Block Pruning with UTXO Aggregation

Taegyu Song*, Kazuyuki Shudo*†
*Tokyo Institute of Technology, Tokyo, Japan
†Kyoto University, Kyoto, Japan

***Abstract*—The ledger of a blockchain is designed to be an append-only data structure, which means that its size only increases. Therefore, to run a node, it is a burden to the node operator to store the entire blockchain ledger. We propose a new method, UTXO aggregation, to reduce the high occupancy of storage. In this method, UTXOs are periodically merged, and a special block including all merged UTXOs is generated. Unlike Bitcoin's block file pruning, this method can allow other nodes to bootstrap and does not require a separate database that stores only UTXOs. This method also has the effect of reducing UTXOs with low economic value. By applying this method to Bitcoin, the storage occupancy of running a node can be immediately reduced to approximately 1/100, and an economic effect of up to 14120 BTC can be expected.**

***Index Terms*—blockchain, block pruning, Bitcoin, storage occupancy**

I. Introduction

Blockchain technology distributes, preserves, and manages data of blocks connected like a chain by hash values by copying them to multiple computers around the world in peer-to-peer protocols. A block contains several online transaction records. In other words, the blockchain is a management technology for a vast distributed ledger that contains all transactions. The blockchain has the following characteristics and problems.

The blockchain assumes a large number of users, and its ledger contains all valid transactions, so a large storage space is required to operate a node. The ledger's size continues to increase because it has an append-only data structure.

In addition, metadata of the ledger and data related to node running also occupy storage. For example, Bitcoin Core is an implementation of Bitcoin and its database system [1], and the internal layout of UTXO set [2] was changed to query data efficiently and bootstrap new nodes.

The node sizes of Bitcoin and Ethereum [3], which are representative blockchains, exceed 383 GB and 1146 GB, respectively. Fig. 1 shows how much size the Bitcoin and Ethereum (Geth) nodes occupy on local storage, and the growth rate of their size is increasing.

In major blockchains, nodes basically store the entire ledger, and the fact that the size of the ledger has grown too large is a burden for many node operators. As the storage occupancy of the running node increases, it becomes difficult to increase the number of node operators. This hinders decentralization of the blockchain.

The blockchain community has suggested various methods to solve this storage size problem. There are methods such as the introduction of sharding, which groups blockchain participants and stores only the ledger of some participants, the proposal of a node type that maintains only a minimal structure for running nodes, and placing a trusted database outside the blockchain network. As an example of a node type that maintains only a minimal structure, Bitcoin's light node [4] does not store the ledger but outsources the validation of blocks and transactions to other nodes and uses SPV [5] to manage the wallet locally.

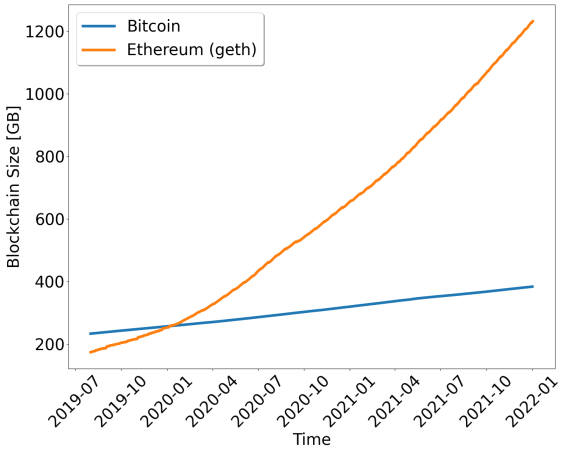

Fig. 1. Storage occupancy of Bitcoin and Ethereum (Geth) nodes.

Bitcoin introduced block file pruning [6], which removes the raw binary data of blocks in the past to alleviate the storage size problem, but it contains other problems. A node with block file pruning cannot help other nodes bootstrap. There is a concern that the node cannot inquire about the block data referenced by some UTXOs and that the node is forced to maintain a separate database storing UTXOs. In this paper, we propose UTXO aggregation, a method that overcomes the limitations of Bitcoin's block file pruning.

This proposal was devised under the influence of the period-oriented settlement system that is frequently used in practice. UTXOs are regularly merged on a user basis, and a special block containing all merged UTXOs, called the aggregation block, is generated and added to the blockchain. This method not only reduces storage occupancy but also has the effect of reducing the number of UTXOs with low economic value.

This paper is organized as follows: In Section 2, we explain the background of Bitcoin. In Section 3, we introduce Bitcoin's block file pruning and related works. In Section 4, the proposed method is discussed. In Section 5, we describe the

Proc. 5th IEEE Int'l Conf. on Blockchain
(IEEE Blockchain 2022), August 2022

requirements for introducing the proposed method to Bitcoin. In Section 6, experiments in Bitcoin of the proposed method and the results are described. In Section 7, we describe the conclusion of this paper.

II. Preliminary

This section describes the data storage and transaction validation of Bitcoin Core, which is the implementation of Bitcoin.

