2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

Jidar: A Jigsaw-like Data Reduction Approach without Trust Assumptions for Bitcoin System

Xiaohai Dai, Jiang Xiao, Wenhui Yang, Chaofan Wang, and Hai Jin
National Engineering Research Center for Big Data Technology and System
Services Computing Technology and System Lab, Cluster and Grid Computing Lab
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
Email: jiangxiao@hust.edu.cn

***Abstract*—The pioneer blockchain platform Bitcoin has drawn massive attention from various sectors in society, with its promising decentralized, irreversible, and trust-free natures. Increasing volume of Bitcoin transactions makes it impractical for each user to store a full copy of whole ledger, especially for a normal user who makes few transactions and owns constrained storage space. There are already some works conducted to reduce the storage overhead by redesigning the system protocol, based on different trust assumptions. An example is running light nodes, which trust the full nodes to store its relevant data and reply to its data request timely. However, it introduces the risks of permanent data loss, as the data relevant to a light node may be tailored by all the full nodes later. Another attempt is conducted by organizing the nodes into several cooperative units based on a trust assumption, which is hard to satisfy in practice. In this paper, we propose a novel *Jigsaw-like Data Reduction* (Jidar) approach. Each node in Jidar only stores transactions of interest and relevant Merkle branches from the complete blocks, just like selecting several pieces from the jigsaw puzzles. A node can maintain and verify all its relevant data locally and safely without any trust assumptions. To verify the validity of a newly proposed transaction, Merkle branches relevant to the inputs are provided along with the transaction by the proposer. In view of the requirement that a user wants to cohere all the fragments stored in different nodes into a complete block, just like stitching all the pieces into a complete jigsaw-puzzle picture, a mechanism to query full data is added to Jidar. Experimental results demonstrate that Jidar can reduce a normal node's storage cost to about 1.03% of the original Bitcoin system at a low communication cost.**

***Index Terms*—Bitcoin, blockchain, data reduction, bloom filter, Merkle tree**

I. Introduction

Envisioned as the pioneer of public blockchain platforms, Bitcoin has received extensive attention for the past few years since its birth in 2008 [1]. Bitcoin users can interact and make transactions with each other reliably and directly without an intermediate, based on a sequential and tamper-proof data structure, known as chained blocks. However, the increasing volume of transactions leads to a rapid increase in Bitcoin on-chain data storage, which is over 210GB at present. Newly generated data for each year takes about 50GB to store, which is unfavorable for a normal user with constrained storage resource to make room for it. What's worse, the size of data storage is proportional to the performance of blockchain system, namely *Transaction Per Second* (TPS) [2]. Since the TPS of Bitcoin is too low now, which is much lower than Visa with over 4000 TPS, lots of proposals to enhance the performance of Bitcoin have been put forward and widely discussed [3] [4]. It can be predicted that TPS will be promoted to a much higher value in the near future, which will bring greater stress on storage of Bitcoin data. Therefore, there is a trend toward reducing the on-chain blockchain data.

There are already some works aiming to deal with the problem of large storage size, by redesigning the blockchain system protocol based on various trust assumptions. The first solution is proposed by Nakamoto [1], namely light nodes. By relying the full nodes to verify transactions, light nodes only need to store the block headers, thus reducing the storage cost. However, a light node must trust other full nodes to store its relevant data and reply to its data request timely. The trust assumption may introduce the risks of permanent data loss, as the data relevant to a light node may be tailored by all the full nodes later. Another attempt tries to reduce the data duplication in a trust unit of nodes (e.g., CUB [5]). By organizing the nodes into several trust units, each node in a trust unit only needs to store one part of the entire blockchain data, whose size is relatively small. However, trust units are usually organized based on the assumption that all the nodes in a trust unit must trust each other, which may be too strong to realize.

Different from the works conducted in the layer of system protocol (e.g., light nodes and CUB), another category makes attempts in the layer of the underlying storage engine (e.g., Forkbase [6] and Ustore [7]). By optimizing the storage engine, works of this category try to reduce the deduplications of data in a node. However, deduplication in a node can bring few effects on blockchain, because the data stored in a blockchain node has a low duplication rate.

In reality, there are massive nodes that make few transactions and have constrained storage space on the blockchain network (i.e., normal nodes in this paper). In general, one of these normal nodes takes the blockchain as a payment platform, who only cares about transactions it is involved without interest in others' data. As a result, it is unreasonable to ask it to keep a complete record of the entire transaction history. Besides, as for a node in the blockchain in the scenario of *Internet of Things* (IoT) [8], which has limited storage capacity, it is impractical for it to store all the blockchain data.

In this paper, we propose a *jigsaw-like data reduction* (Jidar) approach without any trust assumptions for Bitcoin system. With Jidar, only data of interest is stored in each node. To be

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00132

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

specific, each normal node only keeps the relevant data from the entire blocks, which is like selecting several pieces from the jigsaw puzzles. Our approach is designed based on the following understandings:

- Vast majority of users on Bitcoin only care about their personally relevant data, without interests in others' transactions.
- Users have no obligations to store other users' data since they get no direct paid from storing it, which may lead to the irresponsible data tailor by the data preservers (e.g., full nodes).
- Optimizations of a system should not break the original properties, such as decentralized, irreversible, and trust-free natures.

Each node in Jidar only stores the transactions of interest and relevant Merkle branches, thus reducing the storage cost largely. No trust assumptions are made by Jidar, which preserves the characteristics of blockchain. Each user stores his/her relevant data locally, without relying on others, which enhances the users' capacity to protect his/her data. Since a node is not asked to store other nodes' data, the verification of a newly proposed transaction relies on the proof provided by the transaction proposer. In consideration of the requirement for the full data in a block, a mechanism cohering all the fragments stored in different nodes into a complete block is added to Jidar, which is just like stitching all the pieces into a complete jigsaw-puzzle picture. A prototype of Jidar approach is implemented based on Bitcoin implementation in Golang. Experimental results show that Jidar can reduce the data storage to about 1.03% of the original Bitcoin system at an acceptable communication cost for the most normal nodes.

