令和3年度 修士論文

デジタルツインを目指すデータ共有システムに関する検討

Proposal of Data Sharing System for Digital Twin

指導教授 中里秀則 教授

2022年1月24日

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻
分散コンピューティングシステム研究

5120F045-2

高橋　圭介

Keisuke Takahashi

目 次

第1章　序論

1.1　背景

近年、人々の Quality of Life(QoL) や企業や自治体の Quality of Service(QoS) 向上に向け、現実空間をリアルタイムで仮想空間に再現するデジタルツインを利用した取り組みが注目を浴びている。例えば、東京都では、デジタルツインにより、少子高齢化・人口減少、人流・物流の変化、気候変動の危機、地震に備え、人々の Quality of Life(QoL) を向上させるデジタルツイン実現プロジェクトを行なっている [42]。このようなデジタルツインを実現するためには Internet of Things (IoT)、5G/Beyond 5G、セキュリティ技術、参加型センシングといった様々な Information and Communication Technology (ICT) 技術が必要となる。その中でも、本稿では特に IoT プラットフォームとブロックチェーン技術に注目した。

まず、IoT プラットフォームの概要を説明する。IoT プラットフォームとは、大量かつ高頻度に IoT デバイスから生成される IoT データを収集・管理し、IoT サービスの構築を支える基盤となる技術である。IoT プラットフォームを実現する機能として、ネットワークの接続性、IoT デバイスの管理、IoT デバイスの制御、IoT データの蓄積、IoT データの処理といった 5 つの機能が挙げられる。これらの機能によって、収集・蓄積される IoT データを可視化および分析し、IoT デバイスを制御する IoT システムを構築することができる [29]。

よって、上記の記載されている機能を持った IoT プラットフォームによるデジタルツインの実現が期待されている。IoT プラットフォームの実装例を内閣府は都市 OS [40] の参照アーキテクチャにて報告している。都市 OS とはスマートシティ実現に向け、都市に存在する膨大なデータを蓄積・分析し、他の自治体や企業、研究機関などが分野をまたいで連携する仕組みのことである [23]。都市 OS の参照アーキテクチャ[40] では、スマートシティにおける IoT サービスの運用は、IoT サービス間のデータ連携、スマートシティをまたいだスマートシティ間の連携のように、水平方向に連携することが求められていると議論している。そのため、今後は IoT サービス間のデータ連携、スマートシティをまたいだスマートシティ間を連携することができる IoT プラットフォームが必要となる。

また、デジタルツインの実現で期待されているブロックチェーン技術は Satoshi Nakamoto によって Bitcoin [20] と共に開発され、トランザクションを複数のピアで構成される分散型ネットワークで管理する技術である。各ピアで台帳を共有しており、各ピアでトランザクションの要求に対してコンセンサスを行うことによって、トランザクションを検証する。検証されたトランザクションはブロックとして、ブロックを一つなぎに追加していく。このブロックにはトランザクションの他に、その直前のブロックのハッシュ値が保存されている。そのため、ブロックチェーンはデータを追跡し、改ざんすることは困難となっており、セキリュティを担保した台帳技術となっている [22]。

IoT プラットフォームで扱われる IoT データが悪意ある攻撃者によって改ざんされてしまうと IoT サービス提供者が想定したサービスが適切に提供されることが困難になる可能性がある。また、IoT プラットフォームに対して不正にアクセスされ IoT データが盗難される可能性もある。

そこで、よりセキュリティを考慮した仕組みとして、ブロックチェーン技術の適用が期待されている。しかし、ブロックチェーンは、分散台帳からデータを参照したり、台帳にデータを書き込むといった取引情報に対する検証、承認するプロセスにおいて、処理の負荷が大きい。よって、IoT データのような大容量のデータや高頻度のトランザクションを扱う際に、遅延の大きさやスループットなどのスケーラビリティに問題がある。

