**Relatório 1 fase Projeto TFD**

Grupo 66

Henrique Vale 58168

Gustavo Henriques 64361

Leonardo Monteiro 58250

**Introdução**

Esta fase do projeto tinha como objetivo desenvolver um algoritmo Streamlet de consenso com tolerância a faltas de crash.

Para implementarmos isto decidimos usar python. Primeiro criamos ficheiros que representam os diferentes tipos de objetos que vamos usar neste projeto: Block, Message, Transaction e o BlockchainNetworkNode. De seguida encontra se uma pequena descrição de cada estrutura de dados e os seus atributos:

**Block:**

**-** Tem o hash do bloco anterior

- o seu próprio hash

- A época

- A sua posição na blockchain do nó

- Transações

Estes foram os atributos que achamos relevantes. Esta classe vai representar um bloco que vai ser votado, adicionado a listas, ser motorizado etc.

**Message:**

- Tipo de mensagem

- Conteudo da mensagem

- Quem envia

- A sequencia maior vista por um nó

Esta classe representa as mensagens que vão ser mandadas pelo nodes que podem ser do tipo Propose, Vote e Echo.

**Transaction:**

- Quem recebe

-Quem envia

- Id

- Quantia

Classe que vai representar uma transação que vai ser adicionada ao bloco.

**BlockchainNetworkNode:**

- Lista de blocos

- Lista de blocos motorizados

- Lista de blocos finalizados

- Se é líder

- O número de votos

- Lista de mensagens recebidas e enviadas

Esta classe tem mais atributos, mas estes são os mais relevantes. Esta classe é a mais importante pois representa um nó que vai votar e propor blocos e vai comunicar com os outros blocos para chegar a um consenso.

**Implementação**

Agora que já fizemos uma descrição de todas classes podemos falar melhor da nossa implementação. Criamos um ficheiro main que vai ser o ficheiro que vai controlar a execução do programa recebendo o input do utilizador. Esta main vai perguntar ao utilizador um número de nodes e vai gerá-los cada um no seu terminal, também vai obter o número de épocas e o delta que o utilizador pretende. A main vai comunicar com os nodes através de sockets. A main manda certas letras que são recebidas no node, processadas para executar certas funções no node ex: Se quisermos que um node gere uma transação a main manda um “t” para o node que quando é recebido por esse node que ira executar o generate\_random\_transaction().

A screen shot of a computer program

Description automatically generated

Figura 1 Menu do node que é utilizado para a Main comunicar com o node

Depois de obter estas informações vai iniciar uma época. Ao iniciar-mos a época vamos perguntar ao utilizador quantas transações ele quer que sejam geradas e adicionadas ao bloco que vamos dar propose. Depois vemos quem vai ser o líder.

A black background with white text

Description automatically generated

Figura 2 Inicio de uma época onde vemos o número de transações e geramos o líder

A black screen with white text

Description automatically generated

Figura 3 Função para gerar líder de maneira aleatória

Depois disto comunicamos com nó que foi escolhido e mandamos-lhe um “l” que se formos ver na figura 1 vai colocar esse node como líder.



Figura 4 Comunicar com o líder na Main

Agora que já temos o líder gerado e o node com o valor de leader atualizado vamos gerar as transações o número de vezes que o utilizador escolheu e vamos iniciar uma época que vai ter a duração de delta.

A screen shot of a computer code

Description automatically generated

Ao iniciarmos mandarmos ‘s’ para um node ele vai fazer o advance\_epoch().

A screen shot of a computer code

Description automatically generated

Este método vai chamar o propose\_block que vai criar um block novo com as transactions geradas e vai enviar uma mensagem para os outros nós.O método que trata das mensagens tem ira receber esta mensagem de propose e irá seguir o protocolo URB fazendo echo dessa mesma mensagem apenas uma vez, e ira votar nesse bloco se assim fizer sentido. A screen shot of a computer program

Description automatically generated

Figura 5 Método para processar mensagens

Para evitarmos vários echos ou executar a mesma mensagem mais que uma vez implementamos uma message\_queue onde guardamos os ids únicos de cada mensagem e se recebermos ou enviarmos uma mensagem adicionamos a mesma a essa queue.

Como vemos na figura 5 esse node vai mandar um voto para os outros nós ao fazer esta função onde incrementamos também o número de votos desse node. A computer screen shot of a program

Description automatically generated

Figura 6 Função que faz o vote

Agora depois de fazer Broadcast ao nosso voto os nos iram receber uma mensagem voto e como podemos ver na figura 5 ira fazer echo dessa mensagem e vai tentar notorizar.

A computer screen with text

Description automatically generated

Figura 7 Função que faz echo

A computer screen shot of a black screen

Description automatically generated

Figura 8 Função que notoriza

Na nossa implementação para fazermos o finalize criamos uma thread que está sempre a correr a função que faz essa finalização.



Figura 9 thread que faz finalize

Na nossa implementação os nodes comunicam por sockets e a conexão e feita no início onde adicionamos todos os nodes a lista de nós ativos. A computer code with numbers and symbols

Description automatically generated

Figura 10 Adicionar nodes

A screen shot of a computer code

Description automatically generated

Figura 11 Método de broadcast

A screen shot of a computer program

Description automatically generated

Figura 12 Método onde nos ligamos aos nós e mandamos a mensagem devidamente serializada.

Na nossa implementação usamos o pickle para serializar as mensagens e temos uma thread que esta constantemente a ver as mensagens que cria uma thread para processar cada mensagem, isto permite processar múltiplas mensagens ao mesmo tempo.



Figura 13 Thread que inicia o server

A computer screen shot of a program code

Description automatically generated

Figura 14 função trata de criar threads tratam de receber as mensagens

Criamos também uma thread para tratar do finalize.



Figura 15 Thread do finalize

A computer code on a black background

Description automatically generated

Figura 16 Função Finalize

Após o final de uma época cada nó da reset a certos atributos para poder avançar para a próxima época sem problemas e antes de começar uma época nova o utilizador terá a oportunidade de ver o conteúdo de certos atributos de cada nó, tendo um menu para tal.

A computer screen shot of a black screen

Description automatically generated

Figura 17 Função para dar reset

A screen shot of a computer program

Description automatically generated

Figura 18 Menu que nos permite ver atributos de certo nó ou continuar época