TPEA
RAPPORT DU PROJET BLOCKCHAIN

3 novembre 2019

SRENG David
SENG Kelly
GERDAY Nathan

Table des matières

Introduction

Dans le cadre de ce projet de TPEA pour implanter un jeu de mot sous forme d'une blockchain, nous avons choisi d'utiliser le langage Python 3 sur lequel nous avions déjà tous les 3 des connaissances et qui possédait des librairies qui correspondaient bien à nos nécessités pour ce projet.

Toutes les instructions pour lancer le projet se trouvent dans le fichier *README.md*. Au niveau de notre avancée, nous avons utilisé le serveur central fourni pour gérer les communications entre les clients. Nous avons donc bien les auteurs qui s'enregistrent auprès du serveur, reçoivent le sac de lettres et injectent une lettre. Nous avons également des politiciens qui forment des mots à partir des lettres injectées et les injectent eux-même.Toutes les signatures ainsi que les vérifications du serveur de ces signatures fonctionnent.

Enfin, nous avons également mis en place un algorithme de consensus entre les clients afin de déterminer la tête de la chaîne que nous détaillerons lors des explications sur nos différents choix dans le projet.

Tout d'abord, nous allons commencer par décrire rapidement l'architecture de notre projet dans la partie suivante afin de comprendre l'intérêt de chaque fichier et où chercher des éléments intéressants à observer.

Architecture du projet

Dictionnaire de mots

Fichier : dictionary.py

Pour manipuler les mots (ajout d'un mot, existence d'un mot, chargement d'un fichier texte), on a décidé d'utiliser un arbre lexicographique afin d'optimiser la manipulation de ce dictionnaire. Le type est défini de la façon suivante montré ci-dessous. Pour des soucis de clarté, on utilise du pseudo code (en Python, ce serait le type <dict>)

```
type arbre = {"is_word" : bool; "next" : map<string,arbre>}
```

Le champ "is_word" indique si le parcours de la racine jusqu'à ce noeud forme un mot valide. Le champ "next" est un map ayant pour clefs les lettres suivantes possibles pour le préfixe courant et pour valeurs leur sous-arbre lexicographique respectif. La classe Dictionary possède alors des fonctions pour ajouter des mots, lire dans un fichier et tester si un mot existe bien

Trletterpool

Fichier : letterpool.py

Lorsqu'un politicien demande le pool de lettres complet, ce pool est transformé en une structure que l'on a nommé "trletterpool". Elle est de la forme suivante :

```
type trletterpool = map<string,
                        {"authors" : set<string>;"letters" : letter list}>
```

Il s'agit d'un map (dict en python) dont les clefs sont les hash des blocs et dont les valeurs sont une structure ayant le champ "authors" qui liste les auteurs qui ont déjà injecté une lettre sur ce bloc, et le champ "letter" qui est une liste de lettres qui ont la structure originelle (letter, head, author, signature).

Chaque politicien garderont le pool de lettres sous cette forme, elle est utile lorsqu'il faut construire un bloc au dessus d'un autre (il suffit d'avoir le hash de ce dernier pour avoir les lettres qui ont été injectées). De plus trletterpool empêche un auteur d'injecter deux lettres sur un même bloc (bien sûr il pourra le faire mais seulement la première lettre sera gardé dans trletterpool)

Blockchain

Fichier : blockchain.py

Lorsqu'un politicien ou un auteur demande le pool de blocs complets (ie. tous les mots injectés), ce pool est transformé en une structure que l'on a nommé "trwordpool". Elle est de la forme suivante :

```
type trletterpool = map<string,block>
# Il y a aussi une clef "heads" qui donne l'ensemble des heads (ie.
    derniers blocs) de la blockchain
# Il n'y a pas de forks (peu probable), il n'y a qu'un seul head
```

Il s'agit d'une map ayant pour clef hash d'un bloc et pour valeur le bloc correspondant avec la structure originelle (word, head, politician, signature). Pour le bloc genèse, la valeur renvoie None. trwordpool est en réalité la blockchain.