In summary, our major contributions are as follows:

- A detailed analysis on the problem of large storage cost is made both qualitatively and quantitatively, with a summary of related works.
- Three goals about designing a data reduction approach are presented, according to our understandings of blockchain systems.
- An approach aiming to reduce the data storage is elaborated, without making any trust assumptions nor sacrificing the original properties of blockchains.
- A prototype is implemented to evaluate the Jidar approach, whose experimental results demonstrate its feasibility and efficiency.

The remainder of this paper is organized as follows. Section II introduces the motivation and related work about this paper. A basic design including a strawman and bloom filter is elaborated in Section III. To satisfy the requirements of system robustness and functional integrity, the enhanced design of Jidar approach is complemented in Section IV. Details about the prototype implementation of Jidar are described in Section V. A Section VI conducts several experiments to evaluate the Jidar approach. Finally, conclusions are made in Section VII.

TABLE I: Increase of storage cost as TPS increases

TPS	7	10	100	1000	2000
Data (GB/Year)	50	75	750	7500	15000

II. Motivation & Related Work

In this section, we firstly propose the motivations of this paper and analyze the state-of-the-art works related to the reduction of blockchain data. Based on these motivations and analysis, we put forward the overall goals of designing our new system.

A. Motivation

As a Bitcoin full node has to store all the transactions locally, the storage cost increases largely as the time goes by. At the time of writing this paper, the storage size of a Bitcoin full node has reached to 210GB. What's worse, since the system performance of Bitcoin is too low now, at most 7 *Transactions Per Second* (TPS), lots of proposals aiming to improve the performance are presented. It can be predicted that these proposals will further exacerbate the storage problem. Table I reflects the relationship between the TPS and the corresponding cost of extra data storage. The larger is TPS, the larger is storage cost. Particularly, once the Bitcoin's TPS is promoted to 2000 in the near future, the extra storage cost per year is around 15TB, which is too high for normal users [5].

B. Related Work

There are already some works to relieve the stress of data storage of blockchain system. According to the different layers these works are conducted, they can be divided into two categories: system protocol layer and storage engine layer.

1) System protocol layer: One large category of the works tries to redesign system protocols, based on various trust assumptions. An example of this is light node mechanism proposed by Nakamoto in Bitcoin white paper [1]. Light nodes enable users to verify transactions by visiting the full nodes instead of storing the entire ledger data, which reduces the storage size in Bitcoin. This mechanism is widely used in mobile Bitcoin wallets, such as Electrum[1] and BRD[2]. However, there are still some shortcomings in terms of this scheme. Firstly, light nodes have to trust the full nodes to store its relevant data and reply to its data request, which may introduce the risks of permanent data loss. One possibility is that data related to one user may be tailored by all the full nodes later. What's worse, there may be a network bottleneck when multiple light nodes request block data from the same full node.

Another attempt made by CUB organizes different nodes into several *Consensus Units* (CUs) based on the assumption that all the nodes in the same unit must trust with each other [5]. Each node in a CU stores only one part of the blockchain data and all the nodes in a CU maintains a full copy of blockchain data, thus reducing the storage in a node.

[1]https://electrum.org/
[2]https://brd.com/

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

However, the trust assumption may be too hard to satisfy in practice. Besides, granularity of block-level deduplication is a little coarse, which may bring limited effects of data reduction. As a contrast, our approach carries out finer-grained de-duplication without any trust assumptions.

2) Storage engine layer: Another category of schemes tries to eliminate the duplication of data in the layer of underlying storage engine, whose representative works include Forkbase [6] and Ustore [7]. As a universal Storage Engine for forkable applications, Forkbase aims to improve the storage performance and reduce the storage size at the same time. With the aid of chunk-based deduplication, Forkbase succeeds in removing the duplicates of large files with massive common contents. However, as the duplication rates of transactions and blocks stored in a node are usually very low, deduplication at the level of storage engine does a bad job in the scenario of blockchain.

C. Goals

1) Proportional storage: As our primary target, we hope to reduce the storage cost largely for each node. In an ideal world, all the data for a normal user can be stored on a common computer, even on a mobile phone. Meanwhile, a reasonable scheme is that the more transactions are related to a user (i.e., as the sender or receiver of a transaction), the more data should the user store. What's more, the increase of storage size should be independent from the increase of nodes number in the long run, so as to promote the system's scalability.

GOAL 1: Storage size for a user should be in proportion to the number of transactions related to him/her, so as to get reduced largely.

2) Respective storage: By analogy to custody of belongings in our real life, data should be stored and maintained by the stakeholders respectively. It is more reasonable to rely on the stakeholders rather than others to store the data, which can usually provide better safety. If the data gets lost unfortunately, the blame should be placed on the maintainer, namely the stakeholders. From the perspective of users' rights and interests, the rule above is also consistent with the claim that the ownership of data should be returned back to the users [9].

GOAL 2: Data should be stored by the stakeholders respectively, which is consistent with what in our real life and accords with the protection of data ownership.

3) Few assumptions: One of the most important features of permissionless blockchains (e.g., Bitcoin) is no trust relationship among different users. As a clangorous slogan, permissionless blockchains try to establish authentic relationship without trust in each other. Abiding by this slogan, we should make few even no assumptions about the trust among the users.

GOAL 3: Few assumptions about the trust among users should be made, so as to reserve the original advantages of Bitcoin.

III. Basic Design

In this section, we introduce the basic design of our approach in two steps. After elaborating a strawman design, we pose a challenge at the end of Section III-A. To address the challenge, a field based on bloom filter is added up to the block header, which is presented in Section III-B.

A. Strawman design

For the stake of simplicity, we first introduce a strawman design to reduce the storage size at the level of transaction, whose overview is depicted in Figure 1. In the strawman design, each node will store all the block headers, while only storing one small part of each block body. Taking node A as an example, since the transaction *Tx-3* is related to node A (e.g., with node A as one of the receivers of *Tx-3*), the node only stores *Tx-3* and the Merkle branch linking *Tx-3* to the tree root. Other branches including *Tx-0*, *Tx-1*, and *Tx-2* are discarded by it.