A. Bitcoin's data storage

Bitcoin Core stores the following four types of data or databases in local storage:

- `blocks/blk*.dat`
- `blocks/index/*`
- `blocks/chainstate/*`
- `blocks/rev*.dat`

`blocks/blk*.dat` is block data contained in the Bitcoin ledger and is dumped into storage as raw binary data. These data are used to rescan to detect missing transactions from the wallet, to reorganize the chain when a fork occurs and to provide block data to other nodes. `blocks/index/*` stores metadata for all blocks and the location of block data on storage and is implemented as a key-value store, LevelDB [7]. `chainstate/*` stores UTXOs and metadata about transactions referenced by UTXO and is implemented as a key-value store, LevelDB, such as `blocks/index/*`. This database enables nodes to validate new blocks and transactions without scanning the blockchain ledger. If the node does not maintain databases such as `blocks/index/*` and `chainstate/*`, high costs are incurred for data validation and inquiry. In addition, `blocks/rev*.dat` is undo data for `blocks/blk*.dat` and is used when chain reorganization is needed.

B. Bitcoin's script and validation

For a transaction to be generated in the Bitcoin network and contained in the blockchain, it must be validated by nodes including miners. Validation is the process of checking whether a transaction satisfies some rules defined by the Bitcoin network. The following shows some of the rules of validation:

- whether the syntax and data structure of the transaction are correct
- whether the reference output exists and is not consumed for each input
- whether the amount of inputs is greater than the amount of outputs
- whether the `scriptSig` of each input corresponds correctly to the `scriptPubkey` of the reference output

Here, `scriptSig` is a signature for `scriptPubkey`, which is a script used to prove ownership of a UTXO and consume it. The script uses the data structure of the stack and returns a boolean value through `PUSH` and `POP` operations. If `TRUE` is returned as a result of executing the script, ownership of the UTXO is validated. For example, scriptPubkey of pay to script hash (P2SH) [8], one of the Bitcoin scripts, is OP_HASH160 <redeemScriptHash> OP_EQUAL, and scriptSig is [<sig>...<sig>] <redeemScript>. OP_CHECKMULTISIG included in <redeemScript> checks whether the number of signatures required to consume the UTXO is included in the script and signatures are valid and returns `TRUE` or `FALSE` as a result. In other words, validation of a new transaction is the process of checking the inputs of the transaction and reference output of each input from `chainstate/*`, the database UTXO set, and executing the script included in the input and output.

C. Bitcoin's block size

Most of the components of a Bitcoin block have a fixed size. A block consists of the block size (4 bytes), header (80 bytes), number of transactions (1 - 9 bytes), and transactions. The block header consists of the version of Bitcoin (4 bytes), hash value of the previous block (32 bytes), Merkle hash value (32 bytes), timestamp (4 bytes), PoW difficulty (4 bytes), and nonce (4 bytes). A transaction consists of overhead, input, and output. Overhead is a transaction's metadata and consists of Bitcoin version (4 bytes), LockTime (4 bytes), SegWit MarkerFlag (0.5 bytes), the number of inputs (1 - 9 bytes), and the number of outputs (1 - 9 bytes). Input consists of the hash value of the reference transaction (32 bytes), the position of the UTXO on the reference transaction (4 bytes), the length of scriptSig (0.5 bytes) and scriptSig of the UTXO, and nSequence (1 - 9 bytes). The output consists of the amount (8 bytes), length of scriptPubkey (1 - 9 bytes) and scriptPubkey. Then, the block's size is determined by the transactions and scripts included in the block.

III. Related work

The Bitcoin community has introduced block file pruning [6] in its implementation, Bitcoin Core. Block file pruning is a method in which a node removes data that are not related to the validation of a transaction from storage without maintaining it. This includes the past block data, which are the raw binary data stored in the local storage, `blocks/blk*.dat` and `block/rev*.dat`. These raw binary data are used only to relay blocks to other nodes, reorganize the blockchain, inquire into past data, and rescan the Bitcoin wallet. Therefore, a node with block file pruning can remove the rest of the block data, leaving only 288 or more of the most recent block data for blockchain reorganization.

Block file pruning, a method of removing raw binary data from local storage, contains several problems. One problem is that a node with block file pruning cannot help other nodes bootstrap. Since a node with block file pruning does not store past block data in local storage, it cannot provide block data to other nodes. If all nodes in the Bitcoin network execute block file pruning, it becomes impossible for new nodes to participate in the Bitcoin network. Bitcoin's assumed-valid blocks [9] allow new nodes to bootstrap by starting validation from a relatively recent block rather than the genesis block but still require nodes to store all block data. Another problem is

that block pruning forces the node to store a separate database called the UTXO set that stores only UTXOs for validation of new transactions because block data referenced by some UTXOs can be removed from local storage. The fact that block data referenced by UTXOs are removed from local storage may cause inconvenience depending on the user's purpose, such as finding referenced block data for new transaction creation or analyzing past data.

As described in prior studies, a snapshot that proves that the database UTXO set stored by the node is valid is generated periodically and included in the blockchain so that the node that has removed the old block data makes the new node build the UTXO set. In other words, the snapshot enables a new node to bootstrap with block file pruning. However, it still forces the node to maintain a separate database such as the UTXO set, and there is a concern that the block data referenced by the UTXO may not be provided.

