スマート農業における異種データ連携基盤の性能評価

芦田 多香子[1]　Le Hieu Hanh[1]　小口 正人[1]

概要：複数の組織やシステムに分散して存在する多種多様なデータを連携し，高信頼に管理する基盤の重要性が高まっている．このようなデータエコシステムでは，データの改ざん検知や来歴追跡を可能とする検証可能性が重要である．例えば，スマート農業のように環境データや生育情報が継続的に生成・追加される応用では，新規データを追加しながら基盤全体の整合性を維持する必要がある．これまでに，データ間の関係を木構造として表現し，各ノードのハッシュ値をブロックチェーン上で管理することで，データの改ざん検知や改ざん箇所の特定を可能とするデータ連携基盤が提案されている．このような方式では，新たなデータを追加する際，追加データに対応するノードを登録するだけでなく，追加ノードからルートノードまでのハッシュ値を再計算し，ブロックチェーン上の値を更新する必要がある．しかし，既存研究では，主に固定的な木構造や一定分岐・一定深さの木構造を対象としており，実応用で想定されるアンバランスな木構造において，データ追加時の処理時間がどのように変化するかは十分に明らかにされていない．そこで本研究では，Hyperledger Iroha を用いて分散共有基盤を構築しデータ追加時の性能を評価する．特に，一定分岐・一定深さの木構造である一定木と，ノードごとにルートノードまでのパス長が異なる非一定木を比較し，木構造の違いが新規ノード登録処理，ハッシュ値再計算処理，およびハッシュ値更新処理に与える影響を分析した．実験では，5 台の Peer から構成される Hyperledger Iroha ネットワーク上でデータ追加処理時間を計測した．その結果，新規ノード登録処理のコミット時間は木構造によらず約 1 秒で安定した一方，ハッシュ値再計算処理は非一定木で大きく増加し，ばらつきも大きくなった．また，全体実行時間の中央値は約 2.06 秒でほぼ一定であったが，大規模な非一定木ではハッシュ値再計算時間がコミット待ち時間に吸収されないケースが増加し，全体実行時間の平均値および 2.5 秒を超える試行の割合が増加した．本結果より，継続的にデータが追加される分散共有基盤では，追加対象ノードからルートノードまでのパス長を考慮した木構造設計が重要であることを示した．

Performance Evaluation of a Heterogeneous Data Integration Platform for Smart Agriculture

TAKAKO ASHIDA[1]　HIEU HANH LE[1]　MASATO OGUCHI[1]

1. はじめに

1.1 研究背景

社会全体でデータを活用した意思決定が進む中で，複数の組織やシステムに分散して存在する多種多様なデータを連携し，高信頼に管理する基盤の重要性が高まっている．特に，複数主体にまたがるデータを扱うデータエコシステムにおいては，データの改ざん検知や来歴の追跡といった検証可能性が重要な要件となる．

このような背景のもと，データに付随する信頼度や来歴を管理し，任意のデータを検証可能とする「検証可能なデータエコシステム」の研究が進められている [1]．同プロジェクトでは，多様なデータ，データリネージュ，信頼度データ，検証用データを統合的に扱う高信頼データ基盤の実現が目指されている．

本研究では，このような分散共有基盤の応用例としてスマート農業に着目する．スマート農業では，温度や湿度などの環境データや作物の生育情報を収集・分析することで，栽培管理の高度化が期待されている．生育情報や生産履歴を改ざんが困難な分散共有基盤上で管理することにより，農産物のトレーサビリティを確保し産地偽装の防止や消費者に対する信頼性の向上につながる．例えば，小売業者

[1] お茶の水女子大学
Ochanomizu University

が生産者の情報を提示して販売する取り組みにおいても，その情報の真正性を保証する基盤として活用できる．一方で，スマート農業において扱われるデータは，複数の生産者，センサ，分析モデル，外部データベースなどから継続的に生成・追加される．したがって，スマート農業においても多種データを分散的に管理しつつ，データの改ざんを検出できる分散共有基盤が必要である．本研究で想定するスマート農業シナリオでは，生産者，流通業者，小売業者などが分散共有基盤の参加主体となる．将来的には，作物の栽培過程から収穫，出荷，流通，小売に至るまでの情報を複数主体間で共有し，農産物の来歴追跡や改ざん検知を可能にすることを想定している．一方，本稿ではその第一段階として，作物の生育データを対象に，データ追加時の性能評価を行う．

1.2 先行研究と課題

分散共有基盤の実現に向けて，ブロックチェーンを用いたデータ検証機能の研究が行われている．特に，農業分野では複数主体が関与するため，データの透明性や改ざん耐性を確保する技術としてブロックチェーンが注目されている．

Lin ら [2] は，農業分野におけるブロックチェーン応用に向けて農産物のトレーサビリティおよびスマート農業データ管理への利用を示している．また，同研究では農業分野において IoT データなどの大量データを扱う場合，ブロックチェーンのスケーラビリティや既存システムとの統合，プライバシー保護が課題となることが指摘されている．