1) Broadcast and validation of new transactions: Since no full copy of data is stored by each node, the validation of new transaction relies on the proof provided by the transaction proposer. When a user proposes a new transaction, it must broadcast the proof along with the transaction, as shown by the first step in Figure 1. We refer to a transaction plus the relevant proof as a *Transaction Unit* (TU). Figure 2 describes the details of TU, where the transaction *Tx-n* takes the *Output2* from *Tx-3* as the *Input*. As a result, the proof of the TU should include *Tx-3*, the Merkle branch linking *Tx-3* to the tree root, and the *BlockNumber* to identify the specific block.

Once a node received a TU from the network, it will validate the reliability of the new transaction in the TU before further broadcasting it. The validation process is similar to what in the *Simplified Payment Verification* (SPV) mechanism. To be specific, it verifies the Merkle branch from the leaves to the root, and then compares the root in the TU with the tree root in the block header. The second step in Figure 1 depicts the validation process. Since the proof can be provided by the transaction proposer on his/her own, without trusting others, Goal 3 is achieved.

2) Mining and broadcast of new blocks: As a miner in Figure 1, node M does the same thing as that in the original Bitcoin system after validating the TUs. It will create a candidate block based on the transactions collected from TUs and then make *Proof of Work* (PoW) calculation, as the third step in Figure 1 shows.

Once the new block is mined, it will be broadcast across the network by node M, as the forth step in Figure 1 shows. Each node will store the relevant transactions and Merkle branch after receiving the new block, which is only a small part of the total block data. Since the new block contains the transaction proposed by node A, it stores *Tx-n* and the Merkle branch including *Tx-n*. By contrast, node B stores nothing except for the block header, because the newly mined block contains no transactions related to it. Picking the relevant data from a block by a node bears a strong resemblance to the process of splitting out several pieces from a jigsaw puzzle. To be notified, each transaction will be stored at least twice in the whole network, because a transaction usually involves at least a pair of sender-receiver nodes. As shown in Figure 1, both node A and node C

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

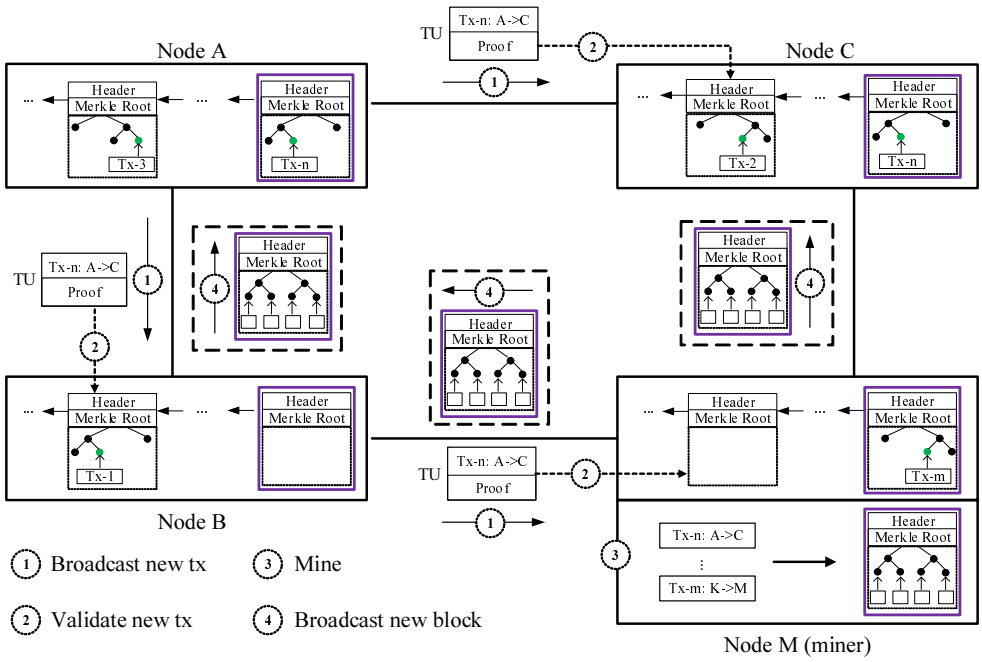


Fig. 1: An overview of a strawman design

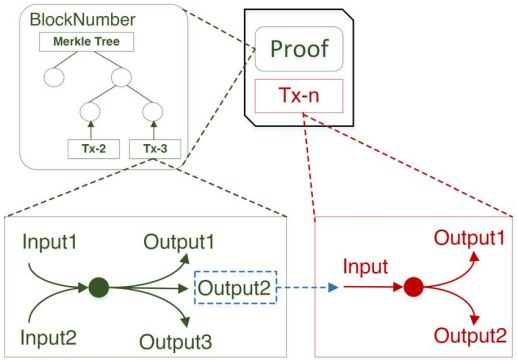


Fig. 2: Schematic diagram of TU

store *Tx-n*. By onlying storing the related data, the storage size can get reduced largely, thus reaching Goal 1. Besides, since different parts in a block are stored separately by different nodes, Goal 2 is achieved.

3) Problems of transaction validation: Although the strawman design seems to work well in reducing the storage cost, it has a vital problem in the process of transaction validation. The Bitcoin protocol requires that the inputs of a new transaction must be one of the *Unspent Transaction Outputs* (UTXOs). A Merkle branch and relevant transaction can only guarantee that the input is one of the transaction outputs, but can do nothing in proving that the output has not been spent before.

CHALLENGE 1: How to prove that the transaction outputs are not spent before?

B. Design with bloom filter

To address the Challenge 1, we draw lessons from bloom Filter [10]. We firstly introduce the adoption of bloom filter in our design in detail, followed by a challenge related to false positive matches in Section III-B1. The solution to tackle the challenge is elaborated in Section III-B2.