Palm et al. [10] proposed block pruning in a permissioned blockchain. In this proposal, only a selected group stores the entire blockchain ledger, and the other groups store only data such as transactions related to them. Since this proposal has security and reliability issues, it is difficult to apply to public blockchains.

In some blockchains, new methods of pruning have been proposed, such as efficiently tracking and storing the status of accounts included in the blockchain network. Bruce et al. [11] and Poelstra et al. [12] adopted the method of replacing Bitcoin's UTXO set with an account tree and then adding the data of this tree to the block. However, in these proposals, the node does not check the availability of the data needed to compute and build the account tree.

In coinPrune [13], miners periodically generate snapshots of the blockchain state, and the snapshots are verified multiple times by miners throughout the network. In this proposal, snapshots verified sufficiently are adopted from the blockchain network, and each node is able to remove blocks prior to the snapshot. This proposal can be introduced to Bitcoin without a hard fork, and a node that newly enters the network can build a UTXO set from a snapshot received from a neighboring peer. However, there is a possibility that a DOS attack may occur while the miner verifies the snapshot.

In addition, securePrune [14] is a proposal of pruning based on the RSA accumulator [15] of the UTXO set and the PoW-based consensus algorithm [5]. In this proposal, an accumulator and NI-PoE [16] are added to the block to verify the validity of the blockchain state. A node newly entering the network can verify whether the UTXO set received from the neighboring node is valid through the accumulator included in the block. However, this proposal has the problem that it is difficult to introduce to an existing blockchain.

In Ethereum, the state of the accounts of the blockchain is maintained as a Merkle Patricia tree, and this tree is updated for each block. Ethereum's node stores all the history of the updated Merkle tree in consideration of the reorganization of the blockchain. The history of such a Merkle tree occupies a large capacity in local storage. Snap sync [17], which was introduced in Geth, one of the implementations of Ethereum, generates a snapshot composed of leaf nodes of the state tree, making it possible to build a Merkle Patricia tree. By snap sync, the node of Geth can remove the past history of the state tree from local storage. Snap sync has greatly reduced the size of the past history occupied in the local storage and the synchronization time of the new node.

IV. Approach

In this section, we propose UTXO aggregation and describe the necessary conditions for introducing this method to blockchain.

A. Necessary conditions

This proposal can be incorporated into a UTXO-based blockchain, and there are necessary conditions for its introduction. It allows a third party other than the owner of the UTXO to consume the UTXO for a limited purpose. To consume the UTXO in the UTXO-based blockchain, an appropriate private key is required for the UTXO, and ownership of the UTXO must be proven by executing the blockchain script. For the introduction of this proposal, there is a condition in the blockchain script, which is that when a user's address owns two or more UTXOs, it must be possible to merge UTXOs into one new UTXO without the owner's approval. This does not mean that UTXOs can be consumed and remitted to other addresses without address approval but rather that UTXOs owned by an address are merged to generate one new UTXO owned by the same address. In other words, the necessary script should operate as if the balances divided into several accounts were collected in one account without the approval of the depositor.

B. UTXO aggregation

In UTXO aggregation as proposed in this paper, all UTXOs of each user are periodically merged into one. A UTXO merging in the proposal means creating a transaction with two or more UTXOs owned by the address as input for each address participating in the blockchain network and one new UTXO owned by the same address as the output. Transactions generated through merging are periodically aggregated in a special block called the aggregation block. The aggregation block is added to the blockchain. In this proposal, there is a period for creating an aggregation block and a period for UTXO merging, and the period for creating an aggregation block is longer than the period for merging. We call the period of creating an aggregation block the epoch. At least once at the time when the epoch changes, UTXOs are merged on a user basis, and an aggregation block containing all merged UTXOs is generated. The aggregation block generated for each epoch intensively shows all UTXOs that exist in the blockchain network. Depending on the purpose of the blockchain, the unit of epoch is set as 1 week, 1 month, or 1 year.

Fig. 2 gives an overview of the proposal. The ledger of the blockchain is divided at regular intervals. Ledger division at regular intervals indicates a specific block, which is the

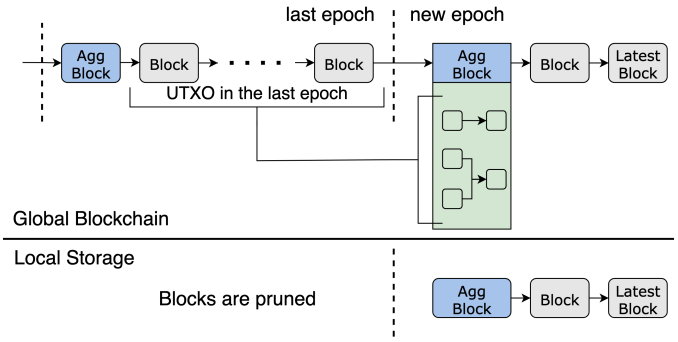

Fig. 2. Architecture of the proposal method and UTXO aggregation outline.

