M2 Informatique - STL

TPEA - Rapport de Projet

Scrabblos

Étudiants :
Nassim Errahoui
Julien Xavier
Pascal Zheng

3 novembre 2019

Table des matières

1 Introduction 2
2 Tour par tour 2
2.1 Centralisé 2
2.2 Décentralisé 5
2.3 Analyse et critique 7
3 Réponse aux questions 7
4 Conclusion 8

1 Introduction

L'objectif de ce projet est d'implémenter un jeu de mot de type Scrabble sous forme d'une blockchain. Ce jeu comporte 2 types d'acteurs différents : les auteurs sont chargés d'injecter des lettres et les politiciens sont chargés de former des mots à partir de ces lettres.

Deux stratégies d'implémentations ont été utilisés. La première consiste à créer un système tour par tour. Une deuxième méthode suit une approche décentralisée permettant de dissocier la *Blockchain* de dite "roue libre", permettant aux acteurs de jouer simultanément.

Une partie se déroule en n tours, lorsqu'il y a n mots dans la blockchain, le programme s'arrête. Durant chaque tour, les auteurs retirent une lettre de leur sac de lettres pour l'injecter dans le pool de lettres. Ensuite, chacun des politiciens va récupérer le pool de lettres et le mélanger afin de créer un mot. Enfin, un consensus se lancera pour sélectionner le mot rapportant le plus de points et un bloc contenant ce mot sera créé dans la blockchain. A la fin de la partie, les auteurs et les politiciens auront des scores. Nous allons récupérer les auteurs et les politiciens qui ont participé à la construction de cette dernière. Le score des auteurs est calculé sur le nombre de lettres qu'ils ont pu injecter dans des mots choisis. Le score des politiciens est calculé sur le nombre de mots qu'ils ont générés et qui ont été choisis.

2 Tour par tour

2.1 Centralisé

Déroulement d'une partie

Pour lancer une partie, nous avons créé une classe *Launcher* qui contient une **HashMap** d'auteurs et une **HashMap** de politiciens. Chaque auteur (respectivement politicien) est associé un **Thread**. Pour que les politiciens puissent connaître les mots autorisés, nous chargeons des dictionnaires sous forme de fichiers texte dans un arbre **Patricia Trie** afin d'accélérer la recherche de mot par les politiciens.

Il existe également un objet *Blockchain* qui est partagé par tous les auteurs et les politiciens suivant un design-pattern **Singleton**. Cet objet, comme son nom l'indique, représente notre blockchain mais elle contient aussi d'autres attributs :

- un verrou utilisé pour gérer la concurrence entre les différents acteurs (**ReentrantLock**)
- deux conditions (**Condition**) sur le verrou décrit ci-dessus pour séparer les tours des auteurs et des politiciens
- une liste thread-safe représentant le letterpool durant ce tour (**Vector**)
- une liste thread-safe représentant le wordpool durant ce tour (**Vector**)
- une liste thread-safe représentant les différents blocks déjà construits ; elle correspond à la blockchain (**Vector**)
- une liste contenant le poids de chaque lettre (**HashMap<Character, Integer>**)

Avant de débuter le premier tour, nous allons lancer les **Threads** des auteurs et des politiciens. Cela va permettre aux auteurs de générer leur sac de lettres et aux politiciens d'obtenir l'arbre **Patricia Trie** des mots autorisés.

Lors de chaque tour, il existe deux phases : une phase auteur puis une phase politicien. Lorsqu'on est dans une phase auteur ce sont les auteurs qui jouent pendant que les politiciens sont en attente et inversement lors de la phase politicien.

Phase auteur

Au début de cette phase, chacun des auteurs va mélanger son sac de lettres. Ensuite, ils vont tenter de récupérer le verrou pour injecter une lettre au pool de lettre de la blockchain. Seulement un auteur pourra récupérer ce verrou, effectuer ses actions et lâcher le verrou, qui sera récupéré par un autre auteur. Les verrous de Java (**Lock**) nous permettent dans ce cas de gérer les accès concurrents à la blockchain par les auteurs afin de s'assurer que tous les auteurs ajoutent une et seulement une seule lettre et qu'ils se mettent en attente.

Pour implémenter une lettre, notre choix s'est porté sur la création d'une classe *Lettre*. Cet objet lettre contient un **Character** représentant la lettre, le hash du dernier mot de la blockchain ainsi que le hash de l'identifiant de l'auteur qui l'a produit.

En complément, nous utilisons un compteur stockant le nombre de lettres injectés dans le pool de lettres la blockchain. Ce compteur est un indicateur pour l'auteur qui jouera en dernier pour lui signaler par sa valeur que le nombre d'injections a été atteint et qu'il doit réveiller les politiciens à l'aide des **Condition** Java afin de passer à la deuxième phase.

Phase politicien

Dans la deuxième phase, les politiciens vont également récupérer chacun leur tour le verrou pour injecter un mot dans la blockchain. Lorsqu'un politicien obtient un verrou, il va créer une copie du pool de lettres et le mélanger.

