Projet Transverse Semestre 8
Parcours SISN

Le Hashgraph : une alternative à la Blockchain ?

Elèves : Muller Bastien, Cambay Vincent

Tuteur : Daucé Emmanuel

25 Mai 2018

Sommaire

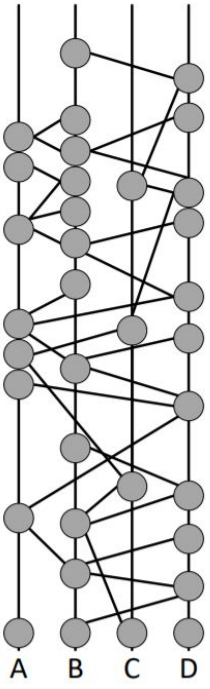
Introduction

Le Hashgraph est une technologie de gestion décentralisée de données. Cette technologie qui a été développée par la société Swirlds se présente comme une alternative efficace à la Blockchain. Les méthodes décentralisées traditionnelles reposent majoritairement sur le principe de preuve de travail ou bien d'enjeu. Le Hashgraph, quant à lui, s'appuie sur des principes différents.

Le but de ce projet est d'étudier et comprendre le fonctionnement d'une implémentation en python de cet algorithme proposée sur le site GitHub, ainsi que d'identifier les améliorations nécessaires pour faire de cette implémentation un système ouvert et distribué.

I) Etude préalable

a) Principe du Hashgraph



Le fonctionnement des Hashgraphs peut être décrit par le schéma suivant. Il y a ici 4 membres (Alice, Bob, Carol et Dave), le hashgraph évolue vers le haut avec le temps. Chaque noeud contient les informations relatives à l'utilisateur. L'objectif de l'algorithme est de permettre le transfert d'informations entre les noeuds (events, transactions…) de manière décentralisée. Il faut pour cela parvenir à un **consensus** entre les membres de la communauté : un accord sur l'état officiel du réseau.

Le principe des Hashgraphs utilise un **protocole de bavardage** (gossip protocol). Cela signifie que chaque noeud envoie régulièrement ses données à d'autres membres aléatoirement. Ainsi, au début, Bob choisit de communiquer avec Dave, ce qui est représenté par un event. Le noeud communique ce qu'il sait, mais aussi de qui il le sait, ce qui permet de retracer l'origine de l'information et ainsi former le hashgraph.

La validation d'une transaction se fait suivant un algorithme de **Virtual Voting** dont le mécanisme est expliqué en annexe 1.

b) Comparaison avec la blockchain

La blockchain possède une méthode de consensus différente : la preuve de travail.

Dans le cas d'une preuve de travail, pour qu'un bloc soit validé, il faut résoudre un problème complexe. Ainsi dans le cas du Bitcoin il faut tester au hasard un grand nombre de codes afin d'espérer obtenir un hash final commençant par un nombre prédéterminé de zéros. C'est ce qui permet de régler la difficulté du problème qui est fixé en fonction de la puissance de calcul des mineurs afin qu'un bloc soit validé en moyenne toutes les 10 minutes. Cette preuve de travail garantit la stabilisation et la non falsification de la chaîne. Cela nécessite cependant une consommation d'énergie. De plus, il se présente un problème d'équité. En effet, les mineurs peuvent favoriser l'ordre de validation des transactions.

La preuve d'enjeu, quant à elle demande à l'utilisateur de prouver la possession d'une certaine quantité de crypto-monnaie pour avoir le droit de valider des blocs supplémentaires. Cependant ce type de preuve, s'il résout le problème de gaspillage de l'énergie, offre moins de sécurité que la preuve de travail. Ainsi ce type de preuve est souvent associé à une preuve de travail (comme dans Peercoin par exemple)

Le hashgraph, lui, est plus équitable, ne possède pas le problème de la consommation d'énergie. Ainsi, à première vue le Hashgraph semble posséder des propriétés intéressantes en vue d'une alternative à la Blockchain. Cependant nous allons voir en dernière partie, après étude d'une implémentation, ce qui pose problème avec le principe de Hashgraph.

II) Une implémentation du principe du Hashgraph

a) Une implémentation fonctionnelle

Ce projet consiste donc en l'étude critique d'une implémentation du hashgraph proposée par un utilisateur de la plateforme GitHub. Le code source nécessite l'installation préalable de librairies en lien avec la cryptographie comme **pysodium** (et **libsodium**) ainsi que **bokeh** qui gère la visualisation interactive des données.