1) Adoption of bloom filter: Bloom filter is a method sacrificing space for efficiency, which is widely used to test if an element is contained in a set. We add a field of bloom filter to the block header, which indicates if an input of a new transaction has been spent as the input before. As shown in Figure 3, a 16-bit bloom filter and three hash functions are adopted. For simplicity, we assume that each transaction consists of one input and one output. By calculating three hash functions of the input of *Tx-0*, the *9th*, *14th*, and *12th* bits of bloom filter are set. The same is done to *Tx-1*, *Tx-2*, and *Tx-3*. When checking if the input of a new transaction has been spent in this block, three hash functions are calculated again. If one of the bits obtained is not set in the bloom filter, we can make sure that the new input is not spent in this block. As an example, the normal case in Figure 3 demonstrates this situation.

As a result, the validation process of the inputs of a new transaction includes two parts: 1) validate if the input is existent as an output of the past transactions (i.e., existence validation); 2) validate if the input has been spent after creation (i.e., unspent validation). Taking the *Tx-n* in Figure 4 as an example, existence validation is done based on the *k-th* block header according to the 'BlockNumber' field in the proof of TU. Unspent validation is done to the blocks from *k+1* to the

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

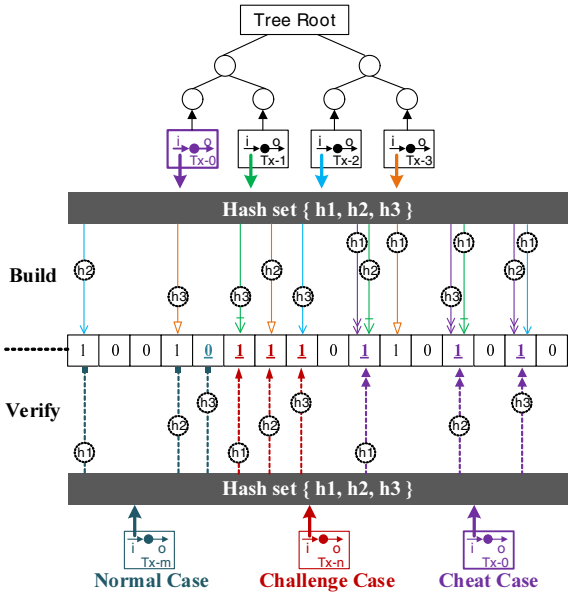


Fig. 3: Schematic diagram of adopting bloom filter to the block header

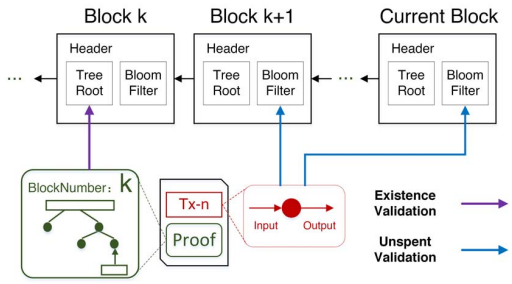


Fig. 4: Two parts of transaction validation process

current block. It should be noted that there may be a long time after the output (i.e., the input of the new transaction) is created, which may lead to a great deal of work of unspent validation. However, it will not take too much time because the calculation of hash function only needs to be done at once, whose results can be stored for later access. Furthermore, validation works can be done in parallel to speed up, which is further proved with experiments in Section VI-A.

One of the most important problems ignored above is the intrinsic false positive matches of bloom filter. Take the *Tx-n* in the 'Challenge Case' in Figure 3 as an example, three bits of calculation results have been set by *Tx-1*, *Tx-2*, and *Tx-3*. As a result, the input of *Tx-n* will be mistaken as already spent, although it is exactly not spent in this block.

CHALLENGE 2: How to deal with the false positive matches of bloom filter?

2) Solutions to deal with false positives: There are two objectives related to deal with the false positive matches of bloom filter: 1) the unspent output (i.e., input) should not be mistaken as spent (i.e., mistake resistance), which has been elaborated above; 2) the spent output (i.e., input) should not be forged as unspent (i.e., cheat resistance). To describe objective of cheat resistance more clearly, we take the 'Cheat Case' in Figure 3 as an example. Three bits of hash calculation results of *Tx-0* are set by *Tx-1*, *Tx-2*, and *Tx-3* respectively. Although the input of *Tx-0* has been spent in this block exactly, it can lie that the corresponding bits are set by other transactions, which is hard to check directly.

The solution still relies on the proof provided by the new transaction proposer. As we all know, each node maintains a plurality of UTXOs (i.e., UTXO set), which is used to make up the inputs of new transactions. If there is no false positive match in the unspent validation process described in Section III-B1, the proof in a TU only needs to contain the Merkle branch and relevant transaction. However, if a false positive match happens in a block, the entire Merkle tree and transactions of the block should be provided as proof. By providing the entire data, an input of a transaction can be proved to be unspent by other nodes. To be specific, the input is compared with all the inputs in the block one by one, which provides not only mistake resistance but also cheat resistance.

Whether it is a Merkle branch or the entire Merkle tree, the data provided as the proof should be stored by the relevant node in advance. As a result, each time a new block is received by a node, it has to check if there is a false positive match between the spent inputs in the block and the outputs in its UTXO set. If true, the entire data is stored. Otherwise, one part of the data is stored. It should be noted that storing the entire data occurs rarely, as the false positive rate is tuned to a very small value.

IV. Jidar Design

Up to now, the system designed above can work like a Bitcoin system in most of the aspects. However, the robustness of the system and some complementary functionalities are ignored. In this section, we firstly present the requirements of the system robustness and functional integrity. To satisfy these requirements, some enchanced solutions are put forward. At the end of this section, we make a summary of our approach (i.e., Jidar).

A. Requirements of system robustness and integrity

As stated in Section III-B2, each node has to find the possible false positive match once receiving a new block respectively. If no false positive match is found, each node will discard the irrelevant data in local to reduce the storage size. This requires the node to receive the new blocks in time when the new blocks are still in the process of broadcast. In other words, it is hard for a node to find the entire data of the blocks from its neighbors after broadcast process. However, unexpected network disconnections or crash of nodes may lead the nodes to miss the blocks, which are common in our daily life.

CHALLENGE 3: How to deal with the unexpected network disconnections or crash of nodes?

