修士論文

ハッシュグラフ技術を用いた
スケーラブルな価値交換システムの
構築とその性能評価

令和 **2** 年 **1** 月 **30** 日 提出

指導教員　　相田 仁　教授

工学系研究科電気系工学専攻
37-186501　三島 潤平

本論文は東京大学大学院工学系研究科に修士号授与の要件として提出した修士論文である.

内容梗概

ハッシュグラフは高速で公平な取引承認が可能な DAG 型の分散台帳技術であると言われているが，合意形成に参加するノード数が増加すると取引承認時間が長くなってしまう．そこで本研究では，小規模なハッシュグラフを複数組み合わせて使う分割ハッシュグラフを提案する．実際の取引を模したシミュレーションを行い，分割ハッシュグラフの取引遅延性能の評価を行った．その結果，分割ハッシュグラフの取引承認時間が既存のハッシュグラフよりも大幅に短くなり，提案手法の有効性が検証出来た．最後に，分割ハッシュグラフの実用化に向けた検討を行い，今後の研究余地について方向性を示した．

目次

図目次

アルゴリズムの一覧

第 1 章

序論

1.1 本研究の背景

金融や流通などの分野で，ブロックチェーンをはじめとする分散台帳技術（DLT）を活用した価値記録・交換システムの導入が検討されている．DLT を用いた価値記録・交換システムは，参加者の信頼性と規模に合わせて 3 種類に分類することができ，参加者の信頼性が高く，規模が小さい順にプライベート型・コンソーシアム型・パブリック型と呼ばれる．そして，いずれのシステムもシステムを形成するノード数や取引量が増えると，取引の承認に要する時間が膨大になってしまうスケーラビリティの問題を抱えている．

取引承認までの時間に関するスケーラビリティの問題は，規模が大きく改ざん耐性の高いパブリック型のブロックチェーンシステムに多く見られる．例えば，ビットコイン [2] の秒間取引性能は 7 TPS [*1] と低く [3]，イーサリアム [4] も 15 TPS 程度にとどまる [5]．プライベート型・コンソーシアム型のシステムでは，参加者の信頼性を活かして取引承認処理の速度を向上させる試みが多く行われているが，パブリック型を視野に入れた試みは少ない [6]．

ハッシュグラフ [1] は，改ざん耐性を備える DLT の中でもマシンパワーに寄らない公平な合意形成と単位時間あたりの取引処理性能に優れた，有向グラフ（DAG）型の DLT として知られている．しかし，ハッシュグラフの合意形成に参加するノード数が増加すると，取引の発行から処理順序決定までの時間が長くなってしまうスケーラビリティ問題が存在し，取引完了までの待ち時間の面で他の DLT に対する優位性を失ってしまう．ハッシュグラフのスケーラビリティ問題に関する研究は少なく，ハッシュグラフのスケーラビリティ問題に対応することが出来れば，ビットコインやイーサリアムと同等の規模を持ち，より短い遅延時間とより秒間取引可能数の多いパブリック型の DLT を実現できる可能性がある．

1.2 本研究の目的

本研究の目的は，ハッシュグラフの合意形成参加ノード数が増えても取引の遂行までの時間に対するスケーラビリティを確保できるようにすることである．

そこで本研究では分割ハッシュグラフを提案する．分割ハッシュグラフではノードをグループ分けし，グループ内とグループ間に関する小規模なハッシュグラフでより高速な合意形成を行う．ノードの属するグループはシャードと呼ばれ，シャードを跨ぐノード間での取引（シャード間取引）については数段階に渡る承認プロセスを経て取引を確定させるプロトコルを考案した．

本研究のもう一つの目的は，既存のハッシュグラフと分割ハッシュグラフについてそれぞれ性能評価を行い，取引確定までの遅延時間に関する分割ハッシュグラフの優位性を示すことである．

[*1] TPS: Transactions Per Second の略

1.3 本研究の貢献

本研究による貢献は以下のとおりである．

- 既存のハッシュグラフの分析と性能のモデル化
- 分割ハッシュグラフの提案
- 分割ハッシュグラフに付随するシャード間取引プロトコルの提案
- 既存のハッシュグラフと分割ハッシュグラフの性能比較

1.4 本論文の構成

本論文は，7 章からなる．第 1 章から第 3 章までは本研究の提案に至る導入部分である．第 1 章にあたる本章では，本研究の背景や目的，貢献について明らかにした．第 2 章ではハッシュグラフに関する解説を行い，続く第 3 章では分割ハッシュグラフの背景にある関連研究について説明する．

後半部の第 4 章から第 7 章では，分割ハッシュグラフの提案と評価を行っている．第 4 章では分割ハッシュグラフの概要とシャード間取引プロトコルの説明をしている．第 5 章ではシミュレーションによる分割ハッシュグラフの性能評価について報告する．シミュレーションの際に既存のハッシュグラフについても性能評価を行い，詳細な性能評価も行っている．第 6 章では分割ハッシュグラフの実用化に必要な機能を挙げ，実現方法について検討を行う．最後に第 7 章で本研究のまとめと今後の展望について論ずる．

第 2 章

ハッシュグラフについて

ハッシュグラフ [1] は，2016 年に Leemon Baird 博士によって発明された DAG 型の DLT 技術である．

ここで言う DLT によって構成されるデータベースの要件は，複製されたステートマシンが存在し，システム全体としてビザンチン耐性を備えることである．ステートマシンは，各ユーザの残高や状態の増減や変化を管理するものであり，システムを構成するそれぞれのノードのメモリやハードディスク上で記録・更新される．各ノードのステートマシンに記録されている内容は非同期的にではあるが，全く同様に遷移することが求められる．また，システムがビザンチン耐性，あるいはビザンチン性を備えるとは，システム全体の合意形成においてビザンチン将軍問題 [7] が発生しないことが証明されていることを意味する．特に DLT におけるビザンチン耐性とは，合意形成に参加するノードに悪意のある攻撃者，あるいは非正常な動作をするノードがいたとして，メッセージの遅延や削除をしたとしても，全体の 1/3 以上のノードがプロトコル通り正常に動作する限り，システム全体としての合意が崩れないことを意味する．

ハッシュグラフは非同期的に動作するノード同士で形成されるシステムでビザンチン耐性を備えた合意形成が可能なシステム，すなわち非同期的ビザンチン耐性を備えた DLT である．

ハッシュグラフは 2018 年にはヘデラ・ハッシュグラフ [8] として，コンソーシアム型の DLT プラットフォームとして商用化もされている．

以下では，[1] の説明を借りて，ハッシュグラフと他の DLT との差異に触れつつ，ハッシュグラフのコンセンサスアルゴリズムについて説明を行う．

2.1 他の DLT との差異

Paxos [9] や Raft [10] は，故障耐性を備えた DLT システムである．これらのシステムは，2 相コミットに代表される故障無しを前提とした DLT システムより高い堅牢度を持つ．これらのシステムではあるノードがリーダーの役割を果たし，特定のプロトコルで他のノードと通信をすることでステートマシンの更新状態を一意に保とうとする．この場合，リーダーのノードに対する DoS 攻撃に対しては脆弱 [11] でビザンチン耐性を備えているとはいえない．ハッシュグラフは特定のリーダーを持たず，全てのノードが平等に情報を交換し合う. そのため，悪意のあるノードから DoS 攻撃を全てのノードに対して行うことは困難であり，DoS 攻撃に対する対する耐性も備えていると言える．

他の DLT システムであるビットコインは proof-of-work（PoW）コンセンサスアルゴリズムを用いたブロックチェーンシステムである．PoW 型のブロックチェーンでは，取引情報の入った後続ブロックの生成を全世界のノードで同時に行う．この場合，各ノード間でブロックの連なりに関する情報の伝わり方に差異があり，後続ブロックの決め方に分岐が生ずる．ブロックチェーンでは，後続するブロックを一意に決定するため分岐したブロックの長い方を採用し，短いブロックを破棄するため，分岐が生じた後も何ブロックか PoW によるブロックの追加を待つ．PoW 型のブロックチェーンでは不正防止のため，ブロックの追加に時間がか

かる仕組みであり，確率的にほぼ間違いないと思われる数のブロックが追加されるまでブロックの採用を待つため，処理性能が低く，不採用となったブロックの PoW 作業に費やしたマシンパワーは無駄になる．ハッシュグラフは，ブロックチェーンと同様に情報が枝分かれして伝わる構造を持つが，枝分かれの選別は行わず枝分かれをそのまま活用する．そのため，ハッシュグラフは承認作業に費やすマシンパワーが 100 % 効率的に運用されていると言える．

2.2 ハッシュグラフの主要コンセプト

ハッシュグラフに登場する主要コンセプトは以下のように列挙できる．

トランザクション 各ノードは署名付きのトランザクション（正確にはイベントに内包されたトランザクション）を生成することが出来る．全てのノードはそのコピーを受け取り，システム全体としてビザンチン合意に到達し，トランザクションの処理順を一意に決定することが出来る．

公平性 システムに属するノードの一部がトランザクションの処理順を操作しようとしても，少数のノードでは困難となっている．

ゴシップ ゴシップとは噂話のことで，各ノードは他のノードをランダムに選択し，過去に知った全ての情報について教えてもらうことが出来る．この操作により，情報は拡散する．

ハッシュグラフ 誰が誰にどんな順序で情報を伝えたかを DAG の形で記録するデータ構造

ゴシップ・アバウト・ゴシップ ノード同士のゴシップは結果として，ゴシップについての記録であるハッシュグラフ自体の同期をとることに相当し，ゴシップについてのゴシップをしていると言える．ノード同士の通信時には，ハッシュグラフ構造体の全てを伝えるのではなく，相手が知らない可能性があるトランザクションに絞って送信するため，ネットワークのオーバーヘッドは大きくない．

仮想投票 従来のビザンチン合意プロトコルでは，ある事実が送信されたかどうかについての合意形成をするためには，全てのノードが実際に投票をする必要があった．ハッシュグラフでは，ハッシュグラフ構造体自体に合意形成に必要な情報が含まれ同期されているため，合意形成に必要な投票を自ノード内で仮想的に再現することが出来，投票のためにネットワーク帯域を使う必要がない．

有名なウィットネス トランザクション情報の入ったイベントの処理順序を決定するためにイベントの並べ替えをする必要がある．n 個のイベントを計算量 $O(n \log n)$ で並び替えるよりも効率的にするために，有名なウィットネスを定義する．ハッシュグラフの頂点情報に当たるイベントのうちいくつかをウィットネスと定義し，多くのノードに知られているウィットネスを有名なウィットネスであるとする．ウィットネスが有名であるかどうかを処理順決定の基準とすることで，処理順決定の計算量を減らすことが出来る．

真に見ることができる ハッシュグラフの頂点 x, y について x が y を真に見ることが出来る，ということを定義する．その定義は，x から y への経路を考えた時に，十分な数のノードを経由することが出来ることである．この概念により，仮想投票の結果の一意性とビザンチン合意が 100 % 成立することについての証明が可能となる．

2.3 ハッシュグラフのコンセンサスアルゴリズム

ハッシュグラフシステム全体の動作は，情報の伝播を担うゴシップ・アバウト・ゴシップと，情報が十分に行き渡ったことを判定し，イベント並びにトランザクションの処理順序を決定する仮想投票の 2 つのプロセ

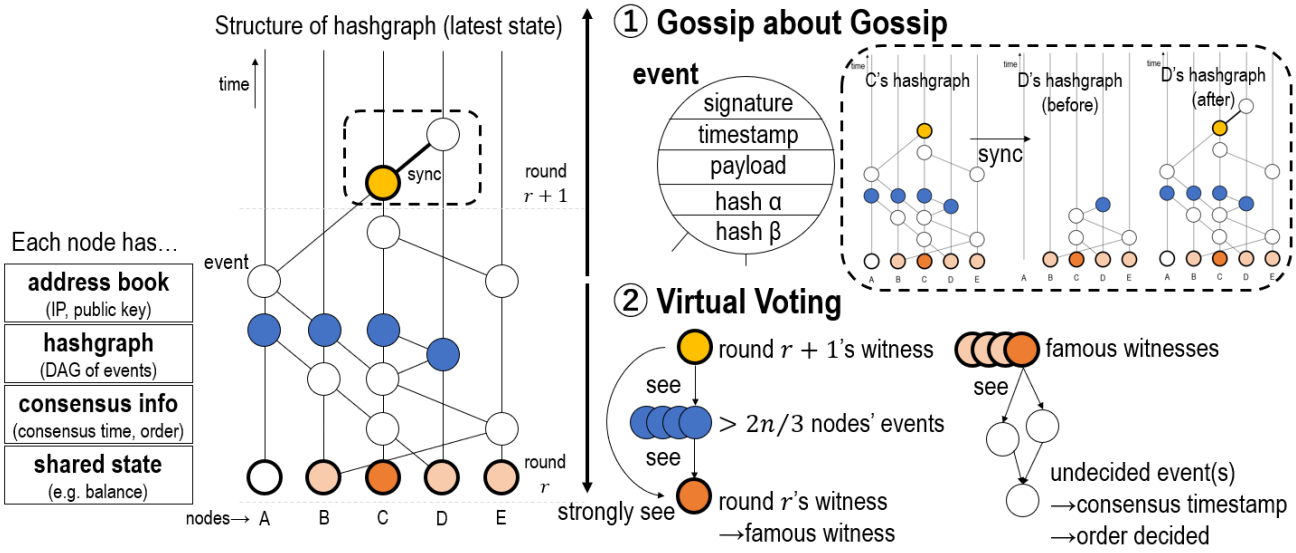


図 1　ハッシュグラフの概要．各ノードは相手の IP アドレスや公開鍵など，通信と暗号化・復号化・署名・署名確認に必要な情報と，情報共有の履歴であるハッシュグラフ，残高や状態の変化を管理するステートマシン，コンセンサスが取れて処理順序が確定したイベントがその順番に入ったコンセンサス情報の配列などをそれぞれ自ノード内のメモリやハードディスク上の情報として持つ．ハッシュグラフのコンセンサスアルゴリズムでは，非同期ながらも各ノードがお互いのハッシュグラフ情報を同期しようと通信を行うことにより，図の左側にあるような「真の」情報共有履歴を表すハッシュグラフのコピーを持とうとする．非同期的にではあるが，全てのノードで同じハッシュグラフのコピーが形成されるので，イベントの処理順序についても一意に確定し，ステートマシンの状態遷移も一意に確定する．情報の伝播はゴシップ・アバウト・ゴシップによる情報の同期要求と新たな頂点であるイベントの追加，処理順の決定は仮想投票による情報の拡散度の判定と，有名なウィットネスのタイムスタンプの情報などを利用して行われる．