動的データ更新を考慮したデータ完全性検証の研究として，Chen ら [3] は，スマートホームを対象に，ブロックチェーンを用いた動的かつ追跡可能なデータ完全性検証方式を提案している．同研究では，複数のスマートデバイスから生成されるデータを集約し，Linear Probing Merkle Hash Tree（LP-MHT）と呼ばれるデータ構造を用いることで，データの追加や変更を効率的に扱う方式を示している．この研究は，IoT 環境における動的データ更新と木構造の更新コストに着目している点で本研究と関連が深い．一方で，対象はスマートホームであり，Hyperledger Iroha 上で農業データを木構造として管理した場合のデータ追加処理性能や，木構造の形状が処理時間に与える影響については扱われていない．

Hyperledger Iroha の応用および性能評価に関する研究として，Ntolkeras ら [4] は，英国の畜産管理・取引プラットフォームに Hyperledger Iroha を統合し，その性能を評価している．同研究では，リクエスト数，応答時間，ネットワーク参加者数を指標として評価を行い，Hyperledger Iroha が畜産分野におけるトレーサビリティや取引管理に利用可能であることを示している．この研究は，農業・畜産分野における Hyperledger Iroha の実応用可能性を示す点で重要である．

堀ら [5] は，自動車部品製造業におけるカーボンフットプリント (CFP) 管理を対象として，ブロックチェーンベースのデータ連携基盤を構築し，データの改ざん検知機能を実装・評価している．ハッシュ木の階層構造と XOR 演算の性質に着目したハッシュ部品木が提案され，改ざんの発生箇所を特定可能なデータ連携基盤が検討されている．この手法では，ハッシュ部品木の生成・更新・検証プロセスを実装し検証プロセスにおいて改ざん特定率 100%を達成したことが報告されている．複数企業を横断する分散データベースシステムにおいて，トレーサビリティを表現し CFP データを集約・保管することの重要性が述べられている．一方で，同研究は主に既存のデータ構造に対する改ざん検知および改ざん箇所特定を対象としており，新たなデータが継続的に追加される場合の更新処理性能については十分に評価されていない．

我々はこれまでに，スマート農業において継続的に生成される動的環境データを対象として，Hyperledger Iroha を用いた分散共有基盤を構築し，既存のハッシュ木構造を農業向けに拡張したデータ追加手法を提案・評価した [6]．同研究では，新規ノード登録処理を実現するために `CommitData`，および `UpdateValue` を組み込みコマンドとして開発し，ハッシュ値再計算処理を `Recompute` として定義した．さらに，実行時間を評価した結果，コミット待ち時間が動的データ追加処理における主要なオーバーヘッドとなること，また `Recompute` は Peer 数の影響を受けにくい一方，`CommitData` および `UpdateValue` は Peer 数の増加に伴って実行時間が増加することを示した．

先行研究により，ブロックチェーンプラットフォームを用いることで，安全性および信頼性を備えた分散共有基盤の構築が可能であることが示されている．しかし，これまでの評価では，一定分岐・一定深さの木構造を中心に検証しており，実応用で想定されるようなアンバランスな木構造に対するデータ追加時の処理時間については十分に明らかにされていない．

1.3 本論文の目的

以上を踏まえ，本論文では，多種データを管理する分散共有基盤において，木構造の違いがデータ追加処理性能に与える影響を明らかにすることを目的とする．実際のデータ連携基盤では，データの種類や生成頻度が一様であるとは限らず，追加対象ノードからルートノードまでのパス長がノードごとに異なる非一定木となる可能性がある．このような非一定木では，ハッシュ値再計算処理の時間やコミット待ち時間との関係が一定木とは異なり，全体実行時間の増加やばらつきにつながる可能性がある．

そこで本論文では，Hyperledger Iroha を用いた分散共有基盤を対象に，一定木と非一定木に対するデータ追加処理

時間を比較する．具体的には，既存ノード数を段階的に増加させた木構造に対して葉ノードを追加し，`CommitData`，`Recompute`，および `UpdateValue` の実行時間を計測する．これにより，木構造の偏りや規模の増加が，ハッシュ値再計算時間，コミット待ち時間，全体実行時間に与える影響を明らかにする．

1.4 本稿の構成

本稿の構成は以下の通りである．第 2 節では，本研究で用いるブロックチェーン，Hyperledger Iroha，およびコンセンサスアルゴリズムについて述べる．第 3 節では，本研究で対象とする異種データ連携基盤システムの構成とデータ追加処理について説明する．第 4 節では，実験環境，実験対象の木構造，計測項目，および実験結果を示す．第 5 節では，本稿のまとめと今後の課題について述べる．

2. 関連知識

本節では，本研究の基盤技術であるブロックチェーンの概要について説明したのち，Hyperledger Iroha の設計思想および特徴について述べる．最後に，Hyperledger Iroha で採用されているコンセンサスアルゴリズムを紹介する．