As one of our understandings of blockchain systems, vast majority of users only care about their personally relevant data,

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

without interests in others'. As a result, the most common requirement for normal users is to query personal data. It is easy and fast for such a query, since the personal data has been stored locally. However, what if a user wants to access the data without local storage? For instance, a late comer wants to get all the data on the blockchain to do further scientific analysis. Although it is not a very common requirement of normal users, querying the full data should be taken into consideration, thus contributing to the functional integrity of the system.

CHALLENGE 4: How to query the data without local storage?

B. Enhanced solutions

In this section, we introduce the enhanced solutions one by one, thus complementing the design of Jidar.

1) Block window: Since both network disconnections and node crashes can be tackled with the same solution, we will take node crash as an example in the following pages for simplicity. According to the duration of crash, the problem of node crashes can be divided into two subproblems: short-term crashes whose durations are less than time T and long-term crashes with durations more than T. A reasonable assumption made by us is that most of the node crashes can be repaired in time T, which means long-term crashes are much rarer than short-term crashes. To address short-term crashes, each node is asked to temporarily store the complete data of blocks received within the recent past time T, which makes up a *block window*. As the time goes by, newly received blocks will be added to the *block window*, while the oldest blocks will be discarded. With the aid of the *block window* mechanism, a node restored from a short-term crash can easily access the missed blocks from its neighbors, thus promoting the system's robustness. The value of T should be set with a trade-off. The larger is T, the robuster the system can be, while the more extra storage will be taken.

2) Multiple instances: Although it rarely happens, problems of long-term crashes should be taken seriously. A simple but viable solution to solve the problem is running multiple instances of nodes by a user. If an instance gets crashed, other instances can still work well to receive the blocks. Obviously, more instances are run by a user, more secure is his/her data storage while more monetary cost is needed. As stated in Section II-C, it relies on the users to maintain their relevant data and ensure the safety of data storage. As a result, it is up to the users to decide how to deploy how many instances. As for the users with a large sum of coins, it is worthy and reasonable for them to distribute multiple instances geographically, either with the convenient help of cloud services [11] or making direct use of physical machines. With respect to users with few coins, they only need to take advantages of their multiple portable electronic devices, such as mobile phones, tablets, and laptops. What's more, since the smart home equipments are becoming increasingly popular, they can be linked to the network as the instances of nodes.

3) Full data query: By far, the relevant data stored by a user only contains the transactions (and corresponding Merkle branches) which takes the user as senders or receivers. In reality, a user can maintain any transactions he/she is interested in, whose only cost is extra storage. If a user wants to or plans to make analysis on the total blockchain data, he/she can keep storing all the data from the very first. As for the late comer in Section IV-A, who has missed the past blocks, a broadcast mechanism to query full data is added to the system. As shown in Figure 5, node D wants to get the complete data of block k. Firstly, it broadcasts the query to the network. Each node returns the fragment stored by it to node D, once receiving the query. After receiving enough fragments, node D will do verifications and cohere the data into a complete block, just like stitching all the pieces into a complete jigsaw-puzzle picture. It should be noted that other nodes may make no response to the data query, as these are not their obligations. Incentive mechanisms can be adopted to encourage users to send the data. Just like the trade rules in our daily life, a node may need to pay others for the wanted data, which further protects the data ownership described in Goal 2. Since the focus of this paper is on the design of data structure, we leave the design of incentive mechanisms as our future works.

C. Summary of Jidar

After elaborating the basic design in Section III and enhanced solutions in Section IV-B, Jidar approach has been described well in detail. Since each user only stores his/her relevant data, Goal 1 and Goal 2 are achieved. During the design process of Jidar, few assumptions about the trust relationship are made, thus achieving the Goal 3.

Compared to the original Bitcoin system, what Jidar makes modifications to is the structure of data stored and transmitted by a node. Including mining, communication, and consensus, Jidar keeps consistent with the original Bitcoin. In other words, Jidar redesigns the blockchain architecture only from the data layer, which avoids lots of security problems, especially the security of consensus protocol.

V. Implemenation

We have implemented a prototype of Jidar on our server machines. Each server contains two eight-core Intel Xeon E5-2670 2.60GHz CPUs, 64GB DRAM, and 8TB HDD, with CentOS 7.2 as the operating system. The prototype is based on the Bitcoin implementation written in Go language, namely btcd[3]. Since the wallet function is excluded from btcd implementation as btcwallet[4], both of these two projects are reimplemented in our prototype, which is called Jidard and Jidarwallet respectively.

As for the bloom filter added in the block header, we adopt the scheme proposed by Adam et al [12]. However, many other optimizations or variations of bloom filter can be used instead, such as Cuckoo Filter [13], Compressed Bloom Filter [14]. To avoid the duplicate hash calculation of the same data, hash results are cached for later use. Besides, validation process of transactions is implemented in a parallel manner, so as to

[3]https://github.com/btcsuite/btcd
[4]https://github.com/btcsuite/btcwallet

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

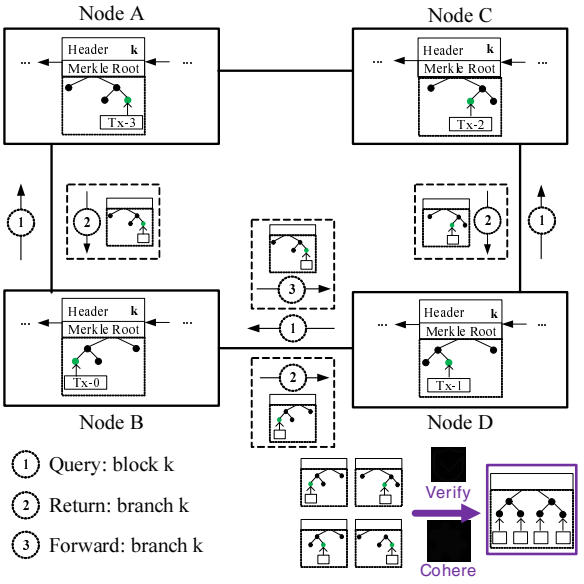


Fig. 5: Broadcast mechanism to query full data

reduce the validation time. To instruct the node to store the data of interest, a parameter containing one or several addresses is specified when the node starts up, which represents what the user is interested in. If the parameter is empty, all the blockchain data will be stored. These addresses are referred to as 'interested addresses' in the following pages.