Chaque acteur (auteurs et politiciens) gardera en local la blockchain. Grâce à cette structure, on peut facilement partir d'un bloc et remonter la chaîne jusqu'au bloc genèse. Dans ce fichier, il existe des fonctions qui permettront de faire le consensus à la fin d'une partie.

Acteur

Fichier : acteur.py

Il s'agit de la classe parent de Auteur et Politicien. On y met toute la partie commune, à savoir :

- La connexion au serveur
- Le chargement du dictionnaire de mots
- le lancement d'un thread qui s'occupera d'écouter le serveur et de modifier l'état de l'acteur suivant les messages. Il s'agit de la classe Listener
- La création du trwordpool en début de partie
- La gestion de la fin de de la partie

Autres

Notre projet comporte d'autres fichiers comme crypto.py qui s'occupe de toute la cryptographie, serv_coms.py qui s'occupe d'envoyer les messages au serveur au bon format, les fichiers auteur.py et politicien.py qui définissent ces acteurs (détaillés plus loin) et des launchers pour exécuter l'intégralité de l'application.

La partie

Informations

On avait commencé par développer la partie en tour par tour. Cependant en avançant, on a rencontré quelques soucis à cause des tours (qu'on détaillera par la suite) donc on a finalement décidé de passer directement à la partie en roue libre.

La partie en roue libre fonctionne de façon simple : tant qu'il y a de nouveaux meilleurs blocs qui sont produits sur la blockchain, on continue. Il y a donc un système de timeout et un nombre limité d'essais qui sont implémentés pour que les acteurs décident s'il faut s'arrêter.

Le système de scoring est simple : le score d'un bloc est l'addition du score donné par son mot (en utilisant les points du scrabble) avec les scores de tous les blocs de la chaîne jusqu'au bloc genèse. Notre algorithme de consensus est donc simple, lorsque deux blocs sont proposés sur un même bloc, on compare leur score et on ne considérera seulement le meilleur (néanmoins ils seront tous les deux dans le trwordpool, le moins bon sera simplement ignoré)

À la fin de la partie, tous les acteurs ont tous la même blockchain (ie. trwordpool) en local étant donnée qu'elle est créée à partir des messages du serveur central. La structure trwordpool et les fonctions qui la manipulent permettent facilement de trouver le meilleur head de la blockchain, de récupèrer tous les blocs depuis ce head jusqu'au bloc genèse et de compter les points des acteurs de cette chaîne.

Auteur

Après s'être enregistré au serveur et avoir construit sa blockchain locale, un auteur commence sa boucle principale. À chaque itération, il commence par trouver le meilleur head dans sa blockchain. Si celui-ci est identique à celui trouvé à l'itération précédente, cela signifie qu'il n'y a pas de nouveaux meilleurs blocs qui ont été produit et l'auteur prend en compte ce manque d'activités. Sinon, il injecte une lettre prise aléatoirement dans son sac de lettres sur ce head. Dans les deux cas, l'auteur se met ensuite en attente d'un nouveau bloc pendant un certain timeout avant de passer à l'itération suivant.

S'il n'y a pas eu de nouveaux meilleurs blocs produits après un certain nombre d'itérations, l'auteur décide de s'arrêter.

Si son sac de lettres est vide (ie. il ne peut plus injecter de lettres), il attend qu'il n'y ait plus d'activités pour s'arrêter.

Il s'agit du thread écoutant le serveur qui s'occupe d'ajouter les nouveaux blocs dans la blockchain et de prévenir l'auteur.

Politicien

Après avoir construit sa blockchain locale et créé son trletterpool, le politicien commence sa boucle principale. À chaque itération, il commence par trouver le meilleur head dans sa blockchain et si celui-ci est identique à celui trouvé à l'itération précédente, le politicien prend en compte ce manque d'activités. Le politicien va ensuite chercher à former un bloc (ie. mot) valide à partir des lettres injectées sur celui-ci (qui sont facilement récupérables grâce au trletterpool).