2.1 ブロックチェーン

ブロックチェーンは，2008 年に Satoshi Nakamoto により投稿された論文 [7] に基づき，暗号通貨 Bitcoin[8] の公開取引台帳としての利用を目的に提案された分散台帳技術である．ブロックチェーンでは，取引履歴をトランザクションとして記録し複数のトランザクションをまとめたブロック単位で管理する．各ブロックは暗号学的ハッシュ関数を用いて鎖状に連結され，過去の履歴を一貫して保持する構造を持つ．各ブロックには，直前のブロックのハッシュ値と新規トランザクションの内容が含まれておりこれらを基に新たなハッシュ値が算出される．この構造により，過去のブロック内容が改ざんされた場合には後続するすべてのブロックとの整合性が失われるため，履歴の改ざんを容易に検知することが可能となる．また，各ブロックやトランザクションには電子署名が付与されており送信者の正当性およびデータの完全性が保証される．さらに，ブロックチェーンは単一の管理者によって保持されるのではなく，複数の Peer に分散して保持される点が特徴である．この分散構成により，一部の Peer に障害や不正が発生した場合であっても他の Peer が保持する台帳との照合を通じて不整合を検出でき，高い耐障害性と可用性を実現している．このような特性からブロックチェーンは信頼性が要求されるデータ管理基盤として金融分野のみならず，サプライチェーン管理や IoT 分野など幅広い応用が検討されている．

一部のブロックチェーンでは，スマートコントラクトが実装されている．スマートコントラクトとは，あらかじめ設定されたルールに従ってブロックチェーン上のトランザクション，もしくはブロックチェーン外から取り込まれた情報をトリガーにして実行されるプログラムのことを指す．スマートコントラクトの情報はブロックチェーン上に記録される．

2.2 Hyperledger Iroha

Hyperledger Iroha[9]（以下，Iroha）は，Hyperledger プロジェクトの一つとして開発されている，次世代のパーミッション型ブロックチェーンプラットフォームである．Iroha は，参加ノードが明確に定義された環境での利用を前提として設計されており，企業や組織間でのデータ共有に適した構成を持つ．このため，非公開データや業務データを安全に管理・共有することが可能である．Iroha の大きな特徴の一つとして，スマートコントラクトが組み込みコマンドとして実装されている点が挙げられる．これにより，汎用的なスマートコントラクトを記述する必要がなく，あらかじめ定義されたコマンドを用いることで軽量かつ高速なトランザクション処理が可能となっている．さらに，Iroha ではコンセンサスアルゴリズムとして Yet Another Consensus を採用している．コンセンサスアルゴリズムについては次節で詳細を説明する．以上の特徴から，Iroha は動的データを扱う分散共有基盤として有効であると考えられる．本研究では，スマート農業における環境データのように頻繁に更新されるデータを対象とするため，軽量なトランザクション処理と高い信頼性を両立できる Iroha を採用している．

2.3 コンセンサスアルゴリズム

Iroha では，Yet Another Consensus（以下，YAC）と呼ばれるコンセンサスアルゴリズムが用いられる．Muratov ら [10] は，YAC をブロックチェーン向けの実用的な BFT コンセンサスアルゴリズムとして提案している．YAC は，従来の BFT 系コンセンサスアルゴリズムにおける非効率なメッセージ交換や，強いリーダへの依存を軽減することを目的としている．また，Iroha において BFT コンセンサスを提供するために利用されているアルゴリズムであると説明されている．YAC における合意形成では，Peer は提案されたブロックそのものではなく，ブロックのハッシュ値に対して投票を行う．各 Peer は提案内容を検証し，正当であると判断した場合に投票を送信する．一定数以上の投票が集まると合意が成立し，対象のブロックがコミットされる．このように，YAC ではブロックハッシュに対する投票により Peer 間で合意を形成し，台帳状態を一貫させる．

YAC を利用したブロックチェーン応用の例として，Hil ら [11] は Social Media Marketing を対象に，CryptoNight mining algorithm と YAC コンセンサスアルゴリズムを組

表 1 CFT と BFT における障害耐性条件

障害モデル	想定する障害	必要ノード数	合意に必要な票数
CFT	クラッシュ障害	$n \geq 2f+1$	過半数
BFT	ビザンチン障害	$n \geq 3f+1$	$2f+1$ 以上

み合わせたブロックチェーンフレームワークを提案している．この研究では，YAC が社会的データの信頼性や改ざん検出を支えるコンセンサス機構として利用されており，YAC が Iroha 以外でも参照されていることがわかる．

コンセンサスアルゴリズムは，想定する障害モデルに応じて Crash Fault Tolerant（以下，CFT）と Byzantine Fault Tolerant（以下，BFT）に分類される．CFT は，Peer の停止や応答不能といったクラッシュ障害を想定する方式である．一方，BFT は，Peer が任意の不正な振る舞いを行うビザンチン障害を想定する方式であり，CFT よりも強い障害耐性を持つ一方で，通信量や処理のオーバーヘッドが大きくなる．

CFT と BFT における障害耐性条件を表 1 に示す．ここで，n はネットワークに参加する全ノード数，f は許容可能な障害ノード数を表す．CFT では，最大 f 台のクラッシュ障害に耐えるために少なくとも $n \geq 2f+1$ 台のノードが必要である．一方，BFT では，最大 f 台のビザンチン障害に耐えるために少なくとも $n \geq 3f+1$ 台のノードが必要である．