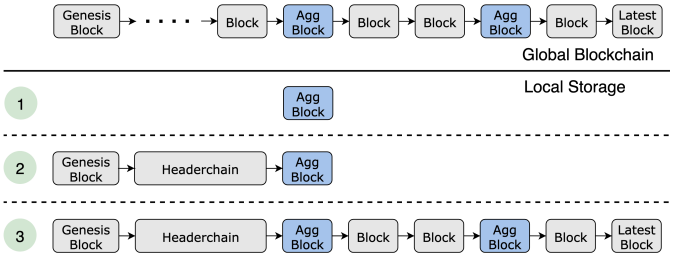

Fig. 3. Bootstrap process of a newly participating node in the network.

point at which the node operator starts to store the ledger in the local storage. This assists multiple node operators in consistently storing the same section of the ledger. Even if the node operator removes the block data before the aggregation block from the local storage, there is no problem in the running node, and the operator's node can store the ledger for the period of interest only. For example, it is possible to simply address the node operator's request, such as requiring data for only the last year or a specific period in the past.

C. Data integrity and UTXO set

In the proposal, since all UTXOs contained in the previous epoch are included in the input of the aggregation block and consumed, all UTXOs must exist in the block of the last epoch. This means that the data to be referenced when a user makes a new transaction must exist in the last epoch. This feature is different from Bitcoin. In Bitcoin's block file pruning, whether the transaction referred to by a UTXO exists in local storage is determined by how many blocks the node stores. Depending on the node, the transaction data referenced by the UTXO may have already been removed from local storage. For this reason, in Bitcoin's block file pruning, if a node does not maintain the UTXO set, which is a separate database that stores only UTXOs, this node is not able to validate new blocks or transactions. However, in the proposal, since all UTXOs and data referenced by UTXOs exist in local storage, blocks and data can be validated even if the node does not maintain a separate database.

D. Bootstrap

For a node that has removed old block data from local storage to be able to validate new data, the node must be able to know all UTXOs without old data. Therefore, in Bitcoin, the node stores the UTXO set in local storage. For a node with block file pruning to enable bootstrapping of other nodes, the node must be able to prove that the UTXO set stored in the local storage is correct. Therefore, in related studies, a method such as a node creating a snapshot, data that prove that UTXO set at a specific point in time is correct, and putting it in a block is used. The bootstrap node can check whether the UTXO set is correct by comparing the snapshot with the UTXO set received from the neighboring node. Since the Bitcoin network does not provide a snapshot, a node with block file pruning cannot allow bootstrapping in new nodes. However, because the aggregation block in the proposal contains all UTXOs at one point in time, this block can serve as a snapshot.

The bootstrapping process in the proposal can be divided into three steps; Fig. 3 shows the outline. First, the bootstrap node finds and obtains the aggregation block in which the node wants to start storing blocks from the blockchain network. Then, the node obtains the headerchain for the period before the aggregation block. Finally, the node obtains the block and validates it for the period after the aggregation block.

E. Policy on merging

In the proposal of this paper, how to make an aggregation block is different according to the timing of UTXO merging. UTXO merging occurs on every block or once an epoch. If it is assumed that merging occurs once in an epoch, merging occurs when the epoch changes, and the result is reflected in the aggregation block. At this time, the aggregation block must be generated by the miner, and this block must be propagated through the blockchain network. If merging occurs on every block, since all nodes store the same data in local storage, each node can generate an aggregation block by itself.

V. Introduction to Bitcoin

There are necessary conditions to introduce UTXO aggregation described in Section IV to Bitcoin. It shares the right to consume the UTXO to the blockchain network. To introduce this method to Bitcoin, a new script must be introduced, and all users must use the new script-based address. We introduce pay to witness aggregation script hash (P2WASH), a script that is based on Bitcoin script pay to witness script hash (P2WSH) [18] and satisfies the requirements.

Fig. 4 shows an overview of Bitcoin's scripts P2SH and P2WSH and the new script P2WASH. The scriptPubkey of m-of-n P2WASH that contains n `public key` and requires m `signature` out of n for UTXO consumption is OP_HASH160 <redemScriptHash> OP_EQUAL. The scriptSig of m-of-n P2WASH is [<sig>...<sig>] <redemScript>, where <redemScript> is OP_0_OR_M [<pubkey>...<pubkey>] OP_AGG_CHECKMULTISIG. P2SH and P2WSH return `TRUE` or `FALSE` by OP_CHECKMULTISIG, but P2WASH returns `SEND`, `AGG`, `FALSE` according to the number of valid

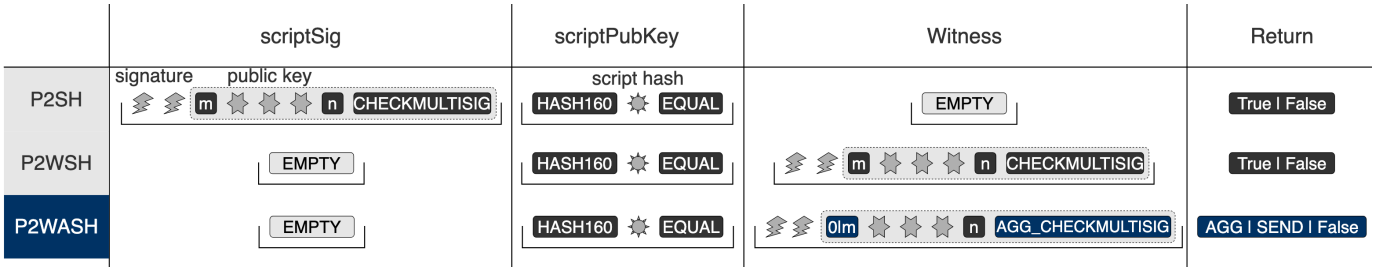

Fig. 4. Comparison of the structure of Bitcoin's scripts and the new script P2WASH.