1.2 研究目的および研究の概要

本稿の研究の目的は二つある。一つは、デジタルツイン実現に必要な、IoT サービス間のデータ連携、スマートシティをまたいだスマートシティ間の連携することができる IoT プラットフォームについて、その利便性を向上するためのプロトコル、認証サービスを提案することである。この利便性向上の対象とする IoT プラットフォームは、早稲田大学が携わっている、スマートシティにアプリケーションに拡張性と相互運用性をもたらす仮想 IoT・クラウド連携基盤の研究開発 (Fed4IoT)[38] が提案している仮想 IoT システム (VirIoT)[12] である。Fed4IoT は、異なるスマートシティ同士、異なる IoT プラットフォーム同士を水平方向に連携する IoT プラットフォームを開発し、各 IoT プラットフォームに対して拡張性と相互運用性を実現することで、スマートシティにおけるアプリケーションの開発、展開を容易にすることを目的とした研究開発プロジェクトである。従って、Fed4IoT プロジェクトでは、研究背景で述べた IoT サービス間のデータ連携、スマートシティをまたいだスマートシティ間を連携することができる IoT プラットフォームに対する取り組みをしている。Fed4IoT が提案する VirIoT は IoT データの蓄積や IoT データに対する処理といった機能を提供する。VirIoT の利便性を向上するために、ThingVisor Factory[33]、[17] が提案されている。ThingVisor Factory は、IoT サービスの設計を簡易化するツールである。VirIoT および ThingVisor Factory の具体的な機能は 2 章にて紹介する。ThingVisor Factory は前もって登録された IoT データに対してのみ機能するため、登録される IoT データをどのように効率的に収集・管理するかは課題のままである。そこで本稿では、、VirIoT や ThingVisor Factory に登録される IoT データを収集・管理するプロトコル、登録時の認証サービスについて紹介する。

、

二つ目の研究目的は、ブロックチェーンをデジタルツイン実現に活用する上で課題となるスケーラビリティの問題を明らかにするため、ブロックチェーンの性能評価を行うことである。本稿では、以下の二つのユースケースを想定し、ブロックチェーン技術の実装例である、Hyperledger Fabric を用いてデータ共有システムを構築し、データ共有システムの性能評価を行なった。

- 人の属性情報や IoT データをデータ共有システムへ登録するユースケース
- データ共有システムに登録されている IoT データを活用するユースケース

また、これらの評価結果から、データ共有システムを用いた動画のライブ配信に関する検討を行う。

1.3 本稿の構成

本論は全 5 章で構成されている。第 2 章では、本研究目的である利便性向上の対象なる IoT プラットフォームやブロックチェーン技術の関連研究について述べる。第 3 章では、Fed4IoT が提案する IoT プラットフォームの利便性向上に向けた取り組みについて紹介する。第 4 章では、スマートシティにおける様々なユースケースの下、ブロックチェーンを活用したデータ共有システムについて検討し、性能評価を行なったことを紹介する。第 5 章で、今後の課題と結論を述べる。

第2章　関連研究

本研究目的である利便性向上の対象なる IoT プラットフォームやブロックチェーン技術の関連研究について述べる。

2.1　IoT プラットフォーム[1]

1 章で述べた、IoT サービス間のデータ連携をするための IoT プラットフォームとして代表とされる FIWARE、oneM2M について紹介する。また、Fed4IoT が提案する仮想 IoT システム (VirIoT)、ThingVisor Factory についても紹介する。

2.1.1　FIWARE

FIWARE[1] は社会および公共分野におけるデータ利活用を実現する IoT プラットフォームである [39]。具体的に、FIWARE は様々なソフトウェアモジュールを備え、これらモジュールを組み合わせることにより柔軟に利用目的に合わせることができる IoT プラットフォームとなっている。FIWARE はこのモジュールの仕様を Generic Enabler（GE）として規定している。GE は移動通信事業者やベンダーが中心となる標準化団体 Open Mobile Alliance (OMA)[5] によって標準化されたインターフェースである NGSLI で規定されている。GE の特徴として、IoT ディスカバリとコンテキストブローカーの二つの機能が挙げられる。IoT ディスカバリは、IoT データの所在や提供元にアクセスするインターフェース NGSI-9 を用いて、どの IoT デバイスからデータを取得するか決定する。コンテキストブローカーは IoT データにアクセスするインターフェース NGSI-10 を用いて、アプリケーションのクエリに応じて IoT デバイスから出力される IoT データの管理を行う。この二つのインターフェースと FIWARE が定めるデータモデルを連携することで高度なデータ検索機能を提供している。

また、FIWARE は NGSI を備えている他の IoT プラットフォームとの連携も可能としているため、分散したデータ管理および異なる分野のサービスに対して FIWARE が管理するデータを連携することができる。FIWARE オープン実装の代表例として Orion Context Broker[7] が挙げられる。