本研究では，Hyperledger Iroha を用いて 5 台の Peer から構成されるブロックチェーンネットワークを構築した．本研究の目的は木構造の違いが提案手法の更新処理性能に与える影響を評価することであり，悪意ある Peer によるビザンチン障害への耐性評価は対象外である．そのため，本研究の実験環境では CFT 構成を採用した．CFT 構成では，Peer の停止や応答遅延といったクラッシュ障害を主な対象とし，悪意ある Peer が矛盾した情報を送信する状況は想定しない．

以上の技術を用いて，本研究ではスマート農業における多種データを検証可能に管理するシステムを構築する．

3. 異種データ連携基盤システム

本節では，本研究で対象とするスマート農業向けデータ管理基盤の概要について述べる．本研究では，スマート農業において継続的に生成される環境データや生育情報を対象とし，データ間の関係を木構造として表現する．例えば，根に相当するノードを栽培ロット，その下位ノードを圃場や栽培工程，さらにその下位ノードをセンサ，分析結果，環境データや生育データとして表現することを想定する．さらに，木構造に対応するハッシュ集約木を構成し各ノードのハッシュ値を Iroha 上で管理することで，データの正当性を検証可能な形で保持する．

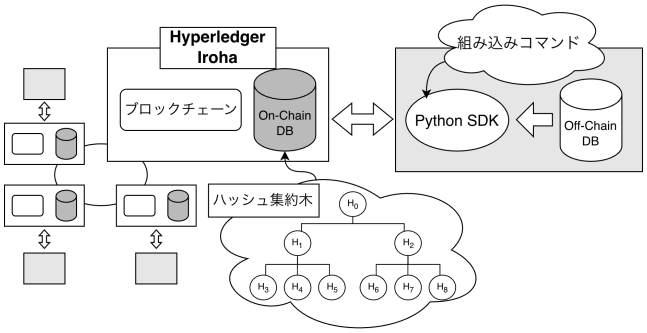

図 1 システム構成図

3.1 システム概要

本研究のシステムの構成図を図 1 に示す．本研究で想定するスマート農業シナリオでは，Peer は分散共有基盤に参加する各組織に対応する．本稿では生産者がそれぞれ Peer を運用し，同一の台帳状態を保持することを想定している．Iroha において On-chain DB とはブロックチェーンで合意されたスナップショットを保持する DB であり，Off-chain DB とは各参加者がデータをローカルに管理する DB である．本研究では環境データを Off-chain DB に保持し，その正当性を検証するためのハッシュ構造のみを On-chain DB に記録することで，処理性能と検証可能性の両立を実現している．あるデータが改ざんされた場合，そのデータに対応するハッシュ値が変化しさらに親ノードからルートノードまでのハッシュ値にも影響が伝播する．そのため，ブロックチェーン上に保存されたハッシュ値と再計算したハッシュ値を比較することで，データの改ざんを検出できる．

3.2 データ追加処理

新たなデータが木構造に追加されると追加ノード自身のハッシュ値を On-chain DB に登録するだけでなく，追加ノードの親ノードからルートノードまでのハッシュ値を更新する必要がある．

本研究用に開発した組み込みコマンドを用いたデータ追加処理の流れを図 2 に示す．本研究におけるデータ追加処理は，以下の 3 つの処理から構成される．

（1） 新規ノード登録処理:`CommitData`

（2） ハッシュ値再計算処理:`Recompute`

（3） ハッシュ値更新処理:`UpdateValue`

`CommitData` では，追加されるデータに対応するノード自身のハッシュ値を算出し，On-chain DB に新規ノード情報として登録する．`Recompute` では，追加ノードからルートノードまでの経路を取得し，その経路上に存在する各ノードのハッシュ値を再計算する．`Recompute` は Application 側で実行されるローカル処理であり，Iroha Network における合意形成を伴わない．`UpdateValue` では，`Recompute`

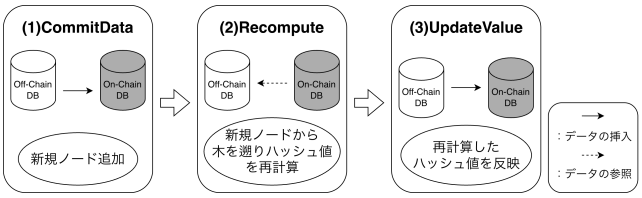

図 2 データベース更新の流れ

コンテナ	Hyperledger Iroha Ver 1.	On / Off-Chain DB
	ubuntu22.04	PostgreSQL
仮想環境	Docker	
OS	Ubuntu20.04LTS	
物理サーバー	CPU：Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz	

図 3 実験環境

によって得られた複数ノードのハッシュ値を On-chain DB に反映する．この処理も Iroha のトランザクションとして実行され，各 Peer において検証された後，コンセンサスアルゴリズムによってコミットされる．

4. 実験

異種データ連携基盤システムにおけるデータ追加処理の性能を評価する．特に，木構造の違いがデータ追加時の実行時間に与える影響を明らかにするため，一定分岐・一定深さの木構造と，実応用に近いアンバランスな木構造を用いて実験を行う．

4.1 実験概要

本実験では，図 2 に示したデータ追加処理を対象として，各処理の実行時間を計測する．本研究では，木構造の違いがデータ追加処理に与える影響を評価するため，以下の 2 種類の木構造を用いる．

