修士論文

時間概念を導入した
分散型台帳技術に関する研究

早稲田大学大学院基幹理工学研究科
情報理工・情報通信専攻

山田雄希

学籍番号	5116F088-5
提出年月日	2018 年 1 月 30 日
指導教授	中島達夫

The Study on Distributed Ledger Technologies based on the Clock Concept

Yuki Yamada

Thesis submitted in partial fulfillment of

the requirements for the degree of

Master in Computer Science and Communications Engineering

Student ID	5116F088-5
Submission Date	January 30. 2018
Supervisor	Prof. Tatsuo NAKAJIMA

Department of Computer Science and Engineering

School of Fundamental Science and Engineering

WASEDA University

概要

本研究はブロックチェーンに時間同期技術を取り入れることで，スケーラビリティに対して性能向上や問題解決が可能かを調査した．近年，分散システムでこれまで前提事項として取得が難しいとされてきたグローバルクロック（全てのノードが共有している同期された時刻）の取得が小型原子時計によって実現に近づいている．この背景を踏まえて改めてグローバルクロックの存在する分散システムについて考えた時，スケーラビリティの問題が指摘されているブロックチェーンに対して有効であると仮説を立てた．特に信頼できるタイムスタンプによってトランザクションやブロックの因果関係を保証することで，従来のブロックチェーンの処理を簡略化し，高速化が図れると考えた．本研究ではブロックチェーンのプロトタイプを作成し，原子時計による時刻取得のシミュレーション（物理グローバルクロックと定義する）に加えて既存技術である論理クロック（ベクタークロックまたはCBCAST）を実装して因果関係の保証を行うバージョン（論理グローバルクロックと定義する），クロックを実装していない通常のブロックチェーンの 4 種類のバージョンを作成して比較した．その結果，グローバルクロックはブロックチェーンのトランザクションに対して因果関係（順序関係）の保証をすることではスケーラビリティの向上に繋がらないことが分かった．また，論理グローバルクロックと物理グローバルクロックを比較した場合，論理グローバルクロックはノード数のスケールアップに応じてデータ量が増えることが判明し，現実的にグローバルクロックを適用するには物理グローバルクロックの方が適していること考えた．この結果からトランザクションの順序保証とスループット値はトレードオフの関係にあることが分買った．一方で，ブロックチェーンのブランチ分岐に対して，タイムスタンプを元に早い時刻でマイニングに成功するブランチを採用する方法に関しては，単位時間あたりのブロック生成数を向上させ結果的にスケーラビリティに貢献できることを示した．

目次

表目次

図目次

第1章　序論

1.1 背景

1.1.1 分散システムにおける時刻概念

コンピュータの世界においても我々の生活する実世界においても，日常的に利用されている時間の概念は仮想的なものである．例えば我々が普段利用している時刻は地球の経度に基づく物理的に正確な時刻の値ではない．我々が利用している時刻の概念は協定世界時（UTC）に基づく現在位置の時間帯の時刻を利用している例が大半である．その理由は日常的に時刻を利用する時において広範囲で共通の時刻を用いる方が効率的であるためである．

このような仮想的な時間の概念は実世界だけではなくコンピュータ上でも広く利用されている．特に重要な実例は，物理的にもネットワーク的にも遠く距離の離れたコンピュータ同士が協調し同期する分散システムである．現在広く用いられている分散システムの大半において時間は曖昧なものと言う前提で設計されている．この大きな理由の一つは常に正確な時刻を取得する保証が無いためである．分散システム内では，コンピュータ自身の故障やネットワークの遅延など様々な要素が複雑に絡み，常に全てのノードが共有されたグローバルな時間（グローバルクロック）を取得できるとは限らない．

このような背景を踏まえ，分散システムにおいて全ノードが共有した正確な時刻（グローバルクロック）を取得することは困難であることが前提として存在してきた．しかしその一方で，時間の概念はイベントの順序整合性など分散システムの一貫性の保証に非常に重要な要素である．[1]による例を参考に説明を行う．例えば銀行の預金システムを考えた時，「自身の口座に 10000 円を預け入れる」処理と「利子で預金残高の 1%に相当する金額を口座に追加する」処理がほぼ同時に発生した場合，この二つの処理のどちらを先に実行するかによって最終的な預金残高の値は異なる結果になる．しかし，ネットワーク的に距離の離れたノード A とノード B が存在したとき，ノード A のデータベースには先に「自身の口座に 1000 円を預け入れる」処理が到着し，ノード B のデータベースには先に「利子で預金残高の 1%に相当する金額を口座に追加する」処理が到着したとする．この場合，処理を到着順に実行して行くと，ノード A とノード B のデータベースはそれぞれ異なった結果を出す．このような状況を防ぐために一貫性の要素は非常に重要である．各々のコンピュータで発生したイベントの因果関係を正しく処理し順序整合性を保証するためには同期済みの時間を用いることが一つの最適な方法である．しかし現実的に正確なタイムスタンプを取得できる保証は無い．そこで正確な時刻は取得できなくとも，イベントの因果関係を保証すればよいという考えのもとで，アルゴリズムによって仮想的な時間を作りだす手法が提案された．この手法は論理クロックと呼ばれ，実際の例としてランポートクロック（Lamport Clock)やベクタークロック（Vector Clock）と言った手法が提案されてきた．[2]

本項をまとめると次のようになる．分散システムにおいて時間同期は重要な技術である．しかし現実的に物理的に正確な時刻（グローバルクロック）を取得することは難しい．そこで，グローバルクロックは取得できないと前提付けて，アルゴリズムによって因果関係だけを保証する論理クロックと言った手法が提案され利用されてきた．このような手法は現在広く分散システムで広く利用されるため，そもそも前提とは逆にグローバルクロックが取得できる環境下での同期技術は議論されてこなかった．

1.1.2　近年の分散システムの時間概念

前項にて前提として分散システムのグローバルクロックは取得できないと説明した．しかし，近年その前提が覆ろうとしている．その一つは原子時計を使った時刻同期の手法である．原子時計は原子や分子の性質を応用し限りなく低い確率でしか時刻のずれが発生しないため，正確な時刻を刻む時計である．例えば[3]によれば,光原子を用いた時計で約 10^-16 秒の精度を実現している．近年，この原子時計が安価になり始めている．これは近い将来，原子時計はコンピュータに搭載可能なほど小型化し，簡単に利用できるようになる可能性がある．例えば，チップスケール原子時計と呼ばれる取り組みが存在する[4]．もし小型の原子時計が実現し広く一般的に浸透すると，コンピュータはより低コストで正確な時刻を取得可能になる．例えば現在コンピュータで時刻同期として広く使われている Network Time Protocol(NTP)は他のサーバーと同期を取りながら時刻を修正している．しかし，原子時計であればコンピュータは同期処理を行わずに単独で常に正確な時刻を取得可能である．つまり，前項で説明したグローバルクロックは取得できないと言う前提が覆ると言える．

原子時計を使った時間同期はすでにクラウドサービス等で利用され始めている．その一例が Spanner[5]や Amazon Time Sync Service[6]である．このことから，原子時計を用いた時刻同期を導入する動きは近い将来より一般的になると考えられる．このように近い将来に従来ではありえなかったグローバルクロックの概念が実現に近づいている．そのような中で，我々は改めてグローバル時間が取得できる場合の同期技術について考える必要があると考えた．

1.1.3 ブロックチェーンへの注目

近年，新たな分散システムの一例としてブロックチェーン(blockchain)が注目されている．ブロックチェーンとは，Bitcoin [18]で提案された基盤技術を一般化した P2P（Peer-to-Peer）分散データベース技術である．またそのデータ構造から分散台帳（Distributed Ledger）と呼ばれることもある．ブロックチェーンの大きな特徴は，”非中央集権型システム(Decentralized system)”と呼ばれる，特定の運用する団体やノードの存在無しに自律した高い信頼性のネットワークを構成できる点である．ブロックチェーンは一定時間ごとに全システム内の未確定のトランザクションを複数選択し，まとめてブロックと言うデータ単位にし，それらを順番に線形リスト式に格納していく構造を持つ．そして全世界のネットワー

