BZIP: A Compact Data Storage System for UTXO-based Blockchain

No Author Given

No Institute Given

Abstract. Unspent Transaction Output (UTXO) set is the foundational model used in many blockchain systems to represent assets. The benefits of UTXO representation include parallel processing, privacy, etc. However, the increasing size of UTXO set is degrading the access performance and severely brings down the validation speed of blockchain further. In this paper, we present a compact storage system for UTXO-based blockchain. Taking Bitcoin as the object of study, we propose two lossless compression techniques to reduce the memory space occupied by UTXO set. Compared with current UTXO database, our mechanism can deliver 2.9-4.5x memory reduction safely. To match with current blockchain system, the database related operations are adopted to make the proposed mechanism easily applied. This compact storage system will improve the performance of current blockchains on verification speed, economic cost and scalability.

Keywords: UTXO · Blockchain · Database.

1 INTRODUCTION

Blockchain, which derives from Bitcoin [1], is regarded as a promising technology serving future economic and social systems because of its trustworthiness and security [2, 3]. Nodes in the same blockchain can trade without help of third-part central administrator anonymously and safely. The mechanism behind this is a distributed trading ledger preserved by blockchain nodes. The ledger only accepts the transaction that is validated and confirmed by most of nodes.

UTXO set [4] is the foundational model used in many blockchain systems, such as cryptocurrencies. UTXO set is the set of all historical transactions which are not spent yet and its up-to-date copies are typically stored in the key-value database of participating nodes. These UTXOs are meant for validating new transactions. When a new transaction is created, the creator needs to provide one or more unspent transactions as inputs to pay. Then nodes check whether the provided inputs are in its UTXO set and decide to accept or refuse this transaction. Therefore, the access performance of UTXO set directly determines node validation speed.

The increasing size of UTXO set, however, is becoming the biggest obstacle for validation efficiency. The main reason is that increasing size of UTXO is pushing its storage device from DRAM to lower-speed disk. Because of the difference between DRAM access time and disk seek time, the UTXO access time would consume from 17 us to 10 ms [5], which leads to hundreds of degradation on validation speed. On the other hand, the requirement to nodes gets higher because of increasing UTXO size. A qualified node has to have more DRAM to accommodate whole UTXO set. That would increase economic cost and decrease the percentage of qualified nodes. For example, the size of Bitcoin's UTXO set has grown beyond 3GB, while a middle-end PC usually only has 4-8 GB DRAM. More importantly, this size is achieved under very low TPS (transactions per second) situation, i.e. seven tps. As a comparison, PayPal handled 7.6 billion transactions in total in a year and tens of thousands of TPS at most [6]. If the trading frequency of Bitcoin catches up with that of PayPal, the size of UTXO would exceed the storage capacity of general PCs.

To answer the question how to relieve and even eliminate the effect of UTXO storage problem on validation efficiency, this paper proposes a compact storage system named UTXO-Virtual (abbr. UTXO-V) for UTXO-based blockchain to minimize the footprint of unspent transactions. Firstly, we target on data of *key* domain and design a shorter representation form of key. The rationale behind this method is the fact that the UTXO size is just a small subset of the space that the keys are able to represent. Secondly, in value domain, we observe that one user usually has many UTXOs but the same user information is repeatedly stored in each entry. Accordingly, a two-level hash mapping technique is proposed for eliminating the redundant data in value domain. What's more, we adopt the read and write operation of database to match existing blockchains and avoid other modifications outside the database. In this paper, we target on the biggest UTXO-based blockchain, Bitcoin. Most of the other blockchain systems are based on Bitcoin, so our proposal is easily to implement on them.

In this paper we make following contributions. We present a compact storage system with two lossless compression techniques to decrease the data size of UTXO set up to 4.5x. Further, it will help to place UTXO set in DRAM other than disk so as to improve validation efficiency, economic cost and scalability of UTXO-based blockchains. Besides, Our proposal is transparent to the upper blockchain applications and well-matched with existing blockchains.