<sig>s contained in scriptSig by OP_AGG_CHECKMULTISIG. If `scriptSig` contains m valid <sig>, the script returns `SEND`, and remittance to another address is possible, similar to the existing script. When `scriptSig` does not contain <sig> and the address of the UTXO owner and the remittance address are the same, the script returns `AGG`, and UTXO merging is possible. Otherwise, the script returns `FALSE`, and nothing happens. That is, the requirement of the proposal is satisfied by the `AGG` returned by P2WASH, and the number of UTXOs owned by each address periodically becomes one. If P2WASH is introduced, the proposed method can be introduced to Bitcoin.

All Bitcoin scripts correspond to P2WASH according to the number of `public keys` and `signatures` contained in the script. For example, scripts such as P2PK, P2PKH, and P2WPKH containing one `public key` and `signature` correspond to x-of-n P2WASH, and scripts such as x-of-n P2MS, P2SH, and P2WSH containing n `public keys` and `signatures` correspond to x-of-n P2WASH.

VI. Experiment

In this section, on the premise of the necessary conditions described in Section V, we examine how effective the proposed method is when incorporated into Bitcoin. To determine the effectiveness of the method, we use Bitcoin data and make some assumptions to calculate and predict changes in items related to storage size.

A. Data and assumptions

The three types of data collected for the experiment are as follows. We use the data of blocks and transactions for one year, and the numbers are 52635 and 99162585, respectively. The script type that occurs during the period is analyzed. We use the data "addresses with nonzero balance" from Glassnode [19]. We use the UTXO data by analyzing UTXO set as of November 2021 using STATUS [20], which Delgado et al. [21] used as an analysis tool for the UTXO set.

We make some assumptions, as we cannot gather all the data to accurately predict the effect on Bitcoin. We assume that the ratio of script types contained in the transaction, the rate of increase of addresses with nonzero balance, and the ratio of input and output of the transaction does not change. We consider only four script types, P2SH, P2PKH, P2WPKH, and P2WSH, occupying approximately 99 % of scripts and ignoring other script types. We also assume that half of P2SH and P2WSH are 1-of-2 and the rest are 2-of-3. We merge UTXOs once in the epoch to simplify the experiment.

TABLE I
Comparison of script size and vSize by script type.

	size [byte]		vSize [vbyte]	
	scriptSig	scriptPubkey	scriptSig	scriptPubkey
P2PKH	107	25	428	100
P2WPKH	107	20	107	80
P2SH (1-of-2)	147	23	588	92
P2SH (2-of-3)	254	23	1016	92
P2WSH (1-of-2)	147	34	147	136
P2WSH (2-of-3)	254	34	254	136
P2WASH (1-of-1)	113	34	113	136
P2WASH (1-of-2)	147	34	147	136
P2WASH (2-of-3)	254	34	254	136

B. Comparison of blocks

The size of the script depends on each type, and Table I shows the details. The average numbers of transactions, inputs, and outputs included in a block are 1884, 5754, and 5976, respectively. The ratios of P2SH, P2PKH, P2WPKH, and P2WSH are 36.2 %, 20.1 %, 40.9 %, and 1.8 %, respectively. Based on the given data and the contents described in Subsection II-C, the size of the block can be calculated. The block sizes before and after the introduction of the method are 1287333 bytes and 1370050 bytes, respectively, which is an increase of approximately 6.4 %. This is because the size of the P2WASH script is larger than that of the P2PKH and P2WPKH scripts.

Changes in the vSize of blocks and the number of transactions that can be included in a block are described in Appendix A.

C. Comparison of the UTXO set

Bitcoin's UTXO set is based on LevelDB, a key-value store. The key of the UTXO set (version 0.15.0 or later) contains the data about the hash value and index of the transaction referenced by each UTXO. The value contains the height of the block, whether the transaction is the coinbase, the amount, and the scriptPubkey of each UTXO. That is, the UTXO set stores information and metadata of UTXOs, and the number

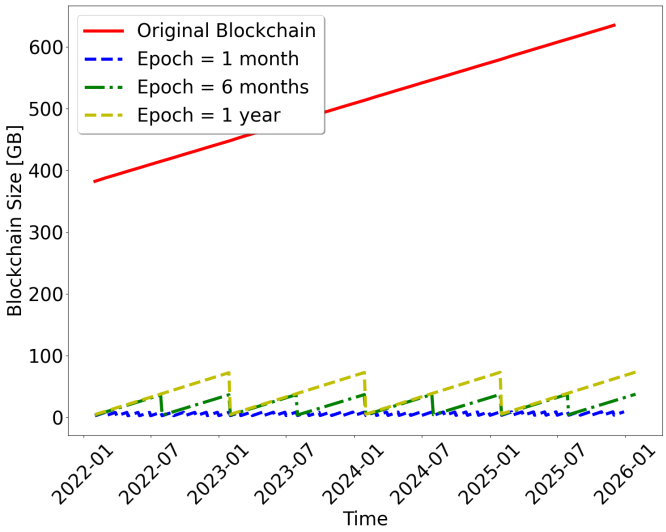

Fig. 5. Changes in size on local storage when a node stores a blockchain ledger of 1 epoch.

