in N:
        $D_x(y)$ = $\forall$v, min{ cost(x, v) + $D_v(y)$ } // v is a neighbour
else: pass
```

Bully Algorithm (Modified)

It is intuitively obvious that we want to use the central node (with the best connectivity to other nodes) as the leader. Once the distance vectors are populated for all the nodes, we use that information to find out the best nodes that can be the leaders.

For the sake of simplicity, using previous notation, let $D_x(y)$ = cost(x,y)

Let M be the mean of all the *non-infinite* distance values in D_x where x is the current node and let the weight be 1/M.

Initialization

```
// Compute the weight as mentioned above
sum, count = 0, 0
for every node y:
    if x == y : continue
    if D_x(y) == INF: continue
    count += 1
    sum += D_x(y)
weight = count / sum       // 1/mean
leader_weight = weight
leader_pids = []
```

Broadcasting Weights and receiving them

Sending thread

```
// here, `weight` is weight of current process
for every node y:
    send(weight, y)
```

Receiving thread

```
// receive `received_weight` and `received_pid` from other process

if (received_weight > leader_weight):
    leader_weight = received_weight
    leader_pids = [received_pid]

else if(received_weight == leader_weight):
    leader_pids.push_back(received_pid)

// Basically maintaining list of potential leaders
```

Finalizing on the leader

```
leader_pid = minimun_pid(leader_pids)
```

The leader elected finally, will be the optimal leader.

Result and Conclusion

We find and elect the optimal leader in the network topology which is essentially the network node which is at the centre of network topology. Using this Node as the leader would ideally help in reducing time involved while exchanging messages while also identifying network topology changes quickly.

This is a proof of concept for our idea of electing a worthy leader where the worth of a leader is determined by the ease of communication with it.

A real life use case:

This algorithm can be deployed into networks processing big data with a centralized leader. Since many processes exchange huge numbers of packets, we notice that the packets travel less distance(cost involved) unlike the vanilla bully algorithm which selects the lowest pid as the leader.

Other uses:

Can be deployed into networks using leader for Mutual Exclusion, Termination Detection and Global Snapshots.