Nous avons rencontré quelques problèmes (au départ inexpliqué) lors de l'exécution du code. Avec l'aide de notre tuteur et sur conseil de l'utilisateur de GitHub à l'origine du programme, nous avons pu trouver leur origine : la librairie libsodium, incluse dans Linux était obsolète.

Le code est organisé en 3 fichiers :

- *swirld.py* qui contient le coeur du programme
- *utils.py* qui définis quelques fonctions additionnelles
- *viz.py* qui gère la visualisation des données en sortie grâce à la librairie bokeh

L'algorithme consiste en une boucle principale (infini) qui simule la création d'events et effectue les mécanismes de gossip et de virtual voting entre les noeuds, qui sont arrangés dans une classe Node

```
def main(self):
    """Main working loop."""

    new = ()
    while True:
        payload = (yield new)

        # pick a random node to sync with but not me
        c = tuple(self.network.keys() - {self.pk})[randrange(self.n - 1)]
        new = self.sync(c, payload)
        self.divide_rounds(new)

        new_c = self.decide_fame()
        self.find_order(new_c)
```

Dans la boucle, l'algorithme utilise :

- *sync(<remote-node-id>, <payload-to-embed>)* qui interroge les noeuds et met à jour les données locales
- *divide_rounds* qui définit les rounds et les témoins des nouvelles transactions
- *decide_fame* qui gère le vote
- *find_order* qui met à jour la liste des transactions en accord avec les résultats du vote

b) Mais incomplète

Cette implémentation fonctionne et permet de simuler le comportement d'un hashgraph composé du nombre de noeuds choisis (propagation d'events grâce au gossip et virtual voting).

Néanmoins, la manière dont fonctionne l'algorithme n'est pas entièrement fidèle au principe théorique du Hashgraph, et ce par plusieurs aspects :

- Les noeuds ne "communiquent" pas réellement entre eux car les events sont gérés de manière "passive". Les noeuds à l'origine d'un events ne le transmettent pas, mais attendent que d'autres noeuds les interrogent par la méthode *ask_sync*.
- Le programme est fait de manière à éviter les threads. La boucle principale passe par les noeuds un à un, alors que les noeuds devraient évoluer et interagir entre eux de manière indépendante.

III) Quelles évolutions ?

- Système distribué : système dont la gestion est traitée par un réseau d'ordinateurs interconnectés qui stockent des données de manière distribuée. Les ressources peuvent être réparties entre les membres du réseau ou bien possédées en totalité par chacun.
- Système ouvert : système configuré pour permettre des accès non restreints par des personnes et/ou des ordinateurs ⇒ n'importe qui peut s'ajouter au réseau, le nombre de noeuds n'est pas connu au préalable.

Certaines évolutions sont nécessaires pour permettre à ce programme de fonctionner en tant que système ouvert et distribué (à l'échelle d'internet et sans autorité centrale) :

- Remplacer la méthode *ask_sync* par un vrai protocole de communication entre les noeuds. Cela nécessite de gérer plusieurs threads d'exécution.

- Implémenter un protocole pour permettre l'adhésion libre au réseau par de nouveaux membres. C'est la condition pour pouvoir qualifier le système d'ouvert, mais ce n'est vraiment pas simple.

Ce deuxième point présente certains problèmes car, comme pour le code présenté dans ce dossier, le système développé par Swirlds n'est pas un système ouvert (aucune démonstration ouverte connue à ce jour).

Pourtant, Swirlds propose plusieurs pistes pour régler ce problème :

- Un système d'invitation où le nouveau membre reçoit une part de la preuve d'enjeu de celui qui invite. Problématique car le premier membre du réseau détient tout le pouvoir.
- Faire appel à des moyens extérieurs comme des portefeuilles de Bitcoin ou introduire une sorte de proof of work → On revient au système qu'on essaye de remplacer.

Au final, vouloir un système ouvert nécessite d'avoir un système "scalable" pour gérer l'arrivée sur le réseau d'un grand nombre de membres, mais on arrive à un problème concernant le "virtual voting" qui nécessite d'une manière ou d'une autre de faire des itérations sur tous les noeuds. Ainsi il ressort que l'algorithme requiert au moins une complexité en espace et en temps en O(n) avec n le nombre de noeuds. Cela rend très compliqué la possibilité d'une base de donnée distribuée de grande taille comme pour le bitcoin par exemple. Mais pour un réseau de petite taille cette complexité est moins problématique.