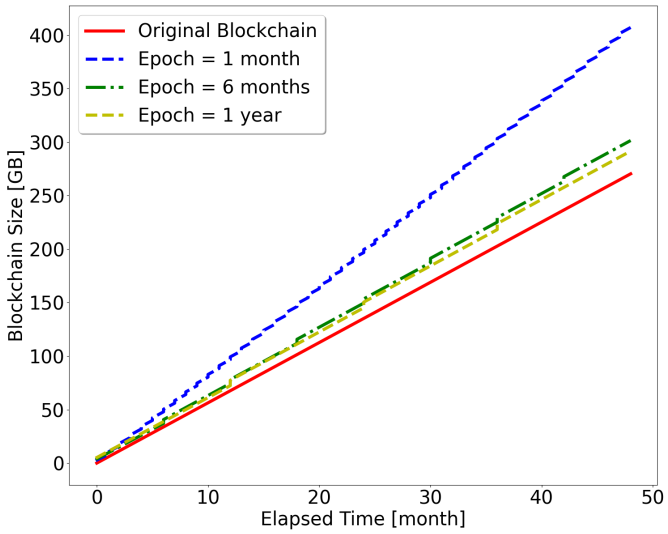

Fig. 6. Rate of increase in the size of the blockchain ledger.

of records in the UTXO set is the same as the number of UTXOs.

If the proposed method is introduced in Bitcoin, the node is not forced to store the UTXO set, but if it does, the number of records changes. Subsection IV-E describes the timing and frequency of UTXO merging, but the change in the number of records differs depending on whether UTXO merging occurs once per epoch or per block. When merging occurs for each block, the number of UTXOs is the same as the number of addresses with nonzero balance, and the number of records of the UTXO set decreases. If merging occurs once in the epoch, the number of UTXOs is the sum of the number of addresses with nonzero balance and the number of UTXOs generated in excess over the epoch. As of December 5, 2021, the number of addresses with nonzero balance was 39240264.

D. Size of the aggregation block

The structure of the aggregation block is the same as that of a general block, and its size is determined by the number of transactions included in the block and inputs and outputs included in each transaction. The aggregation block's input contains existing UTXOs in the previous epoch, and the number of UTXOs is equal to the difference between the number of inputs and outputs in the previous epoch. According to the collected data, the number of outputs included in the block is 222 more than the number of inputs on average. When the epoch unit is 1 month, 6 months, and 1 year, the number of excess outputs is 973076, 5838456, and 11676912, respectively. Only one UTXO of each address is contained in the output of the aggregation block, and the number of addresses with nonzero balance is 39240264.

We can calculate the size of the aggregation block based on the data and assumptions in Subsection VI-A. The size of the aggregation block is approximately 2.27 GB (2270892801 bytes) when the epoch unit is 1 month, 3.19 GB (3194555515 bytes) when the epoch unit is 6 months, and approximately 4.31 GB (4309245744 bytes) when the epoch unit is 1 year.

E. Size occupied by the blockchain ledger on local storage

The change in the ledger size of the blockchain on the local storage can be predicted by the block size calculated in Subsection VI-B and the size of the aggregation block calculated in Subsection VI-D. The size of the ledger increases by the size of one aggregation block and the size of blocks generated during the epoch for each epoch.

When the unit of epoch is 1 month, 6 months, and 1 year, the average number of blocks generated is 4386, 26316, and 52632, respectively. When the unit of epoch is 1 month, 6 months, and 1 year, the size of the aggregation block is 2.43 GB, 3.19 GB, and 4.31 GB, respectively. When the unit of epoch is 1 month, 6 months, and 1 year, the increments of addresses with nonzero balance are 444139, 2911061, and 6314577, respectively. Based on this, we predict the size when storing 1 epoch of blockchain in local storage and obtained the same result as shown in Fig. 5. The size of the blockchain when the proposed method is not introduced is 380 GB, and the sizes are approximately 0.6 - 2.0 %, 0.8 - 9.5 %, and 1.1 - 18.9 % according to the epoch unit after the introduction of the proposed method.

