A method of data synchronization with Ethereum blockchain

Ivan Tarkhanov  
GAUGN; FRC “Computer Science and Control” of RAS  
Russian Federation, Moscow

Obadah Hammoud  
National University of Science and Technology (MISIS)  
Russian Federation, Moscow

Abstract

This research proposes a new method of data synchronization between public blockchain networks and local machines. We discussed the proposed algorithm, and the mathematical model which achieves the shortest delay required for data synchronization. Tests were conducted to verify the correctness of the proposed model. Then a comparison is made with the current available classical synchronization methods. Suggested method may be useful for future DApps applications on Ethereum network.

Keywords list (en): synchronization, blockchain, Ethereum, DApps

Date of publication: 05.09.2020

Citation link:

Hammoud O., Tarkhanov I. A method of data synchronization with Ethereum blockchain // Artificial societies. – 2020. – V. 15. – Issue 3 [Electronic resource].  
URL: https://artsoc.jes.su/s207751800010671-7-1/ (circulation date: 18.03.2021). DOI: 10.18254/S207751800010671-7

Introduction
Blockchain has raised as a secure distributed storage technology, which applications are increasing day by day [6]. Development of distributed applications using blockchain is gaining popularity (Dapps) [9]. In some cases, it’s necessary to have a copy of the data on the local machine, especially in applications which require a very high number of reading requests in a short time. In this type of applications, the need of an efficient synchronization algorithm to keep the data on local machines up to date is required. In this research, we discuss a new method to achieve this purpose in a specific type of applications. These applications should meet the following criteria:

1. High number of data reading processes.
2. Most of transactions are basically new data entries insertion instead of modifying or deleting existed data entries.
3. Having the latest data version is not so critical as long as it’s relatively new data.

An example of such applications would be storing black lists of links, which contain links to be blocked by browsers and operating systems [2]. Users around the world will be requesting data frequently, and there are many links to be blocked.

In some applications, it’s required to get the latest data updates exclusively, like in banking systems, where the account data can’t be out dated. Most of operations are limited to data insertion (adding new entries), with a low number of modifying/deleting entries operations. In this case, having the latest changes (like those made minutes ago) won’t be crucial.

Also, having a local version of this data is important to limit the huge number of requests to blockchain nodes. Another example would be weather climate information storage system, were many requests are made to obtain the weather information, and having a bit old data (like made minutes ago) is not problematic.

If the system doesn’t satisfy these 3 criteria, with high probability, it will not work effectively on a public blockchain.

Data Synchronizing Method

The suggested method is based on splitting existing data records into blocks. As the nature of the studied applications is data entries insertion, it means that the last block probably has all or most of changes, which means that the synchronizing data would be limited to synchronizing the last block instead of synchronizing all of existing data. Blocks with different data will have different hashes, and it will result in having different total hash. Figure 1 explains the proposed concept.
The algorithm goes as the following:

1. Each interval \( t \), compare the total hashes between blockchain and the local version on the client device. If the total hash is the same, stop, if not, go to step 2.
2. Compare blocks hashes from the blockchain with the according ones in the local version.
3. For each local block with a different hash from the blockchain version, do the following
4. Replace it with the blockchain version.
5. Calculate its new hash.
6. Recalculate the total hash of the local version.
7. Stop.
This algorithm is shown in figure 2. The value of the interval $t$ can change according to the nature of the application.

Figure 2. Data synchronizing algorithm
Optimizing the Synchronizing Process

This algorithm makes the data synchronization process more efficient, as it requires transferring less data. This algorithm depends on dividing total data into N blocks. The number of the block is important in this process. It can’t be too big, as it means that more data will be transferred each time, and it can’t be too small, as it will result with having a big number of blocks, which means a high number of hashes comparisons. The number of blocks reflects the size of each block, as total data is consisted of the number of blocks multiplied by the number of entries in each block.

