INFO8002: - Project Report

Sebati Ilias - `s181506`
Miss Gómez Herrera María Andrea Liliana - `s198387`

December 18, 2020

1 Introduction

This project aims to make us implement a consensus between several flight computers. We have to handle several computers which may be deficient. Hence, they may crash.

As a precision, we did this project in a group of two students even after contacting all the students that Mr. Louppe sends us. All of them had already found a group.

We decided to use the `without-ksp` program.

2 Tutorial

The code is composed of four files.

- `without-ksp.py` is the main file for creating the process. But before running this file you first need to create the **n** computers.
- `computers.py` is the file containing the class `FlightComputer`. This file should not be used (i.e: do not run it) .
- `apiLaunched.py` is the file containing the functions to send requests to the different computers.
- `computer.py` is the file representing a computer you can run it with the following command (you need first to go in the `starter-code` directory):

  ```
  $ python3 computer.py fligh_computer_number
  ```

  fligh_computer_number is the number of the flight computer starting from 1 to n. I disabled the server (i.e: computer) to print on the terminal for an efficient purpose. However, you can comment the lines 10, 11, and 12 enables it again.

For instance:
If you want to test the `without-ksp.py` program with 3 computers (this is highly recommended to not go higher). You will create three terminals. From the first terminal to the last one you will do :
First terminal:

```
$ python3 computer.py 1
```

Second:

```
$ python3 computer.py 2
```

...

After creating all your computers you can start the `without-ksp.py` file with the same command as the one you gave us (from another terminal, in the main directory).

```
$ python3 starter-code/without-ksp.py --flight-computers n --correct-fraction fracti
```

3 Algorithm

To fulfill this project, we first decided to make exactly the same code but in a network way. So each computer is a process (i.e: a server) and we send requests using the `requests` API. Each computer is using the REST API `flask`.

After making this part, we tested our code with 3 computers. However, the program runs for 1 hour and then succeeded. My computer is not this powerful. The timestep in the main loop of the `without-ksp.py` needs to go from 0 to 84573. And for each timestep we need to perform requests and other things. Knowing that the program that you gave us with 3 computers run in around 10 seconds we could not expect less with several servers on the same machine.

After seeing that this works, we decided to implement a true leader elector. At each epoch we have a new leader. At the end, all computers will be leaders the same number of time (it depends if the number of epochs is divided by the number of computers).

To be fail tolerant, we decided to use a `TIME_OUT` value. We fixed it at 5 seconds such as the process does not take too much time. We know that you chose a number between 0 and 10 seconds to make the slow computers wait. However, waiting 10 seconds will make the overall process too slow, if a process is slower than 5 seconds we will make the **assumption** that it failed.

Furthermore, we can make the **assumption** that a computer cannot fail when the program just started. Thus, we can always put all the peers in all computers and init the computers.

4 Discussion

The convergence rate of our consensus algorithm is fast. If a process fails, we just elect a new leader and the next leader will take propose an action. However, if the processes are all slow, the program will take a lot of time before reaching consensus.