クに参加しているノードが同じデータを保持して同期し合うことで，データを更新する．ブロックは直前のブロックのデータをハッシュ値として保持する．これをポインタのように扱うハッシュポインタの構造を持つ．これにより，ハッシュ関数の計算困難性によって，特定のブロックを改ざんしようとしたときそのブロックに連なったすべてのハッシュポインタを改めて計算しなければならない．つまり，時間経過とともに改ざんを行う難易度が向上していく構造になっている．また，このデータをネットワークに参加するすべてのノードが比較し合うことで，ネットワークの参加ノードの過半数以上が不正行為を行わなければ改ざんが不可能である．このような仕組みによりブロックチェーンは改ざんが困難なデータ構造を実現している．ブロックチェーンは非中央集権的に自律する分散システムを構築可能な側面を応用し，スマートコントラクト[8]と呼ばれる契約の自動化や低コストでの高可用高信頼なシステムを実現するインフラ技術と言った側面で注目されている．

一方でブロックチェーンは正確な順序整合性を持たない点において問題を抱えている．現在のブロックチェーンにおいてトランザクションの順序や時刻は曖昧である．トランザクションの順番は実質的にそのトランザクションが含まれるブロックの順番とされている．この性質を悪用することで，意図的に二重支払いを行うことが可能である[27]．しかし仮に二重支払い等によって不正なトランザクションや順序がずれたトランザクションが到着してもブロックの生成時に精査されるため，時間の経過と共に解決する問題である．しかし，トランザクションがブロックに含まれることが遅れてしまう現象が頻発すればそれはネットワーク全体の大きな遅延に繋がってしまう．ブロックチェーンや Bitcoin はトランザクションのスケーラビリティ性に問題を抱えていると言われている[7]．ブロックチェーンのスケーラビリティ問題を解消する一つの方法として，可能な限りトランザクションを高速にブロックの中に取り込むことは重要である．他にも，仮想通貨の基盤としてブロックチェーンを利用することを考えた場合，仮想通貨と商品を交換したが，意図的に商品提供者に通貨を与えるトランザクションを削除する行為が可能になる．この対策として，トランザクションがブロックに確実に取り込まれるまで待機する方法があるが，この方法も効率的ではない．

スケーラビリティに関連するもう一つの問題として，ブロックチェーンのブランチ分岐問題が存在する．例えば，同時に 2 箇所でマイニングに成功したノードが現れた場合を考える．この場合，マイニングに成功したノードとその近くのノードはそれぞれのマイニング結果で自身のブロックチェーンを書き換える．この時，ブロックチェーンの最先端のブロックは 2 種類存在することになり，一時的に枝分かれしたと言える．この場合，何かしらのルールによってどちらかのブランチを採用する．例えば Bitcoin の場合はより多くのブロックが連なっているブランチ，つまり長いブランチが採用される．もしブランチの長さが同一だった場合は，確率的に低いマイニングを成功させた方，つまりマイニングの結果で得られたハッシュ値の合計値によって採用する．このように，仮にブランチの分岐が発生しても時間経過と共に最も優れたブランチに収束するように設計されている．しかし，この性質を悪用

し意図的にネットワークを遮断する攻撃手法（eclipse attack）[20]が存在する．また，ブランチの訂正が行われた場合，トランザクションの含まれるブロックの情報が変化する．場合によっては，ブロックに含まれないことになり再度マイニングが成功するまで待つ必要がある．つまり，一度トランザクションを送信しブロックに含まれた後でも，ブランチの分岐と訂正が発生するリスクがほぼ 0 になるまで待たなければトランザクションが確定したとは言えない，と言う現状が存在する．例えば Bitcoin の例で言えば約 6 ブロック，約 60 分の時間が必要になる．このブロックの分岐に関する問題もスケーラビリティを大きく下げる要因になっている．

このような背景を踏まえ，1.1.1 で説明したグローバルクロックの概念を取り入れることは解決につながる可能性がある．グローバルクロックによって正確な時刻を取得もしくはトランザクションの因果関係を定義することで解決できる可能性が存在する．

1.2 研究目的

本研究はブロックチェーンの合意形成に時間同期技術を取り入れることで性能向上や問題解決が可能かを調査することが目的である．特にブロックチェーンのスケーラビリティ問題に対して原子時計等を用いたグローバルクロックの概念を取り入れる手法で解決に貢献可能か調査を行う．

本研究の目的をより詳細に議論するため，以下の 2 つの項目について調査を行う．一つ目に，「グローバルクロックを適用することでスケーラビリティを向上できるか」について，二つ目に「グローバルクロックを実現する手法によりどのような差異が生まれるか」についてである．

本研究ではグローバルクロックを実現する手法として次の 2 つを定義する．一つは論理クロックなど従来の分散システムのアルゴリズムを使ってグローバルクロックを実現する”論理グローバルクロック”，二つ目にチップスケール原子時計を用いて単独で正確な時刻が取得できる環境を想定した”物理グローバルクロック”である．本研究ではこの論理グローバルクロックと物理グローバルクロックの差異を評価することで，原子時計によるグローバルクロックの必要性について議論する．

1.3 論文構成

本論文の構成は以下のようになる.

第 2 章　関連研究

グローバルクロックを用いた同期技術に関する研究，ブロックチェーンのスケーラビリティに関する研究を説明する.

第 3 章　研究の方法論

第 1 章を踏まえ，どのようなシステムとどのような評価を行うべきかを説明する.

第 4 章　実装

第 3 章の調査方法に基づいて, どのようなプログラムによって調査を行うかを説明する.

第 5 章　評価

第 4 章で説明したプログラムを用いて，グローバルクロックを導入したブロックチェーンの性能評価を行う.

第 6 章　考察

第 5 章の結果を元に，この提案がブロックチェーンの性能向上および問題解決の実現に貢献可能かを議論する.

第 7 章　将来展望

今回の結果を元に，今後どのような研究が求められるかを議論する.

第 8 章　結論

本研究の結論を述べる.

第2章　関連研究

2.1 分散システムにおける時刻同期と合意

第 1 章で説明したように分散システムにおいて全てのノードが正確な同一の時間（グローバル時間）を取得することは難しい．グローバルな時間を持っていないと言うことは，分散システムにおける合意形成などの面で致命的となる．これはビザンチン将軍問題[21]と呼ばれる問題にも直結する．しかし一方で，イベントの因果関係を取得できればグローバルクロックは必要ないと言える．実例をあげるならば，現在広く利用されている分散合意アルゴリズムである Paxos[22]や Raft[23], PBFT[24]などは合意形成に厳密なタイムスタンプを用いることなく合意形成を行っている．

分散システムの時刻同期に関して着目する．現在コンピュータ上で広く利用されている時刻は主に NTP を利用したものである．また時刻同期に着目すれば論理クロックなどの手法でイベントの因果関係を簡潔に比較する方法が存在する．実例としてランポートクロックやベクタークロック存在する[2]．これらの仕組みは主にカウンタをノードが保持し，何かコンピュータ内で処理が発生すればその値を加算，この値を他のノードと比較し合うことで順序を比較するものである．

本研究ではこの論理クロックの仕組みが改めてブロックチェーンに効果を持つのか調査することが一つの目的である．ブロックチェーンは曖昧な時間概念を持っている．Bitcoin を例にあげると，Proof-of-Work によって一定間隔で生成されたブロックの連鎖の仕組みをタイムスタンプサーバーとして利用することが可能である[18]．しかし，この方法はブロックの順序が実質的にトランザクションの順序となる曖昧な時間概念の方法である．論理クロックのようにトランザクション一つ一つに対して順序整合性を検証し，現在よりは厳密な順序を保証する仕組みをブロックチェーンに適用し検証することは重要である．

2.2 グローバルクロックを用いた時刻同期

第 1 章で説明した通り，近年原子時計等を用いることでグローバルクロックの実現が近づいている．実例としてチップスケール原子時計[4]が挙げられる．これは原子時計を小型化し，サーバーやコンピュータに搭載可能にする．また近年にはスマートフォンに搭載可能な原子時計[25]を実現する研究が存在する．このような小型原子時計の仕組みの応用例として例えば Spanner[5]と言う分散データベースの TrueTime API のように原子時計を用いた時刻取得や Amazon Web Services の Time Sync Service[6]のような事例が挙げられる．本研究ではこのグローバルクロックをブロックチェーンに適用することでどのような差異が存在するか調査することが一つの目的である．グローバルクロックを適用する事例は多くあるが，ブロックチェーンに対しての実例は少なく検証の余地がある．

