INF-3200 Assignment 2

Introduction

In this assignment the author were asked to implement a solution for leader selection in a distributed system. The system must support at least 10 nodes and must support the get calls getCurrentLeader and getNodes. Where the first one should return the leader of the nodes, and the last one should return a list of nodes and ports. It should conduct a leader election if no nodes are leader. It could also, but not a priority requirement shut down and start up nodes.

Design

The author started by reading up on election algorithms, and the the difference between them. The author then decided that with the given time that the bully algorithm were the best candidate. The bully algorithm works in the way that it starts an election and sends this to nodes of higher order.[1] This means that the nodes with higher index will be contacted and on the picture on the right there is a description of how the election works. The author chose to implement the bully algorithm with some changes, like the coordinator message. This is not sent when elected, but stored by the leader and when needed distributed to nodes. In this way there is always one node that knows of the leader, at that is the leader himself. So when calling upon for example get current leader, if the node does not know the leader it sends a request to the other nodes of higher order for the leader. As there is always at least one node that knows of the leader it will fetch this and store it for further use. So when another node of lower order is calling upon who the leader is there is now two nodes that knows of the leader, and so on.

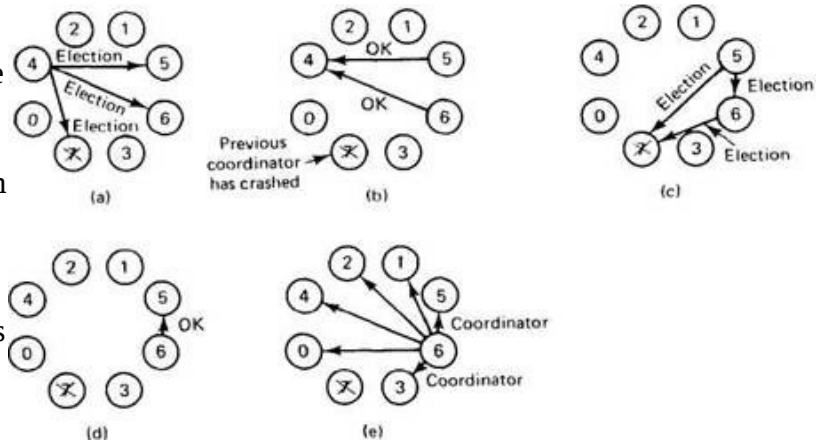

Illustration 1: Bully algorithm

The author based the implementation on the precode from the previous assignment with the same http handlers and requests handlersand made modifications so it will suit the current assignment. Where in the previous the nodes only knew of two nodes, they all know of everyone. The author did not have the time to implement adding and removing of nodes, but had some ideas of how this would work. This would be run from a script like the startup.sh script were you could start up all nodes and then have the choice to remove and start up a new node.

When removing nodes, the node that were meant to be removed then would send a terminate signal to all other so that they would know that this node is shutting down. When this is done there will called a new election so establish that there is a leader at all time. The same goes if there is a new node joining that there must be an election to make sure that the new node is not the leader, if it is then the leader must be changed. And for every election the leader must be reset so, and when the node is in need of the leader it must search for the leader again. This script will have the default number of 10 nodes, where the user could with ease shut down and start up nodes with commands.

The author has also made the decision of making a stop script to shut down every node that runs.

1 http://www.e-reading.club/chapter.php/143358/98/Tanenbaum_-_Distributed_operating_systems.html

Testing

The tests are run on the cluster from the authors own computer using the start up script and the benchmark script provided by the TA's. The time parameter were used to measure time(time python leaderbenchmark.py etc...).

	10 nodes	12 nodes	15 nodes	20 nodes
	10,307	10,354	10,398	10,469
	10,305	10,328	10,335	10,469
	10,308	10,258	10,357	10,506
	10,257	10,275	10,373	10,425
	10,245	10,283	10,389	10,460
	10,294	10,259	10,352	10,426
Average	10,286	10,293	10,367	10,459

The author run the tests with 4 different amount of nodes: 10, 12, 15 and 20. And at the bottom it shows the average time of these test runs. The results will be discussed in the discussion part of this paper.

Discussion

Starting up the discussion by taking a look at the results, and by adding more nodes there is obvious that the time it takes slightly increases. It also clear that the first run are for the most part the run that takes the longest. This is because when the first request for either leader or nodes the node with the lowest index will start an election to find a leader. Also since no node knows (besides the leader himself) of the leader, it will need some time to search for leader. The author could here taken the time to conduct more experiments and tests where the tests were run, then the nodes were shut down. Then run another test to get an average of the first time run.

The author could have chosen some other solution regarding the design choices. The choice of not implementing the whole algorithm, but shorten it down to not broadcasting the elected leader during election. This is a shortcut made by author since the first implementation of broadcasting did not work, and there a lot of bugs that the author did not manage to fix. These bugs were that the election and broadcasting were way too inconsistent. The election were able to find the correct leader, but when the leader were meant to broadcast itself as the leader the problems occurred. The leader had problems when connecting to the other nodes, and when connecting to those nodes it had problems with connecting to everyone. There were some nodes that received the coordinator request, but not all of them. When this happened, the nodes that it did not connect to did not have any knowledge of who the leader were and therefore started a new election. In this way there occurred a endless loop where the election never ended. The author tried to work around to fix this but because of the deadline, this was not managed and the secondary solution of searching and fetching the leader were implemented instead. This solutions works very well but the author does not have any data to compare it with, like for example an original bully implementation. So in terms of comparison with the original the author has no data, and no arguments of which one will work better.

There are also other algorithms that the author could have used, like ring algorithm which has a better worst case scenario then Bully algorithm which has n^2 as worst case in contrary to Ring which has a worst case of 2n. There are also other more complex algorithms that could be used like Paxos. The ring algorithm are like the name, a ring of nodes together which knows the structure of the ring. And when the initiator calls for an election it adds the index to the message and sends it to the next which adds its own index etc. When initiator gets the election message again it chooses the best index(highest) and calls coordinator message to the rests. This is an algorithm that the author considered because of the worst case message notation. The author started to implement this but after some time with this algorithm the author worked more for the Bully algorithm. But in some

way the solution is also at some degree the Ring algorithm. When looking on the election message, the initiator sends it to nodes of higher indices. But when getting an “ok” from any one, it stops sending, so like the Ring algorithm it could end up just sending one election message to higher nodes. This will reduce the worst case scenario of n^2, and ending up with n messages instead if no node is down.

Conclusion

The author has in this paper showed that he has designed and implemented a solution for leader election among nodes in a distributed system. There were some design choices which the author could have done different but the system works. The system are not the fastest or best solution, but the author has learned from this experience.

There were some adjustments that had to be done, to ensure that the solution were complete and working at the end.