This paper is organized as follows. Section 2 justifies the motivation and gives a overview of related work. Section 3 describes the redundancy problem in current model. Section 4 describes the UTXO-V system. Section 5 gives experimental results and discussions. At the end, section 6 concludes.

2 MOTIVATION AND RELATED WORK

Blockchains like Bitcoin have no real "balance" for users/accounts. Instead, they use previous unspent transactions as input coins for users' payments and validation. To avoid using all historical transactions to verify transactions, UTXO set, which is minimal set for valid verification, is invented. The UTXO model enhances validation efficiency and also presents several benefits in terms of privacy , parallel process and potential scalability paradigms. For example, a user

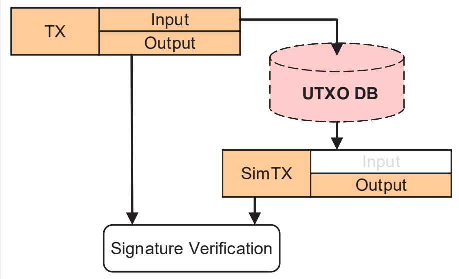


Fig. 1. Overview of the transaction validation.

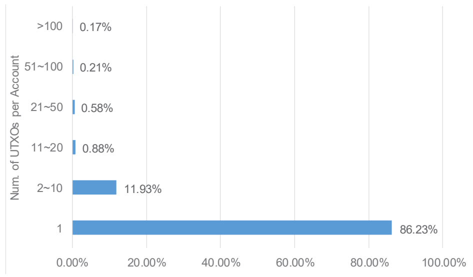
can create a new account for each received transaction to avoid others' tracking. Another advantage is that removing partial data on chain on purpose for scaling won't cripple the whole system [7].

A typical validation procedure of UTXO-based blockchain is shown in Fig. 1. A transaction (abbr. TX) consists of two parts, input and output. Input part mainly includes information of input transactions to pay and signature, while output part includes user account address for verifying signature, coin value and so on. For efficiency, the UTXO set is loaded in a key-value database. When the node receives a new transaction, it will query the UTXO database by keys in input part to check if input transactions are in UTXO set. If so, UTXO database will return a simplified output part of input transactions for following signature verification.

The validation speed is dominated by the access time of UTXO database and time of verifying signature. According to benchmark in Bitcoin [8], the typical UTXO access time on DRAM is 20us and signature verifying time on CPU is about 100 us, which are sufficient to verify 10k transactions per second. However, if we access the UTXO database from disk, the access time will increase to 10 ms which results in only 100 verified TPS and dominates the validation speed.

There are several pitfalls that lead to underestimation on the effect of increasing UTXO size.

Pitfall 1: Since TPS value of most blockchains are very low, the validation speed of 100 tps is good enough. Despite low throughputs of blockchains at this stage, there still is a place for high verification speed. Miners can't start mining on the new best chain until they have fully validated new block. So the mining node of blockchain needs faster validation speed to gain advantage over others. Besides, current blockchains have large space to keep up with actual trading systems, for example, VISA can handle 2,000 tps in



Num. of tx	24,569,579
Num. of UTXOs	50,447,816
Size of UTXO set	3,10 GB
Num. of accounts	18,490,032
Avg num. of UTXO per account	2.21
Avg num. of UTXO per account (remove silent accounts)	9.80

Fig. 2. Analysis on Bitcoin's UTXO set.

average and 56,000 tps at the peak. From the practical perspective, about 10,000 validation speed is necessary for future blockchain systems.

Pitfall 2: The increasing tendency of UTXO size has become slow recently, so it won't bother us in the future. We find that the increasing tendency is not to slow automatically but limited by current throughputs. Taking Bitcoin as example again, We choose the UTXO set until June 2018 and analysis the relation between number of UTXO and user accounts, as shown in Fig. 2. we are surprised to find that over 80% users haven't ever traded since they received their one and only UTXO. We call these accounts, who barely participant in the whole system, as "*silent accounts*". Such high proportion of silent accounts implies that current Bitcoin is far from a fluent system. If we remove the silent accounts, the number of UTXO that each account holds will be 9.8 in average. Based on this, the UTXO size of a fluent Bitcoin will expand to **13.7GB, 4 times** what it is today.