2.3 ブロックチェーンとスケーラビリティ

ブロックチェーンにおいてスケーラビリティの問題が懸念されている[7]．これは言い換えれば，Bitcoin などのシステムはユーザー数やトランザクション量のスケールアップに耐えうることができないと言うことであり，これは暗号通貨などブロックチェーンを用いた分散アプケーションにおいて重要な要素である．ブロックチェーンは smart contracts[8]と呼ばれる契約の自動化の実現や非中央集権アプリケーション（Dapps）[9]と呼ばれる新しいアプリケーションなどの分野に応用されることが期待され，将来の社会基盤となり得る可能性がある．その場合にブロックチェーンは現在よりも更にスケールアップした規模のものになると考えられる．このような事象を考えた場合，スケーラビリティ問題は非常に致命的なものとなる．

例えば Bitcoin を例に取ると，Proof-of-Work の仕組みと難易度調整，そして生成されるブロックのサイズ制限からシステムのスループットや約 7tx/s（トランザクション/秒）と言われている．これは VISA などの既存決済システムの平均 2000 tx/s の値と比較しても大きく差異がある[10]．近年の事例として，ブロックサイズの制限を緩和する Bitcoin Cash プロジェクト[11]やブロックチェーン外（off-chain）でトランザクション処理を行う Lightning network[12]，ブロックチェーンに別のブロックチェーンを接続することでトランザクション処理を行う Sidechains[13]などプロトコルを大きく変更せずに解決する取り組みや，Bitcoin-NG[14]などのプロトコルの改変に関する研究が存在する．1.1.3 で説明したように，トランザクションレベルだけではなくブロックレベルでもスケーラビリティの問題が発生している．近年の事例として，ブロックの生成を分散アルゴリズムで承認されたノード群が行うことで，ブランチの分岐を発生させない Algorand[15]や directed acyclic graph(DAG)のデータ構造をブロックチェーンに導入し，ある程度ブランチの分岐を許容する仕組みになる IOTA[16]などが存在する．

本研究ではグローバルクロックによる順序整合性により，トランザクションの時点でスループットを向上させることでスケーラビリティ問題へ貢献することを考える．ブロック生成時の時点で既に順序が保証された状態であれば，検証にかかる処理を短縮，簡略化することが可能になると考える．

第3章　研究の方法論

3.1 概念

1.2 で説明したように，本研究において論理グローバルクロックと物理グローバルクロックの 2 つの概念をブロックチェーンに適用することで，それぞれどのような差異があるかを評価する．

3.2 実装要件

この評価を行うために本研究ではシミュレーション用のブロックチェーンを作成する必要がある．このプログラムを変更することで 2 種類のバージョンを作ることで論理グローバルクロックと物理グローバルクロックを実現する．また，現時点では物理グローバルクロックの取得は困難であるため，シミュレーションとしての値を利用することが好ましい．

3.3 評価項目

評価を行うべき項目は次の通りである．まずネットワークを想定し，メッセージ量，ユーザー数のスケールアップに対してメッセージのスループットがどの程度変化するかを調査する．次に，メッセージの順序整合性を保証するべきタイミングについて考察するために，トランザクション受信時とブロック生成時の大きく二つで比較を行う．

これらの評価を踏まえて，「クロックが存在することでメリットがあるか」をクロックがある場合とない場合で比較する．次に「クロックの中でも，どのような種類のクロックが最適化」を比較する．この 2 軸を中心に評価を行う．

第4章　実装

4.1 概要

本研究では 3.2 で説明したように，はじめにブロックチェーンを制作し，それにグローバルクロックの実装を行った二つのバージョンを用意することで比較を行う．本研究では論理グローバルクロックの例として，ベクタークロックを実装したバージョンと Causal Multicast（CBCAST）を実装した 2 つの実装を用意した．ベクタークロック実装は単純にトランザクションの因果関係を比較するだけであるが，CBCAST はベクタークロックを応用し順序の遅れたトランザクションを遅延させ正しい順序で受信する仕組みを持つ．この両者を比較することで，トランザクション受信時に順序整合性の厳密な比較を行うことの意義を合わせて調査する．ブロックチェーンにおいてトランザクションはブロックに含まれるまで確定しない（承認されない）．この性質からトランザクション受信時に順序整合性を比較する実装も，ブロック生成時に順序整合性の判定を行うことも可能である．これを踏まえ，トランザクションが未承認の状態で厳密に順序の判定を行うべきか，あるいはブロック生成時のみ順序の判定を行えばいいか比較する．まとめると，本研究では論理グローバルクロックで 2 種類，物理グローバルクロックで 1 種類，また比較用にこれらのクロックを実装してないバージョンで 1 種類の計 4 種類のバージョンを実装し評価した．概要を表 4.1 に示す．ブロックチェーンは Web API に近い形式で動作する Web アプリケーションである．API コールの形で各種関数を呼び出し，簡易的なゴシッププロトコルにより，予め接続されたノードへ結果を送信することでブロードキャストを行う．まとめると，以下の表 4.1 の通りである．また，物理グローバルクロックブロックチェーンは原子時計による正確な時刻が取得可能と仮定したシミュレーションによるものである．

本研究におけるブロックチェーンはオープンソースのソースコードを元に独自に機能を拡張，またグローバルクロックの実装を行った[17]．本研究のソースコードは GitHub にて公開を行っている．（https://github.com/ykymd/blockchain-with-globalclock）

表 4.1 実装するアプリケーションの概要

名称	特徴
クロック無しブロックチェーン	クロックの実装を行なっていない通常のブロックチェーン
論理グローバルクロックブロックチェーン 1	ベクタークロックを実装したブロックチェーン
論理グローバルクロックブロックチェーン 2	CBCAST を参考にした因果的順序マルチキャストを実装したブロックチェーン
物理グローバルクロックブロックチェーン	正確なタイムスタンプが取得可能として実装されたブロックチェーン

4.2 開発環境

本研究ではブロックチェーンの開発プログラミング言語に Python を利用した. また Web フレームワークとして Flask を利用した．サーバーとして uWSGI および nginx を利用し稼働させた．また，評価においてスケーリング等を完結にするために Docker を利用した．コンテナのオーケストレーションとして Docker-Compose を利用した．またデータベースとして MongoDB を利用した．MongoDB はブロックチェーンの値や接続されたノード情報，transaction pool と呼ばれる未承認トランザクションの格納場所など原則的に永続化する情報の大半を管理している．各ソフトウェアのバージョンは以下の表 4.2 に示す.

表 4.2 開発環境およびバージョン一覧

名称	バージョン
Python	3.6.3
Flask	0.12.2
uWSGI	2.0.15
nginx	1.13.7
Docker	17.09.1-ce
Docker-Compose	1.18.0
MongoDB	3.2.11

4.3アプリケーション構成

制作したブロックチェーンの主要なインタフェースについて説明する. 表 4.3 に概要を示す.