VI. Evaluations

In this section, we evaluate Jidard by comparing its behaviors with the original btcd. Since there are some important parameters to be tuned in the prototype, we also study the influence exerted by different values of parameters. To avoid the unnecessary wait of mining, Jidard is running in a simnet[5] mode during the evaluation process. Besides, time of block window is set as one day (i.e., 144 blocks) experientially.

A. Reduction of data storage

1) Effects of the count of addresses: A user can have multiple interested addresses maintained on a node, each of which can get involved in multiple transactions. We firstly study if different number of interested addresses may make a difference to the storage cost. We build up a chain of Jidard consisted of four nodes (i.e., node A, B, C, and D). The number of interested addresses and the number of transactions proposed by each address are listed in Table II. Each block contains one transaction proposed by each node. As a result, 1000 blocks are needed to package all the transactions. To reduce the disturbance from the false positive matches of bloom filter, number of filter bits is set as 16 in this experiment.

As addresses are managed by Jidarwallet, whose storage location is independent from that of Jidard, the storage sizes

[5]https://godoc.org/github.com/btcsuite/btcd

TABLE II: Number of interested addresses and transactions

Node	A	B	C	D
#Interested addrs	1	10	100	1000
#Txs per addr	1000	100	10	1

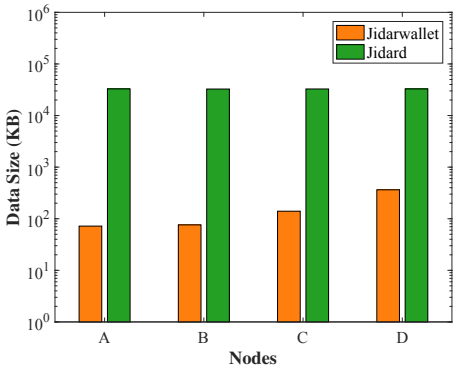


Fig. 6: Storage cost influenced by number of addresses

of Jidard and Jidarwallet are calculated independently. Experimental results are shown in Figure 6. From the results, we can conclude that the number of addresses has much fewer effects on the total storage cost, compared to that brought by transactions. As a result, we suppose that each node only maintains one interested address in the following experiments.

2) Storage reduction on Bitcoin mainnet: To evaluate how large the storage size can get reduced in the real world, we build up another chain consisted of six nodes. We reproduce, broadcast, and package all the transactions on the main network of Bitcoin (i.e., mainnet[6]) from January 9th, 2009 to December 31st, 2018 on our new chain, to simulate the real production environment as far as possible. Since the format of addresses on the mainnet is different from that on the simnet, we do a mapping of addresses. The six nodes are divided into two classes: 1) five observation nodes to be observed (i.e., A, B, C, D, E); 2) one assistance node driving the running of simnet (i.e., F). Each of the observation nodes contains an interested address mapped from the mainnet, which are involved in different numbers of transactions. These interested addresses are listed in Table III. A observation node acts exactly the same as the address on the mainnet, while the assistance node represents all the other addresses to take actions. To compare the total data storage of Jidard with btcd, we ask the assistance node to store all the blockchain data.

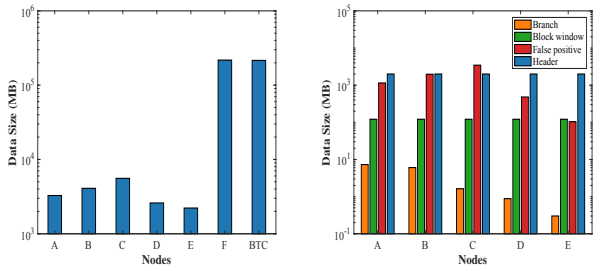
With regard to the setting of bloom filters, we set the size of filter bits as 30000 experientially to reduce the possibility of false positive matches. A detailed analysis about how to set the parameters of bloom filters is made in Section VI-B. Experimental results are shown in Figure 7, where Figure 7a compares the total storage costs of Jidard nodes with btcd nodes and Figure 7b depicts the storage costs of different parts

[6]https://bitcoin.org/en/glossary/main

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

TABLE III: Lists of interested addresses

Node	Address	#Tx
A	336xGpGweq1wtY4kRTuA4w6d7yDkBU9czU	6554
B	3D2oetdNuZUqQHPJmcMDDHYoqkyNVsFk9r	5595
C	18x5Wo3FLQN4t1DLZgV2MoAMWXmCYL9b7M	1554
D	16FSBGvQfy4K8dYvPPWWpmzgKM6CvrCoVy	794
E	3Kzh9qAqVWQhEsfQz7zEQL1EuSx5tyNLNS	296



(a) Total storage cost (b) Different parts of Jidard

Fig. 7: Comparison of storage size between Jidar and Bitcoin

in the Jidard nodes. From Figure 7a, we can conclude that the storage cost of a normal node can be reduced largely. In particular, the storage cost of node E can be reduced to about 1.03% of the original Bitcoin system. As for node F with all the data in Jidard stored, it takes 0.98% percent more in storage cost than the original Bitcoin system, which can be negligible. By comparing different parts in Figure 7b, we can find that the largest parts in storage costs are brought by headers and false positive matches. Both of these two parts are directly related to the setting of bloom filter, which is analyzed with experiments in Section VI-B.