The definition of the optimal size of blocks is based on having the least delay for downloading the last block: \( T \rightarrow \text{min} \)

Assuming that the delay to transfer one entry is fixed (t), the total delay can be defined as the following: \( T = t \times S + t \times H \)

Where S is the block size (number of entries in one block), and H is the number of hashes. The number of hashes is equal to the number of blocks, as each block has one hash. The number of entries in one block is equal to the total number of entries divided by the number of blocks. The formula can be rewritten as the following: \( T = t \times NB + t \times B \)

Where N is the total number of entries, and B is the number of blocks. To make the total delay minimum, the derivative of T should be equal to 0: \( dB \times t \times NB + t \times B = 0 \)

Which results in the following optimal block size: \( B = \sqrt{N} \)

Add or Updating Data Entries

Let’s consider updating or deleting data entries issue. Updating data can be done by simply replacing the old values with the new ones. When deleting entries, its values get updated to null value. The reason of not deleting it directly is:

1. Deleting the entry index requires shifting back all data entries after the deleted value by one step. As a result, it requires a lot of processing, which means that it costs a lot.
2. Shifting data positions means changing the hash of all blocks.
3. The process of updating/deleting entries changes the whole block hash, but it’s not considered as a problem, since the application nature doesn’t require a lot of modifying/updating processes as we discussed in the introduction.

If existing blocks are filled to capacity with data entries (number of data entries = \( B^2 \)), adding new data can raise a problem. Given \( N \) as the number of total existing data entries, B is the number of blocks, it means that there are \( B \) blocks, each block contains \( B \) data entries, so the total number of entries is \( N = B^2 \)

Let’s assume that \( N' \) represents new data entries: \( N'_{\text{new}} = N + N' \)

So, \( B'_{\text{new}} \) can be calculated as the following: \( B'_{\text{new}} = \sqrt{N'_{\text{new}}} = \sqrt{N + N'} \)

As a result, if \( N' \) isn’t big enough, all data blocks will be updated and its hash will be changed just to add relatively small number of entries. Consequently, the user application has to download all data blocks (as an example, the user might have to download all new blocks, just to
download one new data entry in each block). To overcome the problem, the following concept is introduced: 1. New data entries get stored in new blocks of size $B$, until $2B$ blocks are filled in total. 2. If all new blocks are filled, the resulting structure is $2B*B$ ($2B$ blocks, containing $B$ blocks in each). Thus, only new blocks with new data get synchronized by users. 3. New data entries are inserted to existing blocks, starting from the first block. In this step, data can be inserted in empty data slots (an entry which was deleted earlier). This step can be considered effective, as the new data entry (entries) should be inserted to the block, so all changes are done on one block. Also, it removes empty slots caused by data deletion.

1. When the first block is filled (number of data entries inside of it is $2B$), second block gets filled, and so on.
2. The process repeats when the total number of data entries reaches $2B*2B$.

Results

To verify the results, a testing system was built as the following:
- Ethereum Ropsten [7] was used with as the blockchain network with the help of Infura nodes [3]
- URL links were stored as the testing data entries
- Node.js was used to implement the client side, and Web3 library was used to connect with Ethereum
- Different tests were made, having different values for entries total number (7500, 10000, 20000, 50000, 100000).
- For each data entries total number, different scenarios were created.
- In each scenario, a block size was considered.
- Multiple tests were conducted for each scenario.

Table 1. The test results

| N/B  | Test #1 | Test #2 | Test #3 | Test #4 |
|------|---------|---------|---------|---------|
| 7500 | 60      | 90      | 80      | 70      |
| 10000| 90      | 120     | 90      | 100     |
| 20000| 120     | 100     | 130     | 110     |
| 50000| 150     | 150     | 250     | 150     |
| 100000| 300     | 200     | 200     | 400     |

As a result, different values for the optimal block size were obtained, but within a very narrow range close to the suggested value. This differentiation of values can have several causes, like the internet connection state and blockchain nodes state. Figure 3 shows the results obtained.
Conclusion

In this paper, a new method to synchronize data between clients and blockchain is suggested. It reduces data transmission rates noticeable. If a list of 1,000,000 entries in blockchain needs to be synchronized, a 1,000 entries are mostly downloaded instead of all entries in the case of updates existence. This means less network usage, and less data processing on the client side.

