DEIM2022 J34-6

UTXO Aggregation による Block Pruning

Song Taegyu　　首藤 一幸

東京工業大学　〒 152–8550 東京都目黒区大岡山 2-12-1

あらまし　ブロックチェーンの台帳は追記式であり，日々，サイズが大きくなっていく．ノードを運用するためには台帳全体を保持する必要があり，これが運用者の負担となってきている．我々は，この問題を解決する新しい手法 UTXO Aggregation を提案する．この手法は，定期的に UTXO をマージし，マージした UTXO を含むブロックを生成する．Bitcoin の Block File Pruning と異なり，他のノードの起動を助けることもできる．この手法には経済的価値の低い UTXO を減らす効果もある．この手法を Bitcoin に適用することで，ノードのストレージ負担は 1/100 程度に軽減でき，最大 14120 BTC の経済的効果が期待できる．

キーワード　ブロックチェーン，ストレージ占有量削減

1 はじめに

ブロックチェーンとは，多数のオンライン取引記録をまとめて一つのデータであるブロックを構成し，ハッシュ値を用いて以前のブロックとその後のブロックをチェーンのようにつなげた後，この情報の全部または一部を P2P 方式で世界中の複数のコンピュータにコピーして分散保存・管理する技術である．すなわち，ブロックチェーンは全ての取引内訳を含む巨大な分散台帳技術であるといえる．ブロックチェーンは，以下の二つの特徴と特徴に応じた問題点を持つ．

ブロックチェーンは多くの利用者数を想定しており，その台帳は有効なすべての取引内訳を含むため，ストレージの占有量が非常に多い．また，台帳は追記式であるため，そのサイズは増加する一方である．

台帳に付随するデータもストレージを占有する．例えば Bitcoin の実装である Bitcoin Core は，効率的なデータ参照や新規ノードのブートストラップのために，データベースシステムの変更 [1] や UTXO set の内部レイアウトを変更 [2] してきた．

代表的なブロックチェーンである Bitcoin [3] と Ethereum [4] のノードのサイズは，それぞれ 383 GB，1146 GB を超えている．図 1 は，Bitcoin および Ethereum (geth) ノードのストレージ占有量変化を示しており，最近のサイズ増加速度がより速くなっている．

主要ブロックチェーンはノードが全台帳を保存することを基本としている中で，肥大化したブロックチェーン台帳のサイズは，多くのノード運用者にとって大きな負担となってきている．ノード運用の負担が大きいとノード運用者が増えず，ブロックチェーン管理権限の分散化（decentralization）が妨げられてしまう．

複数のブロックチェーンはストレージサイズの問題を解決するために様々な方法を提示している．ブロックチェーン参加者をグルーピングして一部の利用者の台帳だけを保存するシャーディングの導入，ノード運用のために最小限の構造だけを維持する形態のノードの提案，ブロックチェーンネットワークの外に信頼できるリポジトリを設けるなどの方法がある．最小限の構造だけを維持するノードの例として，Bitcoin のライトノード [5] は，台帳を保存せずにブロックチェーンの検証を他のノードにアウトソーシングするが，SPV [3] を使ってローカルでウォレットを管理する．

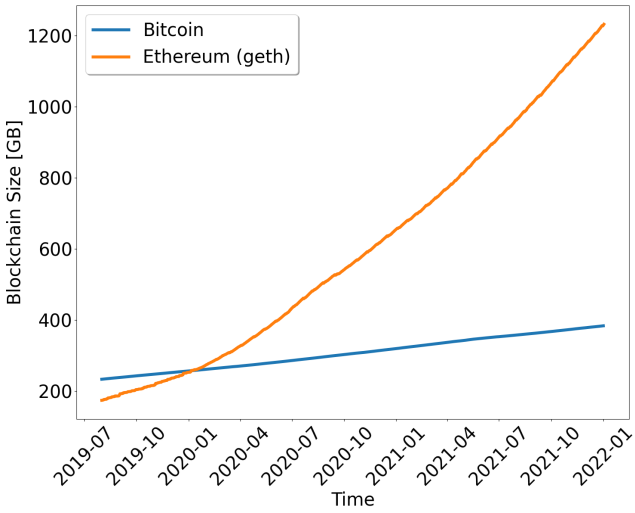

図 1　Bitcoin および Ethereum(geth) ノードのストレージ占有量

Bitcoin は過去ブロックの raw データを削除する Block File Pruning [6] を導入し，その問題を軽減したが，他のノードの起動に助けられない問題があり，UTXO が参照するブロックの照会ができない恐れもある．本稿では，Bitcoin の Block File Pruning の限界を克服する手法 UTXO Aggregation を提案する．

この提案は，実際に頻繁に使用される期間中心の精算システムの影響を受けて考案されている．定期的に UTXO を利用者基準でマージし，マージした UTXO を含むブロックである Aggregation Block を生成する．この手法には経済的価値が低い UTXO を減らす効果もある．