B. Effects of the size of bloom filter bits

The size of bloom filter bits can influence the storage cost from two aspects: 1) direct influence on the size of block headers; 2) indirect influence on the false positive rate, which leads to extra storage of complete blocks. There are three parameters related to the setting of a bloom filter. In the context of Jidar, these three parameters are the number of filter bits, the number of hash functions, and the number of inputs in a block respectively. According to Equation 1 [12], given the number of inputs in a block and the tentative number of filter bits, the optimal number of hash functions can be calculated. Besides, the larger is the number of UTXOs maintained by a node, the more is the false positive matches. As shown in Figure 7b, node A runs into much more false positive matches, since it maintains a large number of UTXOs. In this section, we dig into the influence exerted by the size of filter bits, with two factors changing. We analyze the effects with number of UTXOs changing in Section VI-B1, and the effects with number of inputs changing in Section VI-B2. It should be noted that the costs of block window are removed in the following experiments, since the costs of block window in different Jidard nodes are the same.

$$k = \frac{m}{n} \cdot \ln(2) \tag{1}$$

TABLE IV: Number of UTXOs maintained by observation nodes

Node	A	B	C	D
#UTXOs	10	100	1000	10000

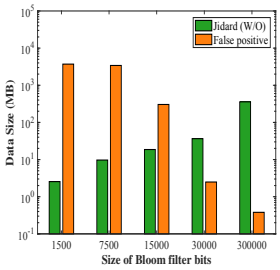
where m, n, and k represent the numbers of filter bits, inputs in a block, and hash functions respectively.

1) Effects with number of UTXOs changing: We make up a chain consisted of five nodes, four of which are observation nodes (i.e., A, B, C, and D) while the other one is the assistance node (i.e., E). At the initial phase of the experiment, node E proposes massive transactions to send coins to the observation nodes, thus creating specific number of UTXOs on each observation node. To mitigate the effects at the initial phase, massive transactions are packaged in very few blocks (i.e., 10 blocks in this experiment). The number of UTXOs maintained by each observation node is listed in Table IV. After the initial phase, node E proposes 1500 transactions every minute, each of which has only one input. All the 1500 transactions will be packaged into a block and then be broadcast to the observation nodes. In order to keep the number of UTXOs in the observation nodes unchanged, all the new transactions are proposed by node E and have nothing to do with the observation nodes. The number of inputs in a block is set as 1500. Five experiments with different size of bloom filter bits are conducted, and number of hash functions is calculated by Equation 1 too. In each experiment, node E proposes 10000 blocks after the initial phase.

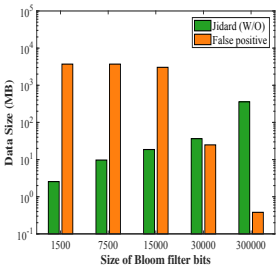
The experimental results are shown in Figure 8. Storage costs brought by false positive matches are calculated separately, while the other parts of Jidard system are calculated together (including the block headers), namely Jidar (W/O) in Figure 8. From the figure, we can find that the larger is the filter size, the fewer is the false positive matches while the larger is the costs of block headers. However, even the size of bloom filter bits is set very small, the extra storage brought by false positive matches has an upper bound, which is the size of the total block data. As for the node with massive UTXOs (i.e., node D), the slight increase from 1500 to 30000 can bring little benefits to reduce the false positive matches. However, if the size of bloom filter bits is set large enough to bring considerable reduction in false positive matches, it will lead to large costs of block headers, which is unfair for other nodes.

To sum up, if most of the nodes on the blockchain maintain only a small number of UTXOs, the size of bloom filter bits should be set as a small value, thus getting the cost of block headers reduced. On the contrary, if most of the nodes maintain massive UTXOs, the size should be set larger. According to our analysis of Bitcoin data, the overwhelming majority of users have UTXOs fewer than 100, which inspires us to set a moderate size of bits (e.g., 30000). As for the users with massive UTXOs, it is reasonable for them to store all the blockchain data, which has a upper bound.

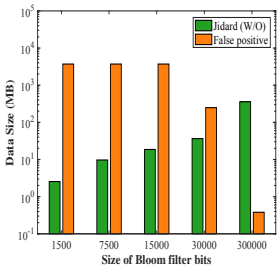
Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.



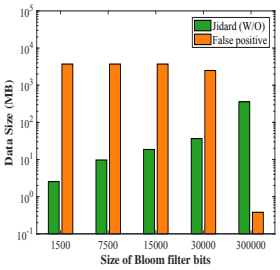
(a) Node A with 10 UTXOs



(b) Node B with 100 UTXOs

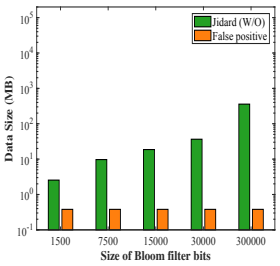


(c) Node C with 1000 UTXOs

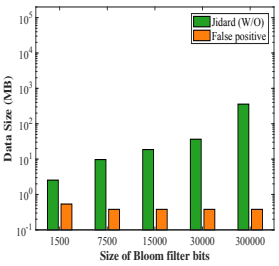


(d) Node D with 10000 UTXOs

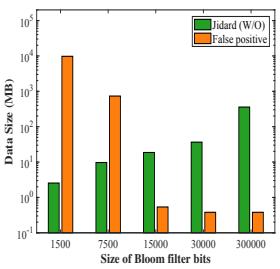
Fig. 8: Effects brought by different sizes of bloom filter bits with number of UTXOs changing



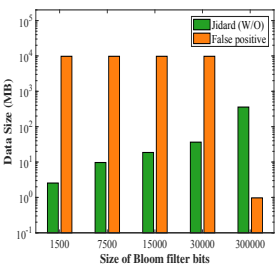
(a) Group 1 with 5 inputs



(b) Group 2 with 50 inputs



(c) Group 3 with 500 inputs



(d) Group 4 with 5000 inputs

Fig. 9: Effects brought by different sizes of bloom filter bits with number of inputs changing

TABLE V: Number of inputs in different experiment groups

Group	1	2	3	4
#Inputs	5	50	500	5000

2) Effects with number of inputs changing: To analyze the effects brought by different sizes of bloom filter bits with number of inputs changing, we keep the number of UTXOs in a node unchanged. Without loss of generality, we set the number of UTXOs as 100 in this section. We make up a chain consisted of two nodes, namely node A (the observation node) and node B (the assistance node). With the change of the number of inputs in a block, four groups of experiments are conducted, as listed in Table V. To ensure the sizes of blocks with different inputs are the same, more outputs are added to the transactions in a block if the number of inputs is small. The initial phase creating 100 UTXOs in node A by node B is similar to that in Section VI-B1. After the initial phase, node B broadcasts a block every minute. All the blocks have the close size, while the inputs of blocks in different experimental groups are different. The number of hash functions is set according to Equation 1 too, and 10000 blocks are proposed in each experiment.