表 4.3 アプリケーションのインタフェース

フィールド	説明
GET /balance	残高を表示する
GET /mine	マイニングによりブロックを追加する
POST /transaction/new	新規トランザクションを発行する
GET /chain	id 番目の投稿データの詳細画面を表示する
GET /nodes	自身と接続しているノードの IP アドレスを表示する
GET /pool	ブロックに追加されていない（未承認）トランザクションを表示する
POST /nodes/register	新しくノードに接続する
POST /nodes/resolve	自身のブロックチェーンを接続されたノードと比較する

次に各インタフェースの説明を行う．まず/balance は自身の残高の取得を行う．各ノードはプロセスの起動時に UUID が割り振られ，それを自身のアドレスとして扱う．/balance を実行した時に，自身が持つブロックチェーン内から自分のアドレスに向けられた，または自分が発行したトランザクションを集計し，現在の残高を計算する．Bitcoin などの暗号通貨は秘密鍵から発行された公開鍵をアドレスとして使うことで複数のアドレスを一つの秘密鍵で管理可能であるが，本研究では簡素化のため一つのノードに一つのアドレスで管理を行う．/mine はマイニングと呼ばれるハッシュの計算を行い，次のブロックの生成に挑戦する処理を実行する．ここで行われる計算は Bitcoin で行われる Proof-of-Work[18]を簡素化したもので，乱数によって生成された値 proof と，直前に存在したブロックの proof 値(lastproof)を並べてハッシュ関数の引数に用いた時，ハッシュの先頭 4 文字が 0000 になるものを次の正しい proof とし，その値を発見するまで試行するという処理である．/transaction/new は新しいトランザクションを発行し，接続されたノードにその情報を送る．/mine によってブロックに取り込まれるまでは，transaction pool と呼ばれる一時的なデータベースに情報が格納される．Transaction pool の値は/pool で確認可能である．/chain は現在のブロックチェーンのデータを表示する．これは/nodes/resolve を呼び出した際に，自身と接続されている他のノードと/chain を比較し合って最も優れたブロックチェーンを採用し，自身のブロックチェーンを上書きするようにしている．ここで言う最も優れたブロックチェーンとは，難易度の高いマイニングに成功したもの，つまり連鎖したブロックが多い（長い）方，または長さが同じであった場合，ブロックの proof 値を数値として見たときに値が小さい方を採用する仕組みになっている．/nodes は自身と接続されたノードの情報（IP アドレス）を表示し，/nodes/register によって新しく登録を行う．

4.4 主要アルゴリズムの説明

本研究で実装したアルゴリズムの概要を説明する．

4.4.1　ベクタークロック

ベクタークロック（Vector Clock）は各ノードがクロック値を持ち，その値を他のノードのものと合わせて保持することで，他のノード内のイベントのクロック値をベクトルのように比較することで因果性を判定する手法である．ベクタークロックは以下の法則に従う．

1. ノードがメッセージを送信する際に，自身のクロック値と他のノードのクロック値の情報をメッセージに乗せる．
2. メッセージ送信後に自身のクロック値を 1 加算する
3. メッセージを受信した場合，メッセージに乗っているクロック値と自身の値を要素ごとに比較する．両者の値のうち大きい方を自身のクロック値として更新する

実際の値を用いて説明する．ノードが 3 台稼働していた時，全ノードのクロック値を v = [0, 0, 0]，個別のクロック値をノード ID が i のとき v[i] = 0 で表現できる．ID:0 のノードが初めてメッセージを送るときには，[0,0,0]の値をメッセージに付与して送信し，送信完了後に自身のクロック値 v[0]に 1 加算する．この例では v=[1, 0, 0]になる．メッセージを受信したとき，例えば 1 番目のノードが[2, 3, 4]を保持していてメッセージに[3, 2, 5]のベクトル情報が載ったメッセージを受け取ったとき，1 番目のノードはクロック値を更新して[3, 3, 5]となる．

また，ベクトルの各要素全てがあるベクトルの全要素より大きい場合，そのベクトルを持ったメッセージは後に発生したことがわかる．例えば，ベクトル[2, 3, 4]はベクトル[1,2,3]のイベントより後に発生したことが分かる．しかし，ベクトル[2, 1, 5]のような場合では比較不可能である．

4.4.2　CBCAST (Causal Multicast)

CBCAST(Causal Multicast)は ISIS プロジェクトで用いられた因果性を持たせたマルチキャストの手法である[26]．本研究では CBCAST を参考に因果性を持ったマルチキャストを実装した．4.4.1 と同じように各ノードは全ノードのクロック値を持ち，以下のルールに従って同期を行う．

1. ノードがメッセージを送信する際に，自身のクロック値と他のノードのクロック値の情報をメッセージに乗せる．
2. ベクトルの要素 v[i]はノード ID: i から正しく受信したメッセージの最後の番号とする
3. メッセージ送信前に自身のクロック値を 1 加算してメッセージに乗せる

4. メッセージを受信した際に，自身の保持するクロック値とメッセージのクロック値を比較する．送信元のノード ID が j であるとき，メッセージ内の v[j]が自身の v[j]より 1 大きい，かつ j 以外の全ての要素が自身の要素より同値または小さい，この条件を満たしたときにメッセージを受け入れる．それ以外はキューに保存する．
5. 3 においてキューに保存されたメッセージは再び 3 の条件を満たした場合に受け入れられる．

実際の値を用いて説明する．4.4.1 と同じように 3 台のノードが存在し，ノード 1(ID:1)のベクトルが[0,0,0]であったとする．この時，ノード 0(ID:0)から[2,0,0]の値を乗せたメッセージが届いた．このときノード 1 はルールに満たないためメッセージをキューに保存する．その後 0 から[1,0,0]のメッセージが届いた．ノード 1 はルールを満たしているためメッセージを受け入れ，ベクトルを[1,0,0]に変更する．その後，[2,0,0]のメッセージは条件を満たすため，受け入れられる．このようにしてメッセージを正しい順序で受信することが可能となる．

4.4.3 物理グローバルクロック同期

本研究の実装において，グローバルクロックは取得可能という前提でシミュレーションを行う．つまり，実際はシステムクロックの値を取得し，それを正確なタイムスタンプと見なして同期を行う．物理グローバルクロックはブロック生成時に正確な順序でトランザクションが整列された状態で処理されることを想定している．しかし第 5 章の評価ではトランザクションによる順序整合性を中心とした評価であり，物理グローバルクロックの値はトランザクション整合性には大きく関与していない．そのため，第 5 章の評価項目においてはクロック無しのバージョンと殆ど機能が変わらない．またブロック生成時の順序保証やタイムスタンプの活用については第 6 章で論ずる．

第5章　評価

本章では評価方法および得られた結果についての説明を行う．まず，調査に用いたシステムの構成を示す．次に，調査の方法と調査項目を説明する．そして，得られた結果について説明する．

5.1 検証環境

調査に用いたシステムの構成を説明する．本研究では Amazon Web Services の EC2 インスタンス上で Docker コンテナを起動し評価を行った．インスタンスの仕様を表 5.1 に示す．

表 5.1 調査システムの仕様

インスタンスタイプ	c4.4xlarge
vCPU	16
メモリ	30GiB
専用 EBS 帯域幅	2000Mbps
OS	Amazon Linux

5.2 調査項目

本研究では第 4 章で示した 4 種類のアプリケーションを調査対象とする．各アプリケーションを Docker コンテナとしてシステム上に配置する．このアプリケーションに対して調査用のスクリプトを実行し，その結果を解析することで調査を行う．
次に検証する項目について説明する．今回は 3 つの調査項目について測定を行う．

5.2.1 ノード数に対してのメッセージ量

はじめにブロックチェーンネットワークに参加するユーザー数（ノード数）に応じて，メッセージ量がどのように変化するかを調査する．Docker- Compose の機能を用いて，複数台のコンテナを起動させる．次に各コンテナ同士を接続し合う．接続するノードは乱数によって決定する．その後，シェルスクリプトを用いて各ノードに対して新規トランザクション発行のリクエストを行う．トランザクションの送信が完了した際に，再び次のトランザクション発行リクエストを送信する．10 分経過後に各ノードの中で最も多く処理を行ったトランザクション数の値を取得し，時間で割ることでネットワークのスループット値とする．これをノード数に応じて複数回行い，結果を取得する．

5.2.2　スループットに対してのメッセージ量

二つ目に，ノードに送信するメッセージ量のスループット値に対して，実際のメッセージスループットの値を調査する．まず，5.2.1 と同じようにサーバー内に 50 台の Docker コンテナを立ち上げそれらを乱数によって接続する．その後，スループット値に合わせて各ノードにトランザクション発行リクエストを送信する．5.2.1 と違って送信完了を待たずにあらかじめ決められたスループットを維持するように並列的に送信する．10 分経過後に各ノードの中で最も多く処理を行ったトランザクション数の値を取得し，時間で割ることでネットワークのスループット値とする．これをスループット値に応じて複数回行い，結果を取得する．