本稿の構成は次のとおりである．2 章で Bitcoin の全般知識について説明し，3 章では Bitcoin の Block File Pruning の関連研究を紹介する．第 4 章では提案手法を述べ，第 5 章では Bitcoin へ提案手法を適用するための必要条件について述べる．第 6 章では実験とその結果を述べる．最後に第 7 章にあとめを述べる．

2 準　　備

本章では，Bitcoin の実装である Bitcoin Core のデータストレージとトランザクションの検証 · 実行過程について述べる．

2.1 Bitcoin のデータストレージ

Bitcoin の実装である Bitcoin Core は `blocks/blk*.dat`，`blocks/index/*`，`chainstate/*`，`blocks/rev*.dat` の 4 種類のデータで構成されている．`blocks/blk*.dat` は，Bitcoin の台帳に含まれるブロックのデータであり，ストレージに raw binary データでダンピングされる．このデータはウォレットから漏れたトランザクションを再スキャン，フォークによるチェーンの再構成，同期中の他のノードにブロックデータを提供するのにだけ必要である．`blocks/index/*` は全てのブロックに対するメタデータとストレージ上の位置が含まれている LevelDB [7] データベースである．`chainstate/*` は使用されていない UTXO と当該 UTXO が含まれたトランザクションのメタデータを簡単に保存する LevelDB データベースである．このデータは新しいブロックおよびトランザクションを検証・実行するのに必要である．検証・実行はブロックデータを直接スキャンすることで可能であるが，そのコストが高いので，別のデータベースで維持する．`blocks/rev*.dat` は，`blocks/blk*.dat` に対する undo データであり，チェーンの再構成が必要なときに使用される．

2.2 Bitcoin のスクリプトと検証

Bitcoin でトランザクションが生成され，ブロックチェーンに含まれるために，Miner から有効性の検証が行われる必要がある．有効性検証とは，Bitcoin ネットワークで定義したルールを満足しているかを確認する過程をいう．取引の構文とデータ構造が正しいであること，それぞれの入力値について参照出力値が存在して消費されていない状態であること，入力値の金額が出力値の金額より大きいこと，各入力値に対する scriptSig により入力値の参照出力値の scriptPubkey が検証されることなどが有効性検証規則の一部である．

scriptSig は scriptPubkey に対する署名であり，当該 UTXO の所有権を証明し出力値が消費されるようにするスクリプトである．スクリプトはスタックのデータ構造を使用し，PUSH 演算と POP 演算を経て `TRUE` 値が導出されれば所有権が証明される．例えば，P2SH (Pay to Script Hash)[8] の scriptPubkey は OP_HASH160 <redeemScriptHash> OP_EQUAL で，scriptSig は [<sig>...<sig>]<redemScript> の形態を持っている．<redeemScript> に含まれている OP_CHECKMULTISIG は，当該スクリプトに必要な数の有効な署名が含まれているかを確認し，`TRUE` あるいは `FALSE` を導き出す．

結局，新しいトランザクションの検証とは，UTXO set である `chainstate/*` からトランザクションの入力と参照出力を確認し，入力と出力に含まれるスクリプトを実行する過程である．

3 関 連 研 究

Bitcoin の実装である Bitcoin Core は Block File Pruning [6] を導入した．Block File Pruning は，ノードがストレージからブロックとトランザクションの有効性検証とは関係ないデータを維持・管理せずに削除する方法である．これに該当するデータはブロックの raw binary データの `blocks/blk*.dat` と `block/rev*.dat` データである．このような raw binary データは，ブロックを他のノードでリレーしたり，チェーンの再構成，過去のトランザクションの照会，Bitcoin ウォレットの再スキャンにのみ使用されるので，有効性検査とは関係ない．したがって，Block File Pruning はチェーンの再構成のために，最新のブロックデータを 288 個以上だけ保存して他のデータを削除する．

Block File Pruning は単にストレージに保存されていた raw binary データを削除する方法であるため，いくつかの問題を含んでいる．一つの問題は，他のノードのブートストラップを助けることができないということである．Block File Pruning をしたノードはストレージに過去のブロックデータを保存していないため，他のノードにブロックデータをリレーすることができない．そして，もし Bitcoin ネットワークのすべてのノードが Block File Pruning を行えば，新しいノードが Bitcoin ネットワークに参加することができなくなる．Bitcoin の assumed-valid blocks [9] は，最初のブロックではなく正しいとされる最近のブロックから検証を始め，新しいノードのブートストラップを可能にするが，依然として過去の履歴を保存することを要求する．もう一つの恐れは，UTXO set に含まれているデータから参照するブロックデータがもう削除され，照会できないことである．利用者によって，UTXO が参照するブロックデータを，取引に参考にすることもしくはブロックチェーンデータの解析することなどにおいて要る可能性がある．

既存研究では周期的にスナップショットを生成して，Block File Pruning のように古いブロックを除去した後もブートストラップを可能にする．しかし，依然として参考になりそうな情報を提供できない恐れがあり，Bitcoin のように既に稼働中のシステムでは相応のアプローチは採用されにくい．