スに分けて説明することが出来る．

2.3.1　ゴシップ・アバウト・ゴシップ

ハッシュグラフでは，承認プロセスに参加するノードそれぞれが他のノードをランダムに選択し，自分が知らない情報について教えてもらうよう要求する．各ノードがこのような要求を繰り返し行うことで，誰がいつ誰に情報を伝えたかの履歴を知ることが出来る．そして，この方法により，ある 1 つの情報が誰かに伝わると，その情報は指数関数的に広がっていき，やがて全てのノードがその情報について知ることになる．

ゴシッププロトコルによって伝えられた情報の履歴は， 図 1 の左側に示すように，時間方向に進展していく有向グラフ（Directed Acyclic Graph; DAG）の形で表すことが出来る．例えば，ノード D からノード C にハッシュグラフの同期要求があると，C は D が持っていないであろう情報を伝え，D は自身のハッシュグラフ情報に未知の情報を加えたのち，D の列に新たな頂点であるイベントを追加する．このとき，新たなイベントは，自身の列の先頭のイベントと同期要求を行ったノードの列の先頭のイベントそれぞれに向けた辺を持つように作成する．

前述したイベントや DAG のデータ構造について説明する．DAG の頂点にあたるイベントは以下に示す複数の情報を持つ．

タイムスタンプ　イベントを作成した時間のタイムスタンプ．

ペイロード　「A から B に残高を 100 移動する」などのトランザクション命令のリスト．キューに追加したいトランザクションがある場合，イベントの作成時にここに書き込む．追加したいトランザクションが無い場合は空となる．

対自ハッシュ値　自分の持つハッシュグラフにおいてその時点で自分の列の先頭にあるイベントのハッシュ値

対他ハッシュ値　同期要求を送ったノードの列の先頭にあるイベントのハッシュ値

署名　タイムスタンプ・ペイロード・対自ハッシュ値・対他ハッシュ値をまとめて自分の秘密鍵で行うデジタル署名

先ほどの，ノード D からノード C にハッシュグラフの同期要求を行った例では，新しいイベントの対自ハッシュ値は，その時点で D の列の先頭にあるイベントのハッシュ値，対他ハッシュ値は，受け取った情報も含めて C の列の先頭にあるイベントのハッシュ値，署名は D による秘密鍵による署名となる．前述の構成によりイベントを記録していくため，各ノードのメモリや記憶領域ではハッシュグラフの DAG 構造は頂点と辺が別々に記録されているようなことはない．イベント情報は，そのイベントのハッシュ値とそのイベントの内容を対応づける連想配列の形で記録され，イベント内容に，親となるイベントのハッシュ値が 0 個（一番最初のイベントの場合）か 2 個記載されているため，親へ親へとイベントを辿っていくことができるようになっている．

ハッシュグラフのゴシッププロトコルでは，お互いのノードがローカルに持つハッシュグラフの情報を補完し合うため，非同期的にではあるが，時間経過により，ノード間でやり取りされた情報の完全な履歴のコピーが各自のハッシュグラフ上で再現されることになる．各自のハッシュグラフ上の履歴情報を利用して，イベントの処理順序を決めるのが，次の小節で説明する仮想投票によるコンセンサスアルゴリズムである．

2.3.2　仮想投票によるコンセンサスアルゴリズム

イベントを承認し，ステートマシンの更新を行うためには，イベントが全てのノードによって知られていることを確認するだけではなく，イベントの処理順序並びにイベントに内包されたトランザクションの処理順序を決定する必要がある（イベントに内包されたトランザクションは通常そのままの順番で処理すれば良いので，イベントの処理順序を決めれば良い）．

多くのリーダーレスでビザンチン耐性を備えたコンセンサスプロトコルでは，各ノードにあるトランザクションを知っているかどうかについて実際に投票を行わせる．その場合，n 人が合意形成するためには，$O(n^2)$ のメッセージを送受信する必要がある．ハッシュグラフのコンセンサスでは，ネットワーク経由での投票プロセスは不要である．ゴシッププロトコルによって各ノードが同じハッシュグラフ情報を持っているとするならば，決定論的方法でイベントの処理順序を定めれば，全てのノードでイベントの処理順序は一意に定まり一致する．当然，ある瞬間においては各ノードは全く同一のハッシュグラフ情報を持つわけではないし，作成されたばかりのイベントがほとんど知られていないということもあり得る．しかし，イベントの親を辿っていくと，昔のイベントについては他のノードのハッシュグラフと認識が一致する部分が現れる．ハッシュグラフでは，イベントの親を辿ることによって得られる手がかりを元にイベントの位置づけを決める仮想投票という方法によってイベントの処理順序を決める．

A が持つハッシュグラフ A と B が持つハッシュグラフ B があるとする．ハッシュグラフ A とハッシュグラフ B には一貫性が成立する．ここでいう一貫性とは，あるイベント x がハッシュグラフ A とハッシュ B の両方に属しているなら，それぞれのハッシュグラフは x の親として全く同一のイベントを持ち，イベント同

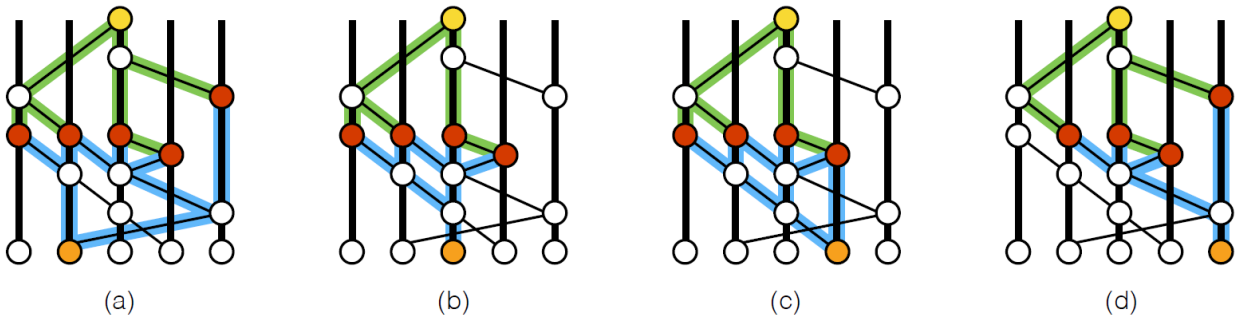


図 2　“真に見る” ことができる状態の図解．[1] より引用．それぞれのハッシュグラフで，一番上の黄色いイベントはそれぞれ下のオレンジ色のイベントを “真に見る” ことができる．ノード数は $n = 5$ であり，$2n/3$ 以上の整数は 4 であるから，“真に見る” ことができるためには黄色からオレンジまでたどり着くルートがのべ 4 ノード分の列を経由できる必要がある．(d) では，右下のオレンジ色のイベントは異なる 4 つのノードの列に乗った赤いイベントの親であり，赤いイベントはそれぞれ一番上の黄色いイベントの親である．そのため，黄色いイベントは右下のオレンジのイベントを “真に見る” ことができる．(a)〜(c) のオレンジ色のイベントについても同様に，黄色いイベントから “真に見る” ことができる．このように黄色いイベントから 4 つの異なるノードの列のオレンジ色のイベントを “真に見る” ことができて，黄色いイベントの生成時ラウンド数が r だった場合，このラウンド数を $r + 1$ に更新することができる（生成時のラウンド数は親のラウンド数を引き継ぐが，黄色いイベントはラウンド数 r の親イベントの $2n/3$ を “真に見る” ことができるので，各ラウンドの最初のイベントであるウィットネスイベントの $2n/3$ 以上を “真に見る” ことができるため）．

士の接続（対自ハッシュ値・対他ハッシュ値）も全く同一であるということである．もし，片方のノードが知らないイベント x があるなら，一貫性を保つように相手のノードにイベント x とそれに連なる情報をゴシッププロトコルで相手と通信を行う際に伝えようとする．

次に，それぞれのノードが自分のハッシュグラフを用いてイベントの処理順を決定するプロセスを説明する．例えばノード A はハッシュグラフ A のあるイベントから他のあるイベントへの投票を行う．この投票は他のノードの列のイベントに関しては，既に受け取っている情報から投票されるか否かを判断して行われ，仮想投票と呼ばれる．

仮想投票の結果に影響を与えようと個々のノードが不正をする可能性があるが，ハッシュグラフの仮想投票では，イベントが他のイベントを “見る” ことができるという状態と “真に見る” ことができるという状態を定義して判定を行うことで不正を防止している．あるイベント w が x を “見る” ことができるとき，x は w から親を辿ることによってたどり着くことができる． 図 2 に “真に見る” ことができるという状態についての図解を示す．ノード数が n のとき，あるイベント w が x を “真に見る” ことができるとは，w が全体の $2n/3$ 以上のノードの列上のイベントを “見る” ことができ，かつそのそれぞれのイベントが x を “見る” ことができる状態を指す． 図 2 の (d) では，右下のオレンジ色のイベントは異なる 4 つのノードの列に乗った赤いイベントの親であり，赤いイベントはそれぞれ一番上の黄色いイベントの親である．そのため，黄色いイベントは右下のオレンジのイベントを “真に見る” ことができる． 図 2 の (a)〜(c) のオレンジ色のイベントについても同様に，黄色いイベントから “真に見る” ことができる．

仮想投票では，イベント w がイベント x を “強く見る” ことができる場合，x から子イベントの w に対して票が投じられる．ハッシュグラフで用いられる一連のコンセンサスアルゴリズムをアルゴリズム 1–4 に示す．

アルゴリズム **1** コンセンサスアルゴリズムの基本ループ [1]

```
1:  loop
2:      (run two loops below in parallel)
3:      loop
4:          sync all known events to a random member
5:      end loop
6:      loop
7:          receive a sync
8:          create a new event
9:          call divideRounds
10:         call decideFame
11:         call findOrder
12:     end loop
13: end loop
```

アルゴリズム **2** divideRounds の処理 [1]

```
1:  procedure divideRounds
2:
3:  for each event x do
4:      r ← max round of parents of x (or 1 if none exist)
5:      if if x can strongly see more than 2n/3 round r witnesses then
6:          x.round ← r + 1
7:      else
8:          x.round ← r
9:      end if
10:     x.witness ← (x has no self parent) or (x.round > x.selfParent.round)
11: end for
```

コンセンサスアルゴリズムの基本ループである アルゴリズム 1 では，あるノード A が別のノード B をランダムに選択し，A が知っている事を B に伝え，B は A から情報を受け取ったことを意味するイベントを自分の列の先頭に加えることを意味する（実際には B が A にハッシュグラフの同期要求を送り，A が B に知っている事を伝えるという流れになる）．ハッシュグラフの同期の後，アルゴリズム 2–4 に示す 3 つの手順をローカルマシン上で実行する．つまり，ネットワークリソースを消費しなくて良い．各手順の for ループでは，各イベントをトポロジカルオーダー，つまり子よりも先に親が並ぶようにした順序で訪れる．一番はじめの for ループでは，ハッシュグラフ上に親のない初期イベント x のみが存在し，$x.round = 1$ と $x.witness = true$ が定義される．また，定数として，合意形成に参加するノード数 n と 2 以上の比較的小さな c を真に見ることが出来る，ということを定義する．

各イベントは アルゴリズム 2 に示すように，親のイベント情報からラウンド数を定義される．基本的に親のイベントのラウンド数が r ならば，そのイベントのラウンド数も r となるが， 図 2 のように，ラウンド数 r のイベントを $2n/3$ 以上 “真に見る” 事ができる場合，自身のラウンド数を $r + 1$ に進める．各ラウンド番号

アルゴリズム **3** decideFame の処理 [1]

```
1:  procedure decideFame
2:
3:  for each event x in order from earlier rounds to later do
4:    x.famous ← UNDECIDED
5:    for each event y in order from earlier rounds to later do
6:      if x.witness and y.witness and y.round > x.round then
7:        d ← y.round − x.round
8:        s ← the set of witness events in round y.round − 1 that y can strongly see
9:        v ← majority vote in s (is truefor a tie)
10:       t ← number of events in s with a vote of v
11:       if d = 1 then {first round of the election}
12:         y.vote ← can y see x?
13:       else
14:         if d mod c > 0 then {this is a normal round}
15:           if t > 2n/3 then {if supermajority, then decide}
16:             x.famous ← v
17:             y.vote ← v
18:             break out of the y loop
19:           else {else, just vote}
20:             y.vote ← v
21:           end if
22:         else {this is a coin round}
23:           if t > 2n/3 then {if supermajority, then vote}
24:             y.vote ← v
25:           else {else flip a coin}
26:             y.vote ← middle bit of y.signature
27:           end if
28:         end if
29:       end if
30:     end if
31:   end for
32: end for
```

