Valorisation de la DHT d'Ethereum pour réduire les besoins de stockage des pairs

Jean-Philippe Eisenbarth, Thibault Cholez, Olivier Perrin

Valorisation de la DHT d'Ethereum pour réduire les besoins de stockage des pairs

Jean-Philippe Eisenbarth[1] et Thibault Cholez[2] et Olivier Perrin[2]

[1]*SnT, University of Luxembourg, L-1855 Luxembourg, Luxembourg*
[2]*Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France*

Les blockchains font face à de multiples défis parmi lesquels l'augmentation continuelle des données à stocker par les pairs qui en supportent l'infrastructure. En particulier, Ethereum se rapproche de la barrière symbolique de 1 To de données à stocker par fullnode. Dans cet article, nous proposons une nouvelle stratégie de stockage basée sur l'exploitation de la table de hachage distribuée d'Ethereum. Nous avons implanté une preuve de concept dans Geth, et évalué que l'économie possible est d'environ 60% du stockage actuel demandé à un pair. Cela représente un gain de l'ordre d'une dizaine de pétaoctets pour le réseau pair-à-pair dans son ensemble. De plus, notre solution est compatible avec les clients actuels et peut-être déployée incrémentalement.

Mots-clefs : Blockchain, Ethereum, stockage, table de hachage distribuée, synchronisation, Geth

1 Introduction

Il est de notoriété publique que les premières blockchains sont des systèmes distribués demandant beaucoup de ressources, notamment à cause de la preuve de travail. En septembre 2022, Ethereum a corrigé cet aspect et réduit radicalement la consommation de ressources calculatoires en passant de la preuve de travail à la preuve d'enjeu. Cependant, la consommation de ressources ne se limite pas au seul calcul, le stockage des données répliquées est également très important.

En effet, en tant que structure de données immuable et répliquée sur tous les pairs, la quantité de données stockée par une bockchain ne cesse de croître. En novembre 2022, un nœud Ethereum qui participe au réseau pair-à-pair (P2P), appelé *fullnode*, a besoin de stocker environ 600 Go de données. Cela rapproche dangereusement les pairs de la barrière symbolique de 1 To de stockage qui peut engendrer des coûts supplémentaires et une baisse du nombre de pairs prêts à supporter l'infrastructure. De plus, à notre connaissance, il n'y a pas d'études sur cette problématique. La communauté scientifique s'est intéressée à l'étude de l'impact [ZJC18, KSE21] et de l'externalisation [NFPSC18, PDBU21] des données des smart-contracts. Une possible solution pourrait être la mise en œuvre du *Sharding* [SZJ18, BFV19, CFB21], qui n'est encore qu'au stade de recherche et dont l'ensemble des détails techniques ne sont pas encore connus. La solution que nous proposons dans cette étude pourrait d'ailleurs être complémentaire avec le *Sharding*.

Dans cette étude, nous proposons une solution pour réduire les besoins de stockage des pairs en exploitant la table de hachage distribuée (DHT) d'Ethereum. En distribuant le stockage des données à long terme (*cold data*), notre solution permet d'économiser 60% du stockage total d'un pair. Les travaux de cette étude sont plus amplement détaillés dans le manuscrit de thèse de Jean-Philippe Eisenbarth [Eis22a].

Le reste de ce document est organisé ainsi : la section 2 présente les méthodes de synchronisation et de stockage actuellement en œuvre dans Geth, le client de référence d'Ethereum. La section 3 présente notre nouvelle stratégie de stockage et de synchronisation et la section 4 conclue ce document.