Cette procédure de recherche d'un mot valide n'est malheureusement pas optimisée. On se contente de choisir des lettres au hasard jusqu'à avoir un mot valide. Elle va néanmoins prendre en compte les lettres qui continuent à être injectées pendant la recherche. Il y a aussi un système qui compte le nombre d'essais et qui va arrêter cette recherche si elle dépasse le nombre autorisé d'essais au cas où elle prendrait trop de temps et que d'autres blocs aient été injectés. Cependant cette limite a été fixée arbitraire, il faudrait trouver un moyen de le faire varier suivant la longueur des mots du dictionnaire.

Si la procédure trouve un mot valide, le politicien injecte le bloc qui correspond. À la fin de l'itération, il se met attente pendant un certain timeout (pour par exemple attendre des injections de lettres ou d'autres blocs) et passe à l'itération suivante.

Comme avec l'auteur, s'il n'y a pas eu de nouveaux meilleurs blocs produits après un certain nombre d'itérations (soit par d'autres politiciens, soit par lui-même pour manque de lettres par exemple), il décide de s'arrêter.

Son thread écoutant le serveur s'occupe d'ajouter les nouvelles lettres à son trletterpool et les nouveaux blocs qui proviennent d'autres politiciens à sa blockchain.

Questions

Question 0

Comment s'assurer que l'auteur n'injecte pas plusieurs lettres pour un bloc ?

Pour s'assurer que l'auteur n'injecte pas plusieurs lettres dans un bloc, nous avons fait en sorte que le pool de lettres récupéré par le politicien soit gardé sous la forme de la structure trletterpool comme expliqué précédemment. Lorsqu'un politicien reçoit deux lettres sur un même bloc provenant d'un même auteur, trletterpool ne va garder que la première des deux.

Cela n'empêche pas un auteur de le faire. Néanmoins, il va simplement gaspiller une lettre. C'est pour ça que nous avons établi un autre moyen de sécurité. A l'intérieur de sa boucle, l'auteur ne va pas injecter de lettres si le meilleur bloc trouvé est le même que celui à l'itération précédente.

Question 1

Implanter un Auteur et un Politicien pour la version tour à tour.

Comme expliqué précédemment, nous avons sauté la partie en tour par tour car nous avons rencontré quelque soucis et certaines aspects ne nous semblaient pas logiques.

Tout d'abord, nous avions un problème pour passer au tour suivant, puis une fois le bug réglé, nous avons eu des soucis au niveau de la synchronisation entre les auteurs et les politiciens. En effet, étant donné que le serveur attend que tous les auteurs aient injecté un mot, puis passe au tour suivant, il n'y a aucune attente au niveau des politiciens ce qui crée un décalage. De plus si un auteur est lancé un peu avant tous les autres, il peut s'enregistrer, injecter une lettre et faire passer le serveur au tour suivant tout seul.

Nous pensons donc qu'il était plus intéressant et aussi par soucis de temps, de passer directement à la partie en roue libre.

Question 2

Implanter un Auteur et un politicien pour la version "en roue libre" avec serveur central.

Déjà expliqué dans la section "La partie".

Question 3

Ce système s'appuie-t-il plutôt sur un algorithme de consensus à Proof-of-Work ou à Proof-of-Stake ? Comment modifieriez-vous ce système pour implanter un algorithme de consensus basé sur l'autre modèle ?

Il s'agit plutôt d'un système de consensus Proof-of-Work. Les politiciens regardent toujours le meilleur bloc afin de construire un mot par dessus. Le premier bloc proposé est souvent celui qui est gardé car les auteurs injecteront des lettres sur celui-ci étant donné qu'il aura le meilleur score. Cela signifie donc qu'il n'y aura plus de nouvelles lettres injectées dans le bloc précédent et donc les politiciens qui continuaient toujours de travailler dessus auront du mal à trouver mieux et de faire en sorte que cette branche rattrape le retard par rapport à l'autre.