F. Rate of increase in the size of the blockchain ledger

In the proposal, unlike before, an aggregation block is additionally generated on the blockchain ledger, so its size increases faster. Fig. 6 shows the rate of increase before and after the introduction of the proposed method. When the proposed method is not introduced, the size increase rate of the ledger is approximately 5.6 GB/month. The rate of increase in the size of the ledger after introducing the proposed method is 8.4 GB/month, 6.2 GB/month, 6.0 GB/month when the

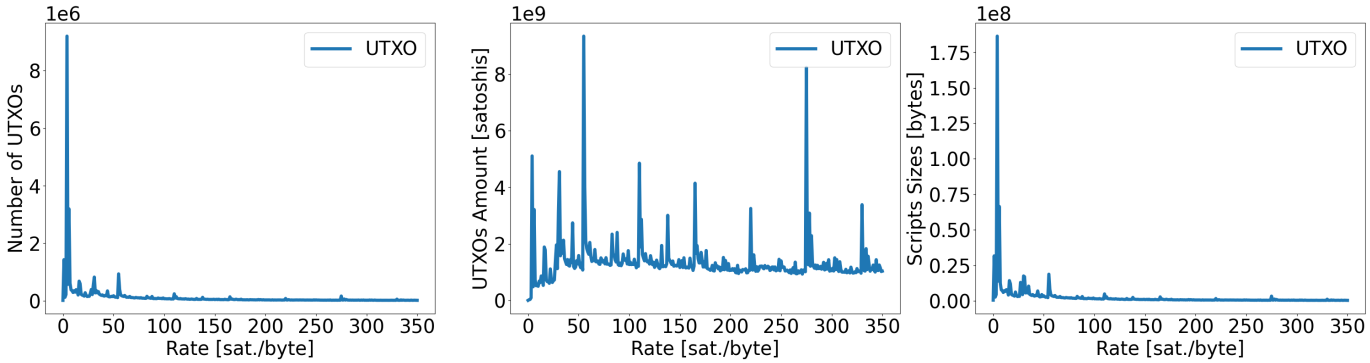

Fig. 7. Distribution of the number, total amount, and script sizes of UTXOs by the UTXO value rate.

epoch unit is 1 month, 6 months, and 1 year, respectively, corresponding to an increase of approximately 50 %, 10 %, and 6 % from before the introduction.

G. Economic effects of UTXO merging

In the proposed method, since each user periodically possesses only one UTXO, the total number of UTXOs decreases, and the average value of each UTXO increases. The value rate of a UTXO can be expressed as a value obtained by dividing the amount included in the UTXO by the script size included in the UTXO. The transaction fee is proportional to the transaction size and the congestion level of the transaction pool, and the transaction fee rate can be expressed as a value obtained by dividing the transaction fee by the transaction size. If the value rate of the UTXO is low or the fee rate of a transaction is high, it is difficult to consume the UTXO. In other words, it can be said that the economic value of a UTXO with a low value rate is small. Delgado et al. [21] classified UTXOs with a small value rate into dust UTXOs and nonprofitable UTXOs. They classified UTXOs where the transaction fee rate exceeds 1/3 of the UTXO's value rate as dust UTXOs, and UTXOs where the transaction fee rate exceeds the UTXO's value rate were classified as nonprofitable UTXOs.

Fig. 7 shows the distribution of the number, total amount, and script size of UTXOs based on the UTXO value rate. According to the distribution, the total amount of UTXOs with a value rate less than 150 sat./byte occupies only 0.012 % of the total Bitcoin amount, but their number occupies 49 %, and their script sizes occupy 50 % of the total.

According to statoshi [22], the maximum fee rates for the last month, one year, and the previous period are approximately 20, 270, 1000 sat./byte, respectively. Fig. 8 shows the cumulative amount of dust UTXO and nonprofitable UTXO according to the fee rate. When the fee rate is 20 sat./byte, the total amount of dust UTXOs and nonprofitable UTXOs is 36 and 30 BTC, respectively. When the fee rate is 270 sat./byte, the total amount of dust UTXOs and nonprofitable UTXOs is 3620 and 2540 BTC, respectively. When the fee rate is 1000 sat./byte, the total amount of dust UTXOs and nonprofitable UTXOs is 14120 and 8620 BTC, respectively.

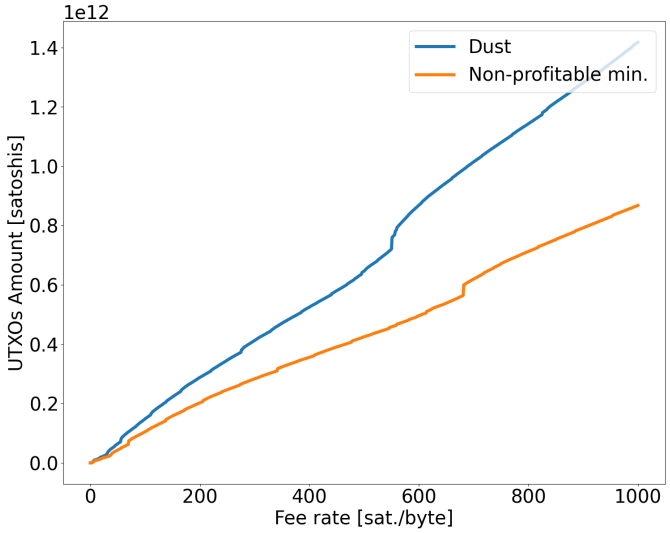

Fig. 8. Cumulative distribution of dust/nonprofitable UTXOs by the fee rate.

