Rapport du projet TPEA

Alpha DIALLO, Reda BERRADA, Sacha MEMMI

Décembre 2020

Table des matières

1. Préambules et problèmes recontrés

Nous avons rencontré plusieurs soucis qui ne nous ont pas permis d'obtenir un code qui fonctionne.

En effet, lorsque nous mettons en commentaire les fonctions de broadcasting, le serveur se comporte normalement même si les clients ne reçoivent plus rien de la part du serveur.

Cependant, dès lors qu'il y a broadcasting, pour que les tours passent, il faut obligatoirement connecter un nouveau client au serveur. De plus, même en passant les tours de cette manière, nous nous apercevons que certains clients ne répondent pas et qu'il y a des mots qui sont émis alors qu'ils ne font pas partie du level courant.

Nous pensons que lors des utilisations de la librairie Lwt dans le code serveur, il y avait des accès concurrents sur les descripteurs des sockets dans l'architecture du code fourni.
Ceci expliquerait pourquoi nous avons un serveur qui tressaute sans afficher de messages d'erreurs.

Voici le scénario qui nous a bloqué, qui montre également la manière de compiler le code :

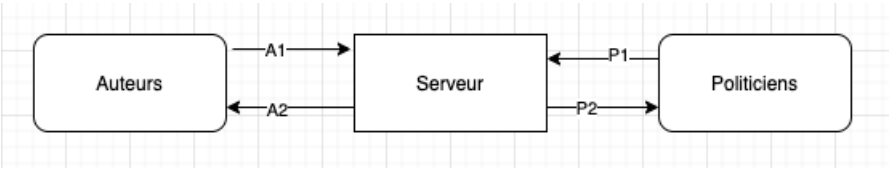
1. Dans un premier terminal, lancer le serveur avec la commande **"./_build/default/src/main_server.exe -timeout 4.0"**.
2. Dans un deuxième terminal, créer 5 auteurs en lançant 5 fois la commande **"./_build/default/src/author.** et 3 politiciens en lançant 3 fois la commande **"./_build/default/src/politicien.exe&"**.

On remarque que le serveur reste en attente de requêtes. Pour obtenir un passage au tour suivant, il faut rajouter un politicien ou un auteur. Le serveur oublie des paquets puis le processus s'arrête au bout de quelques ajouts sans afficher de messages d'erreurs.

2. Tour par tour

2.1 Principes de fonctionnement

2.1.1 Schéma de fonctionnement



Dans ce schéma, c'est le serveur central qui contrôle la transition des tours en utilisant un timer par exemple.
Le serveur envoie à chaque client les pools de mots et de lettres. Les auteurs reçoivent les mots de la pool afin qu'ils puissent ajouter un nouveau mot. Les politiciens quant à eux ont accès au pool de lettres afin de générer des mots à partir des lettres qui le contiennent et de soumettre ce mot au serveur. Les mots de la pool subiront un consensus pour savoir quel mot sera ajouté à la blockchain.

2.1.2 Consensus

On attribue à chaque lettre un score. Pour déterminer les scores d'un mot, nous sommons tous les scores des lettres constituant ce dernier.
Le mot choisi sera celui qui maximisera le score, et celui avec le meilleur score sera ajouté. Si les scores de deux mots sont identiques, nous comparons la signature des blocs afin de déterminer quel mot sera bel et bien ajouté à la blockchain.

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

Une fois la lettre reçue par le serveur, celui-ci vérifie si l'auteur n'avait pas envoyé une lettre lors du tour courant. Dans mempool.ml, il y a injection d'une lettre uniquement si l'auteur n'en a pas déjà soumis.

3. Roue libre

3.1 Modifications apportées

Pour le mode en roue libre, chaque client peut participer à n'importe quel moment. C'est pourquoi des modifications de l'algorithme de consensus doivent être apportées par rapport au mode tour par tour. Dans notre cas, nous avons décidé d'agir sur la méthode de synchronisation des clients sur une même horloge. Cette dernière se décrit comme suit :

1. Le premier client à entrer dans le réseau crée le temps de départ *time_dep* et un intervalle de temps *interval* pour chaque level.
2. Un client connecté demande le temps de départ ainsi que l'intervalle de temps au réseau, il reçoit un ou plusieurs tuples (time_dep, interval) en fonction du nombre de participants, puis prend le premier reçu et commence sa participation au jeu.

