UNIVERSITY OF TROMSØ

INF-3200: Distributed Systems: Assignment 2

Christian Reehorst, Dani Beckmann, Nishant Verma

November 9th, 2015

1 Introduction

In this document we will describe our choice of architecture for implementing leader election to a previously built CHORD peer-to-peer network. This architecture takes into account for a single leader to be elected at any given time, as well as new nodes to join. At this stage it is not possible for neither leader nor nodes to leave the network.

1.1 Requirements

With our chosen architecture from the first assignment, we are to implement a couple of new features in this one:

- Our architecture should support a minimum of 10 nodes
- Support graceful leaving of nodes, along with joining of new ones as the system demand increases
- Anoint a leader through election

2 Technical Background

A general idea of circular single linked list is required as we keep propagating to the next node.

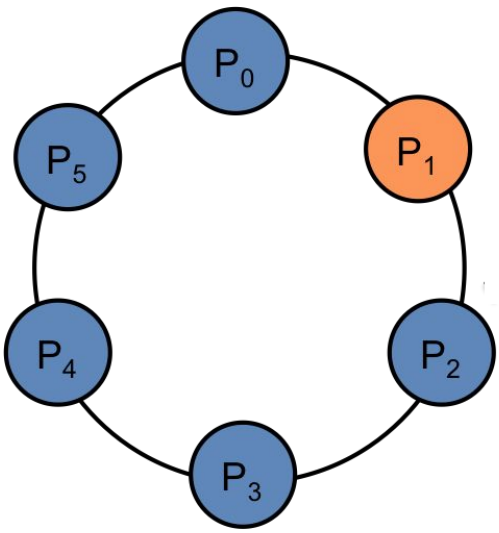
The PUT and GET requests from the front-end uses the HTTP protocol. The module is always implemented on the client side. An instance of the HTTP protocol represents one transaction with a HTTP server. It is instantiated passing a host and an optional port number.

3 Design

Building upon a CHORD architecture we implement a solution based on the *Chang-Roberts algorithm* where communication goes in a clockwise direction in an unidirectional ring. Each node in the network have an unique id, and each node can start the process of selecting a leader if none is present.

When the process of election is started, the node sends out a message containing its own id(in our case is process id itself) to the next node in line, and marks itself as a candidate for leader. The next node then compares the id in the received message with it's own id. If its own id is lower than the one received, the node forwards the message to the next node. If the node have a higher id than the one in the incoming message, it then replaces the id in the message with its own, forwards the message and becomes a candidate for leader. If the node already is a candidate and receive a message with a smaller id, the message is discarded. This stops multiple processes of leader election to happen at once. Lastly, if the node receives a message with an id equal to its own, the node starts to act as leader. This way only the message with the highest id completes the trip around the circle, and only one node can be elected as leader, even if multiple election processes are started.

This is illustrated below, where each node is marked as a process(p). By this, only the node p5 can be elected leader, because a message with id5 will override any lower id's, and any process that hits a node already a candidate is discarded.



Implementation

Our back end node is designed as client and server. It accepts a TCP connection on a certain port and forwards the request to the neighbour node if needed. The entire implementation is based on the chord architecture. If a node joins, we maintain the ring structure by updating its next pointer and new node next pointer and so on.

Whenever a node joins the chord, an election is started and it sends a put request adding its process id in the request body. When a node finds its own id in the request, it declares itself as the leader and announce the result. This is again a PUT request which is circulated to all the nodes in the ring, and every one knows who the leader is. The getNode request just returns the next node that node points to.

4 Discussion

We have chosen to use the Chang & Roberts leader election algorithm in our solution, because of it's simplicity and the accessible documentation on in it. This uniform algorithm is a lot like the bully algorithm in the way that each node have an id, and is actively participating in selecting a new leader, but it reduced the amount of message passing. From what we've read about it we expect the algorithm to use O(n ^ 2) messages and O(n log n) on average, which is very nearly linear behavior.

Our intention was never to implement an architecture that pushed the limit of highest efficiency, but rather to create a robust system we could play with and improve as we see fit. That is not a decision we have regretted as we have failed to implement graceful shutdown of nodes, or in fact any abdication from the system whatsoever.

The Chang & Roberts algorithm is very flexible in the way that you can improve upon it without changing the foundation, and we had an idea to be able to get to the point where we could pass messages both ways in the ring structure.

Also, we would like to mention that we tried to implement the exact algorithm, but it is a bit different and we did not consider the higher id to override the contender with smaller id as we ran into issues we could not figure out. But the simpler version runs fine.

4.1 Evaluation

Using the test in leader_benchmark.py to check our architecture and election algorithm, we find that the nodes in our ring structure unanimously returns the same leader id to the front-end.

5 Conclusion

Building on our architecture from assignment one, we were to implement leaving and joining of new nodes, and an election of a leader. The minimum number of nodes to support should now be 10. In this report we have described our choices linked to implementing these new features, looking at strengths and weaknesses of it. Our leader election is id-based, and they are held upon nodes joining the network.

For the group, we still had some problems overcoming programming language barriers in python and shell-script. We learned a lot during the previous assignment, and that made parts of this one easier, but there is still a lot to learn, and many points that can be improved upon. It has been particularly frustrating not being able to implement nodes leaving. We found early that this part was going to be tricky, and prioritized implementing the other features over this one. As the deadline approached, we had to make a decision, as our attempts at creating a successful leave had all failed, we had to drop this in favor of finishing up the rest of our code. In the end, the rest of the system works well given certain parameters.

References

UK Essays. Comparisons With The Bully Algorithm Information Technology Essay. Accessed 01.11.2015, from: http://www.ukessays.com/essays/information-technology/comparisons-with-the-bully-algorithm-information-technology-essay.php

Wikipedia. Chang & Roberts algorithm. Accessed 01.11.2015, from: https://en.wikipedia.org/wiki/Chang_and_Roberts_algorithm