のイベントのうち，各ノードの列の一番最初のイベントをウィットネスと呼ぶ．以降の仮想投票で票の授受が行われるのはウィットネスイベント同士のみである．

アルゴリズム 3 では，各ウィットネスに対して，“有名” であるかどうかの判定を行う．ウィットネスは，後のラウンドのウィットネスの多くから “真に見る” 事ができる場合に “有名” であると判定される．具体的には，ラウンド $r+1$ のウィットネスがラウンド r のウィットネスを “真に見る” ことができるか投票し，$2n/3$ 以上の票が集まったウィットネスが “有名” なウィットネスとなる．不正が生じても投票が行われるように，

アルゴリズム **4** findOrder の処理 [1]

1: **procedure** findOrder
2: **for each** event x **do**
3: **if** there is a round r such that there is no event y in or before round r that has $y.witness$ = **true** and $y.famous = UNDECIDED$ **and** x is an ancestor of every round r unique famous witness **and** this is not true of any round earlier than r **then**
4: $x.roundReceived \leftarrow r$
5: $s \leftarrow$ set of each event z such that z is a self-ancestor of a round r unique famous witness, and x is an ancestor of z but not of the self-parent of z
6: $x.consensusTimestamp \leftarrow$ median of the timestamps of all the events in s
7: **end if**
8: **end for**
9: **return** all events that have roundReceived not UNDECIDED, sorted by roundReceived, then ties sorted by consensusTimestamp, then by whitened signature

定数 c で定められたラウンド数ごとに，疑似乱数による投票を行い，$2n/3$ 以上の票が集まりうるようにする．これを繰り返すことにより，各ウィットネスはいつか必ず “有名” となる．

アルゴリズム 4 では，各イベントのコンセンサスタイムスタンプの決定，すなわち処理順の決定を行う．まず，イベントの受信ラウンド数を決定する．あるイベント x の受信ラウンド数は，x の子孫にラウンド数 r の “有名な” ウィットネスが全て存在する場合に r と決定される．次に受信時刻を計算する．ラウンド r のある “有名な” ウィットネス y について，y の祖先のうち，x を一番はじめに知ったイベント z のタイムスタンプを t とする．受信時刻，すなわちコンセンサスタイムスタンプは，ラウンド r の各 “有名な” ウィットネスについて，前述の方法で求めた t の中央値となる．最後にイベントの処理順序を確定させる．基本的には受信ラウンド数，コンセンサスタイムスタンプの優先度で並び替えるが，コンセンサスタイムスタンプが一致する場合，イベントの署名の辞書順に並べる．

第 3 章

関連研究

本章では，ハッシュグラフのスケーラビリティ問題の解決にあたり，参考にした研究について紹介する．

3.1 分散台帳技術におけるスケーラビリティ問題への対処例

スケーラビリティの定義 ネットワーク上における価値交換を行うシステムにおけるスケーラビリティとは何かに関してここで定義を行うこととする．ここでは，スケーラビリティを取引のパフォーマンスを表す指標であると考えることにし，具体的なパラメータとして，一定時間当たりに処理される取引量を表す TPS(Transactions Per Second) や取引が発行されてから承認されるまでの遅延を表すレイテンシがあると考えられる．今後は，システムのスケーラビリティを TPS やレイテンシによって測ることとする．

3.1.1 ブロックチェーンのスケーラビリティ問題

ブロックチェーンのスケーラビリティに関して，ビットコインに代表されるパブリックネットワークにおける指標を確認すると，取引の承認速度は 3〜10 TPS 程度である．また，ブロックごとの承認時間はブロックごとの伝搬時間に依存し，平均伝搬時間の 11.6 秒に分岐防止のための確認回数 3〜6 回程度を乗じた時間である 1 分程度となる．ブロックに収めることのできる取引数はブロックサイズの取引のバイトサイズに依存し，現状から逆算した速度が前述の 3〜10 TPS 程度となっている [12]．結果として，大量の取引の要請がある場合，ブロックごとに処理が進むブロックチェーンでは，後の方の取引が承認されるまでに長時間待たされることがあり，スケーラビリティの問題として取り上げられている．

取引システムにおける比較対象として，VISA を見てみると，平均時 2,000 TPS，日々のピークパフォーマンスとして 4,000 TPS，最大達成可能パフォーマンスとして 56,000 TPS という数値があり，平均遅延時間も 7 秒ほどであると言われている [13]．取引システムとして達成すべきスケーラビリティは，例えば選挙システムや自動販売機取引などの一般的用途であれば 1,000〜1,500 TPS，お金の授受に関しては VISA を目安として 5,000 TPS，その他 IoT/M2M 通信を活用するケースにおいて 200,000 TPS 程度を達成すればよいと考えられており，ブロックチェーンのスケーラビリティ問題に対処すべく様々な手法が提案されている．

3.1.2 ブロックチェーンのスケーラビリティ問題への対処例

オフチェーン ブロックチェーン上で行う取引のことをオンチェーンと呼ぶのに対し，ブロックチェーン外で行われる取引のことをオフチェーン取引と呼ぶ．ブロックチェーンのスケーラビリティ問題は，オンチェーン上でのブロック承認速度の遅さに起因するため，チェーン外での取引を可能とし，チェーン外で行われた取引を要約してブロックチェーン上に書き込むという対策が考えられており，これをオフチェーンもしくはレイヤ 2 などと呼ぶことがある．代表例として Lightning Network [14] がある．この技術はブロックチェーン

のトランザクションコストが発生しないため，マイクロペイメントを実現する手段として注目されている一方で，ブロックチェーンへの書き込み時に行われる PoW(Proof of Work) に対するインセンティブを与える機会の損失につながり，システムの維持が結果的に困難になることが懸念されている．

bloXroute　ブロックチェーンでは，ノード全体の 90 % 以上にブロックに関する情報が伝わるのを待つため，伝搬時間の分だけブロックの生成を待つ必要があった．bloXroute [15] は，ブロックの伝搬を高速化するため，ノード間の通信において，従来のようにブロックを受信しきってから次のノードへの通信を開始するのではなく，前のノードからブロックを受信し始めると同時に次のノードへ情報の送信を開始するようなプロトコルを実装することにより，ノード数による伝搬遅延の要因を取り除くことに成功した．この手法による速度の改善は 100〜1,000 倍である．

Zilliqa　Zilliqa [16] は，データベースのシャーディングに着想を得て，ノードをランダムにシャードに割り振ることにより合意形成までの時間を短縮することに成功したシステムである．Zilliqa ではコンセンサスアルゴリズムとして，PoW の代わりに BFT(Byzantine Fault Tolerance) を用いる．また，合意形成の高速化のための更なる工夫としてシュノア署名によるマルチシグニチャーを採用している．また，Etheruem の「Solidity」にあたる、Zilliqa の開発言語「Scilla」も発表されている．Zilliqa のテストネットワークである Red Prawn を使った実証実験では，2,000〜3,000 ノードが参加し，2,000〜3,000 TPS を達成することに成功している．しかしながら，bloXroute にも言えることだが，数千 TPS というトランザクションスピードは高速ではあるものの，IoT/M2M を見据えた大量のトランザクションに対してはまだ改善の余地があると考えられる．

プライベートコンソーシアム　パブリックチェーンでは，不特定多数の参加者を認めて中央集権を回避する設計思想のため認証の仕組みをあえて設けていない分，ブロックの承認プロセスにオーバーヘッドが生じる．そこで，ブロックチェーンの使用を一定の組織内に限定するなど，通信相手に一定の信頼性が置ける条件下でブロックの承認プロセスを向上する手法も検討されている．しかし，このようにして形成されたブロックチェーンコミュニティであるプライベートコンソーシアムは，管理者による汚染を受ける可能性がある他，不特定多数への門戸が開かれていないため，拡張性が低いという問題が懸念される．

3.1.3　ブロックチェーンのスケーラビリティ問題への対処例と本研究との関連性

前述の，ブロックチェーンにおけるスケーラビリティ問題の対処例はいずれもハッシュグラフのスケーラビリティ問題に対処する上で参考になる．

オフチェーンのように取引の要約情報のみをハッシュグラフで承認し記録する方法を取ることでステートマシンの更新回数を減らすことができる．しかし，実質的にはハッシュグラフ構造に占めるトランザクションのデータ量が減るのみで，メインのハッシュグラフにおける承認遅延の根本的解決にはならず，オフチェーン取引をどのように管理するか考える必要もあるため，本研究では扱わないことにした．

bloXroute のように，ネットワーク帯域を最大限活用して情報伝播の効率を上げる手法をハッシュグラフと照らし合わせてみる．ハッシュグラフではいつでも同期の問い合わせが来れば，自身の持つ情報を相手に返すことができる．相手からの通信は重複していても構わず，いつでも通信が来れば，自身の持つ最新のハッシュグラフ情報を相手に伝えることができるため，ネットワーク帯域の利用効率をこれ以上上げることは本研究では扱わなかった．

Zilliqa では，取引の承認プロセスに一定数のノードだけを使用することにして 1 回の承認プロセスあたりの所要時間を短縮している．ハッシュグラフでも，承認プロセスに携わるノード数が多いほど 1 回の承認プロセスあたりの所要時間が長くなることが分かっているので，必要なノード数だけ確保したハッシュグラフで取引の承認プロセスを行えば，取引が発行されてからステートマシンが更新されるまでの時間を短縮する

ことができると考えられる．ただし，複数のハッシュグラフを同時運用する際に，システム全体として残高などの情報の整合性を取るために，新たなプロトコルを考える必要がある．本研究では，シャーディング技術をベースに複数のハッシュグラフを運用し，整合性を保ちつつ取引を承認し実行するプロトコルを考案している．

プライベートコンソーシアムを考慮した開発はヘドラ・ハッシュグラフなどで積極的に行われているが，承認プロセスに参加するノード数の想定が少ない．本研究では，価値交換・記録システムを支える参加者として，承認プロセスに参加すると同時にシステムの正当性を監査したいノードが自由に参加することを想定し，ノード数は 10^4 などのオーダーに到達し得る必要がある．そのため，本研究では，あえてプライベートコンソーシアム型のシステムでしか成立し得ない仮定をできる限り排除して検討することとした．

3.2 ハッシュグラフ技術の改良

JointGraph [17] は，ハッシュグラフのスケーラビリティについてノードの参入容易性と承認時間の短縮の両面から解決を試みた．Jointgraph ではスーパーバイザノードを設定し，スーパーバイザノードが作成したイベントを基準にイベントの処理順序決定や DAG 構造のスナップショット作成を行う．ハッシュグラフに比べて，スーパーバイザノードを信頼することでイベントの処理順序決定に必要なコミュニケーション数を減らすことができ，ハッシュグラフに比べてスループットや取引承認の遅延を約 3 倍改善している．Jointgraph はコンソーシアム型を想定した設計で，ノード数が膨大となったときのスケーラビリティには必ずしも対応できていない．本研究では，ノード数が膨大となっても取引の遂行までの時間が長くなりすぎないことに焦点を当てた提案を行う．

Open Hash Graph [18] は，パブリック型のハッシュグラフシステムの構築を検討している．Open Hash Graph のコンセンサスプロトコルでは，ノードに信頼度を何段階かに設定し，高い信頼度のノードがノード数の調整やネットワークに影響を与えうるノードの排除を実行する権限を持つことで，パブリック型 DLT としてのノードの参入容易性を高めている．また，ハッシュグラフの同期を相互に行うことで，コンセンサスまでの時間短縮を行うという工夫が実装されている．本研究では，ハッシュグラフのスケーラビリティ問題に対処するため，ノード数調整の権限や相互同期についての複雑な実装は行わず，シャーディングによって達成可能な速度改善についての議論に焦点を当てる．

第 4 章

分割ハッシュグラフの基本提案

本章では，分割ハッシュグラフの基本提案を行う．ハッシュグラフのスケーラビリティ問題を解決する方法として，DLT におけるシャーディング技術のうち，特に合意形成時に参加するノード数を一定数以下に調整することでシステム全体を高速化する手法に着目し，合意形成の単位を比較的小規模なハッシュグラフに分割する分割ハッシュグラフを提案する．続く節では，分割ハッシュグラフの概要について説明を行い，その後で分割ハッシュグラフで起きうる取引のうち，シャード間取引と呼ばれる取引に関する合意形成のプロトコルを提案・説明する．

4.1 分割ハッシュグラフ

図 3 に分割ハッシュグラフの概念図を示す．分割ハッシュグラフは，ノードをいくつかのグループに分け，自身のグループ内と自身と他のグループ間のメンバーをそれぞれ合意形成のメンバーとする独立した複数のハッシュグラフと状態管理データベースを持つ．また，ノードが属しているグループのことをシャードと呼ぶことにする．通常のハッシュグラフと同様に，各ハッシュグラフで合意形成・処理順序決定されたイベントの内容に従って対応するデータベースの状態を更新する．別のシャードのメンバーとの取引を可能とするプロトコルが必要だが，1 つ 1 つのハッシュグラフの規模が小さいため，データベースの更新までの時間が通常のハッシュグラフよりも短縮され，取引の遂行までの時間も短くなることが期待できる．