Dès que tout le monde a ces paramètres, chacun peut exécuter le consensus indépendamment et passer au tour suivant, en fonction des données broadcastées sur le réseau. La manière d'attribuer les scores quant à elle reste la même.

3.2 Résistance aux attaques

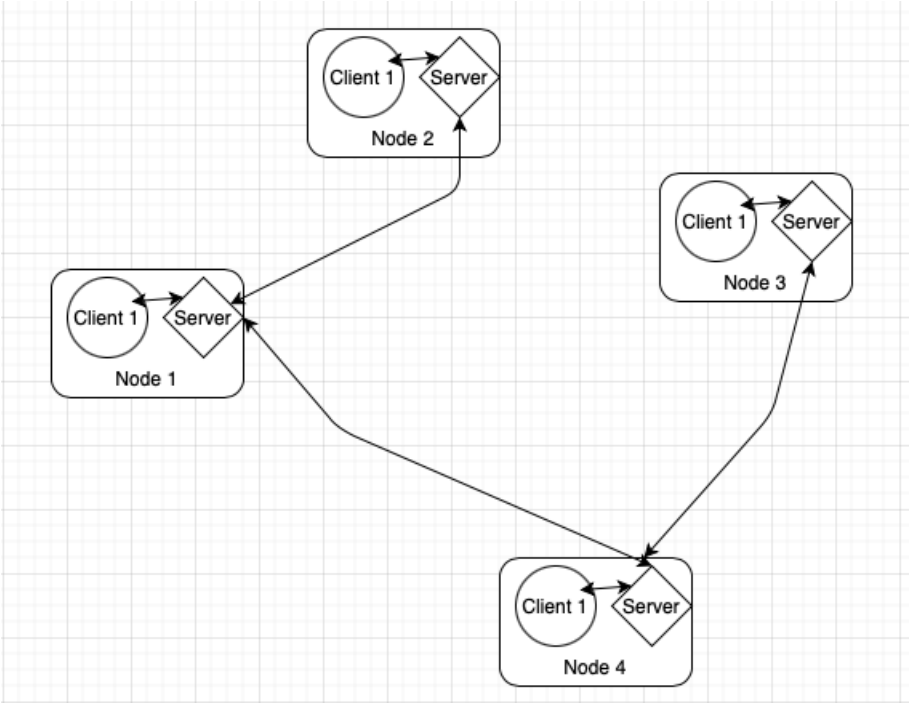
Une possibilité d'attaque serait qu'un des clients donne un faux temps. Par exemple, un client entre dans le réseau et, à la demande, reçoit un faux temps, ce qui biaiserait le bon déroulement du jeu. Pour se défendre d'une telle attaque, il faudrait que le client qui demande un temps reçoive le temps le plus fréquent. Cela dit, cette technique peut être contrée s'il y a entente entre une majorité de clients malhonnêtes.

4. Sans serveur central

4.1 Conception

Pour la conception sans serveur central, on réutilise ce qui a déjà été fait, on construit un noeud en utilisant le code du client et le serveur central. Il est possible dans un même processus d'utiliser deux sockets qui s'interconnectent et échangent des données. Par conséquent, ce qu'il faut modifier dans le serveur est l'interconnection de deux serveurs, c'est-à-dire un serveur en lancement peut se connecter à un autre. Ensuite, chaque message reçu par un noeud A, et provenant d'un noeud B, sera envoyé à tous les noeuds qui sont connectés au noeud A.

Il faut bien sûr que l'extension soit celle de la partie de la roue libre parce que le serveur dans la roue libre ne fait que broadcaster et le consensus se décide entre les clients (auteurs ou politiciens) une fois les horloges synchronisées.



4.2 Proof of Stack

4.2.1 Description

Le Proof of Stack se base sur la possession d'une certaine valeur. Dans notre cas, il s'agit des points récoltés depuis le début par un client (auteur ou politicien). Il s'agit donc de trouver une manière de choisir la tête de la blockchain en se basant sur le score.