2 Contexte : stockage et synchronisation des données dans Geth

2.1 Les données d'Ethereum et leur stockage dans Geth

De façon abstraite, Ethereum cesse d'être un simple registre distribué et s'apparente à une machine d'états distribuée basée sur l'exécution (ou l'application) de transactions. Cette machine d'états utilise différents

types de données pour son fonctionnement. L'état courant d'Ethereum est appelé world state (stocké grâce à une structure de données d'arbre Merkle-Patricia) et la mise à jour de cet état est effectuée en appliquant l'ensemble des transactions contenues dans le dernier bloc qui a été créé. Ces transactions peuvent concerner des échanges de crypto-monnaies ainsi que des appels à des fonctions de smart contracts. Le world state est composé de l'ensemble des comptes créés, sachant qu'un compte est l'association d'une adresse d'un utilisateur (ou d'un smart contract) à un état. Lorsqu'un nouveau bloc est créé et fait consensus, l'ensemble des nœuds du réseau doit générer le nouveau world state. Tous les nœuds partagent alors le même état. C'est pour cela qu'Ethereum est qualifié d'« ordinateur unique et conforme ». À côté du world state, il y a également le stockage des transactions récupérées à partir des blocs de la blockchain. Parmi les éléments importants d'une transaction figurent le coût de son exécution, l'adresse du destinataire et la valeur de la transaction. Enfin, un objet intermédiaire, appelé « reçu » (receipt en anglais), est généré et manipulé par les clients. Il représente le résultat d'une transaction encapsulée dans un bloc.

Dans Geth, les données d'Ethereum sont stockées dans deux bases de données clé-valeur : LevelDB et FreezerDB. LevelDB est utilisée pour le stockage des éléments (blocs, entêtes, reçus, etc.) dont l'utilisation ou la modification est régulière. Dans la pratique, il s'agit du stockage des éléments obtenus depuis moins de trois *epoch* (une *epoch* correspondant à 30 000 blocs). Au-delà de ce seuil, les éléments sont transférés vers la base FreezerDB pour leur stockage à long terme, dans laquelle les accès sont très rares et les modifications impossibles (*append-only*). FreezerDB stocke ainsi la partie ossifiée de la blockchain. Étant donné les propriétés respectives de ces deux bases de données, on qualifie les données de LevelDB comme *hot data* (stockage prévu sur SSD) et celles de FreezerDB sont qualifiées de *cold data* (stockage sur HDD).

2.2 Modes de synchronisation dans Geth

Le client Geth propose essentiellement deux modes de synchronisation permettant d'initialiser un full-node : *Snap* (mode par défaut depuis 2021) et *Full* (mode par défaut historique). Quelque soit le mode de synchronisation, en juillet 2022 un nœud stockait environ 600 Go de données.

Full : Il s'agit de récupérer l'ensemble des blocs, de vérifier leur provenance et la preuve de travail, et de ré-exécuter les transactions qu'ils contiennent pour mettre à jour world state en conséquence. L'intégralité des états intermédiaires depuis le bloc de genèse n'est pas conservée, seuls des points de sauvegarde. Régulièrement le nœud procède à un nettoyage des données qui concernent les précédents états. La synchronisation par ré-exécution de tous les blocs prend cependant un temps considérable (plus d'une semaine) justifiant la conception du nouveau mode de synchronisation suivant.

Snap : La synchronisation d'un nouveau nœud en mode snap se déroule selon cette séquence :

1. téléchargement et vérification des entêtes des blocs ;
2. téléchargement des corps des blocs et reçus, et, en parallèle, téléchargement des données brutes du world state (feuilles de l'arbre Merkle-Patricia) et reconstruction de l'arbre à partir de ces données ;
3. correction de l'arbre à la volée en fonction des nouvelles données qui arrivent (*healing phase*).

Un tel nœud conserve également des points de sauvegarde de l'état, mais pas l'intégralité des états. Ce mode permet d'accélérer la synchronisation d'un nouveau nœud, car il récupère directement les états chez les autres pairs et non plus en ré-exécutant l'ensemble des transactions depuis le bloc de genèse. Ce mode tire parti de la nouvelle structure de données *Snapshot* (accès en $O(1)$) et introduite pour accélérer le partage direct des états, sans avoir à reconstruire ceux-ci depuis les blocs.