（1） 一定分岐・一定深さの木構造: 一定木

（2） アンバランスな木構造: 非一定木

一定木では，葉ノードを除く各内部ノードが同じ分岐数を持ち追加対象となる葉ノードの深さが一定となる．一方，非一定木では，ノードごとに子ノード数やルートまでのパス長が異なる．実際のスマート農業では，データの種類や生成頻度が一様ではないため，アンバランスなデータ構造が発生すると考えられる．そこで本実験では，両者を比較することで，木構造の偏りがデータ追加処理時間に与える影響を明らかにする．

実験環境の構成を図 3 に示す．実験は仮想化環境として Docker を用い，Ubuntu 20.04 LTS コンテナ上に Iroha の動作環境を構築して実施した．また，本実験では単一サーバ上に複数の Peer コンテナを構築し，コンセンサスに参加する Peer とは別にトランザクション送信および計測を行うクライアント用コンテナを用意した．本実験では，5 台の Peer から構成される Iroha ネットワークを用い，CFT 構成で実験を行う．本研究では悪意ある Peer によるビザンチン障害への耐性評価は対象外とし，Peer の停止や応答遅延といったクラッシュ障害を想定する．

4.2 実験対象の木構造

本実験では，あらかじめ葉ノード以外の既存ノードを On-chain DB に登録しておき測定時には葉ノードを 1 つずつ追加する．そのため，以下で示すノード数は登録済みである葉ノード以外の既存ノード数を表す．

一定木では，分岐数を $B = 5$ に固定し，深さ D を変化させた．本研究では，$D = 4, 5, 6, 7$ の 4 種類を用いる．D4B5，D5B5，D6B5，D7B5 における既存ノード数は，それぞれ 156，781，3906，19531 となる．また，追加対象となる葉ノード数はそれぞれ 125，625，3125，15625 である．測定では，これらの既存ノードに対して，追加対象となる葉ノードを 1 つずつ追加し各追加処理に要する時間を計測した．一方，非一定木では，一定木と既存ノード数が同程度になるように既存ノード数を 156，781，3906，19531 に設定し，木構造を生成した．非一定木における追加対象葉ノード数はそれぞれ 98，556，2794，13888 である．非一定木では，各ノードの子ノード数が一定ではなく追加対象ノードからルートノードまでのパス長がノードごとに異なる．なお，本実験では既存ノード数が同程度となるように一定木と非一定木を比較したが，追加対象葉ノード数および総トランザクション数は完全には一致していない．そのため，本稿では各追加処理 1 回あたりの実行時間を中心に比較し，総処理時間ではなく平均値，中央値，標準偏差，および total time が 2.5 秒を超える試行の割合に着目して評価する．ここで，2.5 秒は通常時の total time の中央値が約 2.06 秒であることを踏まえ，典型的な処理時間から明確に増加した試行を抽出するための閾値として設定した．

4.3 計測項目

本実験では，5 台の Peer のうち 3 台でコミットが確認された時点を 3/5 commit，5 台すべてでコミットが確認された時点を 5/5 commit と定義する．3/5 commit は，過半数の Peer においてコミットが確認された状態であり，合意成立までの代表的な時間として扱う．一方，5/5 commit は，全 Peer にコミットが反映されるまでの時間として扱う．5/5 commit は最後にコミットする Peer の遅延の影響を受けるため，本研究では 3/5 commit と 5/5 commit を区別して評価する．

本実験では，新規ノード追加時の各処理について以下の時間を計測した．

- `CommitData` の 3/5 commit 時間
- `CommitData` の 5/5 commit 時間

表 2　一定木および非一定木における実行時間の比較 [s]

木構造	構成	既存ノード数	Commit3/5	Commit5/5	Recompute	Update3/5	Update5/5	total5/5 平均	total5/5 中央値
一定木	D4B5	156	1.000	1.081	0.330	0.676	0.759	2.213	2.064
	D5B5	781	1.005	1.033	0.406	0.601	0.631	2.113	2.064
	D6B5	3906	1.004	1.082	0.480	0.527	0.614	2.219	2.065
	D7B5	19531	1.005	1.052	0.552	0.456	0.507	2.154	2.065
非一定木	N=156	156	1.001	1.007	0.502	0.506	0.513	2.066	2.064
	N=781	781	1.004	1.048	0.590	0.465	0.511	2.192	2.065
	N=3906	3906	1.005	1.059	0.763	0.395	0.452	2.317	2.065
	N=19531	19531	1.006	1.018	0.906	0.478	0.492	2.459	2.069

- Recompute の実行時間
- UpdateValue の 3/5 commit 時間
- UpdateValue の 5/5 commit 時間
- 全体の実行時間：total time

本稿で示す total time は，CommitData の送信開始から，Recompute を経て，UpdateValue の 5/5 commit 確認が完了するまでの時間である．したがって，total time はデータ追加処理全体が全 Peer に反映されるまでの合計時間を表す．一方，3/5 commit 時間は，各トランザクションにおいて過半数の Peer でコミットが確認されるまでの時間として個別に評価する．ここで，CommitData および UpdateValue は Iroha のトランザクションとして実行されるため，各 Peer におけるコミット確認までの待ち時間が含まれる．一方，Recompute は Application 側で実行されるハッシュ値再計算処理であり，Iroha におけるトランザクション発行およびコミット処理を伴わない．また，各ノードについて平均値，中央値，標準偏差，および total time が 2.5 秒を超える割合を算出し，木構造および規模の違いが実行時間に与える影響を評価した．