Pitfall 3: Existing caching techniques can handle bigger size of UTXO set. Caching techniques are efficient to enhance access speed of UTXO database by reserving outputs that were created recently in cache. However, This kind of methods trade-off QoS of users who trade less frequently for average QoS. Moreover, the situation that the proportion of silent accounts is large are more beneficial for the caching technique, so this technique may fail in more active-trading systems.

In industry domain, UTXO set are usually stored in a simplified format to reduce its footprint [3, 4]. The actual database only maintain the necessary data for indexing and signature verifying. Another technique is aggressive caching [8][9], i.e. putting as more UTXOs in DRAM as memory device can afford. These techniques have applied in latest blockchains but failed to solve the problem completely. Recently researchers invent the analysis tools [4][7] for UTXO set and find there existing many dust outputs or unprofitable outputs, whose values are below the fee to pay miners. However, these work's proposals are more closer to detail analysis and potentially useful suggestions than a complete solution. Besides, using other candidate models such as account-based model[10, 11] is incompatible with existing systems and less secure.

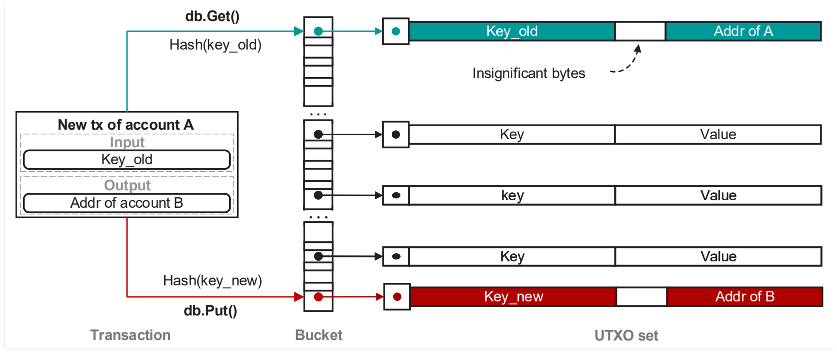


Fig. 3. The working process of database in blockchains.

3 ANALYSIS on REDUNDANCY of UTXO DATABASE

The UTXO database stores key-value pairs for the node to read and write unspent transaction records. the descriptions of key and value are as follows:

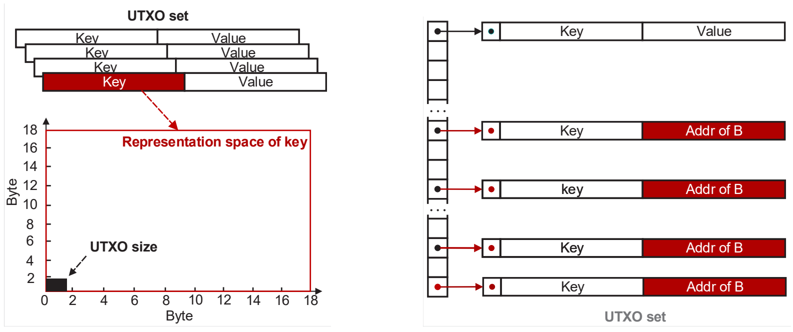
- *key*: Blockchains use the transaction ID and an index as its *key*. Transaction ID is a 32-byte hash value typically derived from SHA-2 encryption of transaction data. The index is a 4-byte value indicating the location of target output in transactions.
- *value*: The *value* consists of several types of data for transaction validation, including 20-bytes account address, 2-bytes coin value, 3-bytes block height, etc. Among these, account address dominates the storage of value domain, which occupies over 70%, so we call the other bytes ”insignificant bytes”. The total byte length of one value is 28 byte.