3 Nouvelle méthode de synchronisation et de stockage

Depuis l'introduction du mode de synchronisation *Snap*, les états sont donc directement téléchargés depuis les autres pairs et non plus générés à partir de l'exécution des transactions, ce qui ne justifie plus que **tous les nœuds** possèdent **tous les blocs**. De plus, les données des 90 000 derniers blocs sont directement accessibles dans LevelDB et les données au-delà sont transférées dans la base de données à long terme FreezeDB. Nous avons également remarqué, à travers une expérience [Eis22a] qu'un nœud contacte très peu de pairs (environ 350) lors de sa synchronisation et que seule une très faible proportion (environ 10%)

TABLE 1 : Évaluation théorique du gain d'espace de stockage pour un *fullnode*. La ligne rouge représente l'état actuel du stockage dans le réseau P2P et la ligne bleue représente un objectif atteignable en adoptant notre proposition.

Préfixe (bits)	Facteur de réplication	Stockage des blocs dans FreezerDB par pair(Go)	Gain sur FreezerDB	Gain total par pair	Stockage total des blocs dans le réseau (Go)
0	36 000	358,90	0,00%	0,00%	12 920 400
1	18 000	179,45	48,91%	30,05%	6 460 200
2	9000	~89,73	73,36%	45,08%	3 230 100
3	4500	~44,87	85,58%	52,59%	1 615 320
4	2250	~22,44	91,70%	56,35%	807 660
5	1125	~11,22	94,75%	58,22%	403 830

TABLE 2 : Résultats de l'évaluation de notre implémentation

Synchronisation	Option	Temps de synchronisation	MAD [†] du nombre de blocs téléchargés
Classique	∅	27 s	8.5
Complète par DHT	`--syncFromDHT`	27 s	2.0
Optimisée par DHT	`--syncFromDHT --storeInDHT`	7 s	0.0

transmet réellement des données. La charge de travail est donc très inégalement répartie. Enfin nous avons remarqué [ECP22] que les clients Ethereum implantent une DHT basée sur Kademlia qui est pleinement fonctionnelle mais inexploitée à ce jour car elle ne stocke aucune information en son sein.

Partant de ces constats, nous proposons de répartir de manière efficace le stockage de FreezerDB, en particulier les corps des blocs et listes de reçus, sur l'ensemble des pairs du réseau grâce à la DHT d'Ethereum. Cela diminuera les besoins en capacité de stockage pour les fullnodes. Notre solution ne concerne pas les données présentes dans LevelDB, car celles-ci sont nécessaires pour la génération d'un nouvel état et sujettes à des accès fréquents en lecture et écriture qui requièrent un stockage local et performant. Pour la répartition du stockage, nous utilisons la fonction de distance XOR, héritée de Kademlia, entre l'ID réseau des pairs (hash Keccak256 de la clé publique ECDSA du pair) et le hash Keccak256 des entêtes des blocs. Ainsi, on peut définir une distance en deçà de laquelle un pair devient responsable de certains blocs, garantissant un certain facteur de réplication, c'est-à-dire un nombre de pairs responsables des mêmes blocs. Ce facteur permet de définir un compromis entre le gain d'espace et la robustesse du stockage répartis au churn et aux attaques éclipse. Une fois la synchronisation terminée, un pair continue de recevoir tous les nouveaux blocs émis mais ne stocke que ceux dont il est responsable.

Nous avons ainsi implanté et évalué deux nouvelles méthodes de synchronisation au sein du client Geth [Eis22b], `--syncFromDHT` et `--storeInDHT`. La première option permet de récupérer les blocs souhaités uniquement chez les pairs qui en sont responsables, la deuxième permet de récupérer uniquement les blocs dont le pair est responsable. Ces deux options peuvent être combinées pour permettre au client de fonctionner de manière optimisée, à savoir en récupérant uniquement les blocs dont il est responsable de par son identifiant réseau depuis les autres pairs qui en sont responsables. De plus, ces nouvelles méthodes sont rétrocompatibles et ne perturbent pas le fonctionnement de la blockchain Ethereum.

En adoptant cette nouvelle méthode de synchronisation et de gestion du stockage des données, le réseau P2P Ethereum peut économiser beaucoup d'espace de stockage. Étant donné une taille de stockage des corps de blocs et listes de reçus dans FreezerDB de 358,9 Go (juillet 2022), et une taille du réseau de 36.000 pairs, le gain attendu en fonction du facteur de réplication est indiqué dans le tableau 1.

