Assignment 2

Nicolai Bakkeli, 11.06.2015

Introduction

This paper will briefly explain the design and implementation of a leader election scheme placed on top of a peer-to-peer network. It will also briefly explain how nodes are added and removed to/from the network.

Technical Background

Leader election

Leader election is a procedure that is embedded in every node of the distributed system. Any node which detects the failure of the leader node can initiate a leadership election. The election concludes its operation when a leader is elected and all the nodes are aware of the new leader and agree on that.[1]

Bully algorithm

The bully algorithm is a programming mechanism that applies a hierarchy to nodes on a system, making a process coordinator or slave. This is used as a method in distributed computing for dynamically electing a coordinator by process ID number. The process with the highest process ID number is selected as the coordinator.[2]

Design

Node integration

Each node have the same functionality and can initiate itself alone without help. But each node need to communicate in order to create the network.
At the start up of a new node will this node access a directory that is available for every node in the network. It will create a file named after its own address and store this file in the given directory. Since every node does this can the addresses of other nodes be gathered by looking through this directory. So the new node reads the list of files in order to get the other node addresses. These addresses is then stored in a table that contain the addresses and whether or not the nodes are active or not.
The new node will at this point broadcast to every other node that it wishes to join the network. Nodes at the receiving end adds the new node to their tables and returns that they are active. Each answering node will be updated in the new nodes table as active.
Some nodes might not answer at this time so the new node waits before committing to the network until the table is filled with active entries. It does this by broadcasting the same message to the nodes that did not answer, and continues to do this until they do.
When all this is done is the new node considered ready for work. At this point will the new node start a coordinator election (even if it exist a coordinator).

Election

A node that start an election will broadcast an election message to every node that has a higher ip address relative to itself. Nodes that receive this message know that it's time for a new election and will start the same process as the initial node.
Each node that have broadcasted in this way will wait for replies from the other nodes. Nodes know that they are not supposed to be coordinators whenever another node answer their broadcasts. If no other node answer the broadcast can the given node take the role as the new coordinator. A broadcast is sent out to every node by the new coordinator to tell them that this node is the new coordinator. Other nodes that know that they are not coordinators will wait for a reply from whoever that will be the new coordinator. The waiting slave nodes will wait for a while in hope to hear from the new coordinator, if no node clam to be the coordinator (within a timelimit) will a new election start.

Coordinator check and shutdown

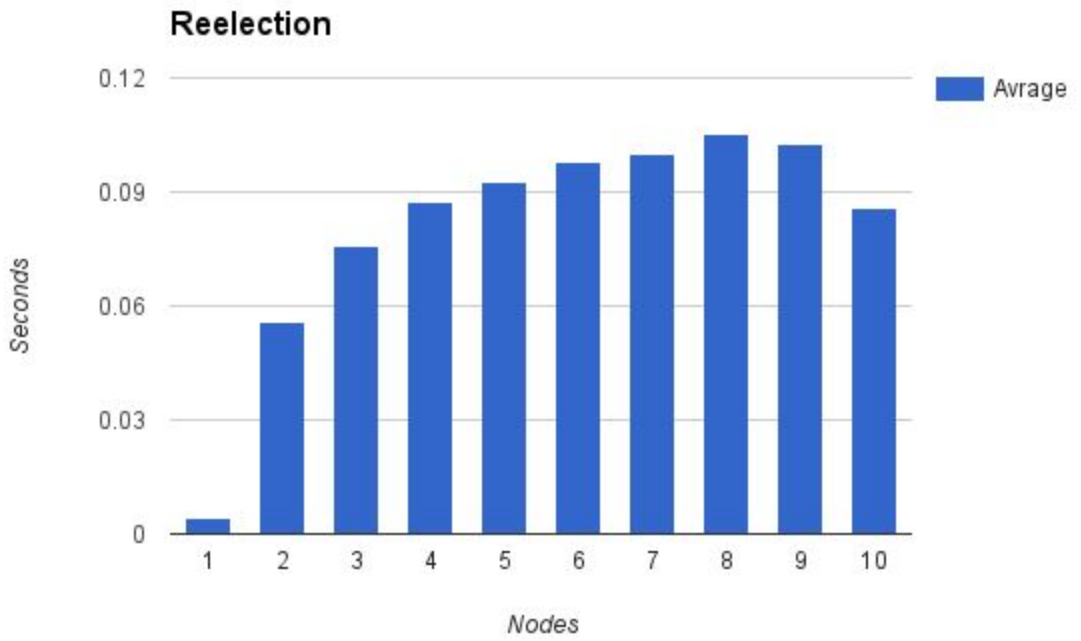
Each slave node will within a given time limit check on the coordinator to check if it is alive or not. A new election will start whenever a node can't get any response from the coordinator.
Whenever a node shuts down will it broadcast to every other node that it is leaving the network, and it will destroy its own file on disk (containing the address of the node) in order to keep future nodes to ignore it.