Palm ら [10] は許可型ブロックチェーンでの Block Pruning を提案した．選択された集団によって全体のブロックチェーンが維持され，他の集団は自分たちと関連する取引内訳のみを保存する方法である．この方法はセキュリティと信頼性の問題からパブリックブロックチェーンには適用しにくい．

一部のブロックチェーンでは，効率的に状態を構成するために存在する account とそれらの残額だけを追跡するなどの方法で構造を単純化した Pruning [11]，[12] が提案された．Bruce

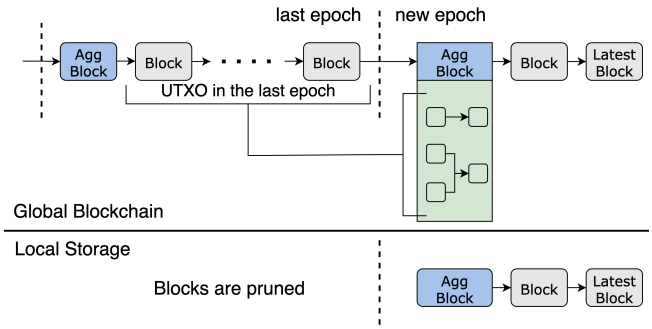

図 2　UTXO Aggregation

ら [11] と Poelstra ら [12] は Bitcoin の UTXO set を account ツリーに置き換え，ブロックに結合する．しかし，この方法はノードが必要なデータの可用性なしに account ツリーを計算することを想定する．

coinPrune [13] は，周期的に状態のスナップショットを生成し，生成されたスナップショットは全ての Miner から複数回の確認を受ける．スナップショットが十分に確認された後，以前のブロックを除去する．この方法はハードフォークなしに Bitcoin に適用でき，スナップショットの存在から新しいノードのブートストラップも可能にする．しかし，miner がスナップショットを確認する過程で DOS 攻撃が発生する可能性がある．

securePrune [14] は，UTXO set の RSA accumulator [15] および PoW-based 合意アルゴリズム [3] をベースとした Pruning を提案する．この提案は，ブロックに accumulator と NI-PoE 証明 [16] を結合して有効性を検証し，ブロックの accumulator の存在から新しいノードのブートストラップが可能である．しかし，既に稼働中のブロックチェーンには適用することは困難である．

Ethereum は account の状態情報をマクルパトリシアツリーで維持し，ブロックごとに更新するが，変化したツリーの以前の状態を削除していなかった．Ethereum の snap sync [17] は，State Trie の leaf node からなるスナップショットから，マクルパトリシアツリーを再構築できるようにし，State Trie の過去 node を削除するようにした．この方法で必要なストレージサイズとブロックチェーンネットワークとの同期化時間を大幅に減らした．

4　提 案 手 法

本章では，UTXO Aggregation による Block Pruning を提案し，手法の必要条件について説明する．

4.1　必 要 条 件

本提案は UTXO-based ブロックチェーンに適用でき，適用には必要条件がある．それは UTXO 消費の権限の一部をネットワークに共有することである．

UTXO-based ブロックチェーンで UTXO を消費するためには，当該 UTXO に対して適切な pravite key を所有しなければならず，private key の唯一所有者のみがブロックチェーンスクリプト実行により所有権を証明することができる．本提案の適用のためには，2 つ以上の UTXO が同一のアドレスによって所有されるとき，所有しているアドレスの許可なしに UTXO らを 1 つに合わせることを可能にするブロックチェーンスクリプトが必要である．これは，アドレスの権限なく，UTXO を消費して他のアドレスに送金できることを意味せず，同じアドレスにのみ送金を許容できることを意味する．すなわち，必要スクリプトは，銀行システムで複数の口座に分かれていた残高を一つの口座に集めるように動作し，預金者の許可なく動作する．

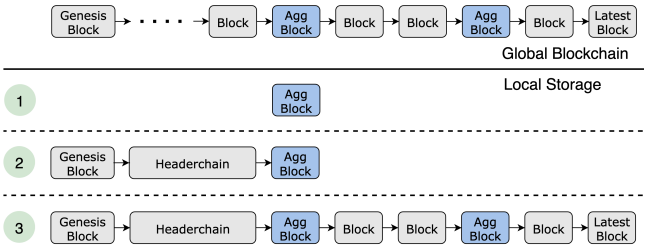

図 3　ブートストラップ

4.2　UTXO Aggregation

本稿で提案する UTXO Aggregation による Block Pruning は，周期的に利用者の UTXO をマージする．提案ではこの周期を epoch と呼ぶ．そして提案においてマージとは，すべてのアドレスに対して，一つのアドレスが所有している 2 個以上の UTXO を input とし，同一のアドレスが所有者となる 1 個の UTXO を output とするトランザクションを生成する行為を意味する．このように生成されたすべてのアドレスのトランザクションは，Aggregation Block というブロックに含まれてから，ブロックチェーンに含まれる．つまり，少なくとも epoch ごとにブロックチェーンのすべてのアドレスがたった 1 つの UTXO を所有するようになり，その内容は Aggregation Block に含まれて生成され，ブロックチェーンに追加されるのである．epoch ごとに生成された Aggregation Block は，過去の履歴の結果を集約的に示し，単位期間を基準に精算するシステムと類似した面を持っている．ブロックチェーンの目的に応じて，epoch を 1 週間，1ヶ月，1 年のように設定できる．