通常のハッシュグラフでは，全てのノードが他のノードの残高情報などを持ち，1 回のトランザクションの実行でお互いの残高やデジタル資産の受け渡しが可能であった．分割ハッシュグラフでは，ハッシュグラフの同期に必要な IP アドレスや公開鍵の情報は全ノード分持っているが，分割されたハッシュグラフのそれぞれが、対応するノードの残高情報を保有するわけではない．なぜなら，図 3 の Group1 のノードは，Group2 と Group4 の間で行われた残高などの授受の情報を知ることができず，Group2 や Group4 の残高情報を保有していたとしても正確に更新することができないためである．従って，Group1 では Group1 のノードのみに関する残高などの情報を Group1 のシャードに対応するデータベース（ステートマシン）に保有する．このままでは異なるシャードのノード間での残高やデジタル資産の授受ができないため，自シャードと他シャードの両者が参加するハッシュグラフを活用する．具体的な説明は次節で行うが，異なるシャードのノード間の取引は，残高が充足していたかどうかなど，取引を実現する上で中間的に確認すべき事項を周知するトランザクションを発行し，取引の実現状態を何段階かに分けて更新していくことで実現する．自シャードと他シャードの両者が参加するハッシュグラフは取引の実現状態の更新に必要なトランザクション情報が行き交うハッシュグラフであり，対応するステートマシンにも，残高などではなく，取引の実現状態を記録・更新する．また，取引の実現状態の変化に応じて，必要なトランザクションを他のハッシュグラフのキューに追加する．

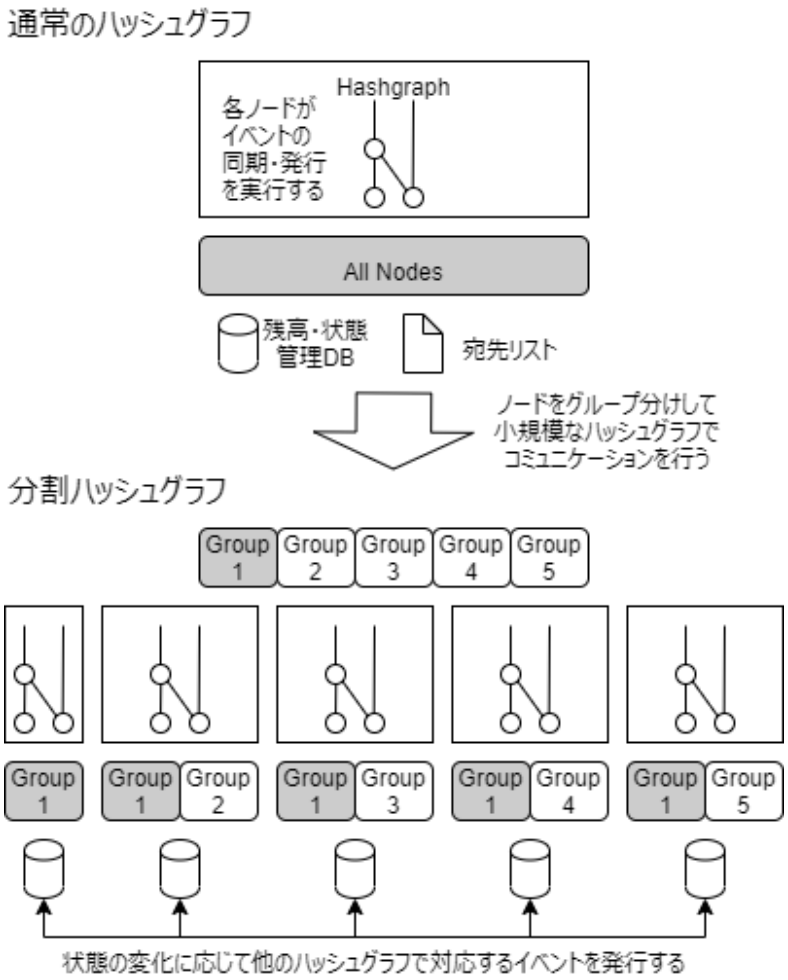


図 3　分割ハッシュグラフの概要図．分割ハッシュグラフでは，シャードと呼ばれるグループにノードを割り当て，各ノードが小規模なハッシュグラフとそれに対応するステートマシンを持ちながら合意形成プロセスを実行する．ハッシュグラフ 1 つ 1 つのノード数が少ないため，システム全体として合意形成に掛かる時間を短縮することができると考えられる．この図の Group1 のノードは，Group1 に閉じたハッシュグラフと，Group1 と他のグループのペア同士で合意形成を行うハッシュグラフの計 5 つのハッシュグラフを使い分ける．

4.2　シャード間取引のプロトコル

分割ハッシュグラフでは，既存のハッシュグラフと比較して，異なるシャードのノード間での残高やデジタル資産の授受を行うためのプロトコルが不足している．このような，異なるシャードのノード間での残高やデジタル資産の授受に関する取引とそれを実現するプロセスのことを指して，シャード間取引と呼ぶことにする．本研究の基本提案では，もっとも基本的なケースとして，シャード 1 のノードからシャード 2 のノードに残高を移動する取引の実現に必要なプロトコルを提案する．この取引では，シャード 1 のノードの残高が不足していた場合にシャード 2 のノードの残高を増やしてしまうことがないという，取引の一貫性に関わる必要最低限の条件を達成することのみを考える．

図 4 にシャード間取引のうち，シャード 1 のメンバー A からシャード 2 のメンバー B に残高 100 を移動する取引のプロトコルを示す．大まかな流れは以下の通りになる．

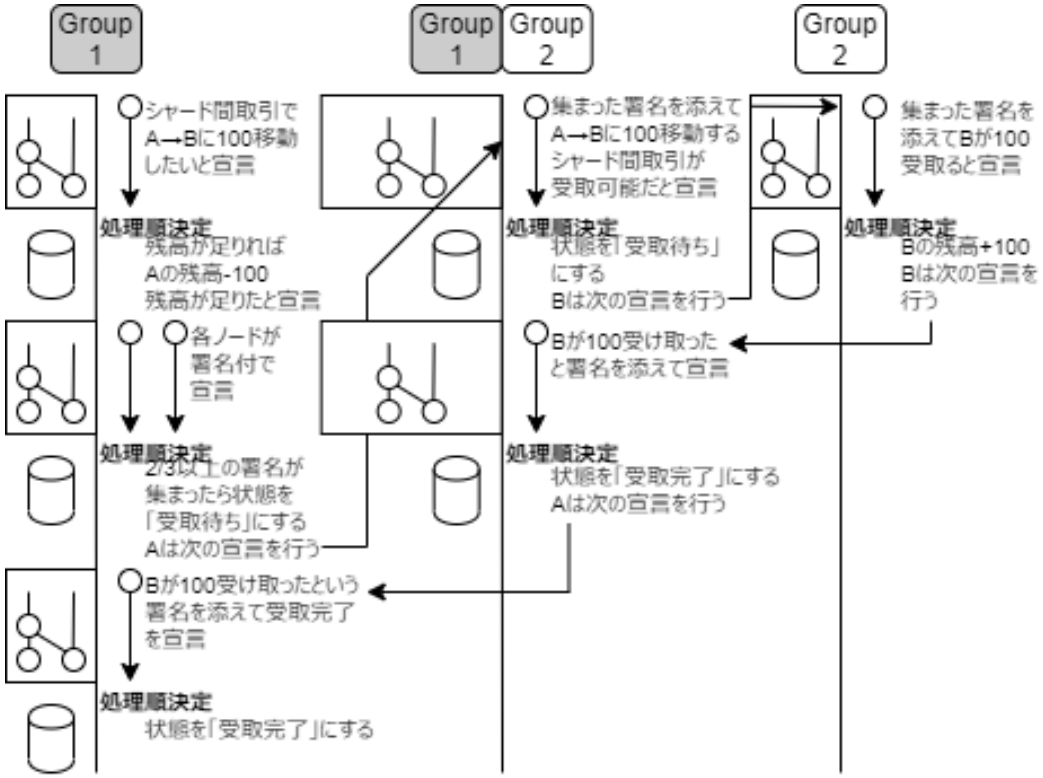


図 4　シャード間取引のプロトコルの概要図．計 6 回のトランザクションの承認・処理順決定・ステートマシンの更新を経て取引が完了する．

1. A が自シャード内でシャード間取引を宣言する．
2. シャード 1 のノードは A の残高を仮引き落とし出来たらその旨を宣言する．
3. シャード 1 の $2n/3$ 以上のノードから仮引き落とし完了の宣言が集まったら，A はシャード 1・2 間のハッシュグラフで残高が受け取り可能であると宣言する．残高不足だった場合は仮引き落とし完了の宣言が集まらないため，この先に進むことはない．
4. B は残高が受け取り可能である宣言を処理する際に，自シャードでシャード間取引の残高受け取りを宣言する．
5. シャード間取引の残高受け取りが終わったら，B はシャード 1・2 間のハッシュグラフでシャード間取引の残高受け取り完了を宣言する．
6. A はシャード間取引の残高受け取り完了の宣言を確認したら，自シャードでもシャード間取引の残高受け取り完了の宣言をする．

上記のプロセスの宣言はトランザクション文の形を取り，例えば以下のように定義できる．

1. ISTXsend A B 100
2. ISTXdeposit hash1 sig1
3. ISTXreceivable A B 100 hash1 sigs
4. ISTXreceive A B 100 hash1 hash3 sigs
5. ISTXreceived hash3 sig4
6. ISTXend hash1 hash3 sig4

ここで，ISTX とはシャード間取引を英語で表した Inter Shard Transaction の略であるとする．トランザクション文中の hash1 は 1. のトランザクションが含まれていたイベントのハッシュ値，hash3 は 3. のトランザクションが含まれていたイベントのハッシュ値である．ステートマシンに ISTX の処理進行状態を記録する際に，これらハッシュ値 hash1，hash3 をキーとして使用する．sig1 は，hash1 や取引金額の情報に対する処理者のデジタル署名で，確かに宣言通り処理した証拠として扱う．sigs はシャード 1 の $2n/3$ 以上のノードによる署名 sig1 を集めたもので，確かに残高の仮引き落としが完了している証拠として扱う．sig4 は，hash1 や取引金額の情報に対して B が行うデジタル署名である．

各ハッシュグラフに対応するステートマシンに記録された状態は，宣言 1.〜6. の処理により，次のように変化する．

シャード 1 のハッシュグラフに対応するステートマシン

- 宣言 1.
  残高：
  A：900 → 800
  状態：
  hash1：
  State:“waiting deposit”
  SuccessfulDepositSig:{}
- 宣言 2.
  状態：
  hash1：
  State:“waiting receive”
  SuccessfulDepositSig:{
  ’xxx.yyy.zzz.www’: sig1,
  ’aaa.bbb.ccc.ddd’: sig1’,...
  }
- 宣言 6.
  状態：
  hash1：
  State:“ISTX end”
  SuccessfulDepositSig:{
  ’xxx.yyy.zzz.www’: sig1,
  ’aaa.bbb.ccc.ddd’: sig1’,...
  }

シャード 1・2 間のハッシュグラフに対応するステートマシン

- 宣言 3.
  状態：
  hash3：
  OriginalEventHash: hash1

State:“deposit announced”

- 宣言 5.

  状態：

  hash3：

  OriginalEventHash: hash1

  State:“received”

シャード 2 のハッシュグラフに対応するステートマシン

- 宣言 4.