5.2.3　ブロック順序整合性によるブランチ分岐への対処

5.2.1, 5.2.2 はグローバルクロックによるトランザクション順序整合性を調査しているが，本項目はブロックチェーンのブロックの順序整合性を調査する．ブロックチェーンのスケーラビリティ問題には 1.1.3 で述べたようにブランチ分岐の問題が存在する．本項ではブランチ分岐でのグローバルクロックの対処を考察し，簡易的な評価を行う．本調査項目は予めシナリオを定義し考察することで導出した．シナリオは 5.3.3 に示す．

5.3　分析および結果

本節では評価によって得られた結果を説明する．また 5.3.3 ではシナリオを定義し分析をすることで評価を行った．

5.3.1　ノード数に対してのメッセージ量

得られた結果を表 5.2 に示す．no-clock はクロック無し実装，vector はベクタークロックを実装したバージョン，cbcast は CBCAST を実装したバージョン，physical は物理グローバルクロックを実装したバージョンである．表 5.2 の単位は 1 秒あたりのトランザクション数（tx/s)である．また，表 5.2 をまとめたグラフを図 5.1 に示す．図 5.1 の横軸はノード数，縦軸は 1 秒あたりのトランザクション数（tx/s)を表す．

表 5.2 ノード数に対してのメッセージ量

ノード数	no-clock	vector	cbcast	physical
1	724	747	759	759
5	682	711	719	720
10	631	558	363	549
50	395	298	64	286
100	277	217	32	331

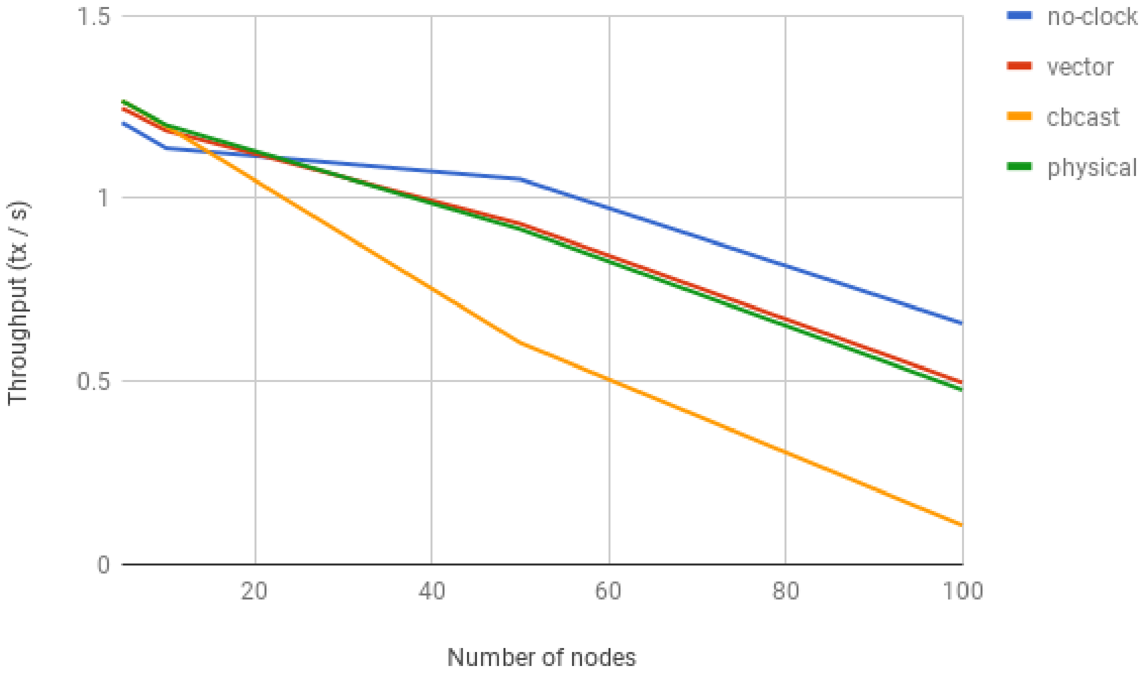

図 5.1 ノード数に対してのメッセージ量

5.3.2 スループットに対してのメッセージ量

得られた結果を表 5.3 に示す．送信したメッセージスループット値（tx/s)に対して，実際のブロックチェーンが時間内に処理したメッセージ数を表す．凡例の名称は 5.2.1 と同一である．また表 5.3 をグラフとして図 5.2 に示す．図 5.2 の横軸は送信した 1 秒あたりのトランザクション数（tx/s)，縦軸は実際の 1 秒あたりのトランザクション数（tx/s)に対する横軸の割合を表す．つまり本来想定していたスループット値と比較して実際のスループット値は何%であったかを表す．

表 5.3 スループットに対してのメッセージ量

送信したメッセージスループット(tx / s)	no-clock	vector	cbcast	physical
1	600	600	58	600
2	910	288	120	210
4	83	59	231	306
5	320	300	134	198
10	72	119	141	356

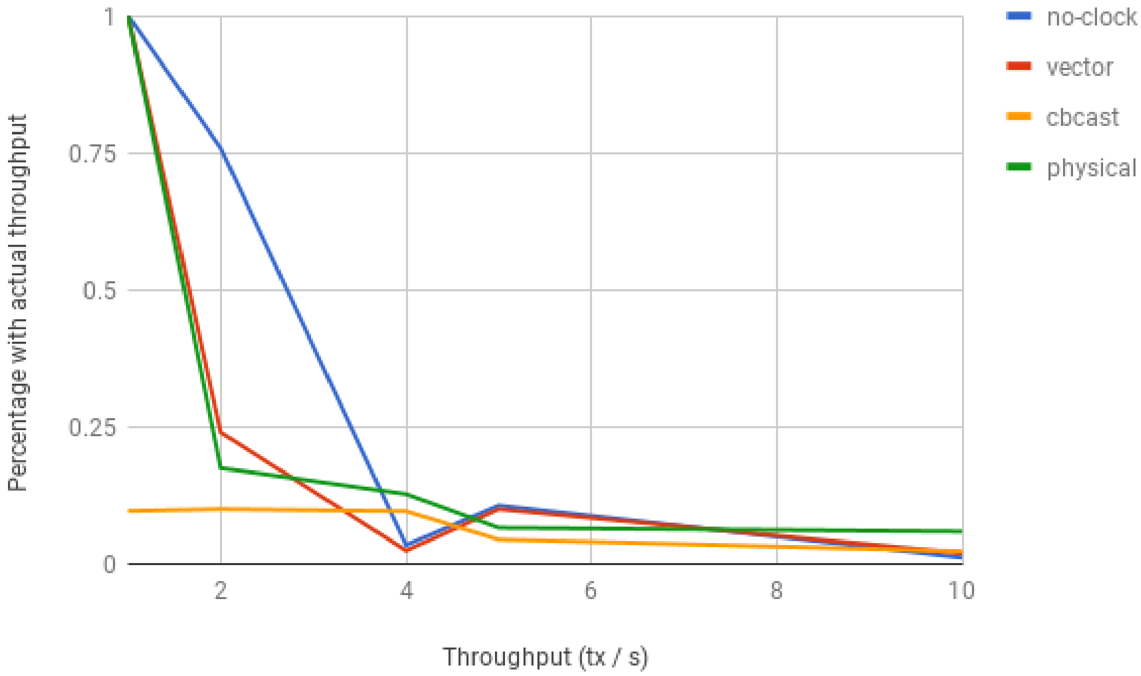

図 5.2 スループットに対しての実際のメッセージ量との割合

5.3.3 グローバルクロックを用いた分岐ブランチ選択

本項ではグローバルクロックによるタイムスタンプを用いたブロックのブランチ選択についての結果を示す．本項は，はじめにシナリオを定義し，それを分析することで評価を行った．次項より説明する．

5.3.3.1 ブランチ分岐が発生する場合のシナリオ

ブランチ分岐が発生する場合の大きな要因は同時にマイニングに成功したノードが発生し，メッセージが届くまでに遅延がある場合と，ネットワークが分断される場合である．ここでは同時にマイニングに成功した場合のブランチ分岐の動作を暗号通貨を例にして簡単にシナリオとして説明する．