Fig. 3 illustrates the working process of database when a new transaction is created and then becomes a new unspent transaction. Suppose the user A creates a transaction to pay the user B, A will include the key related to an old transaction which he wants to spend in the input part and add the account address of user B in the output part to prevent others spending this new transaction. Then the validation process and possible update process are completed through the interaction with key-value database. The main related operations in database are *db.Get()* and *db.Put()*:

- *db.Get()*: This operation is responsible to read data from database according to the provided key. Firstly, it use fast hash function to map the key to its bucket location in database and then check whether a record exists. If so, stored key will be compared with input key to handle collisions. At last, the data in value domain is returned for the following transaction validations.
- *db.Put()*: If the new transaction is validated, it will be written into database as a new unspent output, which is completed by db.Put(). Through hash mapping, db.Put() writes the new key and value record into corresponding bucket location. If collisions occur, the record will be written in another location based on collision resolution strategies such as open addressing and chaining.

3.1 Key Domain Redundancy

The transaction ID and index which key stores represents and determines an unique identity of the transaction output. However, such representation doesn’t take the status of transaction outputs into account, that is to say, such a transaction output may have been spent or have not been created yet. This representation method is imperative to distinguish certain output in all transaction outputs space but unnecessary in UTXO database. Because UTXO database only need to store the present unspent transaction outputs, which is actually a very small set of all transaction outputs.



(a) Key domain (b) Value domain

Fig. 4. The redundancy problem in UTXO database.

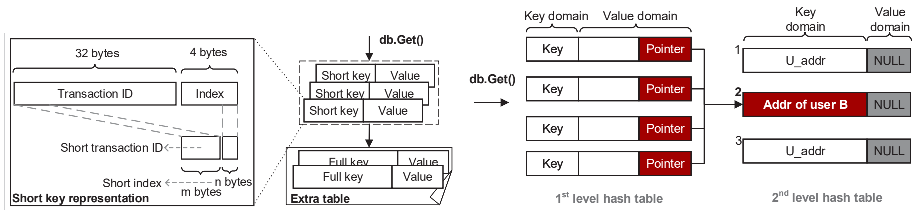
Fig. 4 (a) shows the difference between the size of UTXO set and representation space of key in 2D space. The word length of key is 36 bytes, which are capable of standing for 2^{288} unique transaction outputs. As a contrast, The max UTXO size ever is about 50 million, which ideally can be represented by only 4 bytes, only 1/9 of key length.

3.2 Value Domain Redundancy

The redundancy in value domain comes from duplicated records of the same account address. From the analysis of Bitcoin (Fig. 2), Every account holds 2.21 UTXOs in average, and if we remove the silent accounts, this number will increase to almost 10 utxos/account. Because of the NoSQL property of key-value database, different UTXOs with same account address are regarded as totally irrelevant records and allocated memory separately. As shown in Fig. 4 (b), It causes that the same account address has multiple copies. Considering the high storage proportion of account address in value domain, these redundancy copies burden the storage system heavily.

4 COMPACT STORAGE SYSTEM DESIGN

4.1 short representation design



(a) Short key representation (b) Short value representation

Fig. 5. The design of short representations in UTXO database.

Short representation of key Using a shorter representation of key is an intuitive but risky method. The challenge is that short representation brings "collision hazard", in which different keys lie in the same bucket after hash mapping, and simultaneously, have the same short representation. Because the key is meant for collision resolution, the same short representation of different keys would result in database failing to detect possible collisions.

We propose a simple but quite efficient representation design of key, as illustrated in Fig. 5 (a). The proposed representation also consists of two parts: short transaction ID and short index. The short transaction ID takes highest m bytes of the original transaction ID while the short index selects lowest n bytes of the original index. The transaction ID is a hash value, so the selected location of transaction ID is trivial. The index value is usually small, so we take the least bytes of index.

Furthermore, an extra table is presented to detect possible collisions. We add all the records with collision hazard in this extra table so that the possible collisions will be detected in this table. In this section We will demonstrate the memory overhead of the extra table is very low firstly, then the detail detection process will be explained in section 4.2.