  残高：

  B：200 → 300

上記のプロセスを経ることで，シャード 1 のノードからシャード 2 のノードに残高を移動する取引を実現することが出来る．このようなプロセスにより，残高の仮引き落としと受け取りのタイミングを，残高を管理する自シャードのハッシュグラフの処理順決定プロセスによって確定させることが出来るので，他のシャードとのシャード間取引が並行して進行していても，残高の過不足の勘定を一貫性を持って実行することが出来る．

第 5 章

提案手法の性能評価

本章では，第 4 章で提案した分割ハッシュグラフの取引発行から処理順決定・処理完了までの遅延時間に関する性能評価について報告する．まず，シミュレーションの環境や条件に関する設定について述べ，既存のハッシュグラフに相当する単一シャードハッシュグラフで行った性能評価実験結果について報告する．単一シャードハッシュグラフの性能評価は，コンセンサスアルゴリズムを 3 過程に分け，各過程ごとに掛かった時間を詳細に調査したため，ノード数がスケールした場合のコンセンサス所要時間についてモデルを構築することが出来た．そのため，このモデルについての説明が続く．その後，100 ノードの場合における分割ハッシュグラフの性能評価を行い，ノード数がさらに増加した場合に想定される性能について議論を行う．

5.1 シミュレーション設定

本節では，ハッシュグラフの性能評価に際して行ったシミュレーション実験の環境や条件に関する設定について述べる．

5.1.1 ノードの初期化

ノードは，仮想的にハッシュグラフシステムのエントリーポイントとなる端末にアクセスし，ハッシュグラフのコンセンサスプロセスに参加しているノード数とそれらのノードの IP アドレス・公開鍵・所属シャードの情報を持っているものとする．また，全てのノードについて，取引できる初期残高を 1000 とする．分割ハッシュグラフでは，各ノードは等しい割合でシャードに割り当てられ，各シャードのノード数に偏りがないようにする．ハッシュグラフはイベント数 0 の初期状態から開始し，自ノードの列の一番先頭の初期イベントとして，トランザクションなし・対自ハッシュ値なし・対他ハッシュ値なしのイベントを追加する．以降，ハッシュグラフの同期処理をいつでも受け付けることが可能であり，自らハッシュグラフの同期要求を送る状態に移行する．

5.1.2 ノードの動作

ノードは アルゴリズム 1 に則り，ハッシュグラフとステートマシンを更新していく．単一シャードハッシュグラフと分割ハッシュグラフとで少々動作方法と終了条件が異なる．

単一シャードハッシュグラフの場合

1. 一定確率でランダムなノードへ残高を移す取引を発行し，次のイベントに含めるトランザクションのキューに追加する．

2. ランダムなノードを選択し，ハッシュグラフの同期要求をする．
3. ネットワークを通して相手から情報を受け取る．（sync/IO）
4. 自分のハッシュグラフを更新し，新しく生成される先頭イベントにキューにあるトランザクションを含める．（sync/update）
5. イベントのコンセンサス判定を行い，処理順が決定したイベントのトランザクションを実行してステートマシンを更新する．（consensus）

分割ハッシュグラフの場合

1. 自身が管理しているハッシュグラフそれぞれについて，以下を繰り返す．
2. 一定確率でランダムなノードへ残高を移す取引を発行し，次のイベントに含めるトランザクションのキューに追加する．相手が同じシャードに属するかどうかでトランザクション文を適切に構成する．
3. ランダムなノードを選択し，ハッシュグラフの同期要求をする．
4. ネットワークを通して相手から情報を受け取る．（sync/IO）
5. 自分のハッシュグラフを更新し，新しく生成される先頭イベントにキューにあるトランザクションを含める．（sync/update）
6. イベントのコンセンサス判定を行い，処理順が決定したイベントのトランザクションを実行してステートマシンを更新する．シャード間取引の処理で必要な場合は，適切なシャードのトランザクションのキューに適切なトランザクションを追加する．（consensus）

いずれの場合も，新たな取引トランザクションの発行は各ループごとに確率 10 % で発生し，取引金額は 0 ～100 の範囲でランダムに選択されるようにした．

5.1.3 ノードの動作終了条件

単一シャードハッシュグラフの場合　10,000 個のイベントの処理順序が決定し，ステートマシンの更新が終了すること
分割ハッシュグラフの場合　100 個のシャード間取引が終了状態になること

5.1.4 ノードの設定

ノードとして，以下のスペックの PC を実験に使用した．

マシン　MacBook Pro（13 インチ，2017 年モデル）
CPU　Intel Core i5 2.3 GHz
メモリ　8 GB

5.1.5 ハッシュグラフの同期通信に関する設定

各ノード同士は実際にはネットワークを通じてハッシュグラフの情報をやり取りする必要があるが，本シミュレーションでは時間短縮のため，データの送受信はプログラムの関数上で擬似的に行い，代わりにネットワーク環境を設定し，データの送受信に掛かる時間の推定値をシミュレーション結果に加算した．本研究では，国内の高速なネットワーク回線の環境を想定し，ノード間の通信遅延は片道 10 ms，通信速度は 100 Mbps とした．

5.1.6 シミュレーション方法

シミュレーションは上記の設定で動作するノードと同じ動作をするプログラムを「ノードの設定」で記載した PC を用い，シングルスレッドで実行した．動作すべきノードは複数あるため，動作ループを 1 周させるノードをランダムに選択し実行するようにして，全てのノードのループが一通り終了したら次のループへ進むようにした．

システム時刻をハッシュグラフのイベントにそのまま使用してしまうと，全てのノードが同時並行的に動作していることに反してしまうため，対策を行った．本シミュレーションでは，ノードの動作開始時刻をシステム時刻とし，以後，全ての処理とネットワーク処理に掛かる時間を計測（ネットワーク処理に掛かる時間は送受信データ量から計算）し，システム時刻に加算していく擬似的なシステム時刻を各ノードが所有し，この擬似的なシステム時刻をイベントのタイムスタンプに使用することで，シングルスレッドで多数のノードを複数同時に動作させているのと同じ時間の進行状況を再現した．

また，分割ハッシュグラフのシミュレーションでは，各シャードのプロセスごとにノードの CPU を割り当て，並行処理させることができるため，本シミュレーションではノードの CPU は無限のコア数を持ち，いくつでもプロセスを並行処理できると仮定した．よって，擬似的なシステム時刻を各ハッシュグラフごとに所有し，各シャードごとに擬似的なシステム時刻を進めることでプロセスを並行処理している状況を再現した．ノードのループ処理においてあるハッシュグラフの擬似的なシステム時刻が進みすぎてしまわないよう，ハッシュグラフの処理順は擬似的なシステム時刻が小さい順とした．さらに，単一シャードハッシュグラフと比較して，ネットワーク帯域をシャード分割数分だけ多く使うと考えられるので，ネットワークの通信速度はシャード分割数分の 1 として設定した．

5.1.7 測定内容

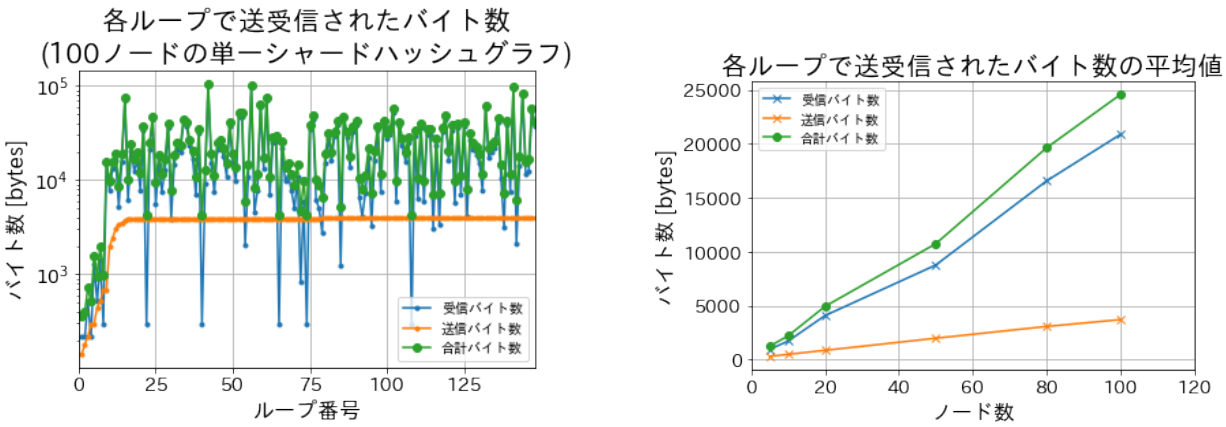
本シミュレーションで測定した数値は以下の通りである．

- sync/IO に要した時間（送受信データ量から計算）
- sync/update に要した時間
- consensus に要した時間
- イベントの発行から処理順決定までに要した時間（finality time）
- （分割ハッシュグラフのみ）シャード間取引の発行から終了状態になるまでに要した時間（ISTX finality time）

5.2 単一シャードハッシュグラフの詳細評価

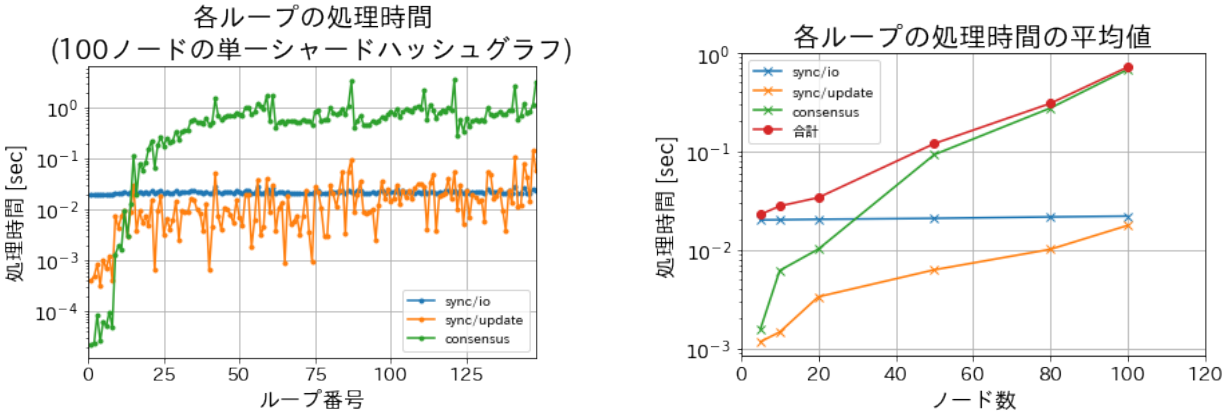
単一シャードハッシュグラフについて，前節のシミュレーション方法に則り，ノード数 2,5,10,20,50,100 の場合についてシミュレーションを行った． 図 5–8 に，100 ノードのシミュレーション時の詳細な結果と各ノード数ごとの平均値のデータを示す．

図 5 によれば，ハッシュグラフの各ループでは受信バイト数が送信バイト数を大きく上回る．送信バイト数に数えられるデータは，同期要求時に要求者がハッシュグラフの情報をどこまで知っているかについて，各ノードの列の先頭のイベントのハッシュ値を取り出して伝えているものであり，高々ノード数に比例するデータ量しかない．対する受信バイト数に数えられるデータは，相手から帰ってくる複数のイベントの集合であり，イベントのハッシュ値以外のトランザクションなどの情報も含むため，送信データに比べて相対的に膨大



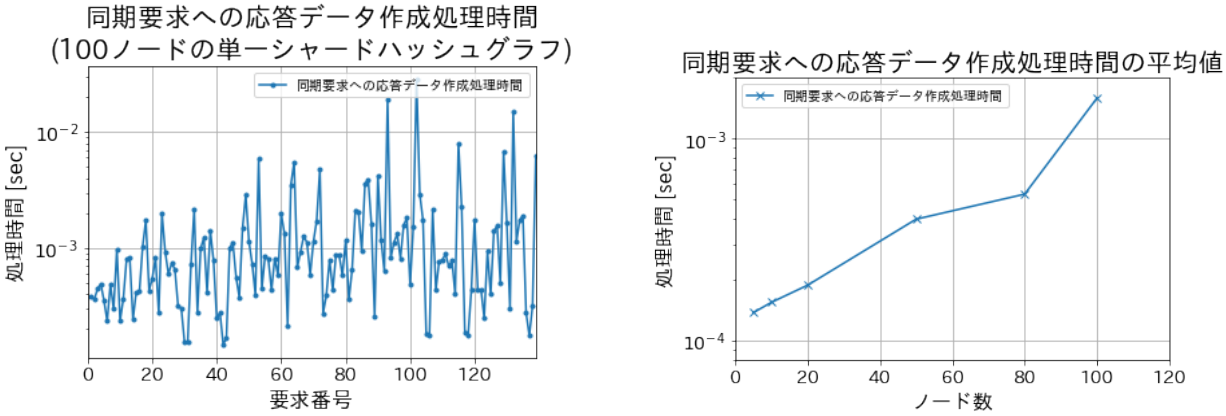
(a) 100 ノードでのデータ　　(b) 各ノード数ごとの平均値

図 5　単一シャードハッシュグラフの各ループで送受信されたデータ量.



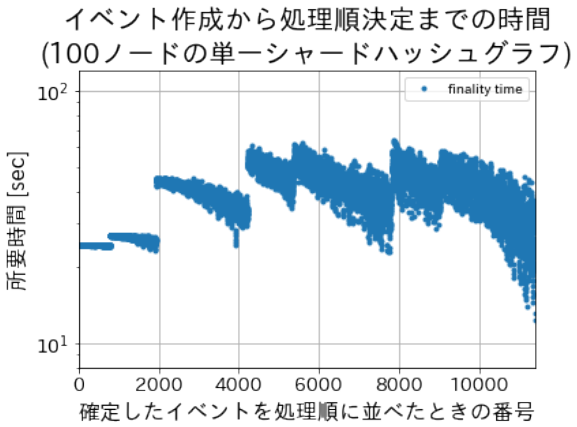
(a) 100 ノードでのデータ　　(b) 各ノード数ごとの平均値

図 6　単一シャードハッシュグラフの各ループの処理時間.

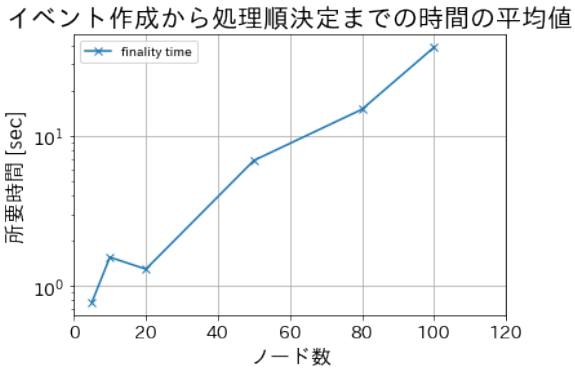


(a) 100 ノードでのデータ　　(b) 各ノード数ごとの平均値

図 7　単一シャードハッシュグラフの同期要求への応答データ作成処理時間.



(a) 100 ノードでのデータ



(b) 各ノード数ごとの平均値

図 8　単一シャードハッシュグラフのイベント生成から処理順決定までの時間.

