The Arctic University of Norway

Report Inf-3200
Distributed Systems Fundamentals

Leader Election

9.11.2015

Tim A. Teige

Fredrik Høiseter Rasch

E-mail: tte008@post.uit.no, fredrik.h.rasch@uit.no

Introduction

This report describes the implementation and design of a leader election in an unstructured distributed network. The leader election algorithm used is the Yo-Yo algorithm.

Leader Election

A leader election selects a leader in a network. It is not always known which node is the leader node in the network when it is initialized, therefore the leader election is often the process to be run first. This is done so it is possible to organize the network. The leader is responsible for delegating tasks for the current job and synchronization between nodes. Some nodes in the network might not be able to directly communicate with the leader, it is therefore vital to be able to recreate a possible path for communication.

Design

The network is not organized in an architecture. This is done to show that the Yo-Yo election algorithm does not need a specific structure. The nodes are connected to random nodes when they connect to the network. A frontend server is used to log on and off nodes and visualize the connected nodes. It does not manage anything during the election algorithm.

Yo-Yo Leader Election

This algorithm is a minimum finding algorithm. The Yo-Yo algorithm consist of two parts. A preprocessor phase, Yo-Yo. The network is treated like a directed graph. Where the nodes in the network are the nodes in the graph and the connection between nodes are the edges.

Preprocessing

The preprocessing phase consists of nodes exchanging ids with all their neighbours. Each pair of nodes orients their edge towards the node with the smallest id. A node where all the edges point inwards is known as a sink. A node where all the edges points outwards is known as a source. All other nodes are known as internal nodes. When all nodes have completed this phase, the next step can take place.

Yo-Yo

The Yo-Yo phase consists of two parts, Yo- and -Yo

Yo-

This phase may contain multiple iterations, based on the result of an iteration. The first step is for the source to send its id to all nodes connected to it. The internal nodes then waits to retrieve all the values from the nodes that it is connected to. Then it calculates the minimum of all the values it has received and sends this value to all its outgoing connections. When the sinks have received all the minimum values from all the neighbours, it then calculates the minimum.

-Yo

The sink now sends a yes to the neighbours which gave it the smallest value and no to all others. The internal nodes sends yes to all the neighbours it received the smallest value from and where the edge was oriented inwards, else it sends a no.
If the internal node received one and only one no, it sends no to all the connected nodes. The sources wait until they receive all the votes, if they are all yes, it is still a candidate.

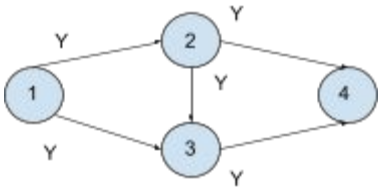
Restructuring the graph

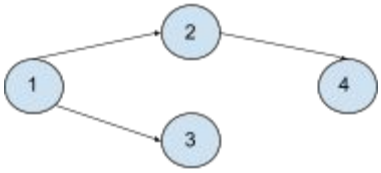
This step is necessary to make only the sources which are still candidates sources.
If a node sends no to a neighbour where the edge is oriented inwards to the node, the edge is flipped. $x \rightarrow y$ now becomes $y \rightarrow x$

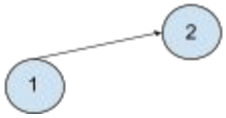
If there is multiple candidates left, a new iteration of the algorithm is initiated.
The algorithm will terminate when only the leader remains.

Pruning the Graph

During the algorithm, nodes check if they only have a single edge oriented inwards, and no edges oriented outwards, if they do, they ask to be removed from the voting. In other words, have the edge removed. If the connected node with the edge oriented outwards now is now in the same situation, it also asks to be removed. These nodes have no impact on the voting, since they can only send out a single yes and no no’s.
If a node receives the same values from multiple nodes while receiving the smallest id, it will ask all of them except one to remove the edge. This ensures that the least amount of messages has to be passed, while conserving the consistency of the voting.







In this situation the index 1 will be spread to all nodes and all the nodes will compute it as the minimum. All nodes will then agree that it is the minimum and vote yes. Edges which is redundant are removed. The iteration starts again, all vote yes and sink nodes which are leaf nodes are removed. The next step would be to remove node 2, since it is a leaf sink node.

Implementation

The frontend is implemented using asp.net and the peer to peer backends are implemented C#. The backends are compiled using mono.

Discussion

The Yo-Yo algorithm has a maximum of O(mlogn) messages passed to elect a leader, where m is the number of edges and n is the number of nodes in the graph. The actual complexity has not yet been established, this is still an open research question, but in an unstructured network it is mainly dependant on how loosely connected the graph is.

There are two main algorithms used for unstructured networks, this is Yo-Yo and Mega-merger. The Mega-merger algorithm calculates the root of the network by applying an algorithm close to that of creating a minimum spanning tree of the network. This has a maximum of O(m + nlogn) messages passed to elect the leader. Compared to the Yo-Yo algorithm the main difference lies in ratio between number of nodes and how many these are connected to.
When comparing to a structured network such as a ring where each node has a unique id the number of messages passed to elect a leader is O(n^2) in a unidirectional ring and O(nlogn) in a bidirectional ring. In comparison to the Yo-Yo algorithm the amount of messages needed to elect a leader is the double. In the bidirectional ring the amount of edges in the graph would be twice the amount of nodes, and therefore is equivalent to O(2nlogn). Due to the Yo-Yo algorithm being universal it has the possibility to be deployed on any type of network.

Due to the way the network is created, there is no guarantee that all the nodes in the network is connected. It is possible to form separate subnetworks that are separate from each other. This would cause these networks to operate individually and not share the same leader. This could be solved by a better strategy for creating the network. For instance a crawler that traverses all the nodes in the network, if it can't reach all the nodes connected, either connect the two subnetworks or recreate the network and redeploy the crawler.

The implementation of the Yo-Yo algorithm does not work well for most cases. It is possible to connect nodes and disconnect them. Finding the leader does not work because of the flaws in our implementation. The front end displays the nodes and the connected nodes. Due to this there is no analysis. With our current implementation the amount of messages will reach above the optimal number of message O(mlogn).

Conclusion

The design is a good solution for unstructured networks with the right ratio of edges to nodes, else the Mega-merger is a viable option. Currently the implementation does not work for most cases, but there are edge cases where it might work.