図 2 は，提案の概要を示している．提案により，ブロックチェーンのブロックは一定の間隔に分けることができる．一定の間隔は，ノード運用者がブロックの保存開始時点を指定できる指標となり，複数のノード運用者が同じ区間を一貫して保存できるように補助する．そして，ノード運用者はストレージから Aggregation Block 以前のブロックを削除しても運用には問題がなく，運用者が関心を持った期間のブロックのみを保存することが可能である．例えば，直近 1 年間あるいは特定期間などのニーズに簡単に対応できる．

4.3　完　結　性

提案により，Aggregation Block は以前の epoch に含まれる全ての UTXO を input にして消費するので，UTXO は常に最後の epoch のブロック中に存在することになる．従って，トラ

ンザクション生成に関連するすべての情報は最後の epoch の期間中に含まれることを保証することができ，この特徴は既存の Bitcoin の Block File Pruning との差別点である．Bitcoin の Block File Pruning で，UTXO が参照するブロックがストレージに存在するかはストレージにどれだけ多くのブロックを保存するかの設定により決定される．Block File Pruning で，UTXO set というデータベースがなければ，ブロックとトランザクションの実行・検証ができなくなる．しかし，本手法では全ての UTXO および UTXO が参照するブロックが最後の epoch に存在することを保証するので，UTXO set のような別のデータベースを維持しなくても過去ブロックデータを削除するのが可能である．

4.4 ブートストラップ

ストレージから過去ブロックデータを除去したノードが新しいブロックとトランザクションの検証および実行をするためには，ブロックチェーンネットワーク上の全ての UTXO を把握する必要がある．そして，新しいノードのブートストラップを助けるためには，隣接ノードもしくは自分が把握している UTXO らが正しいであることを証明できなければならない．Block Pruning でスナップショットはそのデータベースの正しいかを証明するために使用されるデータをいう．様々な代案では，ブートストラップのために UTXO set のスナップショットを生成し，それに関する情報をブロックチェーンに含めるなどの方式を採用している．したがって，ブートストラップするノードはスナップショットを通じて他のノードから受け取った UTXO set のデータが正しいであるかを確認することができる．

Bitcoin の Block File Pruning はスナップショットの不在で他のノードのブートストラップを助けることができない問題がある．提案の Aggregation Block は，全ての UTXO を含む UTXO set であるので，スナップショットの役割をすることができる．

提案のブートストラップは図 3 のように三段階に分けられる．第一に，ブートストラップするノードは，ブロックチェーンネットワークでストレージに保存し始めようとする時点の Aggregation Block を探して得る．第二に，Aggregation Block 以前の期間に対する Headerchain を得る．最後に，Aggregation Block 以降のブロックを得て検証する．

4.5 方　　針

本稿の提案はマージのタイミングで Aggregation Block の生成方法が異なる．マージが行われる時点は全てのブロックまたは epoch に一度である．もし，epoch に一度マージすると想定するならば，Aggregation Block を生成するタイミングで行われる．そして，Aggregation Block は必ず miner によって生成され，ブロックチェーンネットワークに伝播される必要がある．もし，ブロックごとにマージをするなら，全てのノードがストレージに共通のデータを持っているため，Aggregation Block を自ら生成することができる．

5 Bitcoin への適用

4 章で記述した提案手法である UTXO Aggregation による Block Pruning を Bitcoin に適用するための必要条件がある．したがって，UTXO 消費の権限を共有するスクリプトがあれば，提案手法を適用することができる．新しいスクリプトの導入は，Bitcoin のすべての利用者が新しいタイプのアドレスを使用しなければならないことを意味する．

Bitcoin で UTXO 消費の権限を共有するために，Bitcoin のスクリプト P2WSH (Pay to Witness Script Hash)[18] をベースとした P2WASH (Pay to Witness Agggregation Script Hash) を導入する．