VII. Conclusion

In this proposal, we proposed UTXO aggregation, which can be introduced into a UTXO-based blockchain. Block pruning and related research proposed in some blockchains, including Bitcoin, have the problem that nodes cannot bootstrap. There is a possibility that the block referenced by a UTXO or the UTXO has been removed from the local storage, forcing the node to maintain a separate database that stores only UTXOs. An aggregation block that is periodically generated in the proposed method serves as a snapshot of the UTXO set by allowing the bootstrap node to identify all UTXOs. After the aggregation block is generated, there is no UTXO in the period before this block, so the node does not need to maintain a separate database that stores only UTXOs. The proposed method not only solves the above problems but also reduces the number of UTXOs with low economic value on the blockchain network because UTXOs are regularly merged.

For the blockchain to become a sustainable system even after a long period of time, it is necessary to reduce the

storage occupancy required to run a node. In the experiment of this paper, using Bitcoin data, the blockchain ledger size was calculated and compared when the proposed method was introduced to Bitcoin. As a result of the experiment, it was confirmed that although the increase rate of the ledger size is somewhat faster, the size of the ledger stored in the storage can be significantly reduced. Therefore, it can be said that the proposed method is effective in reducing the running cost of the node.

To introduce the proposed method, a script that can share a part of the right to consume UTXOs to the network is needed, and all users must use this script-based address. We conclude that the proposed method can be easily introduced in the situation of designing a new blockchain, but consensus on the network is required to introduce it to the existing blockchain. If the consensus conflicts with the existing rules of the blockchain, the blockchain must be hard forked, as occurred with Bitcoin Cash and Bitcoin Gold.

APPENDIX

A. Changes in block vSize and transaction throughput

In Bitcoin, there is a limit on the size of a block. The limit was initially 1000000 bytes, but after the introduction of segWit [18], it became 4000000 vbytes. segWit puts the script's signature in the witness space, not the scriptSig space. In calculating the vSize of a block, the vSize is quadrupled except for the signature part of the segWit-based script. In other words, if a segWit-based script is used, more transactions can be contained in a block, and the use rate of the segWit-based script has recently increased.

The virtual size of the block was 3787198 vbytes before applying the proposed method, but it became 1973890 vbytes after application, which is an approximately 47.9 % decrease. Therefore, the actual size of the block increases somewhat, but there is room for vSize. A larger number of transactions can be contained in a block.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP21H04872.

REFERENCES

[1] "Bitcoin project, "bitcoin-qt version 0.8.0 released"," https://bitcoin.org/en/release/v0.8.0, accessed: 2021-12-16.
[2] "Bitcoin project, "bitcoin core version 0.15.0 released"," https://bitcoin.org/en/release/v0.15.0, accessed: 2021-12-16.
[3] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[4] "Bitcoin project, "lightweight node"," https://en.bitcoin.it/wiki/Lightweight_node, accessed: 2021-12-16.
[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.
[6] "Bitcoin project, "bitcoin core version 0.11.0 released"," https://bitcoin.org/en/release/v0.11.0, accessed: 2021-12-16.
[7] Google, "Leveldb," https://github.com/google/leveldb, 2012.
[8] "Bitcoin project, "bitcoin core version 0.10.0 released"," https://bitcoin.org/en/release/v0.10.0, accessed: 2021-12-16.
[9] "Bitcoin project, "bitcoin core version 0.14.0 released"," https://bitcoin.org/en/release/v0.14.0, accessed: 2021-12-16.
[10] E. Palm, O. Schelén, and U. Bodin, "Selective blockchain transaction pruning and state derivability," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 31–40.
[11] J. Bruce, "The mini-blockchain scheme," *White paper*, 2014.
[12] A. Poelstra, "Mimblewimble," 2016.
[13] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, "How to securely prune bitcoin ' s blockchain," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 298–306.
[14] B. S. Reddy, "secureprune: Secure block pruning in utxo based blockchains using accumulators," in *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2021, pp. 174–178.
[15] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 480–494.
[16] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Annual International Cryptology Conference*. Springer, 2019, pp. 561–586.
[17] "Ethereum foundation, "geth v1.11.0"," https://blog.ethereum.org/2021/03/03/geth-v1-10-0/, accessed: 2021-12-16.
[18] "Bitcoin project, "bitcoin core version 0.13.0 released"," https://bitcoin.org/en/release/v0.13.0, accessed: 2021-12-16.
[19] "Glassnode, "bitcoin: Number of addresses with a non-zero balance"," https://studio.glassnode.com/, accessed: 2021-12-16.
[20] S. D. Segura, "Statistical analysis tool for utxo set," https://github.com/sr-gi/bitcoin_tools/tree/v0.2/bitcoin_tools/analysis/status, 2018.
[21] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Analysis of the bitcoin utxo set," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 78–91.
[22] "statoshi.info, "recommended transaction fee for target confirmation in x blocks"," https://statoshi.info/, accessed: 2021-12-16.