The experimental results are shown in Figure 9. From Figure 9a and Figure 9b, we can find that the increase in size of bloom filter bits brings little profits to the groups with few inputs, since their false positive rates are already very low. Similar to the phenomenon in Section VI-B1, on the one hand, slight increase of filter bits has few effects on the group with massive inputs, as shown in Figure 9d. On the other hand, excessively large size of filter bits can bring considerable reduction to false positive rates, but it also leads to large storage costs of block headers. Since the numbers of inputs in the vast majority of blocks are smaller than 1000, the size value of bloom filter bits is recommended to set as 15000 or 30000, which is consistent with the conclusion in Section VI-B1. Besides, the size of bloom filter bits may be adjusted dynamically according to the number of inputs in the recent blocks, which is similar to the adjustment of mining difficulty. The dynamic adjustment of bloom filter will be included in our future works.

C. Communication Cost

In reality, Jidard reduces the storage overhead at the cost of higher network overhead. In this section, we make a quantitative analysis of communication cost brought by Jidard. Since the size of bloom filter bits will affect the false positive rate, thus changing the size of proof in TUs, we conduct multiple experiments with the size of bloom filter bits changing. The initial configuration of the nodes is similar to that in Section VI-B1. The differences include: 1) new transactions after the initial phase are proposed by the observation nodes; 2) to avoid the bad influence exerted by each other, the transactions are proposed by the single observation node in each experiment; 3) to keep the number of UTXOs in a observation node unchanged, the observation node makes transactions with itself as the receiver; 4) mining time is set as ten minutes to simulate the real production environment and duration of each experiment is set as ten hours. Two groups of experiments are conducted, with the number of newly proposed transactions different. The speed of transactions proposed in the first experimental group is set less than 4 (e.g., 3 in our

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.

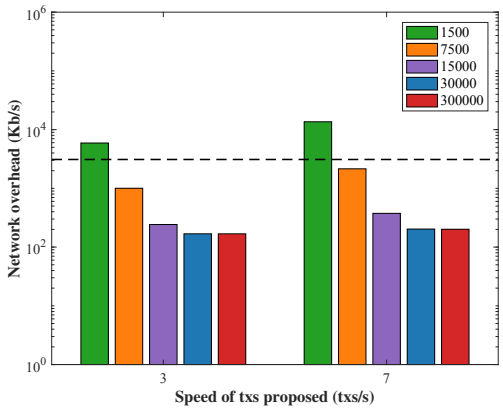


Fig. 10: Communication costs

experiment), according to the statistics of mempool status[7]. The speed of transactions proposed in the second group is set as 7, which is the maximum TPS in the Bitcoin system.

The experimental results are shown in Figure 10. From the figure, we can find that the larger is the size of bloom filter bits, the smaller is network overload. Besides, the larger is the size of bloom filter bits, the smaller is the increase in network overhead brought by the increase in the speed of transactions proposed. In particular, the network overhead is increased by 130% when the size of filter bits is 1500, while it is increased by 19% when the size is 300000. Since the lower bound on most nodes' bandwidth is about 3.03Mbps [15], which is shown as the black dashed line in Figure 10, the size of bloom filter bits should be set larger than 7500. In other words, if the size of bloom filter bits is set larger than 7500 (e.g., 30000), the network overhead brought by Jidard is negligible and can be acceptable by almost all the normal nodes.

VII. Conclusion

The goal of this paper is to present the design of Jidar to address the storage problem of the Bitcoin system without sacrificing the original characteristics of Bitcoin system. By only storing its relevant data, which is like selecting pieces from the jigsaw puzzles, storage cost of a normal Bitcoin node gets reduced largely. For others to verify the reliability of a newly proposed transaction, the transaction proposer must broadcast the proof along with the transaction. Experimental results demonstrate that Jidar can reduce the storage cost to about 1.03% of the original Bitcoin system at a low communication cost. It should be noted that Jidar can be extended widely to other blockchain systems adopting a Merkle tree or its variation.

Acknowledgment

This work is supported by National Science Foundation of China under Grant No. 61702203, Hubei Provincial Natural Science Foundation No. 2018CFB133. Jiang Xiao is the corresponding author of the paper.

[7]https://blockchair.com/bitcoin/

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf/, 2008.
[2] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," *ACM SIGMOD Record*, vol. 25, no. 2, pp. 173–182, 1996.
[3] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2016, pp. 45–59.
[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.
[5] Z. Xu, S. Han, and L. Chen, "Cub, a consensus unit-based storage scheme for blockchain system," in *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering*. IEEE, 2018, pp. 173–184.
[6] S. Wang, T. T. A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, B. C. Ooi, and P. Ruan, "Forkbase: An efficient storage engine for blockchain and forkable applications," in *Proceedings of the VLDB Endowment*. VLDB Endowment, 2018, pp. 1137–1150.
[7] A. Dinh, J. Wang, S. Wang, G. Chen, W.-N. Chin, Q. Lin, B. C. Ooi, P. Ruan, K.-L. Tan, and Z. Xie, "Ustore: a distributed storage with rich semantics," *arXiv preprint arXiv:1702.02799*, 2017.
[8] M. Samaniego and R. Deters, "Blockchain as a service for iot," in *Proceedings of 2016 IEEE International Conference on Internet of Things and Green Computing and Communications and Cyber, Physical and Social Computing and Smart Data*. IEEE, 2016, pp. 433–436.
[9] Y. Guo and C. Liang, "Blockchain application and outlook in the banking industry," *Financial Innovation*, vol. 2, no. 1, p. 24, 2016.
[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[11] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.
[12] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," *Random Structures & Algorithms*, vol. 33, no. 2, pp. 187–218, 2008.
[13] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 75–88.
[14] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 604–612, 2002.
[15] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. G. Sirer, "On scaling decentralized blockchains," in *Proceedings of International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 25,2020 at 14:30:15 UTC from IEEE Xplore. Restrictions apply.