図 4 は，P2SH，P2WSH，P2WASH の概要を示している．n 個の `public key` を含み，消費のために m 個の `signature` が必要な m-of-n P2WASH の scriptPubkey は OP_HASH160 <redemScriptHash> OP_EQUAL である．そして，scriptSig は [<sig>...<sig>] <redemScript> の形態を持ち，<redemScript> は OP_0_OR_M [<pubkey>...<pubkey>] OP_AGG_CHECKMULTISIG から構成される．P2SH と P2WSH の OP_CHECKMULTISIG が `TRUE` あるいは `FALSE` をリターンするのとは違い，OP_AGG_CHECKMULTISIG は scriptSig に含まれた <sig> の数に応じて，`SEND`，`AGG`，`FALSE` をリターンする．もし `scriptSig` に m 個の有効な <sig> を含めば，スクリプトは `SEND` をリターンし，既存のスクリプトと同様に他の address に送金可能になる．`scriptSig` に <sig> を含まず，送金の対象となる adress が input の address と同じなら，スクリプトは `AGG` をリターンし，マージができる．以外のすべての場合についてスクリプトは，`FALSE` をリターンし，何もしない．つまり，P2WASH がリターンする `AGG` はある address の許可なく，address が所有している 2 つ以上の UTXO を消費して 1 つの UTXO に合わせる．P2WASH の導入で提案手法を Bitcoin に適用できる．

Bitcoin のすべてのスクリプトはスクリプトに含まれている `public key` と `signature` の個数により P2WASH で対応できる．例えば，1 つの `public key` と `signature` のみを含む P2PK，P2PKH，P2WPKH 等のスクリプトは 1-of-1 P2WASH に対応し，n 個の `public key` と `signature` を含む x-of-n P2MS，P2SH，P2WSH 等のスクリプトは x-of-n P2WASH に対応する．

6 実　　験

本章では，5 章で述べた必要条件を前提として，提案手法を Bitcoin に適用する時の効果について記す．

6.1 実験データと仮定

Bitcoin で提案手法の実験するために収集したデータは 3 種類である．1 年間 (2020-12-05 2021-12-05) のデータであるブロック 52635 個とトランザクション 99162585 個を使用し，期間中に発生したスクリプトタイプを分析した．Glassnode [19]

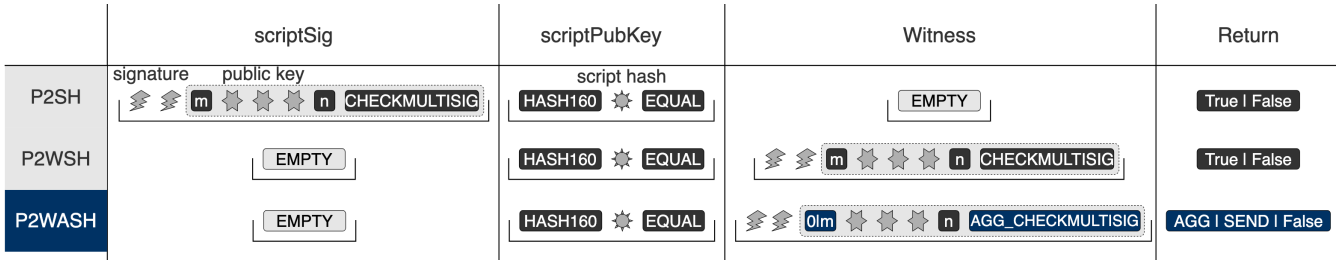

図 4　Bitcoin の既存スクリプトと P2WASH

の残高があるアドレスのデータを使用した．[20] で UTXO set の分析ツールとして使用された STATUS [21] で 2021 年 11 月時点の UTXO set を分析し，UTXO データを使用した．

時間の流れに応じた手法の効果推移を予測するため，実験のいくつかの仮定を行う．ブロックとトランザクションに含まれるスクリプトタイプの割合，残高があるアドレスの増加速度，トランザクションの input と output の割合は変化しないことにする．スクリプトタイプの約 99 % を占める P2SH，P2PKH，P2WPKH，P2WSH のみを考慮し，以外のスクリプトタイプは無視する．そして，m-of-n P2SH，P2WSH の 50 % は 1-of-2，残り 50 % は 2-of-3 と仮定する．実験の単純化のため epoch に一度マージすることにする．

6.2　ブロックの比較

スクリプトは種類によってそのサイズが異なり，表 1 はその内容を示す．ブロックに含まれるトランザクション，input，output の平均個数はそれぞれ 1884，5754，5976 個であり，P2SH，P2PKH，P2WPKH，P2WSH の割合はそれぞれ 36.2 %，20.1 %，40.9 %，1.8 %である．このときブロックのサイズは手法適用前は 1287333 bytes であり，適用後は 1370050 bytes となり約 6.4 %増加する．これは P2PKH，P2WPKH に比べて P2WASH のスクリプトサイズが大きいためである．

手法適用によって変化する vSize とブロック当たりの処理量変化は，付録 A で記述する．

6.3　UTXO set の比較

Bitcoin の UTXO set は key-value store である LevelDB として実装されている．Bitcoin(0.15.0 バージョン以降) の LevelDB の key は，ある UTXO が含まれたトランザクションのハッシュ値と index に対する情報を含む．value には，UTXO が含まれているブロックの高さ，トランザクションが coinbase であるか，金額，ScriptPubkey が含まれている．すなわち，LevelDB には UTXO の数の分だけ record を含んでいる．