4.4　実験結果

表 2 に，各木構造における実行時間の集計結果を示す．表中の Commit は新規ノード登録処理である CommitData を表し，Update はハッシュ値更新処理である UpdateValue を表す．本節では，まずデータ追加処理全体の実行時間である total time の傾向を述べた後，その内訳として CommitData，Recompute，および UpdateValue の結果を分析する．

4.4.1　全体の実行時間について

まず，データ追加処理全体の実行時間である total time について述べる．表 2 より，total time の中央値はすべての条件で約 2.06 秒となった．この結果から，典型的なデータ追加処理時間は，木構造の規模や形状によらず約 2 秒で安定していることが分かる．一方，平均値に着目すると非一定木では規模の増加に伴い total time が増加した．特に非一定木 N=19531 では，D7B5 の平均 2.154 秒と比較して約 0.305 秒，割合にして約 14.2%大きくなった．これは，非一定木では Recompute が長くなるノードが増加し，コミット待ち時間による吸収が十分に働かない試行が増えたためであると考えられる．

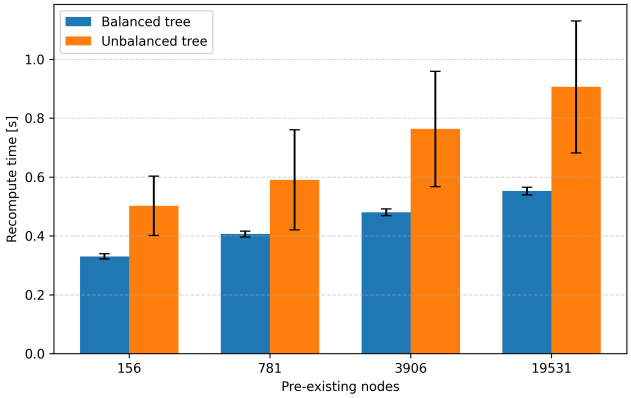

図 4　Recompute の平均実行時間と標準偏差

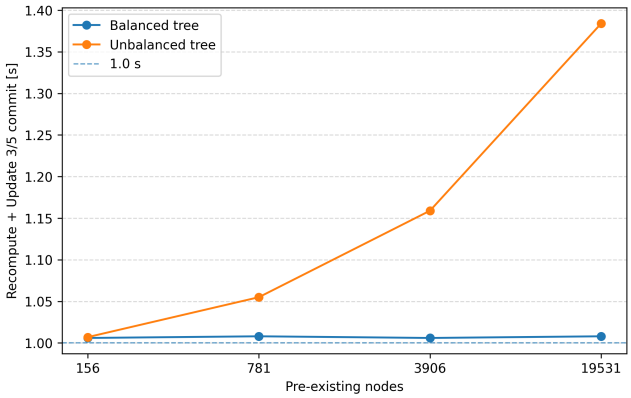

図 5　Recompute 時間と UpdateValue の 3/5 commit 時間の和

4.4.2　CommitData の結果について

表 2 より，CommitData の 3/5 commit 時間は，一定木・非一定木のいずれの条件においても約 1.0 秒であった．この結果から，CommitData の 3/5 commit 時間は，木構造の規模や形状にはほとんど依存しないことが分かる．これは，CommitData が新規追加ノード自身の情報を登録する処理であり，追加対象ノードからルートノードまでのパス長や木構造全体の大きさに直接依存しないためであると考えられる．一方で，5/5 commit 時間では標準偏差や最大値が大きくなる傾向が確認された．これは，5 台すべての Peer でのコミット確認を待つため，最後にコミットする Peer の遅延の影響を受けるためである．

表 3　total time が各閾値を超えた試行の割合

木構造	構成	> 2.5 秒	> 3.0 秒	> 4.0 秒
一定木	D4B5	25/125 (20.0%)	0/125 (0%)	0/125 (0%)
	D5B5	36/625 (5.7%)	4/625 (0.64%)	0/625 (0%)
	D6B5	268/3125 (8.6%)	268/3125 (8.6%)	212/3125 (6.8%)
	D7B5	757/15625 (4.84%)	757/15625 (4.84%)	630/15625 (4.03%)
非一定木	N=156	0/98 (0%)	0/98 (0%)	0/98 (0%)
	N=781	88/556 (15.8%)	25/556 (4.5%)	0/556 (0%)
	N=3906	571/2794 (20.4%)	571/2794 (20.4%)	131/2794 (4.7%)
	N=19531	**5318/13888 (38.3%)**	**5318/13888 (38.3%)**	119/13888 (0.86%)