Implementation

The code was written in the programming language python. The design was implemented on some previous work that had some fundamentals ready such as the base server and a frontend to communicate between user and nodes. The implementation was tested with “leader_benchmark.py” and “owntest.sh” script created for testing election time.

The leader benchmark gave information on which nodes are connected with each other and which node is the current leader. It asks each node what node is the leader to see if there are any inconsistencies. By running this test on the implementation did it answer that the leader is the same between all the nodes and that every node know each other. I don't intend to claim that this is always the case but in the tests i did run were there no inconsistencies. The test have stopped a few times because of communication issues between the test and the network. This will be discussed in the discussion.

The time of elections have been timed and can be viewed in the image below. These election times have been done on random nodes by testing the time on ten nodes, then removing a random node to do the test again. This process continues until no nodes are left, then every node is restarted and the test can be issued again. The request is also sent to random nodes among the available nodes so there is multiple layers of radom in this test. The test were run 3000 times for each “node count” in order to make the dataset more correct due to all the random elements in the test. The dataset will be discussed in the discussion.



Discussion

The implementation works to some degree and have some ugly uses of “try” in python to avoid some of the minor issues that might occur. This is of course bad practice but i chose to use them this way in order to complete the task within the deadline.

The implementation uses two threads, one for the main server and one for sending messages. The sender thread was meant to be used the leader election alone. The idea of having two threads was to have one communicate with the client while the other was doing the background communication. This is not really necessary but i left it this way anyways.

Every node in the network need access to the disk in order to get knowledge of the other nodes. If the implementation was placed on an open network would the directory be prone to sabotage as any user could remove files from the network in order to keep new nodes from getting knowledge of the existing nodes. The other problem is how to solve the accessibility of the directory for nodes outside the inner cluster. When the scale of the network increase will there be problems with the amount of messages sent across the network for each election.

On to the test results. There occured a problem when running the leader benchmark. A plausible reason this might be is that the server is receiving a SIGPIPE when writing to a socket. The problem usually occurs when data is written to a closed socket. So it is probable that a socket is closed before all the data from an end has been completely received. This might be why the hiccup happened.
When looking at the election time from the tests can we see that the time increases as the nodes increases. This is at least true until the last two “node counts” (9 and 10 nodes). Having an increase in time when the nodes increase is expected as the chain of election requests will be longer. But the wierd thing is the decrease in time on the two last tests. The reason this might be is that the first tests are conducted on the highest amount of nodes. Each time the test restart will there be a new set of processes thus having “fresh” processes to work on. The problem with the others might be that they suffer from memory leaks or that the nodes are reacting to the previous node leaving while the new test is issued. A way to fix this is to reduce the frequency of the tests and do half the tests in the opposite direction (from few nodes to many nodes).
Testing on more than 10 nodes were problematic as the cluster started asking for my password. This made it hard to automate. But the password could of course be written in the script but i felt that it would be unwise so the tests ended on 10 nodes. On that note is it possible to run the implementation on every node on the cluster as it was tested manually.

Conclusion

The leader election has been implemented and can support more than 10 nodes. It supports graceful shutdown and the addition of new nodes. Some bash scripts have been created to startup and shutdown the nodes. It's advised to read the readme in order to use these.

References

[1] Enhanced Bully Algorithm for Leader Node Election in Synchronous Distributed Systems, mdpi: Date retrieved: 6 November 2015 19:36 UTC, https://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&cad=rja&uact=8&ved=0CFIQFjAGahUKEwjDir63tvzIAhULqXIKHUXHB6s&url=http%3A%2F%2Fwww.mdpi.com%2F2073-431X%2F1%2F1%2F3%2Fpdf&usg=AFQjCNEdW98EzahD5hdTl4Jf15G9ngU24g&sig2=eDq2jphzE5taj23yT9RuSg

[2] Bully algorithm, Wikipedia: Date retrieved: 6 November 2015 18:40 UTC, https://en.wikipedia.org/w/index.php?title=Bully_algorithm&oldid=678440989