提案手法を適用すると，LevelDB の record の数が変化する．4.5 節でマージの頻度数について言及したが，epoch に一度マージするか，ブロック毎にマージするかでその record 個数の変化の様相が異なる．もしブロックごとにマージするなら，UTXO の個数が残高があるアドレスの個数と同じなので，残高があるアドレスの個数だけの record だけ保存すればよい．しかし，epoch に一度マージするなら，UTXO と各アドレスの残高をすべて保存しなければならないので，残高があるアドレスの数に

	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

表 1　種類によるスクリプトの size と vSize

epoch 中に超過発生した UTXO の数を加えた値になる．2021 年 12 月 5 日の基準で残高があるアドレスの数は 39240264 で約 3900 万である．

6.4　Aggregation Block

Aggregation Block のサイズは，一般的なブロックのようにトランザクションの個数と input，output の個数によって決定される．Aggregation Block の input は，前の epoch 期間中のブロックで発生した input と output の違いによって決定される．そして，1 つのブロックで output の個数が input の個数より平均 222 個超過して発生する．epoch が 1ヶ月，6ヶ月，1 年の時に超過して発生する output の個数は，それぞれ 973076，5838456，11676912 で，この値は Aggregation Block の input の個数となる．Aggregation Block の output の個数は残高があるアドレスの個数で 39240264 である．

与えられた input，output，スクリプト比率，トランザクション数によって Aggregation Block のサイズを計算することができる．epoch が 1ヶ月のときは 2270892801 bytes で約 2.27 GB，epoch が 6ヶ月のときは 3194555515 bytes で約 3.19 GB，epoch が 1 年のときは 4309245744 bytes で約 4.31 GB である．

6.5　ストレージ上のブロックチェーンサイズの変化

ストレージ上のブロックチェーンのサイズの変化は，6.2 節で求めたブロックサイズと 6.4 節で求めた Aggregation Block のサイズから予測できる．一般的なブロックチェーンサイズに epoch ごとに Aggregation Block のサイズを加える結果である．

ローカルストレージに保存されるのは epoch 毎に，Aggregation Block 1 個と epoch 中に発生したブロックである．epoch

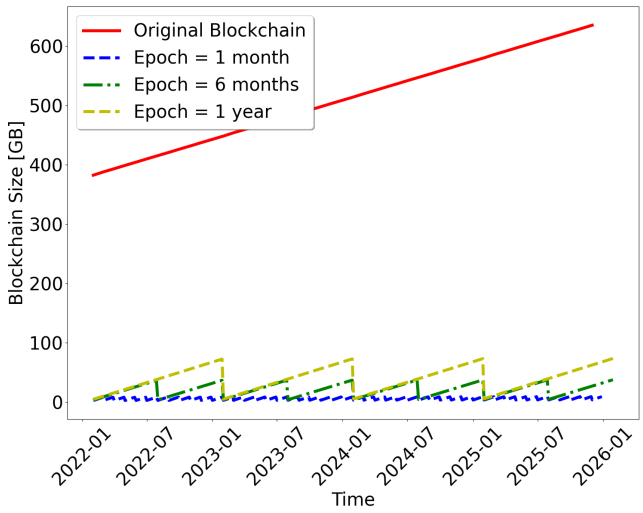

図 5　ストレージに 1 epoch のブロックチェーンを保存したときのサイズ変化

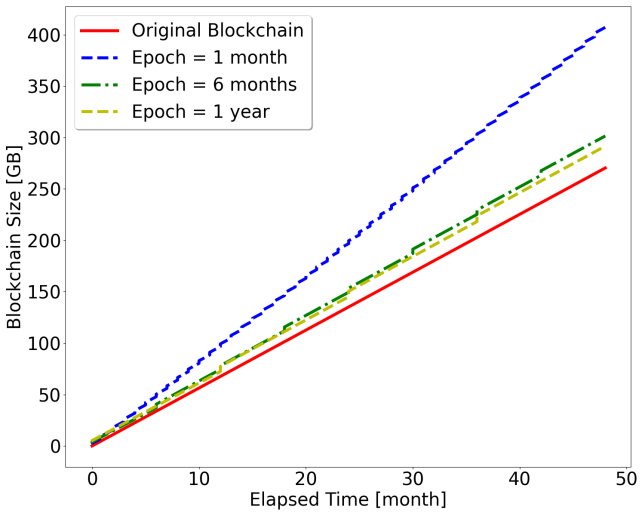

図 6　ブロックチェーンサイズの増加速度

が 1ヶ月のとき，発生するブロックの個数は平均 4386 個であり，Aggregation Block のサイズは約 2.43 GB である．残高があるアドレスは約 444139 個増加する．epoch が 6ヶ月のとき，発生するブロックの個数は平均 26316 個であり，Aggregation Block のサイズは約 3.19 GB である．残高があるアドレスは約 2911061 個増加する．epoch が 1 年のとき，発生するブロックの個数は平均 52632 個であり，Aggregation Block のサイズは約 4.31 GB である．残高があるアドレスは約 6314577 個増加する．これに基づいてローカルストレージに 1 epoch 分量のブロックチェーンを保存したときのサイズの変化を予測すると，図 5 のようになる．結果として epoch が 1ヶ月，6ヶ月，1 年で 1epoch のみをストレージに保存する際のブロックチェーンサイズは，従来のブロックチェーンサイズの 0.6 - 2.0 %，0.8 - 9.5 %，1.1 - 18.9 % 程度となる．