Ensuite, pour chaque lettre, ce politicien va chercher si cette lettre correspond à un élément du noeud de l'arbre **Patricia Trie**. Si c'est le cas, on considère qu'on a trouvé un mot et on descend dans les fils pour voir s'il existe un mot plus grand avec la lettre suivante. Sinon, on renvoie le mot courant ou la chaîne vide si nous sommes à la racine.

A partir de là, on enlève les lettres qui n'ont pas été utilisés dans notre mot et on construit un objet *Mot* qui contient la liste des lettres de ce mot, le hash du dernier mot de la blockchain et le hash de l'identifiant du politicien.

Enfin, le politicien va ajouter le mot à la liste partagé de mots de la blockchain si et seulement si la taille de se mot est supérieur ou égale à la difficulté. Puis, il se mettra en attente. Comme pour les auteurs, le dernier politicien a un rôle spécial : lorsqu'il a terminé l'ajout de son mot, il lance l'algorithme de consensus.

Consensus

Pour permettre l'élection d'un nouveau bloc, notre algorithme se base sur la valeur de chaque lettre défini dans la classe *Blockchain* et identifie 3 cas. Dans les deux premiers cas, on parcourt la liste de mots, on renvoie le mot ayant le plus haut score afin d'associer ce mot au nouveau bloc. Cependant, on augmente la difficulté de 2 si plus de 10% des politiciens ont trouvés un mot sinon on diminue la difficulté de 2. Dans le troisième cas, personne n'a trouvé de mots alors on ne créé pas de bloc.

Puis, si la partie n'est pas terminée on réveille les auteurs en attente sur leur **Condition** et on recommence à partir de la phase acteurs. Sinon, le thread principal reprend la main et affiche le score de chacun des acteurs et politiciens.

— {'k','w','x','y','z'} valent 10 points.
— {'j','q'} valent 8 points.
— {'f','h','v'} valent 4 points.
— {'b','c','p'} valent 3 points.
— {'d','g','m'} valent 2 points.
— {'a','e','i','l','n','o','r','s','t','u'} valent 1 points.

```
public void Consensus() {
    Mot max = words.get(0);
    int score = getScore(max);
    for (int i = 1; i < words.size(); i++) {
        Mot courant = words.get(i);
        int scoreCourant = getScore(courant);
        if (score < scoreCourant) {
            max = courant;
            score = scoreCourant;
        }
    }
    blockchain.add(new Block(max));
}
```

FIGURE 1 – algorithme de consensus

2.2 Décentralisé

Selon le même principe que la partie centralisée, la partie se déroule en tour par tour, cependant, dans l'implémentation ci-dessus, nous avions un objet *Blockchain* qui était une instance partagée entre chacun des auteurs et politiciens qui contenait les mots et lettres injectés par tour ainsi que la blockchain courante. Cette fois-ci, nous n'avons que les mots et les lettres de chaque tour partagés, mais la blockchain est désormais unique à tous les auteurs/politiciens et chacun doivent alors à la fin d'un tour d'appliquer un consensus afin de mettre à jour sa blockchain.

Déroulement d'un tour

De la même manière qu'en centralisé, nos auteurs et nos politiciens appliquent leurs actions, cependant nous avons désormais deux phases de plus :

1. Chaque auteur applique l'algorithme de consensus sur sa blockchain, le dernier réveille les politiciens.
2. Chaque politicien applique l'algorithme de consensus sur sa blockchain, le dernier réveille les auteurs pour le prochain tour.

Ajouts sur la phase auteur

Le tour de l'auteur se déroule de la même manière qu'auparavant mais on ajoute un morceau de code permettant l'application du consensus. En effet à la fin du tour de l'auteur, nous avons ajouter le code suivant :

```
//block d'update
bc.getLock().lock();
try {
    bc.getAuteurCondition().await();
    if(bc.getWords().size() != 0) {
        bc.Consensus();
    }
    cptUpdate++;
    if (bc.getNbAuteur() == cptUpdate) {
        cptUpdate = 0;
        bc.getPoliticienCondition().signalAll();
    }
    bc.getAuteurCondition().await();
```

FIGURE 2 – ajouts à Auteur.java

Ici nous faisons en sorte que l'auteur se bloque non plus qu'une seule fois à la fin du tour afin de gérer le passage au tour suivant, mais deux fois afin de faire dans un premier temps le consensus à la fin du tour d'injection des politiciens, et une seconde fois afin de faire le passage au prochain tour lorsque les politiciens auront fini leur consensus. Qui plus est, nous avons un compteur ***cptUpdate*** qui nous permet de vérifier que seulement le dernier auteur réveille les politiciens à la fin de leur consensus.