4.4.3　Recompute の結果について

一定木および非一定木における Recompute の平均実行時間と標準偏差を図 4 に示す．一定木では，既存ノード数の増加に伴い Recompute の平均実行時間が増加した．D が 1 増加するごとに実行時間は約 0.07 秒から 0.08 秒増加しており，深さに対してほぼ線形に増加していることが分かる．一方で，標準偏差は小さく，一定木では処理時間が安定していることが読み取れる．これは，一定木では追加対象となる葉ノードからルートノードまでのパス長が一定であり，各試行で再計算対象となるノード数がほぼ同じであるためと考えられる．

一方，非一定木では，Recompute の平均実行時間およびばらつきが一定木よりも大きくなった．同規模の一定木と比較すると，非一定木の Recompute は約 1.5 倍から 1.6 倍の時間を要した．また，非一定木の標準偏差は一定木と比較して大きい．これは，非一定木では各ノードの子ノード数やルートノードまでのパス長が一定ではなく，追加対象ノードごとに再計算対象となるノード数が異なるためである．特に N=19531 の非一定木では，Recompute の最大値が 1.859 秒となり，1 秒を超える試行が全体の 34.7%確認された．この結果から，非一定木では規模が大きくなるにつれて，パス長のばらつきが実行時間の増加および不安定化として顕在化することが分かる．

4.4.4　UpdateValue の結果について

一定木では，木の規模が大きくなるにつれて，UpdateValue のコミット待ち時間は短くなる傾向が見られた．これは，UpdateValue の処理自体が軽くなったことを意味するのではなく，直前に実行される Recompute の実行時間が増加したことで UpdateValue トランザクションを送信する時刻が次の proposal 作成タイミングに近づいたためであると考えられる．本実験環境では，Iroha の proposal_creation_timeout を 1000ms に設定している．これは，本研究で用いた Iroha 環境における標準的な設定値であり既存の実験条件と比較可能にするためである．そのため，トランザクションのコミット待ち時間には約 1 秒周期の影響が現れる．図 5 より，一定木では既存ノード数が増加しても和が約 1.0 秒で安定している一方，非一定木では規模の増加に伴って和が大きくなることが分かる．このことから，一定木では Recompute の増加分が UpdateValue のコミット待ち時間の短縮によって吸収され，合計時間が約 1 秒に保たれていると考えられる．一方で，非一定木では Recompute が長くなり proposal 作成周期内に処理が収まりきらないケースが増加したためであると考えられる．

4.4.5　処理時間の詳細分析

さらに，total time が 2.5 秒，3.0 秒，4.0 秒を超える試行を集計した．表 3 に，total time が各閾値を超えた試行の割合を示す．一定木では，D7B5 のように木の規模を 19531 ノードまで大きくした場合でも，total time が 2.5 秒を超える割合は 4.84%であった．一方，非一定木では，規模の増加に伴い total time が 2.5 秒を超える試行の割合が増加した．N=156 では 2.5 秒を超える試行は確認されなかったが，N=781 では 15.8%，N=3906 では 20.4%，N=19531 では 38.3%が 2.5 秒を超えた．特に N=19531 では，3.0 秒を超える試行も 38.3%であり，2.5 秒を超えた試行のほとんどが 3.0 秒以上まで増加していた．これは，コミット待ち時間が約 1 秒周期で増加するためであると考えられる．すなわち，通常時の total time は約 2.06 秒であるが，トランザクションが次の proposal 作成タイミングに間に合わない場合，さらに約 1 秒待機するため total time は約 3 秒以上となる．このことから，非一定木では規模が大きくなるにつれて，Recompute の長いノードが増加し，コミット周期を逃す試行が多くなると考えられる．

2.5 秒を超える要因を明らかにするため，2.5 秒以下であった試行と 2.5 秒を超えた試行における各処理時間を比較した．一定木 D7B5 では，total time が 2.5 秒以下であった試行における Recompute の平均が 0.552 秒であるのに対し，2.5 秒を超えた試行における Recompute の平均は 0.554 秒であり，ほとんど差がなかった．一方で，2.5 秒を超える試行では CommitData の 5/5 commit 時間および UpdateValue の 5/5 commit 時間が大きく増加していた．したがって，処理時間が増加した試行は木構造由来の再計算コストではなく，主に 5/5 commit における最後の Peer の遅延によって発生していると考えられる．一方，非一定木 N=19531 では，total time が 2.5 秒以下であった試行における Recompute の平均が 0.768 秒であったのに対し，2.5 秒を超えた試行では 1.127 秒まで増加してい

た．また，UpdateValue の 3/5 commit 時間も 2.5 秒を超える試行では大きく増加していた．このことから，非一定木において処理時間が増加した試行は，Recompute の増加と UpdateValue のコミット遅延の両方に起因すると考えられる．特に，大規模な非一定木では追加対象ノードからルートノードまでのパス長が長いノードが存在し，そのようなノードではハッシュ値再計算処理が長くなる．その結果，UpdateValue トランザクションが次の proposal 作成タイミングに間に合わず，total time が約 1 秒単位で増加したと考えられる．