6.6　ブロックチェーンサイズの増加速度

提案提案では，ブロックチェーンに Aggregation Block が追加的に発生するので，ブロックチェーンのサイズは従来のブロックチェーンよりも早く増加する．図 6 は，手法適用前後の増加速度を示している．手法を適用しなかったとき，ブロックチェーンのサイズ増加速度は約 5.6 GB/month である．手法を適用したときのブロックチェーンサイズの増加速度は，epoch が 1ヶ月，6ヶ月，1 年のとき，それぞれ 8.4，6.2，6.0 GB/month であり，これは手法適用前の増加速度より 50，10，6 %増加した数値である．

6.7　経 済 効 果

提案提案は周期的に UTXO をマージするので，経済的価値の低い UTXO を減らす効果がある．UTXO の経済的価値は，UTXO の金額をスクリプトサイズで割った値で表現することができるが，[20] では経済的価値の低い UTXO を dust UTXO,non-profitable UTXO に分類する．dust UTXO はトランザクション生成のための 1 byte あたりの手数料が，UTXO が持つ 1 byte あたりの価値の 1/3 を超える UTXO であり，non-profitable UTXO は 1 byte あたりの手数料が 1 byte あたりの価値を超える UTXO である．

図 7 は，1 byte あたりの価値を基準として，UTXO の個数，スクリプトサイズ，総額の分布を示したものである．分布により，150 sat./byte 以下の UTXO の総額は全体 Bitcoin 価値の 0.012 %のみを占めるが，UTXO の個数は全 UTXO 個数の 49 %，スクリプトサイズは全スクリプトサイズの 50 %を占める．

トランザクション生成の手数料はスクリプトのサイズと transaction pool が混雑した程度によって決定されるため，時間の流れとともに変化する．[22] によるとこの 1 ヶ月間に最大の手数料は約 20 sat./byte，この 1 年間の最大の手数料は約 270 sat./byte，歴代最大の手数料は約 1000 sat./byte 程度である．図 8 は手数料による dust UTXO と non-profitable UTXO の総額を示す．手数料が 20 sat./byte の時に発生する dust UTXO と non-profitable UTXO はそれぞれ 36，30 BTC であり，手数料が 270 sat./byte た時は，それぞれ 3620，2540 BTC，手数料が 1000 sat./byte の時はそれぞれ 14120，8620 BTC である．

7　ま　と　め

本研究では，UTXO-based ブロックチェーンで適用できる UTXO Aggregation による Block Pruning を提案した．Bitcoin を含めた一部のブロックチェーンで Block Pruning をした際，あるいは関連された他の提案で，他のノードのブートスタップを助けることができず，UTXO の参照ブロックとトランザクションのクエリーが不可能な問題を含んでいる．提案手法は，周期的に UTXO をアドレスごとにマージして Aggregation Block を生成するが，Aggregation Block は他のノードのブートスタップを可能にするようにする UTXO らのスナップショットの役割を遂行することができる．そして周期的な Aggregation Block の生成は，UTXO の参照，ブロックにトランザクションがいつも最後の epoch に存在するようにして，関連情報を常に