なデータ量となる．以上より，受信バイト数が送信バイト数を大きく上回ることが理解できる．

図 6 は，各ノードの同期や仮想投票に関するループ処理にかかる時間の分析結果である．ハッシュグラフのプロセスで 1 ループあたり何の処理にどれだけの時間が使われているかについて，実際に計測と分析を行った例はこれまでになかった． 図 6a を見ると，ノード数が増加すると consensus のプロセス，すなわち仮想投票によるイベントの処理順決定にかかる時間がほとんどを占めることが分かる．ネットワーク環境を比較的高速なものに設定しているため，各ループで送受信されるバイト数に対する送受信の時間はほとんど通信遅延の 10 ms のオーダーとなっている．

図 7 を見ると，同期要求への応答データ作成処理時間のオーダーは 10^{-4}〜10^{-3} s と，各ループの処理時間のオーダーである 10^{-2}〜10^{0} s に対して非常に小さく，無視できると考えられる．

図 8 を見ると，ノード数が増加した時，イベント作成から処理順決定までの時間はノード数に比例するように増加していることが分かる．本シミュレーションでは，非常に時間がかかってしまうという理由から 1,000 ノードや 10,000 ノードでのシミュレーションを行っていないが，ノード数 100 の時点で平均所要時間が 20 秒以上のため，スケーラビリティの観点から，単一シャードハッシュグラフは何千何万ものノードで実用的な運用が出来ないことが再確認された．

また， 図 5a，6a，7a，8a より，10,000 イベントの処理順確定までを終了条件とした本シミュレーションでは，ループ番号の増加に従って，計測値が一定の範囲の値をとるようになっていることから，その平均値を使ってノード数の増加に対する計測値の変化の傾向を確認していることは問題ないと考えられる．

5.3　ハッシュグラフの承認遅延時間の仮説的モデルの構築

本節では，前節の単一シャードハッシュグラフの性能評価を受けて，よりノード数が多いハッシュグラフの処理とイベントの処理順決定までの所要時間の推定を行うモデルの作成を行う．

図 9 に単一シャードハッシュグラフの各ループの処理時間の推定曲線を示す．ノード数 n のとき，各プロセスにかかる時間の推定値は以下のようになった．

- sync/update: $3.483 \times 10^{-5} n \log n$ s
- sync/IO: $(37.97n + 203.4n) \times 8/100 \times 10^{6} + 0.01 \times 2$ s

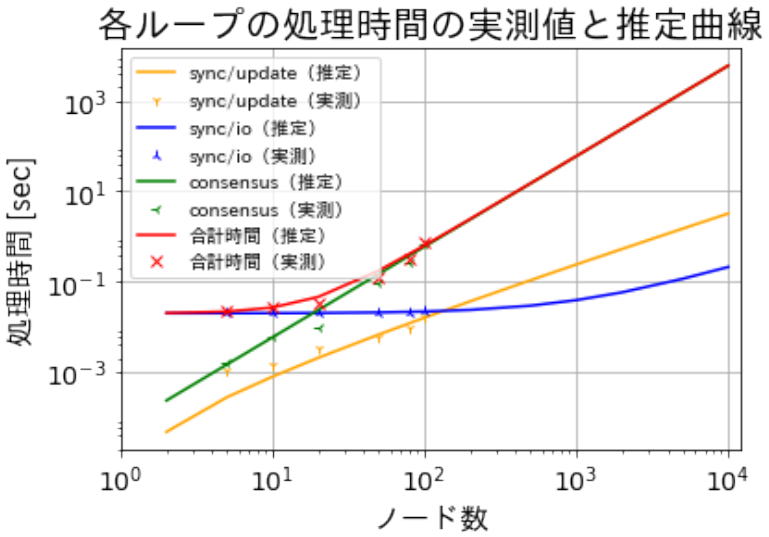


図 9　単一シャードハッシュグラフの各ループの処理時間の推定曲線.

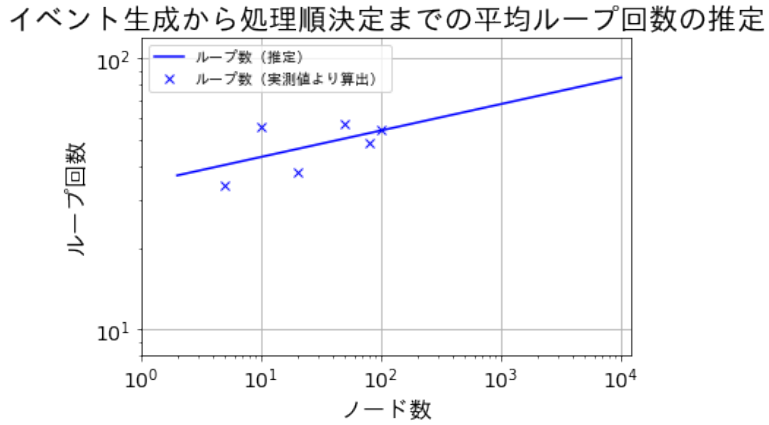


図 10　単一シャードハッシュグラフのイベント生成から処理順決定までの平均ループ回数の推定.

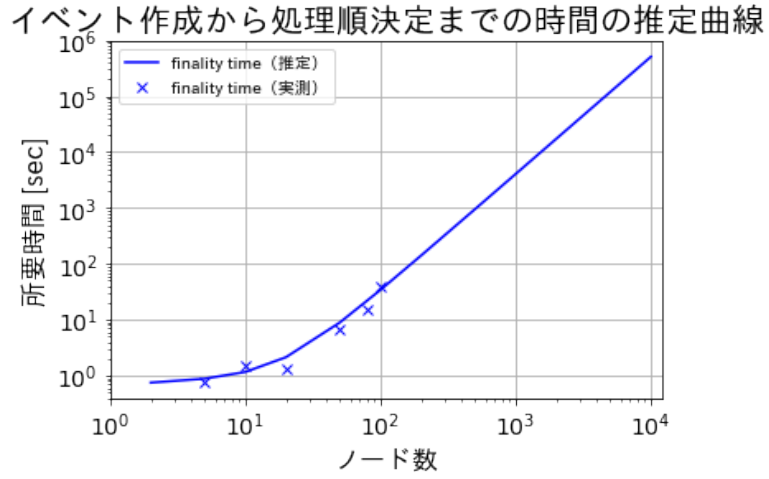


図 11　単一シャードハッシュグラフのイベント生成から処理順決定までの時間の推定曲線.

- consensus: $5.936 \times 10^{-5} n^2$ s

sync/update では，イベントをハッシュグラフに追加する際に，まずノード数に比例するイベントのトポロジカルソートでの並べ替えに $O(|V| + |E|)$ を要する．頂点の数 V と辺の数 E は両方ノード数に比例するため，計算量 $O(n)$ である．また，連想型配列へのイベントの追加に $O(\log n)$ を要するため $O(n \log n)$ での近似とした．

sync/IO のうち，$37.97n$ は送信データのバイト数，$203.4n$ は受信データのバイト数である．送信データや受信データのバイト数はノード数に比例するため 1 次近似を行った．送信データや受信データのバイト数はノード数に比例することは，ハッシュグラフ内のイベントの総数がノード数に比例することから分かる．

consensus は，仮想投票の際に深さ優先探索を行い，探索の回数と探索の計算時間がそれぞれノード数に比例する（なぜなら，ノード数の分だけ起点となるウィットネスが存在し，多くのノードを通るパスを見つけきるにはおよそノード数分だけグラフをたどる必要があるため）かた，2 次近似を行った．

以上の近似から算出された各ループの処理時間の推定曲線は 図 9 によれば，実測値から大きく外れておらず，まずまずの近似であると言える．

前述した各ループ処理時間のモデルに基づき，イベント作成から処理順決定までの時間の推定を行いたい．ここで，イベント生成から処理順決定までの平均ループ回数の推定を行うことによって，イベント作成から処理順決定までの時間の推定を行うことにした．イベント生成から処理順決定までの平均ループ回数のサンプル値は，実際の処理順決定までの所要時間を 1 回のループあたりの平均処理時間で割ったものを使用した．図 10 で 1 次近似を行ったところ，平均ループ回数は $34.6 \times n^{0.09769}$ と近似された．この近似を利用して，イベント作成から処理順決定までの時間の推定を行ったところ， 図 11 のように，実測値から大きく外れない近似となった．

以上の結果から，よりノード数が多いハッシュグラフの処理とイベントの処理順決定までの所要時間を推定するモデルが作成できたと考える．

5.4　分割ハッシュグラフの性能評価

本節では，分割ハッシュグラフについて，ノード総数を 100 に固定し，シャード分割数 2,3,5,10,20 の場合についてシミュレーションを行った結果をまとめる．

単一シャードハッシュグラフのシミュレーション結果からも想像がつくように，分割ハッシュグラフでは 図 12–15 から分かるように，シャード分割数が増えるほど 1 つのハッシュグラフあたりのノード数が減り，やり取りされるデータ量や処理時間，個々のイベントの処理順決定までの時間が短くなっていることが分かる．

また， 図 13 の sync/IO の処理時間を見る限りにおいて，分割ハッシュグラフがネットワーク帯域を複数のハッシュグラフで使用することによる通信速度低速化の影響は出ていないと考えられる．

図 15 を見ると，他シャードとのハッシュグラフにおけるイベントの処理順決定までの時間は自シャードのハッシュグラフよりも大幅に長いことが分かる．これは，他シャードとのハッシュグラフの参加ノード数が自シャードのハッシュグラフの参加ノード数の 2 倍になっていることに起因する．本シミュレーションでは，イベントの発行相手はシャード内外問わずランダムに決められているため，他シャードとのハッシュグラフでの処理の占める割合が多く，各イベントの処理順決定までの時間の平均値は他シャードとのハッシュグラフでの処理準決定までの時間に近くなっている．

図 16 はシャード間取引の作成から完了までの時間の平均値を表すグラフである．シャード間取引のプロ

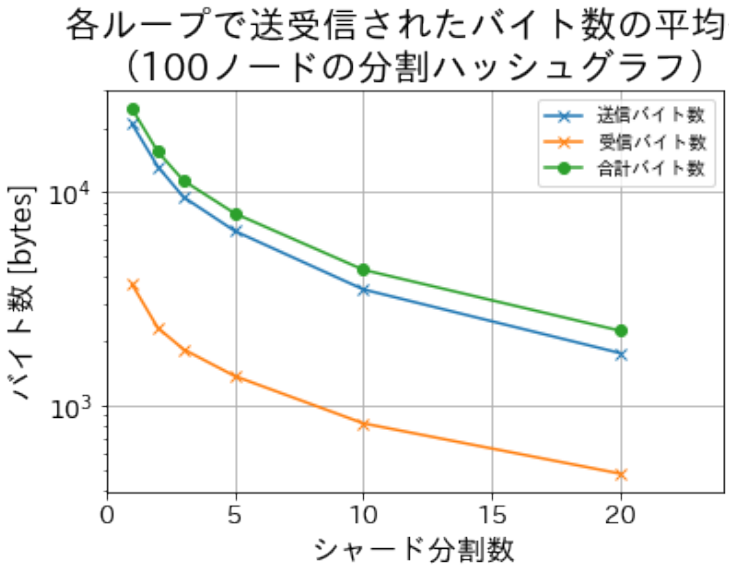


図 12　分割ハッシュグラフの各ループで送受信されたデータ量.

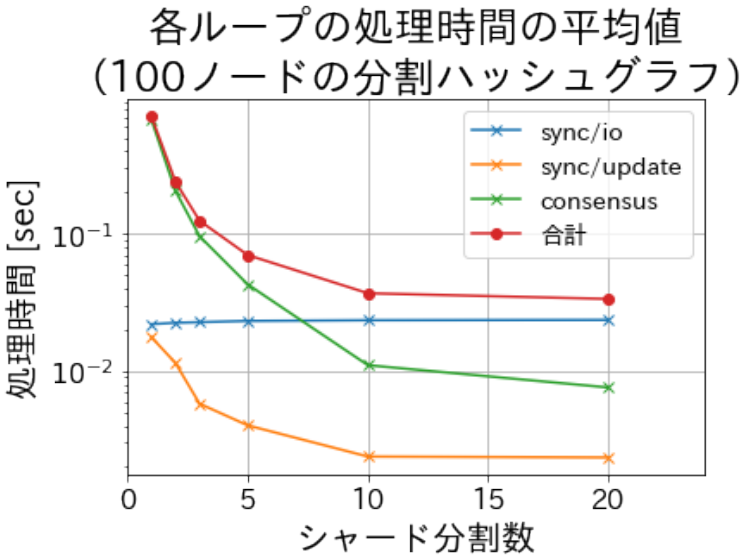


図 13　分割ハッシュグラフの各ループの処理時間.