The size of extra table is proportional to number of conflicting records. So the target is to evaluate the expected number of collisions that fail to be detected by database after hash mapping and short representation. The related parameters are defined in Table. 1. The target problem can be divided into two traditional collision problems, which is well-studied and concluded as the birthday problem. The two collision problems are as follows:

- If n_{ht} UTXOs is hashed into n_{hs} slots, what the expected number of collisions, i.e. $E(\xi_h)$, will be.
- If $E(\xi_h)$ conflicting records use short representation $\{n_{st}, n_{sx}\}$ and all lie in the same slot, what the expected number of collisions, i.e. $E(\xi_{sh})$, will be.

According to the formula on excepted number of collisions of birthday problem, the results of above sub-problems are:

$$E(\xi_h) = \frac{C^2_{n_{ht}}}{2^{n_{hs}}} \tag{1}$$

$$E(\xi_{sh}) = \frac{C^2_{E(\xi_h)}}{2^{n_{st}+n_{sx}}} \tag{2}$$

Referring to design of Bitcoin system, we estimate the $E(\xi_{sh})$ by assigning the values of parameters as follows: $n_{ht} = 50,447,816$, $n_{hs} = 32$, $n_{st} = 24$ (i.e. 3 bytes), $n_{sx} = 8$ (i.e. 1 bytes). Substitute these parameters into Eq. 1 and Eq. 2, then we get $E(\xi_{sh}) \approx 32$. Therefore, theoretically an extra table capable of 32 full records would be collision-resoluble for 4 bytes short key representation.

Table 1. variable notation

Name	Description
ξ_h	Num. of collisions among keys with hash mapping
ξ_{sh}	Num. of collisions with both short representation and hash
n_{ht}	Num. of unspent transaction outputs
n_{hs}	Log base 2 of num. of slots assigned in database
n_{st}	Bit length of short representation of transaction ID
n_{sx}	Bit length of short representation of output index

Short representation of value Key-value databases are incapable of merging redundant copies of user addresses. However, they allows building a hash table with null values. So we present a two level hash table to reduce the redundancy data in value domain, as shown in Fig. 5 (b). Every unique user address is stored as a key in the second table whose value domain is set to null. In this way, we remove unnecessary copies, meanwhile, the read and write speed of user addresses keeps fast. As for the first hash table, we fill the value domain with a pointer pointing to user address data so that we can acquire user address based on key.

The memory overhead mainly comes from the footprint of pointers. Each point would occupy 4 bytes or 8 bytes depending on underlying architecture. We set this footprint as 8 bytes conservatively and assume the average number of utxos per user holds is x. Besides, the bit length of user address is known as 20 bytes. Then the ratio between the memory overhead and the original footprint would be $(2x+5)/5x$, which is less than 1 when $x > 1$ and smaller as x increases.

4.2 database operations adoption

To match current blockchain systems, we adopt the read and write operations of database by adding new functions to db.Get() and db.Put().

Suppose provided key is K and the desired value is V, the new $db.Get(K, \&V)$ completes following procedure:

1. By hashing K, access the target bucket in the first level hash table. If the short representation of K doesn't match with the stored short key, terminate the procedure. Otherwise, acquire the data (including pointer and insignificant bytes) in value domain, and go to step 2.
2. Check if K matches with any keys stored in extra table. If so, assign corresponding value to V and terminate. Otherwise, go to step 3.
3. According to pointer from step 1, access the target user address in the second level hash table. Then assembly the insignificant bytes and user address as V, return V and terminate.

Similarly, suppose provided key is K and provided value is V, the new $db.Put(K, V)$ completes following procedure:

Table 2. platform information

Part Name	Information
Bitcoin Core	Version 0.16.1
LevelDB	Version 1.20
Keys	36 bytes
Values	28 bytes
Entries	40,948,331
CPU	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
Cache Size	25600 KB
DRAM	32 GB

1. By hashing K, access the target bucket of the first level hash table. If the bucket is empty, fill short representation of K in key domain and go to step 3. Otherwise, go to step 2.
2. Compare the short representation of K with stored short keys in the same bucket, if there are no collisions, acquire a suitable bucket to fill, go to step 3. Otherwise, go to step 4.
3. Decompose the V into insignificant bytes and user address, fill the insignificant bytes, go to step 5.
4. Store the K, V in the extra table and terminate the procedure.
5. By hashing user address, access the target bucket of the second level hash table. If the target bucket is empty, fill the user address in key domain. Assign the bucket location to the pointer in the first level table. Terminate.