あるユーザーが暗号通貨を送金するためにトランザクションを発行した．トランザクションリクエストはゴシッププロトコルを通じて，時間をかけて全ユーザーへブロードキャストされた．このリクエストはマイニングを行っているノードであるマイナーA とマイナーA からネットワーク的に遠隔地にあるノードであるマイナーB にも到着した．マイナーA およびマイナーB は次のブロックを生成するためにマイニングの計算を行っている．この時点でマイナーA とマイナーB には既に 99 個のブロックのデータが最新のブロックチェーンとして存在しているため，もしマイニングに成功すればこのブロックは 100 番目になる．やがて 10 分経過後，マイナーA がマイニングに成功しハッシュ値 0000abcd を発見した．

すぐさまマイナーA はブロック 100 を生成し，自身に近いノードにその情報をブロードキャストした. 一方で, マイナーA が発見して数秒後に, 偶然マイナーB がハッシュ値 0000ef12 を発見した．すぐさまマイナーB はブロック 100 を生成し，自身に近いノードにその情報をブロードキャストした．ここでネットワーク上に存在するあるノード C を考える．ノード C はあるとき自身に近いノードからマイナーA が生成したブロックの情報が送られてくる．ノード C はそのブロックの情報を検証し，正しいことが分かったため，自身のブロックチェーンにブロック 100 として追加した．その後，また自身に近いノードから今度はマイナーB が生成したブロック 100 の情報が送られてきた．ノード C は両者のブロック 100 を比較し，マイナーB の発見したハッシュ値 0000ef12 の方が優れていることが計算によって判明した．ノード C は自身のブロックチェーンをマイナーB のブロック情報で書き換えた．この優れたブランチを判定する計算方法は静的であるため，どのノードが計算しても必ず同じ結果になる．やがてネットワーク上に参加するノードは段々とマイナーB の作成したブランチを採用し，最終的に多数派となりマイナーB の作成したブランチに収束する．このようにして，時間経過と共にブロックチェーンは特定のブランチに収束する仕組みになっている．

5.3.3.2 ブランチ分岐が発生する場合の最悪のシナリオ

次にブランチ分岐がスケーラビリティに影響を及ぼす最悪のケースを考えてみる．何かしらの理由でブロックチェーンネットワークが分断され全ユーザーの 49%が所属するネットワーク A と 51%が所属するネットワーク B が出来た．その時，両ネットワークで共通していた最新のブロックのブロック番号は 99 だったとする．ある日，ネットワーク A に所属するユーザーが送金のためにトランザクション A'を発行した．そのトランザクション A'はネットワーク A に所属するマイナーA によってブロック 100 に含まれることで確定した．やがて時間経過と共にマイニングが様々なノードで行われブロック 200 までが生成された．一方で，ネットワーク B でも独自でマイニングが行われ続け，ブロック 201 までが生成された．ネットワーク B のノードにトランザクション A'は到達することがなかったため，時系列的にブロック 100 に含めるべきであっても，ネットワーク B 上のブロックチェーンには存在しない．ある日，ネットワークの分断が解消されネットワーク A とネットワーク B は統合された．この時，優れたブランチは長さの長いネットワーク B のブランチ（ブランチ B）であった．やがて時間経過と共に，かつてネットワーク A に所属していたノードはネットワーク A 内でマイニングされ続けていたブランチ A を破棄し，ブランチ B を採用した．ネットワーク A に所属していたノードはかつてブランチ A のブロックに含まれていたトランザクションは未承認トランザクションとしてブロードキャストされ，ブロック 201 以降に含まれるように待つことになった．この時，今までブランチ A 内に存在していたトランザクションはブロック 100 個分である．このトランザクションがネットワークに再び放出された時，スループットは大幅に低下する．また，未承認トランザクションが多いと言

うことは新しくトランザクションを発行した場合も次に生成されるブロックに含まれる確率が低くなり，ブロックに含まれるまで何回も待つ必要が出る．更に，ブランチ A 内でマイニングの報酬を受け取っていたノードがその報酬を使ったトランザクションを発行していた場合，ブロックそのものが破棄されるため報酬そのものが無かったことになり，報酬を使ったトランザクションは全て無効トランザクションとなる．すると，無効トランザクションを使ったトランザクションはまた無効になる連鎖が続き，大量のユーザーの残高から通貨が無かったことになる．このような矛盾とその修正が連鎖することが最悪のシナリオとして考えられる．

5.3.3.3 グローバルクロックを用いたブランチ分岐問題への対処

グローバルクロック概念を用いることでブランチ分岐問題の対処を考察する．まず，グローバルクロックによってブロックの順序を保証することで 5.2.3.2 のシナリオに対処可能であるとは考えにくい．その理由はブロックチェーンのデータ一貫性の保証の問題である．ブロックに正確なタイムスタンプが割り振られることはネットワークが分岐する状況であっても各ブランチのブロックに明確な順序を割り振ることが可能である．しかし，ブロックは多数のトランザクションの集合である．トランザクション同士にも因果関係が存在し，単純にブロック 101 はブランチ A，ブロック 102 はブランチ B と言ったブロックを交互に採用するような方法では整合性を保てない．また Proof-of-Work などでブロックを生成した報酬を利用して新たにトランザクションを生成する場合があれば，そのトランザクションが無効になり，それに連なるトランザクションも同時に無効になる．このようにブロックの順序やトランザクションの順序を判断可能であっても，ブロックチェーン全体の整合性を保つことは非常に難しい．このことから現実的にグローバルクロックを用いることでブランチ分岐問題を根本的に解決することは難しいと考える．

一方で，小規模なブランチの分岐であればシステム全体のスループットを少量であるが向上させることが可能であると考えられる．ブランチの分岐を解決する際に時刻的に先に生成されたブロックを採用することにする．ブロックの生成が早いということはこのブロックの後にブロックが連なることを考えると，単位時間あたりのブロック生成数は先に生成されたブロックのブランチの方が多いと言える．ブロックの平均生成時間は一定であるように調整されているため，少しでも生成時間が短かったブランチを採用する方が，結果的にシステム全体の単位時間あたりのブロック生成数の値は向上するためである．例えば，ブロック 100 の次に連なるブロック 101 が 2 種類存在した時の採用するブランチを考える．ブロック 100 の生成時刻が Unix 時間で 1000 であり，ブロックの平均生成時間が 10 分であったとする．2 種類のブロック 101 の時刻は Unix 時間で 1100 と 1500 であったとする．この時，次のブロック 102 の生成が完了する予想時刻はそれぞれ 1700 と 2100 になる．その場合，2 ブロック生成にそれぞれ 700, 1100 要したことになる．1 つのブロックに含むことができるトランザクションの数を 100 とした場合，単位時間あたりのトランザクション

のスループットは　200/700 = 0.286 tx / s, 200 / 1100 = 0.182 tx /s となり，約 0.1 tx / s の差が出る．ブロック 200 完了時のスループットの差は 100 * 100 / (1100 – 1000 + 600 * 100) = 0.166 tx / s, 100 * 100 / (1500 + 600 * 100) – 1000 = 0.165 tx / s となり約 0.001 tx / s となる．このように微々たる差ではあるがシステムのスループットを向上させることが可能である．一般化すると，ブロックに含まれるトランザクション数の平均値が s, ブロック生成時間の平均が m, 採用すべきブロック候補のブロック生成時間がそれぞれt_1，t_2 ($t_1 < t_2$)とする時，n ブロック先のスループット差は

$$\frac{sn}{t_1 + m(n-1)} - \frac{sn}{t_2 + m(n-1)}$$

となる．Bitcoin の実例の値に当てはめる．Blockchain.info[19]によれば，s≒2000, m≒600 となる．この時，$t_1 = 0$, $t_2 = 600$ であったとき,6 ブロック先では約 0.667tx/s のスループットが発生する．Bitcoin トランザクションスループットは最大で 7tx/s と言われており，s=2000, m=600 の場合でのスループット値は 3.33tx/s である．微々たる差ではあるが，Bitcoin がグローバルクロックを採用することでスループット値の向上に貢献が可能であると言える．

他にもブランチ採用の際に時刻を利用することでより信頼性の高いブランチを採用可能である．生成されたハッシュ値の確率的な難易度とそのハッシュ値を生成するまでに要した時間を計算することで，より難易度の高いマイニングに成功したブランチを選択可能と言える．