トコルとして，自シャードのハッシュグラフで 4 回，他シャードとのハッシュグラフで 2 回の計 6 回イベントの承認が必要なため，そのオーバーヘッドに勝る利点が分割ハッシュグラフにあるか知る必要がある．シャード分割数 2 の場合は，分割無しの場合よりも取引の完了にかかる時間が長い．これは，分割前と同じノード数のハッシュグラフと半数のノードによるハッシュグラフを両方使い，多段階でシャード間取引を実行するオーバーヘッドがあるためであると考えられる．それ以外は，シャード間取引の完了までにかかる時間は分割無しの場合よりも短く，シャード分割数が増えるほど短くなる．これにより，分割ハッシュグラフは，シャード間取引のプロトコルを取り入れてなお，単一シャードハッシュグラフよりも高速に取引を完了できることが分かった．

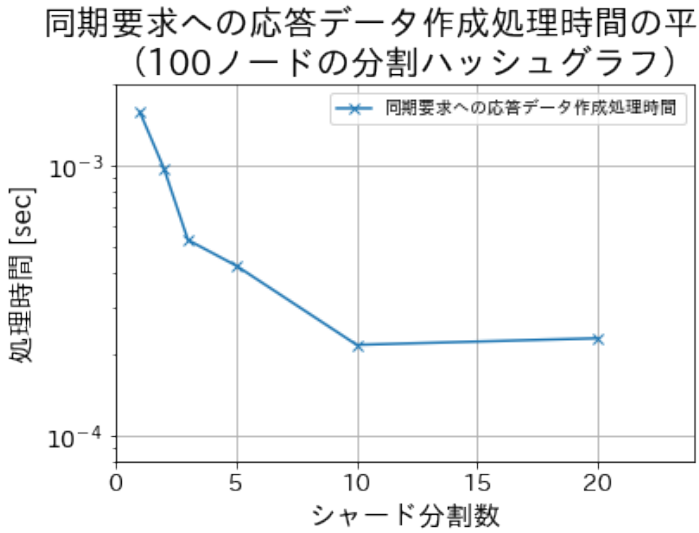


図 14　分割ハッシュグラフの同期要求への応答データ作成処理時間.

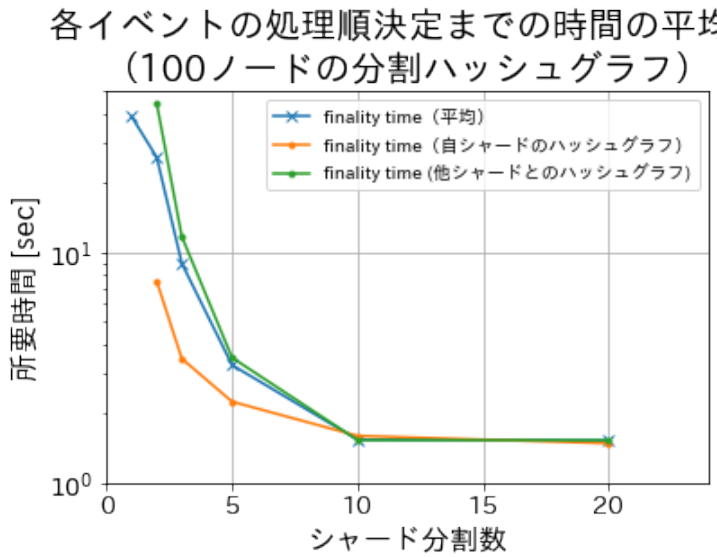


図 15　分割ハッシュグラフのイベント生成から処理順決定までの時間.

5.5　分割ハッシュグラフの性能限界の推定

本節では，これまでのシミュレーション結果を受け，よりノード数が多い場合に分割ハッシュグラフを適用すると，どれほど取引の完了時間を短縮できるのかについて検討を行う．

まず，シャード間取引の完了に要する時間の推定を，単一シャードハッシュグラフで求めたモデルを用いて行う．シャード間取引のプロトコルでは，自シャードのハッシュグラフで 4 回，他シャードとのハッシュグラフで 2 回の計 6 回イベントの処理順の決定とトランザクションの実行を繰り返す必要がある．1 シャードあたりのノード数が m ならば，ノード数 m のハッシュグラフで 4 回，ノード数 $2m$ のハッシュグラフで 2 回コンセンサスに到達する必要があるため，ノード数 n のハッシュグラフでイベントの発行から処理順決定までにかかる時間を $t(n)$ とすると，シャード間取引の完了にかかる時間は $4t(m) + 2t(2m)$ であると推定できる．ただし，$t(n)$ はシャード分割数によるネットワーク速度低下を考慮した関数である必要がある．つまり，

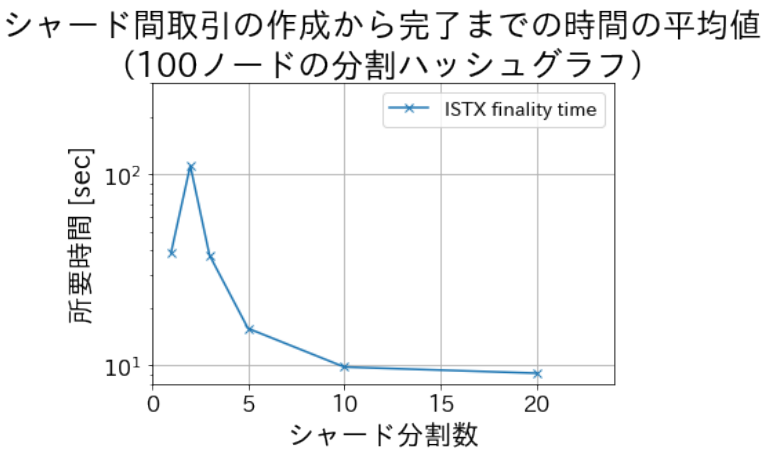


図 16　分割ハッシュグラフのシャード間取引の作成から完了までの時間.

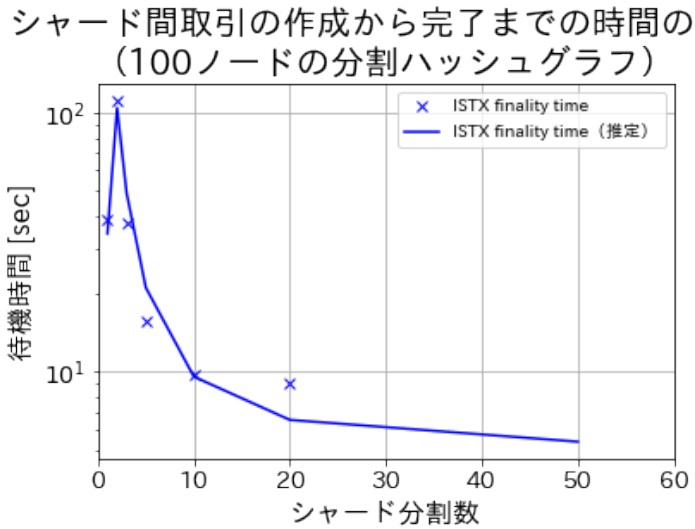


図 17　分割ハッシュグラフのシャード間取引の作成から完了までの時間の推定.

実際には引数にシャード分割数 k を与えるというようなことをする必要がある．$4t(m) + 2t(2m)$ によって推定された，100 ノードの分割ハッシュグラフでの，シャード間取引の完了にかかる時間の推定結果を 図 17 に単一シャードハッシュグラフの各ループの処理時間の推定曲線を示す．この推定結果はおよそ実測値と一致することが分かる．

次に，ノード数が増加した場合のシャード間取引の完了にかかる時間の最適化を行うことを考える．単一シャードハッシュグラフで求めたモデルと $4t(m) + 2t(2m)$ によるシャード間取引の完了にかかる時間の推定推定によって，シャード分割数を様々に変化させながら，シャード間取引の完了にかかる時間が最小となるシャード分割数を探すことが出来る．100,200,400,...,51200 ノードの分割ハッシュグラフそれぞれについて，シャード間取引の完了にかかる時間の最小値と最適なシャード分割数における 1 シャードあたりのノード数を調べた結果を 図 18 に示す． 図 18 を見ると，分割ハッシュグラフで達成できるシャード間取引の完了にかかる時間の最小値は，単一シャードハッシュグラフよりも圧倒的に小さく抑えられることが分かる．約 10,000 ノードの場合の最小値は約 100 秒であり，十分高速であるとは言い切れないが，単一シャードハッシュグラフの場合の 10^6 秒よりは圧倒的に短くなっていることが分かる．なお，最適値を達成するときの

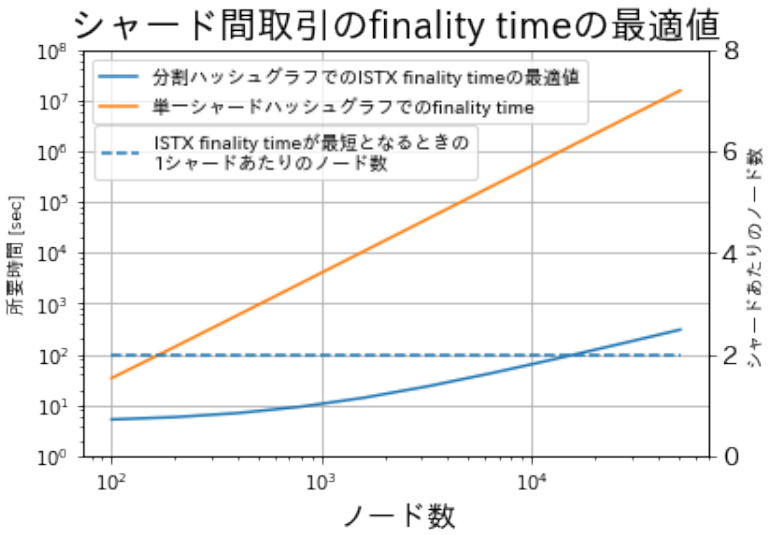


図 18　分割ハッシュグラフのシャード間取引の finality time の最適値と最適シャード分割数.

シャードあたりのノード数を見てみると，理論上の最小値である 2 ばかりであり，可能な限り細かいシャードに分割しようとしてようやく達成できる数値であると分かる．

しかし，可能な限り細かいシャードに分割した場合，ハッシュグラフの合意形成においてビザンチン耐性を達成するための条件である $2n/3$ 以上の正直なノードが集まらない可能性が出てきてしまう．あるノードの母集団に確率 p で悪意のあるノードが存在するとし，ランダムで n 台のノードを選出した場合に，$2n/3$ 以上のノードが無害，つまりコミュニティがビザンチン耐性的に健全である確率は以下のようにして計算される．

$$\sum_{r=0}^{\lfloor \frac{n}{3} \rfloor} {}_nC_r p^r (1-p)^{n-r}$$

これを様々な確率 p について計算した結果を 図 19 に示す．

不正の起こる確率を 10^{-8} 以下に抑えるには，$p = 0.1$ の場合，100 ノード以上がハッシュグラフに存在する必要があることが分かる．並行処理に必要な CPU も無尽蔵にコア数を増やせるわけではないことからも，分割ハッシュグラフのシャード分割数は一定数以上のノードを含んで安全性を担保しつつ，並列処理可能な範囲内で調整する必要がある．

図 20 に，1 シャードあたり 100 ノード以上となる条件付きでシャード間取引の完了にかかる時間の最小値と最適なシャード分割数における 1 シャードあたりのノード数を調べた結果を示す．この場合，ほとんどのケースで 1 シャードあたり，最低値である 100 ノードが属するように分割を行った場合にシャード間取引の完了にかかる時間が最小化されることが分かる．この場合でも，単一シャードハッシュグラフよりも取引の完了に要する時間は圧倒的に早いと言える．

5.6　総合的な評価

以上の性能評価結果より，分割ハッシュグラフは単一シャードハッシュグラフと比較して，確かに取引の完了に要する時間を短縮する効果のある手法であることが分かった．

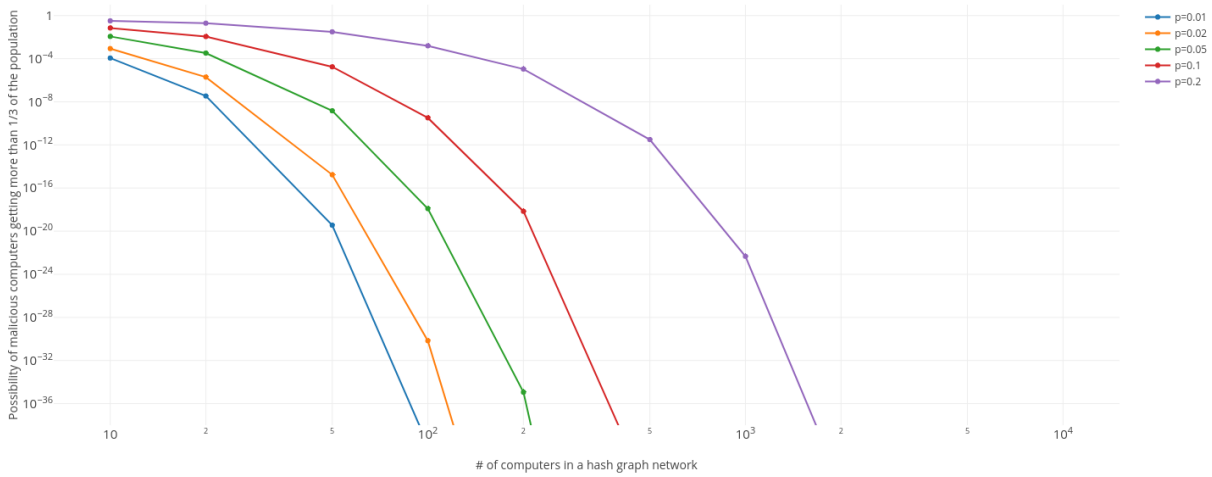


図 19　有害なノードが混ざりうる時にハッシュグラフで健全な合意形成ができなくなる確率.