5. まとめと今後の課題

本稿では，継続的に生成される動的データを対象として，Hyperledger Iroha を用いた分散共有基盤におけるデータ追加処理の性能評価を行った．特に，データ追加時に必要となる新規ノード登録処理，追加ノードからルートノードまでのハッシュ値再計算処理，および再計算結果の更新処理に着目し，木構造の違いが実行時間に与える影響を評価した．実験では，一定分岐・一定深さの木構造である一定木と，ノードごとに子ノード数やルートまでのパス長が異なる非一定木を用意し，既存ノード数を 156，781，3906，19531 と変化させて比較した．その結果，新規ノード登録処理である CommitData の 3/5 commit 時間は，木構造やノード数によらず約 1 秒で安定しており，木構造の違いによる影響はほとんど見られなかった．一方で，ハッシュ値再計算処理である Recompute には木構造の影響が明確に表れた．一定木では，深さの増加に伴って Recompute の実行時間がほぼ線形に増加したが，標準偏差は小さく処理時間は安定していた．これに対し，非一定木では，同規模の一定木と比較して Recompute の平均実行時間が大きくばらつきも大きくなった．特に，追加対象ノードからルートノードまでのパス長の違いが再計算時間に大きく影響することが分かった．また，データ追加処理全体の実行時間については，中央値がいずれの条件でも約 2.06 秒となった．これは，本実験環境において Iroha の proposal_creation_timeout を 1000 ms に設定しており，CommitData のコミット待ち時間と，Recompute 後の UpdateValue のコミット待ち時間がそれぞれ約 1 秒周期の影響を受けるためであると考えられる．ただし，大規模な非一定木では，Recompute の増加がコミット待ち時間に吸収されないケースが増加し，total time の平均値および 2.5 秒を超える試行の割合が大きくなった．特に，既存ノード数 19531 の非一定木では，total time が 3 秒を超える試行が全体の約 38%に達し，大規模な非一定木では木構造由来の再計算コストが全体実行時間にも影響することが確認された．

以上の結果から，スマート農業のように多種データが継続的に追加される応用においては，木全体の規模だけでなく，追加対象ノードからルートノードまでのパス長を考慮したデータ構造設計が重要であることが分かった．特に，非一定木ではパス長のばらつきが Recompute の実行時間および全体実行時間が 2.5 秒を超える試行の割合に影響するため，実運用においては木構造が過度に深くならないように管理する必要がある．また，コミット待ち時間が全体実行時間に大きく影響するため，トランザクションの送信回数や proposal_creation_timeout の最適化も重要である．

今後の課題として，実運用に近い環境での評価があげられる．本実験では Docker 上に構築した Iroha を用いて評価を行ったが，実際のスマート農業環境では，複数の Peer が物理的に離れた場所に配置され，ネットワーク遅延や通信の不安定性が発生する可能性がある．そのため，複数の物理サーバ上に Peer を配置した環境で評価を行い，ネットワーク遅延や Peer 数の増加がデータ追加処理に与える影響を検証する必要がある．このような評価を通じて，動的データ処理におけるボトルネックをより正確に特定し，さらなる性能改善につなげることが期待される．

参考文献

[1] 検証可能なデータエコシステム. https://www.kde.cs.tsukuba.ac.jp/crest/.

[2] Weijun Lin, Xinghong Huang, Hui Fang, Victoria Wang, Yining Hua, Jingjie Wang, Haining Yin, Dewei Yi, and Laihung Yau. Blockchain technology in current agricultural systems: From techniques to applications. *IEEE Access*, 8:143920–143937, 2020.

[3] Chunliang Chen, Liangliang Wang, Yu Long, Yiyuan Luo, and Kefei Chen. A blockchain-based dynamic and traceable data integrity verification scheme for smart homes. *Journal of Systems Architecture*, 130:102677, 2022.

[4] Konstantinos Ntolkeras, Hossein Sharif, Soroush Dehghan Salmasi, and William Knottenbelt. Performance analysis of a hyperledger iroha blockchain framework used in the uk livestock industry. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 456–461, 2021.

[5] 堀遥, Le Hieu Hanh, and 小口正人. 改ざんが特定可能なサプライチェーン全体でのデータ連携基盤の提案. In **マルチメディア，分散，協調とモバイル** *(DICOMO2025)* **シンポジウム**, pages 1F–2, 母畑温泉, June 2025.

[6] 芦田多香子 and 小口正人. スマート農業の動的環境データを対象とした分散共有基盤の構築と性能評価. In **第** *18* **回データ工学と情報マネジメントに関する**

フォーラム *(DEIM2026)*, pages 1D–03, 神戸国際会議場, 2026.

[7] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[8] Bitcoin. https://bitcoin.org/ja/.

[9] Hyperledger iroha blockchain for enterprise. https://soramitsu.co.jp/iroha.

[10] Fedor Muratov, Andrei Lebedev, Nikolai Iushkevich, Bulat Nasrulin, and Makoto Takemiya. Yac: Bft consensus algorithm for blockchain, 2018.

[11] Anwer Mustafa Hil, Fahd N. Al-Wesabi, Hadeel Alsolai, Ola Abdelgney Omer Ali, Nadhem Nemri, Manar Ahmed Hamza, Abu Sarwar Zamani, and Mohammed Rizwanullah. Cryptonight mining algorithm with yac consensus for social media marketing using blockchain. *Computers, Materials and Continua*, 71(2):3921–3936, 2021.