Une idée pour un Proof-of-Stake serait de modifier le système de score de la façon suivante : le score d'un bloc correspond à l'addition du score du mot, du nombre de blocs que le politicien a proposé dans cette chaîne multiplié par un coefficient et du score des blocs précédents de la chaîne. De cette façon, le bloc proposé par un politicien ayant déjà beaucoup de blocs dans cette chaîne aura un meilleur score que le même bloc proposé par un politicien ayant moins de blocs dans la chaîne.

Question 4

Comment modifieriez-vous le système pour ne plus utiliser le serveur central. (en PoW ou Pos, au choix)

Il faudrait mettre en place un système de communications directes entre les clients en pair à pair. Pour cela, chaque client doit posséder une liste d'autres clients auxquels il est connecté. Lorsqu'une nouvelle personne s'enregistre, il peut lui envoyer une partie de la liste, puis ceux qui ont été envoyés peuvent également fournir une partie de la liste et ce jusqu'à ce que le nouveau client soit connecté à suffisamment d'autres clients pour ne pas être isolé.
Une fois tous les clients connectés ensemble, il suffit d'envoyer le message à chaque client de sa liste et ainsi de suite pour le diffuser à l'ensemble de tous les clients.

Une fois ce procédé de communication établi, le reste peut fonctionner de la même manière si on garde le système que nous avons implanté en roue libre étant donné que le serveur central ne servait dans ce cas vraiment plus qu'aux communications.
Il faudrait cependant faire attention à ce que chaque message passe par plusieurs noeuds à la fois, afin de vérifier qu'un attaquant ne peut pas décider de manipuler la blockchain et les différents messages pour s'avantager.

Améliorations possibles

Étant donné le soucis de temps, beaucoup de points restent à améliorer. On pourrait notamment programmer la partie en tour par tour pour qu'elle marche correctement. Actuellement il est possible de la lancer de cette façon mais il y a des problèmes de périodes lorsque l'on injecte des lettres et des mots et cela provoque l'arrêt des acteurs alors qu'il serait possible de continuer. Par exemple, le serveur centrale vérifie qu'un auteur n'injecte pas deux lettres pendant la même période même sur des blocs différents et donc la deuxième lettre n'est pas broadcastée. L'auteur pense avoir injecté la lettre au deuxième bloc donc il ne peut plus en injecter une autre sur ce bloc. Il manque alors une lettre au bloc qui aurait aider à construire un nouveau bloc.

Il y a également des améliorations à faire au niveau de la mémoire. Ce n'est pas très bien optimisé. Par exemple, exécuter 20 acteurs suffit pour entraîner une surconsommation de la mémoire RAM et éventuellement planter un ordinateur. En effet, les structures trwordpool et trletterpool prennent énormément de place, surtout au fil du temps. Il faudrait trouver un moyen de ne pas stocker les blocks en entier (word, head, politician) mais cela semble difficile puisqu'il faut les connaître afin de savoir les scores. Le meilleur moyen serait de stocker toutes les informations dans des bases de données.

L'algorithme de recherche d'un mot n'est pas efficace. Il cherche aléatoirement jusqu'à tomber sur un mot valide. Cela implique une surconsommation du processeur, qui est inutile, et prend beaucoup de temps à s'exécuter. Il faudrait implémenter un algorithme efficace qui permettrait de trouver un mot valide dans un arbre lexicographique à partir d'un ensemble de lettres disponibles.

Le système avec les timeouts et les nombres d'essais limités pour mettre en place la fin de partie n'est pas très propre et est individuel. Il faudrait une façon qui permettrait de dire à tout le monde de s'arrêter à un moment précis et que tout le monde soit d'accord avec la décision prise à cet instant.