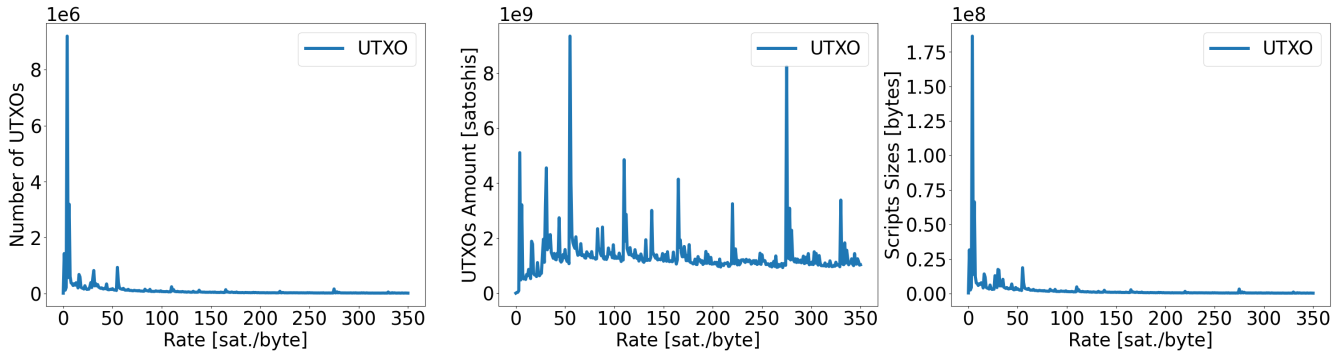

図 7　UTXO の単位価値による度数・総額・スクリプトサイズ分布

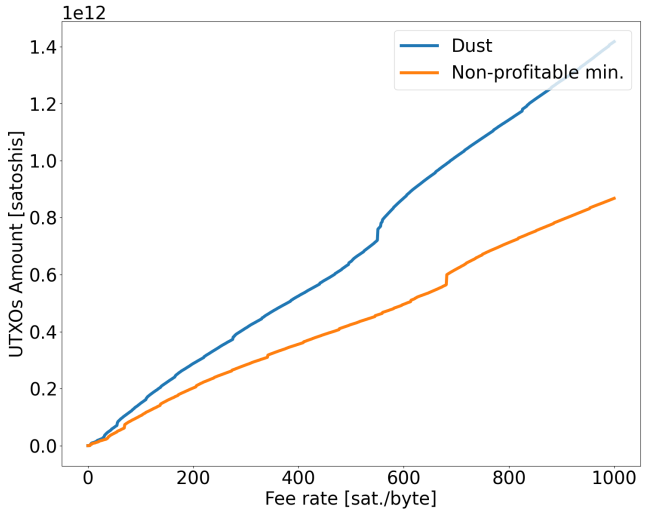

図 8　dust/non-profitable UTXO の累積分布

クエリーできるようにする．これはノードに UTXO set のような別のデータベースの保存を強要しないことを意味する．提案手法は既存の問題を解決するだけでなく，周期的に UTXO をマージすることで，経済的価値が低い UTXO を減らすのに役立つことができる．

10 年あるいは 100 年後のような長い時間を経て，ブロックチェーンが持続可能なシステムとなるためには，ノード運用のストレージ負担が必ず減らされるべきである．本稿の実験では，Bitcoin に提案手法を適用前後のブロックチェーンサイズを算定し比較した．ブロックチェーンサイズの増加速度は多少速くなったが，ストレージに保存すべきブロックチェーンのサイズは大幅に減少させることが可能であった．したがって，提案手法はノードの運用負担を減らすのに有効であるといえる．

提案手法を適用するためには，UTXO 使用の権限の一部をネットワークに共有しなければならない．そしてアドレスごとで UTXO をマージするにはブロックチェーンネットワークでの合意が必要である．このような理由から，既に動作中のブロックチェーンに提案手法を適用するためには，Bitcoin Cash, Bitcoin Gold のようにハードフォークする必要がある．もし，新しいブロックチェーンを設計するような状況では，提案手法がノードの運用負担を軽減するのに役立つ．

付　　録

A　ブロックの vSize と処理量

Bitcoin はブロックサイズには制限があるが，この制限は 1000000 bytes から segWit[18] の導入後，4000000 vbytes に変わった．ブロックの vSize を計算するとき，SegWit スクリプトの scriptSig を除いたすべてのデータに対して 4 倍を行う．

ブロックが占める vbyte は提案手法適用前の 3787198 vbytes であり，適用後は 1973890 vbytes になって 47.9 % 減少する．この違いは，すべてのアドレスが SegWit ベースであるために発生する．したがって，ブロックの実際のサイズは増加したものの，むしろブロックの vSize には余裕が生じ，既存にブロックに含まれるトランザクション量より追加で約 91 % のトランザクションがブロックに含まれることができる．

謝辞　本研究は JSPS 科研費 JP21H04872 の助成を受けたものです．

文　　献

[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] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[4] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[5] Bitcoin project, ”lightweight node”. `https://en.bitcoin.it/wiki/Lightweight_node`. Accessed: 2021-12-16.

[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] Emanuel Palm, Olov Schelén, and Ulf Bodin. Selective blockchain transaction pruning and state derivability. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 31–40. IEEE, 2018.

[11] JD Bruce. The mini-blockchain scheme. *White paper*, 2014.

[12] Andrew Poelstra. Mimblewimble. 2016.

[13] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. How to securely prune bitcoin ' s blockchain. In *2020 IFIP Networking Conference (Networking)*, pages 298–306. IEEE, 2020.

[14] B Swaroopa Reddy. secureprune: Secure block pruning in utxo based blockchains using accumulators. In *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 174–178. IEEE, 2021.

[15] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997.

[16] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Annual International Cryptology Conference*, pages 561–586. Springer, 2019.

[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] Sergi Delgado-Segura, Cristina Pérez-Sola, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. Analysis of the bitcoin utxo set. In *International Conference on Financial Cryptography and Data Security*, pages 78–91. Springer, 2018.

[21] Sergi Delgado Segura. Statistical analysis tool for utxo set. https://github.com/sr-gi/bitcoin_tools/tree/v0.2/bitcoin_tools/analysis/status, 2018.

[22] statoshi.info, "recommended transaction fee for target confirmation in x blocks". https://statoshi.info/. Accessed: 2021-12-16.