Nous avons également évalué notre implémentation dans un réseau privé composé de 16 pairs répartis uniformément sur la DHT et se répartissant le stockage de 7500 blocs avec un facteur de réplication de 1.

†. *Median absolute deviation*, écart absolu à la médiane

Le tableau 2 montre le temps de synchronisation et l'écart absolu à la médiane du nombre de blocs envoyés par pair pour chaque stratégie. Outre le gain de stockage, notre solution permet de réduire le temps de synchronisation et de mieux répartir la charge des téléchargements de blocs.

4 Conclusion

À travers l'étude du code source de Geth et des écrits techniques de la communauté Ethereum, nous avons constaté que les pairs n'exécutent plus aujourd'hui les transactions des blocs qu'ils téléchargent lorsqu'ils se synchronisent. C'est pourquoi il n'est plus nécessaire pour les pairs de stocker l'ensemble des données de la blockchain. Nous avons proposé de répartir la charge du stockage des anciennes données. Nous avons implémenté notre solution en tant que preuve de concept dans Geth et évalué le gain de cette nouvelle méthode : avec un facteur de réplication de 1125 pairs, la réduction du stockage pour chaque pair du réseau représente 95% du volume de FreezerDB et 58% du volume de stockage total. À l'échelle du réseau P2P, cela représente environ une dizaine de pétaoctets. De plus, notre nouvelle stratégie de synchronisation peut être déployée progressivement, chaque client pouvant choisir sa méthode de synchronisation, toutes étant compatibles. Nos travaux futurs s'intéresseront à la complémentarité de notre solution avec le *Sharding*.

Références

[BFV19] Mirko Bez, Giacomo Fornari, and Tullio Vardanega. The scalability challenge of ethereum : An initial quantitative analysis. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 167–176, April 2019.

[CFB21] Mikel Cortes-Goicoechea, Luca Franceschini, and Leonardo Bautista-Gomez. Resource Analysis of Ethereum 2.0 Clients. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 1–8, September 2021.

[ECP22] Jean-Philippe Eisenbarth, Thibault Cholez, and Olivier Perrin. Ethereum's Peer-to-Peer Network Monitoring and Sybil Attack Prevention. *J Netw Syst Manage*, 30(4) :65, October 2022.

[Eis22a] Jean-Philippe Eisenbarth. *Analyse, Valorisation et Protection Des Réseaux Pair-à-Pair de Blockchains Publiques*. These de doctorat, Université de Lorraine, December 2022. Sous la direction de Olivier Perrin et de Thibault Cholez.

[Eis22b] Jean-Philippe Eisenbarth. Fork of the Official Go implementation of the Ethereum protocol supporting new sychronization modes exploiting the DHT, May 2022.

[KSE21] Periklis Kostamis, Andreas Sendros, and Pavlos Efraimidis. Exploring Ethereum's Data Stores : A Cost and Performance Comparison. In *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 53–60, September 2021.

[NFPSC18] Robert Norvill, Beltran Borja Fiz Pontiveros, Radu State, and Andrea Cullen. IPFS for Reduction of Chain Size in Ethereum. In *2018 IEEE International Conference on Internet of Things (iThings)*, pages 1121–1128, July 2018.

[PDBU21] P. Poornima Devi, Srinivasan Ananthanarayanan Bragadeesh, and Arumugam Umamakeswari. Secure Data Management Using IPFS and Ethereum. In *Proceedings of International Conference on Computational Intelligence, Data Science and Cloud Computing*, Lecture Notes on Data Engineering and Communications Technologies, pages 565–578, Singapore, 2021. Springer.

[SZJ18] Daniel Sel, Kaiwen Zhang, and Hans-Arno Jacobsen. Towards Solving the Data Availability Problem for Sharded Ethereum. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, SERIAL'18, pages 25–30, New York, NY, USA, December 2018. Association for Computing Machinery.

[ZJC18] Huijuan Zhang, Chengxin Jin, and Hejie Cui. A Method to Predict the Performance and Storage of Executing Contract for Ethereum Consortium-Blockchain. In *Blockchain – ICBC 2018*, Lecture Notes in Computer Science, pages 63–74, Cham, 2018. Springer.