Compared to traditional counterpart, the adopted *db.Get*() increases the operation time on access to extra table and the second hash table. On the other hand, the adopted *db.Put*() increases the operation time on comparison with possibly conflicting keys, write to extra table and access to the second hash table. The time overhead of the modified operations mainly comes from multiple access to hash table. Thanks to the small size, the read and write time of extra table would be quite low as long as we constrain the number of collisions small.

5 EVALUATIONS

The experiments are performed on Bitcoin Core 0.16.1, which employs levelDB v1.20 [5] to store UTXO set. We collect the UTXO data before July 2018 and generate bigger data based on it for evaluation. The detailed information is listed in Table. 2. We compare our proposal with the original UTXO model on occupied memory and access performance. We also evaluate the effects of short representation of key and value respectively and jointly.

5.1 Memory Reduction

Fig. 6 illustrates the data size of various solutions with different UTXO quantities, whose number of utxo per user (abbr. utxos/user) are fixed at 2.21 and

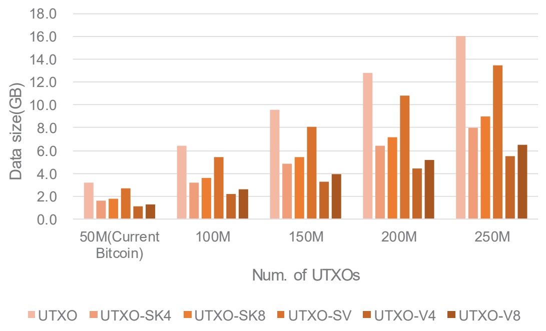


Fig. 6. Comparison on occupied memory among different solutions.

consistent with current Bitcoin. UTXO-SK4 means we use 4 bytes short keys and the original value domain. Similarly, UTXO-SK8 employs 8 bytes short keys. In UTXO-SV, the short value representation is employed with full key representation. The last two solutions combines UTXO-SK4 and UTXO-SK8 with UTXO-SV, respectively.

As shown in Fig. 6, the proposed method can reduce the memory size of current Bitcoin by about 2.9x and are scalable for different data size. Limited by the low value of utxos/user, the short value representation is less efficient than short key representation and only contributes 1.2x memory reduction. However, if we increase the utxos/user from 2 to 10, we find that the short value representation reduces the data size further, as indicated in Fig. 7. The compression ratio is up to 4.5 when the utxos/user is 10, which is close to the situation of active users in Bitcoin. It implies we can reduce the memory size from 13.7G to 3.0G, which enables current nodes to achieve higher validation throughput without any hardware updates. In addition, it's also helpful to the implementation of blockchain on IoT devices.

5.2 Performance

The adoptions on read and write operation increase the execution time. The increased time mainly comes from related operations to the extra table and the second level table. The extra table are closely relevant to short key representation. Therefore, we collect the actual collisions of database under different representations of the key above all. An interesting phenomenon is that the

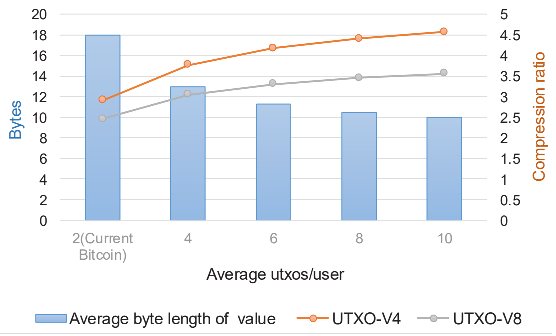


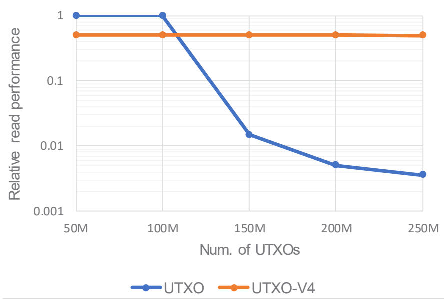
Fig. 7. The impact of utxos/user on compression ratio and value size.

number of detected collisions are less than the theoretical value. For example, there are 32 collisions for four-byte key representation in theory while the testing value is zero. The reason is that we make a conservative assumption in which all conflicting records lie in the same bucket. However, such situation are hard to happen in reality. We find that all the buckets have no more than three collisions in the above example.