4.2.2 Consensus

On considère les plus riches à un instant de level comme étant ceux qui ont accumulé plus de score que la moyenne et inversement pour les pauvres. Par exemple, on note le plus riche parmi les auteurs et le plus riche parmi les politiciens. Le choix du mot sera déterminé en deux points :

1. Pour un level pair, on choisit les cinq meilleurs mots soumis par les politiciens plus riches. Et parmi ces cinq mots, on choisit le mot de l'auteur dont la somme des scores depuis le début de la partie est le plus faible.
2. Pour un level impair, on choisit les cinq meilleurs mot parmi les politiciens plus pauvres. Et parmi ces cinq mots, on choisit le mot de l'auteur dont la somme des scores depuis le début de la partie est le plus faible.

En cas d'égalité, on les départage par la comparaison dès la signature du bloc (mot).

Pour la modification du code source, nous pensons que, comme la synchronisation des horloges est déjà gérée dans la roue libre, le consensus peut facilement être modifié sans toucher à l'architecture décentralisée décrite plus haut (**section 4.1**).

4.2.3 Résistance

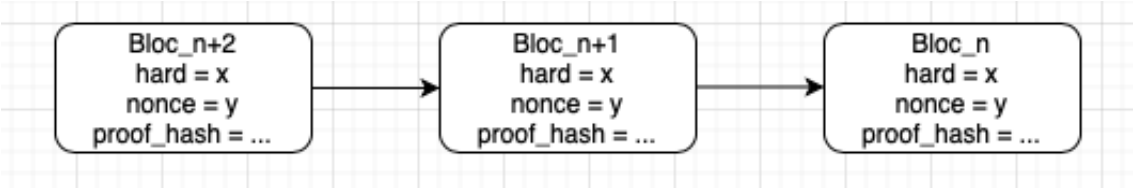
L'une des attaques possibles en plus de l'attaque sur l'horloge est d'isoler certains noeuds du réseau. Il s'agit d'une attaque classique des blockchains. Pour se prémunir de cette attaque, il faudrait que chaque client se connecte à plusieurs noeuds afin d'effectuer des vérifications (s'appuie sur la redondance).

4.3 Proof of Work

4.3.1 Description

Le Proof of Work se base sur la compétition de mineurs pour trouver le bon hash d'un bloc, et ces derniers s'attendent à une récompense en retour de ce travail.

4.3.2 Consensus



Ici, chaque fois qu'un politicien injecte un mot, il présente un hash dont la difficulté est dans le head de la blockchain et qui représentera sa "preuve de travail". Quant aux hash et la nonce, ils seront aussi intégrés dans un mot injecté pour que les autres puissent vérifier la validité du bloc. Si plusieurs clients ont pu accomplir la tâche, on sélectionne le mot en suivant le consensus du début.

Là aussi pour la modification du code, on partira sur la section 4.1 avec les horloges déjà synchronisées, puisque le modèle est déjà décentralisé.

4.3.3 Résistance

En plus de l'attaque classique d'isolement des noeuds sur les blockchains, un attaquant peut rajouter de grandes difficultés dans les blocs sur le modèle qu'on a décrit, et cela ralentirait ou bloquerait totalement le réseau. Pour ce prémunir, on peut demander une difficulté dans le réseau et prendre la moyenne.

Selon la difficulté du hashage le timeout, une fois depassé cela pose des problèmes sur l'exatitudes de quel level on trouve, dans ce cas on conceptiontualise les levels comme des cycles d'intervalle du timeout de départ.

Conclusion

Malgré les problèmes que nous avons pu rencontrer lors de l'implémentation du projet, nous avons pu élaborer une architecture de blockchains pouvant fonctionner à la fois avec un serveur central mais aussi de manière décentralisée (avec un acteur jouant le rôle du serveur).

Nous avons fait une ébauche des différentes possibilités d'attaque des structures que nous avons développées ainsi que les manières que nous pouvions employer afin de les déjouer.

Il serait intéressant de faire varier dynamiquement la difficulté de minage en fonction du nombre de politiciens afin que le serveur ne soit pas surchargé lorsqu'il y a beaucoup de clients connectés.