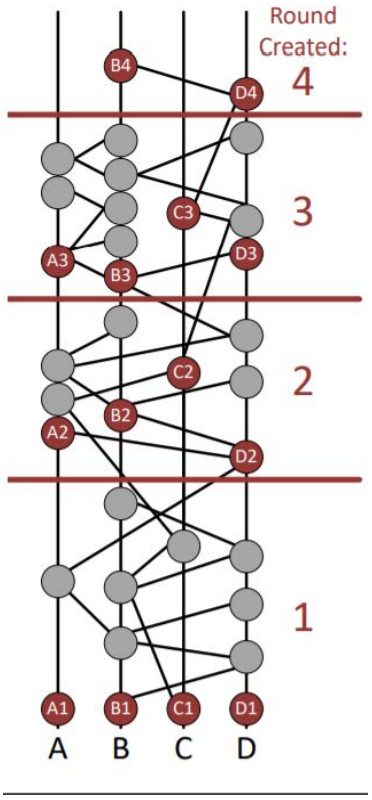
Conclusion

Il s'avère donc que la technologie hashgraph ne peut pas se positionner comme une alternative à la blockchain car il n'est pas prouvé qu'elle fonctionne dans le cadre d'un système ouvert et distribué à grande échelle. Le Hashgraph semble plus adapté pour des bases de données distribuées de taille moyenne avec une sorte de système d'authentification centralisé, comme pour des entreprises par exemple.

Ce projet nous a permis de mieux comprendre la complexité et les enjeux autour de la construction de systèmes ouvert et distribué. Enfin nous voulons remercier notre tuteur Emmanuel Daucé qui nous a aidé et conseillé.

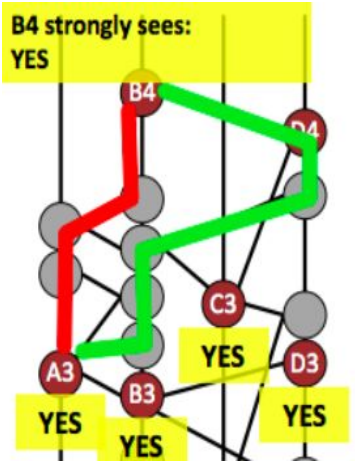
Annexe : précisions sur le virtual voting

Chaque évènement appartient à un round. Le premier évènement de chaque round pour chaque membre du réseau est qualifié de témoin. Il faut pour chaque témoin déterminer si il est “connu” ou pas.



Pour cela on regarde les témoins du round suivant. On regarde alors si les témoins de ce round descendent du témoin à “valider”. Par exemple si on cherche à valider ”B2”. On regarde A3,B3,C3 et D3. On se rend compte qu’ils ont tous voté positivement pour désigner B2 comme témoin “connu”.

Mais cela ne suffit pas. Il faut décider quels sont les voix que l’on prendra en compte pour le scrutin. Pour cela il faut regarder un témoin quelconque du deuxième round qui suit. En reprenant l’exemple de B2, on choisit de regarder B4 (on aurait pu essayer aussi avec D4). Pour qu’un vote soit pris en compte, il faut qu’il soit “fortement vu” par ce témoin. Pour cela il faut qu’il existe un chemin entre ce témoin (B4) et le témoin votant (A3 par exemple) qui passe par plus de ⅔ de la population totale, soit 3 ou 4 membres dans notre exemple. Le chemin vert passe par 3 membres, le vote de A3 est donc pris en compte.



Enfin, il faut compter les votes. Pour qu’un témoin soit qualifié de connu ou non, il faut déjà qu’il y ait une majorité qualifiée (un nombre supérieur à ⅔ de la population totale) de oui ou de non. Ici B2 est bien un témoin connu car on a 3 votes positifs pris en compte (C3 n’est ici pas pris en compte).

Si B4 était incapable de décider, il faudrait ré-essayer avec D4. Et si on ne trouve toujours pas de majorité qualifiée, il existe une solution technique qui permet de trancher.

Cette notion de témoin connu est importante. En effet pour qu’une transaction soit validée il faut qu’elle appartienne à un noeud ancêtre de chaque témoin connu d’un round de rang égal ou supérieur au sien.