5.3.3.4 グローバルクロックを用いたブランチ分岐解決の評価

5.3.3.4 で述べたスループットの向上について簡易的に評価を行う．第 4 章のブロックチェーン（物理グローバルクロックバージョン）を利用し，マイニングが早く完了したブランチの単位時間あたりのブロック生成数を求める．本評価ではバージョンを 2 種類用意する．片方は，あらかじめ取得した正解値を初期値に入力し即座に 1 回目のマイニングを完了させる．一方もう片方のバージョンは通常通りマイニングを行う．2 回目以降は同様に両バージョン通常通りのマイニングを行う．本評価はまず平均ブロック生成時間を取得した上で 2 バージョンの比較を行う．

まず，マイニングにかかる平均時間を取得する．本評価では，ハッシュ値の先頭に 0 が連続して 5 つ並ぶハッシュを正解とする（1/1048576 の確率）．この設定で得られた平均マイニング完了時間は 1.295 秒であった．この設定で物理グローバルクロックありと無しの 2 バージョンで 10 ブロック採掘する時間を計測した．結果を以下の表 5.4 に示す．

表 5.4　グローバルクロックを用いたブランチ採択方法による採掘時間

	10 ブロック採掘にかかった時間（s）	単位時間当たりのブロック生成数（block / s）
初期値あり	18.56	0.539
初期値なし	21.16	0.473

第6章　考察

第 5 章で得られた結果を元に考察を行う.

6.1 クロック概念を導入する意義

図 5.1, 図 5.2 より，クロックを実装していないバージョンの方が物理グローバルクロック，論理グローバルクロックの実装に比べて高いスループット値を出した. 論理グローバルクロックに関して，クロック同期を実装している場合はメッセージの送受信に対して計算処理を行うため，この処理がオーバーヘッドとなりスループット値が低下していると考えられる. また, 物理グローバルクロックは本実装においてシステムクロックの値を取得しているため，その処理がオーバヘッドになったと考えられる.

このことから, 確かに順序整合性を保証することが可能であっても，システム全体のスループットは低下する結果となった. 例えば物理グローバルクロック,論理グローバルクロックとクロック無しのスループットは約 0.05 tx/s の差が発生した. そのため，クロックの概念がトランザクションのスケーラビリティに大きく貢献するとは言えない結果となった. ただし，順序整合性を保証すること自体はブロックチェーンの一貫性をより高めることになる. このスループットの差を許容すべきかのトレードオフの関係とも言える.

6.2 各状況に適したクロックの種類

次に，クロック同期の種類による差異を考察する. 評価結果からクロック同期を取り入れることでブロックチェーンそのもののスループットは処理のオーバーヘッドで低下すると言える. しかし，評価を行う際に判明したのが，スケールアップに応じて増加するデータ量であった. 論理グローバルクロックで用いたベクタークロックや CBCAST は各ノードのカウンタを保持している. つまり，ネットワークに参加するユーザー数に比例して一回のメッセージあたりのデータ量は大きくなる. このことから，ブロックチェーンのスケールアップに対応しきれないため，論理グローバルクロックを活用するは現実的に困難と言える. ただ, 逆に言えば小規模なネットワークやプライベートなネットワークでブロックチェーンを運用する場合, データ量の問題が与える影響は少なくなるため，論理グローバルクロックは適していると考えられる. ベクタークロックと CBCAST の実装を比較する. ベクタークロックと CBCAST との実装面での差異は，主に CBCAST がメッセージの受信可否を判定して拒否する場合はキューにメッセージを移動させ，再び受信可能になるまで待機することにある. 第 5 章の結果から分かるように，CBCAST は全般的にベクタークロックに比べメッセージスループット値などで大きく劣る結果となった. このことからも, トランザクション受信時に順序整合性の処理を実行させることは現実的に得策でないと考える. 特に実際の Bitcoin ネットワークなどを考えた場合，大量のノードからブロードキャストされたデータが届くため，順序整合性によるオーバーヘッドは莫大なものとなると考える.

物理グローバルクロックの場合を考える．評価結果から得られた通り，物理グローバルクロックはクロックがない場合の実装よりスループットが低い結果となった．このことから，物理グローバルクロックを実装する場合，特に時刻取得処理のオーバーヘッドが非常に重要な要素であることが得られた．ブロックチェーンが動作する処理の中で頻繁に時刻の取得や比較が行われるため，時刻取得処理の遅れはシステム全体の遅れにつながる．もし原子時計をコンピュータに搭載する場合，プログラムが時刻取得を要求してから，プログラムが利用可能なデータ形式になるまでの変換を可能な限り短くしなければならない．もしくは，ハードウェアレベルで時計をサポートし，例えば CPU レジスタ内に時刻データを保持し，常に原子時計がその値を書き換えると言った構成が必要であると考える．次に物理グローバルクロックの必要性について考える．物理グローバルクロックは実質的に信頼できるタイムスタンプとして扱われたが，タイムスタンプデータのみでは完全な順序整合性は保証できない．ただし，ブロック生成時のトランザクションが承認されるタイミングでトランザクションを時刻に応じて整列させることで，より不正なトランザクションの検証処理の高速化が狙えると考える．他にも二重支払いなど意図的に既に送られたトランザクションを打ち消す行為に対しても，時系列順で先に生成された方を優先することで防止することが可能である．

論理グローバルクロックと物理グローバルクロックを比較する．先ほど述べたように論理グローバルクロックはスケールアップするにつれ増加するデータ量の問題があると指摘した．しかし物理グローバルクロックはこの現象は発生しないと言える．なぜならトランザクションには論理グローバルクロックのように，他のノードの情報を乗せることがないためである．物理グローバルクロックとクロック無し実装の違いはトランザクションにタイムスタンプ情報が存在するかである．そのため両者を比較してもデータ量に大きな差異はない．ユーザー数がスケールアップしても 1 トランザクションあたりのデータ量は増加することがないということは，クロック情報に関してはユーザー数のスケールを大きく受けないと言えスケーラブルであると言える．ただし，評価結果から得られる通り，単純に大量のノードから情報を受け取ったことによる処理のオーバーヘッドが比例することになり，単純に物理グローバルクロックを導入しただけでブロックチェーンそのものがスケーラブルになるという結果は得られなかった．

まとめると次のようになる．クロック同期技術はトランザクションの順序保証を可能にするが，同期処理のオーバーヘッドが発生することからブロックチェーンの性能やスケーラビリティを向上させるわけではないと言う結果が得られた．特に，論理グローバルクロックはユーザー数に比例したデータ量の問題が存在するため，実際にグローバルクロックを適用する際はやはり単独で正確な時刻を取得可能な物理グローバルクロックの方が適していると言える．また，実際に運用することを考えた場合，時刻取得のオーバーヘッドによる性能低下があり得るため，順序保証とスループットはトレードオフの関係であることがわかった．

6.3 グローバルクロックを用いたブロックチェーンのブランチ分岐解決

表 5.4 から確かに初期値が与えられた, つまりマイニングが高速に完了したブランチの方が結果的に単位時間あたりのブロック生成数は高い数値を出すことが分かった．このことからも，ブランチ選択をタイムスタンプ順で早い方を選択する方法は，結果的に単位時間あたりのブロック生成数を向上させ，スケーラビリティの向上に貢献できると言える．

第7章　将来展望

7.1 クロック値の耐改ざん性と一貫性について

本研究では，順序整合性の保証においてクロック値の改ざんが行われた場合を考慮していなかった．論理グローバルクロックおよび物理グローバルクロックの値はそれぞれ信頼できるものとして同期処理を実装したが，これらの値が変更された場合システムの破綻を招く可能性があった．そこで本研究の成果を踏まえて，よりセキュリティを考慮した同期技術を考える必要がある．または，これらの同期処理をハードウェアレベルで安全な場所で実行することである．例えば，CPU の TrustZone や TEE と言った環境下であれば，クロック値の改ざんが行われにくいと考えられる．このようなハードウェアレベルなセキュア領域でのブロックチェーンの実行および同期処理についてより一層議論をする必要がある．