Ajouts sur la phase politicien

Le tour du politicien se déroule aussi de la même manière qu'auparavant mais on ajoute un morceau de code permettant l'application du consensus. Nous avons ajouté le code suivant :

```
//block d'update
bc.getLock().lock();
bc.getPoliticienCondition().await();
if(bc.getWords().size() != 0) {
    bc.Consensus();
}
cptUpdate++;
if(bc.getNbPoliticien() == cptUpdate) {
    if(bc.getBlockchain().size() <= nbTours+1) {
        bc.getLetters().clear();
        if(bc.getWords().size() > (int)bc.getNbPoliticien()*0.1) {
            bc.setDifficulte(bc.getDifficulte()+2);
            System.out.println("On augmente la difficulté à "+bc.getDifficulte());
        }else {
            bc.setDifficulte(bc.getDifficulte()-1);
            System.out.println("On baisse la difficulté à "+bc.getDifficulte());
        }
        bc.getWords().clear();
        String word = bc.getBlockchain().lastElement().getMot().get_full_word();
        System.out.println("Le mot "+word+" à été choisi.");
        bc.getAuteurCondition().signalAll();
    }
    cptUpdate = 0;
}
```

FIGURE 3 – ajouts à Politicien.java

Ici nous faisons en sorte que le politicien se bloque à la fin de son injection de mot plutôt que de faire appel au consensus de l'objet partagé blockchain qui était défini auparavant dans la partie centralisée. Désormais, nous appliquons le consensus à tous les politiciens et le dernier (aussi repéré par ***cptUpdate***) vide la liste de lettres et de mots du tour courant et affiche le mot qui a été choisi, et réveille les auteurs pour passer au tour suivant.

2.3 Analyse et critique

Lors du cours, nous avons vu qu'il existait plusieurs types d'algorithme de consensus : proof of work, proof of stake, etc.
Proof of work étant une méthode de validation par recherche d'un *Nonce* permettant de retrouver le hash correspondant et permettant de valider un block et proof of stake par *mining*.
Proof of stake lui est une méthode de validation qui se base sur le fait qu'un acteur peut prouver la possession d'une certaine quantité de crypto-monnaie (leur « participation » dans la crypto-monnaie) pour prétendre à pouvoir valider des blocs supplémentaires dans la chaîne de bloc et de pouvoir toucher la récompense.
Sur ce projet, nous avons décidé de partir sur une approche plus du style proof of work. En effet, nous avons une liste de politiciens qui envoient chacun un mot si ils sont de taille supérieur à la difficulté. Ainsi, dans l'algorithme de consensus, on n'as aucune chance que l'on ajoute un mot dont la taille

Afin de faire un consensus à partir d'un proof of stake, nous avons deux axes imaginables :
- Dans un premier temps, si les auteurs augmentent leur score car leurs lettres ont étés choisies, alors on peut facilement leur permettre de poster plus d'une lettre par tour , par exemple de permettre une lettre supplémentaire tous les 10 points supplémentaire, idem pour les Politiciens avec leurs mots.
- Dans un second temps, on peut faire un système de monnaie à la place du score, avec une roulette de sélection aléatoire, et plus le politicien met d'argent en jeu, plus le politicien prend de la place dans la roulette (par exemple sur une roulette à 100 Scrabble-coin, alors si un politicien met 51 Scrabble-coin en jeu, il à alors 51% de chance de se faire élire).

3 Réponse aux questions

- **Comment s'assurer que l'auteur n'injecte pas plusieurs lettres pour un bloc ?**
On est dans un système de tour par tour, où l'auteur ne peut injecter qu'une seule lettre puisqu'il se bloque sur un verrou après l'injection jusqu'au prochain tour.

- **Comment obtenez-vous un consensus entre les acteurs du jeu ?**
Nous obtenons un consensus à l'aide d'un **Singleton** représentant la blockchain partagé par tous les acteurs de la partie ; c'est ce singleton qui va décider sur quel block chaque auteurs, et politiciens vont injecter des lettres/mots .

- **Est-ce qu'un attaquant peut faire changer le consensus en sa faveur ?**
Un attaquant pourrait en effet faire changer le consensus en sa faveur : étant donné que nous n'avons pas de vérification faite sur les mots envoyés par les politiciens, un acteur malveillant peut par conséquent envoyer des mots absents du **Patricia Trie** disponible à tous les autres acteurs. Étant donné qu'on ne sécurise pas non plus les clefs, il est tout à fait possible pour un acteur d'envoyer un message à la **Blockchain** en signant avec l'id d'un autre acteur et ainsi, le bloquer pendant le tour courant. Si ce comportement est poussé à l'extrême, un attaquant pourrait bloquer tous les acteurs autre que lui pour le tour courant et ainsi être le seul à envoyer un message valide.

4 Conclusion

Pour conclure, notre blockchain suit un système "tour par tour" permettant à chacun des acteurs de pouvoir prendre part à la construction de la blockchain. Cela signifie, que l'on peut contrôler les différents acteurs. Nous avons un réussi à avoir un début de peer-to-peer avec notre approche décentralisée. Il nous manque pour cela la création d'un réseau entre les acteurs, la propagation des informations entre les voisins sans qu'on ai une base de donnée commune a tout le monde via une instance de la *Blockchain*. Nous avons pu comprendre les différences entre des systèmes de validation proof of work et proof of stake. Il en existe plusieurs autres, telles que proof of authority, proof of hold, etc. Cependant, nous n'avons pas eu le temps d'explorer la possibilité de faire un système en "roue libre", faute de temps, ce qui nous empêche d'avoir une approche "temps réel" comme pourrait le faire les différentes blockchain de nos jours.