MIE-DSV Project

Victor Moreno Arribas
morenvic@cvut.cz
@morenvic

Czech Technical University

Faculty of Information Technology

2021-2022

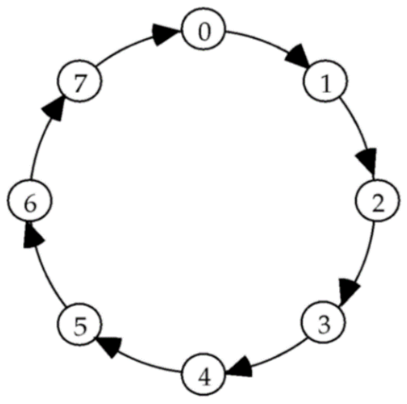
Table of contents

Project description

This project is part of the MIE-DSV Distributed Systems and Computing subject. During the course each student is assigned on of the distributed algorithms that we studied during the course to implemented using the Google RPC (Remote Procedure Call) also called the gRPC framework. In my case, I had to implement the **Unidirectional Link Eager Leader Election Algorithm**.

Algorithm description

The algorithm implemented for this solution is the Unidirectional Links Eager Leader Election Algorithm in a distributed system using a ring topology.



This algorithm consists of a given list of nodes that make a ring, like the picture on the left, and the algorithm has to start in a Unique Initiator and analyze each of the

nodes pairs one by one. In each comparison between two nodes, if the sender's value is lower than the receiver's value, then the receiver is eliminated from the topology

and the in the followings iterations it will be skipped. The idea is that in each iteration at least one node is eliminated until only one node is left which is the leader.

We have followed the most common design pattern as shown in class that consists of communicating each node only with the one in the right from the node view as the links are

unidirectional, and there each node can only be connected to another node (but just one, not two or more).

1. The first node from the given list is considered as the Initiator and therefore as a sender.
2. Sender sends a message to the receiver (next node)
3. If sender's value is lower than receiver's value, then receiver is eliminated. If not, continue to next pair.
4. If a node is eliminated, the sender has to skip to the next available node in the topology.
5. If sender and receiver are the same node, then is a leader. Finish.

Example of the algorithm

3,1,2,4

Nodes = [3,1,2,4]

Iteration 1:

¿3 < 1? False, therefore continue with next pair of nodes.

¿1 < 2? True, therefore eliminate node 2 and new receiver of node 1 is node 4 because nodes list is now [3,1,4]

¿2 < 4? True, therefore eliminate node 4 and new receiver of node 1 is node 3 because nodes list is now [3,1]

Iteration 2:

¿3 < 1? False, therefore continue with next pair of nodes.

¿1 < 3? True, therefore eliminate node 3 and new receiver of node 1 is node 1 because nodes list is now [1]

¿1 == 1? True, therefore **Node 1 is leader**.

Architecture description

The architecture of the project is based on different parts:

gRPC working files	eager_pb2.py eager_pb2_grpc.py eager.proto
node file running a single node	node.py
main program	manager.py
data files	general.txt received.txt sent.txt

GRPC framework

What’s RPC?

Remote Procedure Call is a technology that regulates inter-process communication, i.e., the exchange of information between system processes. RPC in a synchronous mechanism "that transfers control flow and data between two address spaces over a narrowband network as a process call". By traversing the address space, processes can be started on a remote computer connected to the network and external instances can be included in computation and data processing processes in an operational manner. The RPC communication process consists of sending parameters and returning a function value. It is often not limited to a single call, since in practice many requests are processed in parallel.

Ultimately, the concept behind the remote procedure call is intended to harmonize processing levels: ideally, for programmers and users, it should not make any difference which procedure call is involved. In other words: remote calls should be as easy to implement as local ones (transparency principle), at least in theory. In client-server networks and architectures, RPC calls represent a bidirectional, request-oriented communication process and complement purely message-based communication, which follows the input-output paradigm (use of I/O functions).

What’s Protocol Buffers?

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.

What’s gRPC?

gRPC is a framework that uses RPC to communicate. RPC is not Protobuf but instead Protobuf can use RPC and gRPC is actually Protobuf over RPC.

Proto file definition

First of all, we have to implement a service that let us create a communication between nodes and that is done in the eager.proto file. The service is called Eager and it contains a SendMessage function and a Finish function. The first one, is the main function between both since we will use it to send all the messages between the nodes that compose the ring topology, instead the Finish function is only used once by the leader to finish the program and to count the total number of messages sent. Both functions return an empty message since we don’t use the return values for the gRPC service.

The messages sent by the **Send** function have the following parameters:

- Sender – saves the node value from the sender
- Receiver – saves the node value from the receiver

The messages sent by the **End** function have the following parameters:

- Leader – returns the node value from the leader

This .proto file allows us to automatically generate the **eager_pb2.py** and **eager_pb2_grpc.py** files with the command:

```
python -m grpc_tools.protoc --proto_path=. ./eager.proto --python_out=. --grpc_python_out=.
```

Results obtained