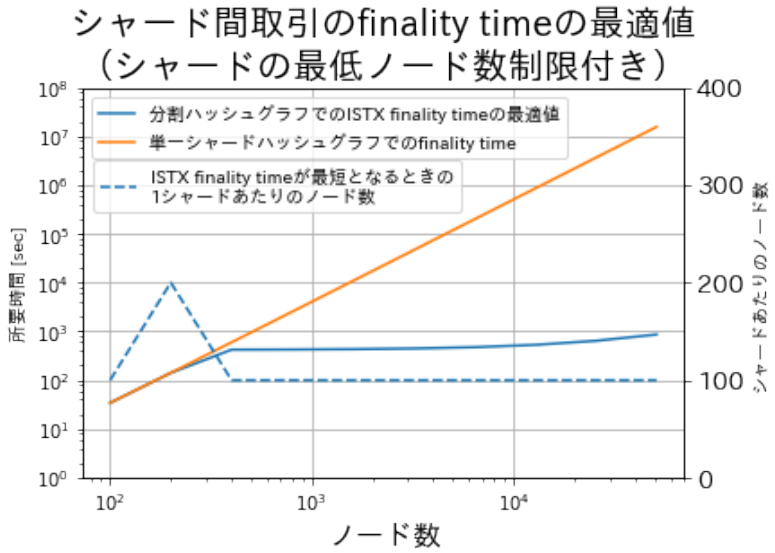


図 20　分割ハッシュグラフのシャード間取引の finality time の最適値と最適シャード分割数（シャードの最低ノード数の制限付き）.

第 6 章

分割ハッシュグラフの実用化に必要な機能の考察

本章では，分割ハッシュグラフをパブリック DLT として実用化レベルに到達させるために必要であろう機能のうち，シミュレーションで実装しなかった内容について検討を行う．

6.1 状態の保存と新規ノード参加

パブリック DLT では，ノードは自由に合意形成の輪に参加したり離脱出来たりすることが普通であると考えられている．本研究では，仮想的なエントリーポイントを想定してノードの初期化を行ったが，ノード数は固定されており，ノードの自由な参加や離脱のプロトコルが考慮されていない．分割ハッシュグラフをパブリック DLT として運用するためには，ハッシュグラフの状態の保存の機能と新規ノード参加のための機能の実装が必要である．新規ノード参加は，ハッシュグラフの状態の保存が出来れば自ずと可能になると考えられる．

まず，ハッシュグラフの状態の保存方法について検討を行う．Open Hash Graph を参考にすると，ハッシュグラフの情報を，あるイベントの過去と未来とで分断して考えるという手法がとられている．ハッシュグラフの情報を保存したい場合，毎回全ハッシュグラフの情報のコピーを取ることは非効率である．そのため，重複のないようにイベントの集合をハッシュグラフから切り取って名前を付けて保存するという方法が考えられる．イベントの集合をハッシュグラフから切り取るのに，過去と未来とを分断する特別なイベントが使える．

ハッシュグラフの状態を保存したいと思ったノードは，チェックポイントイベントと呼ばれるイベントを発行する．Open Hash Graph とは異なり，チェックポイントイベントは全てのノードによって発行されている必要はない．チェックポイントイベントの処理順が決定すると，ハッシュグラフのスナップショットの保存が実行される．このとき，チェックポイントイベントが存在する列のノード以外のノードの先頭を探す．具体的には，チェックポイントイベントから “見る” ことができるイベントのうち，チェックポイントイベントにもっとも近いイベントを各ノードについて選択する．そして，ハッシュグラフのスナップショットには，各先頭のイベントから “見る” ことの出来るイベントのうち，1 つ前のスナップショットの先頭までのイベントと，チェックポイントイベントの処理時点でのステートマシンの状態，そして 1 つ前のチェックポイントイベントのハッシュ値を保存する．これにより，ハッシュグラフのスナップショットは，ハッシュグラフのイベントの集合とステートマシンの状態のセットがチェーン状に連なった単方向の連結リストとなる．

次に，ハッシュグラフのスナップショットを活用して新規ノードが参加するプロトコルを考える．新規ノードがハッシュグラフに参加したい場合，新規ノードは既存のノードに参加要求を出す．既存ノードはスナップショットイベントを新規参加ノードの IP アドレスや公開鍵の情報を含めて発行する．スナップショットイベントの処理時に既存ノードはスナップショット情報を新規参加ノードに渡し，新規参加ノードはスナップ

ショットに含まれるイベントとステートマシンで自身を初期化する．スナップショットイベントの後では通信が可能なノードの一覧に新規参加ノードが加わり，新規参加ノードはスナップショットイベントの後のイベントのみについて処理順の決定と処理を行えばよいため，整合性が保たれる．

6.2　シャード間取引の中断処理

本研究のシミュレーションでは，シャード間取引について残高不足の場合にそれ以上取引が進行しないという一貫性を実現することだけを要件としていた．しかし，残高が充足している場合でも，ある取引の進行状態に到達して以降ネットワークの不具合などによってイベントがネットワークに流れなくなると，残高の仮引き落としが終わっているが，相手が受け取ったかどうか分からない状態に陥ってしまう．残高の仮引き落としが終わったノードは，状況によってはシャード間取引をキャンセルし，仮引き落としされた残高を取り返したいかもしれない．しかし，相手が既に残高を受け取っている可能性がある以上，安易に仮引き落としを取り消すプロトコルを実装することはできない．これは既存のハッシュグラフの取引ではなかった問題であり，キャンセル処理が可能であるなど，広義に一貫性のある取引を実現するプロトコルについて検討する必要がある．

6.3　シャード間取引のプロトコルの集約署名による高速化

本研究のシミュレーションで扱ったシャード間取引のプロトコルをもう一度検討すると，手順 3. で送信するデジタル署名のリストはトランザクション文に占めるバイト数の割合が大きく，ハッシュグラフ情報を逼迫している．本研究のシミュレーションでは高速で安定したネットワークを仮定したが，実際のネットワークが低速である可能性も勘案すると，送受信されるデータ量は少なくなることが望ましい．

BLS 署名 [19] は，署名検証にペアリングと呼ばれる双線型写像を用いている署名方式である．そして，BLS 署名は複数名による署名を 1 つの署名に集約することができる性質を持つ．従って，手順 3. で送信するデジタル署名を BLS 署名のうち，集約署名とすることによって，1 つの署名と同等のデータ量のオーバーヘッドのみで済ませることができる．すなわち，他の手順で送信される署名と同じオーバーヘッドとなる．BLS 署名を使用する場合，ペアリング計算を行わなくてはいけない関係上，他の署名方法と計算時間を比較する必要があり，慎重に検討するべきである．

6.4　シャード間取引を回避するシャード割り当て

本研究では，取引を行うノードをランダムに選択した．しかし，現実に行われる取引を考えると，特定の相手と頻繁に取引を行ったり，稀に 1 度きり取引を行う相手がいたりするというように，取引をする相手と取引をする頻度に偏りがある．本研究で提案したシャード間取引プロトコルを利用すると，6 回の処理順決定プロセスを経るオーバーヘッドがあるため，可能であればシャード間取引を回避する方が良い．

シャード間取引を回避するためには，シャード割り当ての段階で取引頻度の高いノード同士を同じシャードに割り当てれば良い．シャード間取引が回避できれば，シャード間取引に関わるトランザクション情報をネットワーク帯域を使って送信する頻度が減るため，平均的なネットワーク通信速度が多少改善したり，ハッシュグラフ情報を保存するのに必要な記憶容量も減らすことができる．すなわち，コンセンサスのループ処理に要する時間を削減でき，取引の完了までの所要時間を削減できる可能性もある．

ただし，特定の相手との取引の頻度という恣意的に調整可能な基準でシャードの割り当てを行うと，悪意のあるノード同士が本来の確率以上に集まって，ハッシュグラフの合意形成の健全性を損なう可能性がある．従って，取引頻度の高いノードの集合のうち，共通部分をもたない複数の集合を合わせたシャードを形成するなどといった工夫が必要である．様々な取引傾向のあるノードに合わせて，分割ハッシュグラフで安全で効率の良い取引を実現するためのシャードの割り当てについて検討することは，分割ハッシュグラフの今後の重要な研究課題であるといえる．

第 7 章

結論

7.1 本研究の成果

本研究では，DLT のシャーディング技術に着目し，小規模なハッシュグラフを複数組み合わせて使う分割ハッシュグラフを提案した．ハッシュグラフの取引承認までに要する時間は，承認プロセスに参加するノード数が少ないほど短いため，分割ハッシュグラフによってシステム全体で平均的な取引承認までの時間を短くすることができると予想することができた．

本研究では，これまでになかった，ハッシュグラフのコンセンサスループの各プロセスの処理に要する時間に関する詳細な検証を行い，ノード数からハッシュグラフの取引承認までの時間を算出するモデルを構築することができた．そして，分割ハッシュグラフについてシミュレーションを行い，シャード間取引プロトコルを運用しつつも，ノード数 100 以上の場合に単一シャードハッシュグラフよりも大幅に取引承認までの時間を短くすることができた．また，取引承認までの時間を短くするシャード分割数について分析を行い，1 シャードあたりのノード数を 100 以上にするというセキュリティ上の条件をつけても，10,000 ノード程度の場合，単一シャードハッシュグラフよりも取引承認までの時間が短いことが確認できた．本研究により，パブリック型のスケーラブルな DLT システムをハッシュグラフで実現する場合，分割ハッシュグラフを適用することが有効であることを示すことができた．

7.2 今後の展望

本研究の今後の方向性を 2 つ述べる．

1 つ目は，より多様な条件設定による分割ハッシュグラフのシミュレーションによって，分割ハッシュグラフの特性を詳細に分析することである．本研究では，分割ハッシュグラフについて最もシンプルだと考えられる条件で実験を行った．そのため，ネットワーク遅延の増大など，実際の通信で考えられるシナリオをできる限り網羅したシミュレーション設定で実験することにより，あらゆる環境に対応した分割ハッシュグラフの設計に役立てることが考えられる．また，100 ノード以上の場合に，処理時間のモデルで計算した取引承認までの時間の推定結果が合っているかを実際に確かめる必要がある．

2 つ目は，分割ハッシュグラフをパブリック型の DLT システムとして実用化するに当たって，第 6 章で述べたような要件について，あり得るシナリオを想定し，分割ハッシュグラフの適当性を検討することである．分割ハッシュグラフがどのようなシナリオに対応できて，どのようなシナリオでは対応できないかを明確化することによって，分割ハッシュグラフの改良や応用先について適切な提言が可能になると考えられる．

謝辞

初めに，毎週の全体ミーティングや個別ミーティングで多大なるご指導をいただいた相田仁教授にこの場でお礼申し上げます．研究の方向性で悩んだ際に親身になって考えていただくなど，大変お世話になりました．また，秘書の元岡みさ子氏，技術専門職員の千葉新吾氏には日々の研究室生活において様々な手助けをいただきました．元助教である古宇田フミ子氏は研究内容に関して多くの疑問や意見をぶつけてくださり，自身の研究を見ていてくれる人がいるという支えになるとともに，研究を進める力となりました．最後に，日頃から研究室で話や相談に乗ってくれた後輩の中村三郎君や長坂明紀を始め，研究室の先輩，B4 の皆様にも感謝いたします．

令和 2 年 1 月 30 日

三島 潤平

参考文献

[1] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds, Inc. Technical Report SWIRLDS-TR-2016*, Vol. 1, pp. 1–28, 2016.

[2] Satoshi Nakamoto, et al. Bitcoin: A peer-to-peer electronic cash system, 2008.

[3] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 3–16, 2016.

[4] Gavin Wood, et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, Vol. 151, No. 2014, pp. 1–32, 2014.

[5] LM Bach, Branko Mihaljevic, and Mario Zagar. Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1545–1550. IEEE, 2018.

[6] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where is current research on blockchain technology? – a systematic review. *PloS one*, Vol. 11, No. 10, pp. 1–27, 2016.

[7] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pp. 203–226. 2019.

[8] Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: A governing council & public hashgraph network. *The trust layer of the internet, whitepaper*, Vol. 1, pp. 1–97, 2018.

[9] Leslie Lamport. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*, pp. 277–317. 2019.

[10] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC '14)*, pp. 305–319, 2014.

[11] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 31–42, 2016.

[12] BitcoinStats. Bitcoinstats. `http://www.bitcoinstats.com/network/propagation/`, January 2019. (Accessed on 01/17/2019).

[13] Kyle Torpey. Bitcoin needs to scale to levels higher than visa, mastercard & paypal. https://coinjournal.net/bitcoin-will-need-to-scale-to-levels-much-higher-than-visa-mastercard-and-paypal-combined/, October 2015. (Accessed on 01/17/2019).

[14] Thaddeus Dryja Joseph Poon. lightning network paper. `https://lightning.network/lightning-network-paper.pdf`, January 2016. (Accessed on 01/17/2019).

[15] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. bloXroute: A Scalable Trustless Blockchain Distribution Network WHITEPAPER. Technical report.

[16] The Zilliqa Team. whitepaper.pdf. `https://docs.zilliqa.com/whitepaper.pdf`, August 2017. (Accessed on 01/17/2019).

[17] Fu Xiang, Wang Huaimin, Shi Peichang, Ouyang Xue, and Zhang Xunhui. Jointgraph: A dag-based

efficient consensus algorithm for consortium blockchains. *Software: Practice and Experience*, pp. 1–13, 2019.

[18] Myongsu Choe. Open hashgraph: An ultimate blockchain engine, 2018.

[19] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 514–532. Springer, 2001.

発表文献

[1] 三島潤平, 相田仁. ハッシュグラフ技術を用いたスケーラブルな価値交換システムの構築とその性能評価. 情報処理学会 第 82 回全国大会, March 2020. 発表予定.