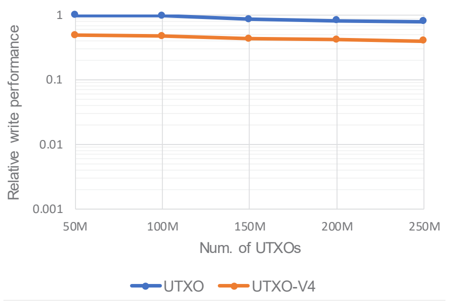
Table 3. COLLISIONS UNDER DIFFERENT KEY REPRESENTATIONS

Collisions	Number of UTXOs					
	10M	**50M**	**100M**	**150M**	**200M**	**250M**
1B	464	**1924**	3956	8012	16145	**33004**
2B	15	**77**	152	243	310	**475**
4B	0	**0**	0	0	4	**7**
8B	0	**0**	0	0	0	**0**

We limit the DRAM size as 8GB and then evaluate the access performance by feeding one million key-value pairs to the adopted database. Fig. 8 shows the relative read and write performance between two cases: the orignal UTXO method and UTXO-V4. We find that the read performance of original method faces dramatic degradation when UTXO size exceeds 100 million. The reason is that large UTXO size makes parts of data stored in low-speed disks. On the other hand, the UTXO-V4 method benefits from data compression and low collision rate so it keeps stable access performance with varying UTXO sizes. In fact, The adopted operations lead to about 2x delay for both read and write performance. However, because the access time of original operations is fast enough, the 2x delay is acceptable and still sufficient for tens of thousands of operations per second. In summary, the proposed system would supply more stable and faster access speed for big UTXO set and sufficient access speed for small UTXO set.



(a) Read operation



(b) Write operation

Fig. 8. Read and write time performance of UTXO-V.

6 CONCLUSIONS

In this paper, we propose an efficient and compact storage system for UTXO-based blockchain. We present two lossless compression method to represent key and value domain of UTXO database with less memory. To accommodate existing blockchain database, the read and write operations of database are adopted with little overhead. The experiments on Bitcoin shows that we can deliver 2.9-4.5x reduction on data size at the cost of 2x time delay, which is still sufficient for current system and more beneficial to larger UTXO set. The achieved benefits can deliver higher validation throughput, reduce cost of blockchain nodes and enable broader application scenarios.

References

1. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
2. K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security.* Springer, 2016, pp. 106–125.
3. T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.
4. S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartı, "Analysis of the bitcoin utxo set," in *Proceedings of the 5th Workshop on Bitcoin and Blockchain Research Research*, 2018.
5. J. D. Sanjay Ghemawat, "Leveldb is a fast key-value storage library written at google," https://github.com/bitcoin/bitcoin/tree/master/src/leveldb, accessed Sep 16, 2018.
6. PayPal, "2018 annual meeting of stockholder and proxy statement and 2017 annual report," https://investor.paypal-corp.com/eventdetail.cfm?eventid=188727, accessed Sep 16, 2018.
7. C. Pérez-Sola, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartı, "Another coin bites the dust: An analysis of dust in utxo based cryptocurrencies," 2018.
8. "Bitcoin core v0.16.2," https://github.com/bitcoin/bitcoin/tree/master/src, accessed Sep 16, 2018.
9. Y. Sakakibara, K. Nakamura, and H. Matsutani, "An fpga nic based hardware caching for blockchain," in *Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies.* ACM, 2017, p. 1.
10. G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
11. C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, vol. 310, 2016.