7.2 正確なタイムスタンプを取得することの意義

本研究では物理グローバルクロックを取得可能である前提でのシミュレーションを行った．ここで利用されていたクロックの値はシステムクロックを利用したものであった．NTP などのプロトコルは現在広く用いられていることから，システムクロックはある程度信頼できる時刻を指していると言える．すると実際に物理グローバルクロックとシステムクロックの差異は非常に短い時間の誤差であると考えられる．その場合，ミリ秒やナノ秒ほどの短い単位で正確な物理グローバルクロックの精度は本当に必要なのか，どの程度の誤差ならシステムクロックで代用可能かを議論する必要がある．例えば，6.3 で述べたタイムスタンプを用いたブランチ選択は現在の Bitcoin の実装でも実現可能な手法である．現在の Bitcoin のソースコードによればクロック値は自身と他のノードのシステムクロックの平均値を利用している．この方法によって，ある程度クロックの値は信頼可能であり，Bitcoin の平均マイニング時間が 10 分であることから 0 秒未満の厳密なタイムスタンプが必要とされないケースが多いと考えられる．もしグローバルクロックで無くともある程度信頼できるタイムスタンプで導入可能なことが分かれば，本研究の実現可能性をより高めることにつながる．

また 6.2 で述べたように，実際にタイムスタンプを取得する際のオーバーヘッドは重要な問題である．オーバーヘッドを無くすためにハードウェアレベルでクロックをサポートする構成，もしくは遅延を予測したタイムスタンプ取得方法についても議論や研究を行う必要がある．

7.3 ブロックチェーンの種類

本研究における評価は，簡易的なブロックチェーンを自ら作成した評価を行った．しかし，現在の暗号通貨やオープンソースのブロックチェーンプロジェクトを見た場合，明確なブロックチェーンの定義は確定しておらず，その種類も多岐に渡る．例えば Bitcoin や

Ethereum[1]など暗号通貨の基盤技術としてのブロックチェーンや Hyper Ledger[2]などプライベート環境でのブロックチェーンを利用するプロジェクトなどが存在する．他にも DAG の適用[16]のようにブロックチェーンを応用した技術も存在する．グローバルクロックを適用したブロックチェーンを実現することを考えた場合，より実世界で利用されているブロックチェーンで同じようにグローバルクロックが存在する場合の合意形成を考えることは，重要である．現実的に運用することを想定し，各ブロックチェーンの特徴と照らし合わせて評価を行い，そこから得られた結果を議論することは，本研究の実現可能性をより高めることに繋がると考える．

1 https://www.ethereum.org
2 https://hyperledger.org

第8章　結論

本研究はブロックチェーンの合意形成に時間同期技術を取り入れることで性能向上や問題解決が可能かを調査した．特にブロックチェーンのスケーラビリティ問題に対して原子時計等を用いたグローバルクロックの概念を取り入れる手法で解決に貢献可能かを中心に議論を行った．本研究では，グローバルクロックを実現する方法として論理グローバルクロック，物理グローバルクロックを定義し，それぞれを比較することでより適した時間同期についても議論した．結果としては，論理グローバルクロックはユーザー数のスケールアップに応じてデータ量が増大する理由から，現実的に適していないことが分かった．また，物理グローバルクロックは時刻取得時のオーバーヘッドなどの理由で，大きくスループット値などを向上する結果は得られなかった．このことから，ブロックチェーンのトランザクションにグローバルクロックを適用することはスケーラビリティの解決には繋がらないことが分かった．ただ，その一方でグローバルクロックによってトランザクションの順序保証は可能であるため，性能を犠牲にしてより高い一貫性を保証するかどうかのトレードオフの関係であると言える．一方で，ブロックチェーンのブランチの選択を行う場合，タイムスタンプを参考に早くマイニングに成功したブランチを取り入れる手法は，ネットワーク全体の単位時間あたりのブロック生成数を増やすことになり，スループットを向上できる可能性を示した．以上のことから，ブロックチェーンにグローバルクロックの概念を適用することはトランザクションにおいては，順序整合性を実現する代わりにシステム性能とのトレードオフの関係であるためスケーラビリティの向上には適さないこと，ブロックにおいては，ブロック選択手法に時間概念を取り入れることで全体的なブロック生成時間を早めることが可能という点でスケーラビリティの向上に貢献可能であると結論付ける．

謝辞

本研究を行うにあたり，適切な指導をいただきました中島達夫教授に深く感謝の意を申し上げます．また，ともに勉学および研究に励んできた中島研究室の皆様に感謝いたします．最後に私生活を支えていただきました家族に深く感謝の意を申し上げます．

参考文献

[1] TANENBAUM, Andrew S.; VAN STEEN, Maarten. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[2] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21*(7), 558-565. (1978).

[3] Yamaguchi, A., Fujieda, M., Kumagai, M., Hachisu, H., Nagano, S., Li, Y., ... & Katori, H. Direct comparison of distant optical lattice clocks at the 10-16 uncertainty. Applied physics express, 4(8), 082203. (2011).

[4] Knappe, S., Schwindt, P. D. D., Shah, V., Hollberg, L., Kitching, J., Liew, L., & Moreland, J. A chip-scale atomic clock based on 87 Rb with improved frequency stability. Optics express, 13(4), 1249-1253. (2005).

[5] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., ... & Hsieh, W. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, *31*(3), 8. (2013).

[6] Amazon Time Sync Service のご紹介, https://aws.amazon.com/jp/about-aws/whats-new/2017/11/introducing-the-amazon-time-sync-service/ (Retrieved January 2018).

[7] Sompolinsky, Y., & Zohar, A. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. IACR Cryptology ePrint Archive, 2013(881). (2013).

[8] Szabo, N., Formalizing and Securing Relationships on Public Networks, First Monday, Vol. 2, No. 9. (2007)

[9] Swan, M. *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc. (2015).

[10] Visa Inc. "Visa Inc. at a Glance" https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf (2015). (Retrieved January 2018).

[11] Bitcoin Cash, https://www.bitcoincash.org (Retrieved January 2018)

[12] Poon, J., & Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments. Technical Report (draft). (2015). (Retrieved January 2018).

[13] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timon, J., Wuille, P., "Enabling blockchain innovations with pegged sidechains", https://blockstream.com/sidechains.pdf, (2014) (Retrieved January 2018).

[14] Eyal, I., Gencer, A. E., Sirer, E. G., & Van Renesse, R. Bitcoin-NG: A Scalable Blockchain Protocol. In *NSDI*(pp. 45-59). (2016, March).

[15] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. "Algorand: Scaling byzantine agreements for cryptocurrencies." In *Proceedings of the 26th Symposium on Operating Systems Principles* (pp. 51-68). ACM. (2017, October).

[16] Serguei P., “The Tangle” https://iota.org/IOTA_Whitepaper.pdf (2017) (Retrieved January 2018)
[17] Daniel, F ., “Learn Blockchains by Building One”. https://hackernoon.com/learn-blockchains-by-building-one-117428612f46 (2017) (Retrieved January 2018)
[18] Nakamoto, S., “*Bitcoin: A peer-to-peer electronic cash system”,* https://bitcoin.org/bitcoin.pdf. (2008) (Retrieved January 2018)
[19] BLOCKCHAIN LUXEMBOURG S.A. “ビットコインチャート” https://blockchain.info/charts (Retrieved January 2018)
[20] Heilman, E., Kendler, A., Zohar, A., & Goldberg, S. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *USENIX Security Symposium* (pp. 129-144). (2015, August).
[21] Lamport, L., Shostak, R., & Pease, M. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *4*(3), 382-401. (1982).
[22] Lamport, L. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, *16*(2), 133-169. (1998).
[23] Ongaro, D., & Ousterhout, J. K. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference* (pp. 305-319). (2014, June).
[24] Castro, M., & Liskov, B. Practical Byzantine fault tolerance. In *OSDI* (Vol. 99, pp. 173-186). (1999, February).
[25] 国立研究開発法人情報通信研究機構, “原子時計をスマートフォンに搭載できるくらいの超小型システムへ”, https://www.nict.go.jp/press/2018/01/23-1.html (2018) (Retrieved January 2018)
[26] Birman, K., & Cooper, R. The ISIS project: Real experience with a fault tolerant programming system. *ACM SIGOPS Operating Systems Review*, *25*(2), 103-107. (1991).
[27] “Irreversible Transactions”, https://en.bitcoin.it/wiki/Irreversible_Transactions, (2017) (Retrieved January 2018)