Alternative existing methods would be:

- Full data synchronization (Unidirectional synchronization) [7]. In this method, the whole data entries from blockchain are downloaded every time a change is applied (event driven synchronization) [4], or on a regular time intervals basis (time driven synchronization). This leads to high data transmission rates, and a higher pressure on blockchain nodes.

- Changes synchronization (Bidirectional synchronization) [5]. In this method, a log with all changes can be maintained on blockchain side. Users compare with the last checkpoint existed on the blockchain side, and perform actions related with unsynchronized checkpoints. It has the following downsides: 1) It requires much extra storage on the blockchain side to maintain the log. The log most likely has much more entries than the number of data entries. The reason of that, is maintaining a list requires logging operations not required to have the final list, like inserting a list, and then modify it or deleting it. 2) Unnecessary actions can be executed on the local machine, like adding a link and then modifying/deleting it. 3) Extra delay in blockchain side to search for local node latest checkpoint.

The main limitation in the proposed method is that it targets a very specific type of applications, which have low number of data modifications and deletion, and where it's not critical to get the latest data version. As a result, it can't be used as a general method which fits all synchronization situations.
In this research, the following results were obtained:

- A new method to synchronize data between blockchain network and local machine is proposed.
- A mathematical model which ensures the minimal delay for this method to work. This model can become the basis for promising distributed applications on the Ethereum.

In future, we consider extending this work to have it more generalized and applicable on a wider range of blockchain-based applications types.

References:

1. Debajani M. Ethereum for Architects and Developers. Apress. ISBN-13 (pbk): 978-1-4842-4074-8 – 2018.

2. Hammoud O. R., Tarkhanov I. A. A Method to Prevent Tracking Browsing History with the Use of Browser Extension //2019 4th International Conference on Computer Science and Engineering (UBMK). – IEEE, 2019.

3. Lee W. M. Using the metamask chrome extension //Beginning Ethereum Smart Contracts Programming. – Apress, Berkeley, CA, 2019.

4. Michael E, Pierre G, Rupak M, and Fernando R. Analysis of Asynchronous Programs with Event-Based Synchronization. Springer-Verlag Berlin Heidelberg – 2015.

5. Neil F. Differential synchronization //Proceedings of the 9th ACM symposium on Document engineering. – 2009.

6. Puthal D. et al. The blockchain as a decentralized security framework [future directions] //IEEE Consumer Electronics Magazine. 2018, T. 7, №. 2.

7. Rajni J. Review paper on database synchronization between local and server // International journal of engineering sciences & research technology 2016.

8. Sinnott, R. O., Chadwick, D. W., Doherty, T., Martin, D., Stell, A., Stewart, G., ... & Watt, J. Advanced security for virtual organizations: The pros and cons of centralized vs decentralized security models // Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID) – IEEE – 2008

9. Wu K. An empirical study of blockchain-based decentralized applications //arXiv preprint arXiv:1902.04969. – 2019.
A method of data synchronization with Ethereum blockchain

Тарханов Иван Александрович
ГАУГН; ФИЦ ИУ РАН
Российская Федерация, Москва

Хаммуд Обадах Рамадан
НИТУ МИСиС
Российская Федерация, Москва

Аннотация

This research proposes a new method of data synchronization between public blockchain networks and local machines. We discussed the proposed algorithm, and the mathematical model which achieves the shortest delay required for data synchronization. Tests were conducted to verify the correctness of the proposed model. Then a comparison is made with the current available classical synchronization methods. Suggested method may be useful for future DApps applications on Ethereum network.

Ключевые слова: synchronization, blockchain, Ethereum, DApps

Дата публикации: 05.09.2020

Ссылка для цитирования:

Тарханов И. А., Хаммуд О. Р. A method of data synchronization with Ethereum blockchain // Искусственные общества. – 2020. – Т. 15. – Выпуск 3 [Электронный ресурс]. URL: https://artsoc.jes.su/s207751800010671-7-1/ (дата обращения: 18.03.2021). DOI: 10.18254/S207751800010671-7