2.1.2　oneM2M

oneM2M[4]、[34] は欧米日中韓の標準化団体や複数の組織によって研究開発されている IoT プラットフォームである。

oneM2M では通信方式に依存しないアーキテクチャと通信方式を仕様化している。oneM2M のアーキテクチャでは、アプリケーションを示す Application Entity (AE)、IoT プラットフォームにおいて重要となる機能の集合体を示す Common Service Entity (CSE) が規定されている。データのやり取りは、AE と CSE 間、CSE 同士での通信のように、必ず CSE を介した通信を行うように規定されている。CSE はクラウド、エッジやゲートウェイなどに設置されており、データ管理、デバイス管理、セキュリティなど 12 種類の機能を持っている。これらの機能は外部から利用できるように、リソースとして表されている。リソースの操作は、生成、取得、更新、削除、通知の 5 つに限定する

[1]本節は電子情報通信学会にて発表した「効率的な IoT データ共有および IoT サービス配備のための IoT メタ情報管理に関する検討」[35] に基づいている。

ことによって、シンプルなプロトコル設計を実現している。oneM2M での通信方式は、複数の通信プロトコル、複数のシリアライゼーションから選択することができる。これにより、システム開発の際、柔軟に用途に応じた通信方式を選択することができる。oneM2M は、OpenMTC [6] や OM2M [15] など多くオープン実装がなされている。

2.1.3 Fed4IoT が提案する仮想 IoT システム VirIoT の構成

本節では Fed4IoT が提案する仮想 IoT システム VirIoT の構成 [12],[38],[32] ついて、概要を説明する。図 2.1 に Fed4IoT が提案する仮想 IoT システム (VirIoT) を単純化した図を示す。図の VirIoT の外にあるルートデータドメインは、IoT プラットフォーム (FIWARE や oneM2M) が管理している外部からの接続を有効とするデータブローカーや物理 IoT デバイスで構成されており、VirIoT に対して物理 IoT データを供給する役割を担っている。VirIoT の構成を単純化すると、ThingVisor, vSilo の二つが代表的な構成要素となる。ThingVisor は、ルートデータドメインからデータを取得し、必要なデータ処理を施した後、NGSI-LD[13] などの中立データフォーマットへ変換し、仮想 IoT デバイス (vThings) としてデータを出力する機能を提供する。このように、仮想 IoT デバイスを生成するソフトウェアが ThingVisor である。vSilo は ThingVisor が出力する vThings の取得、中立データフォーマットから各 IoT プラットフォーム固有のデータフォーマットへの変換およびテナント (ユーザ) へのデータ提供機能で構成される。また、vSilo は、テナントから見たときに、仮想 IoT デバイスがテナント間で互いに干渉せず独立して扱えるようなアプリケーション実行環境を提供し、各 vSilo はテナント独自の IoT プラットフォームのデータブローカー機能を提供する。この ThingVisor と vSilo の間を連携するために、VirIoT では、図 2.1 のように MQTT のような Pub/Sub モデル型の通信プロトコルを想定している。

2.1.4 Fed4IoT が提案する ThingVisor Factory

本節では、VirIoT の利便性を向上する ThingVisor Factory についての概要を説明する。

ThingVisor Factory は、テナントや ThingVisor 開発者が独自の ThingVisor を設計しネットワーク内に自動配備する仕組みである。ThingVisor Factory は、図 2.2 のように、サービス提供者やスマートシティの市民が、Web インターフェースを通して、直観的に自分の希望する ThingVisor を設計することができるツールである。具体的に、図 2.2 のノードの一覧はセンサーノード、サービスファンクションノード、コネクタノードで構成されている。センサーノードでは、温度計や監視カメラなどの IoT デバイスが出力するデータを示す。サービスファンクションノードは、ネットワーク仮想化技術の一つであるサービスファンクションチェイニング技術のサービスファンクションを示しており、人物検知や顔認識などの IoT データに対する処理機能を示す。コネクタノードは、Pub/Sub モデルや Information Centric Networking (ICN) などの通信プロトコルを示す。

図 2.2 のように、これらのノードを、ドラッグアンドドラップで連結させることで ThingVisor が作成され、ネットワーク上に ThingVisor が展開される。

2.2 ブロックチェーン技術の関連研究

2.2.1 ブロックチェーン技術の概要

ブロックチェーン技術は Satoshi Nakamoto によって Bitcoin[20] と共に開発され、トランザクションを管理者を仲介せずに、複数のピアで構成される分散型ネットワークで管理する技術である。ブロックチェーンでは、トランザクションの要求があるとブロックチェーンネットワークの参加者間でトランザクションの要求に対してコンセンサスを行い、トランザクションが承認される。そして、承認されたトランザクションをブロックとして、一つなぎに台帳に追加し、参加者間で台帳を共有する

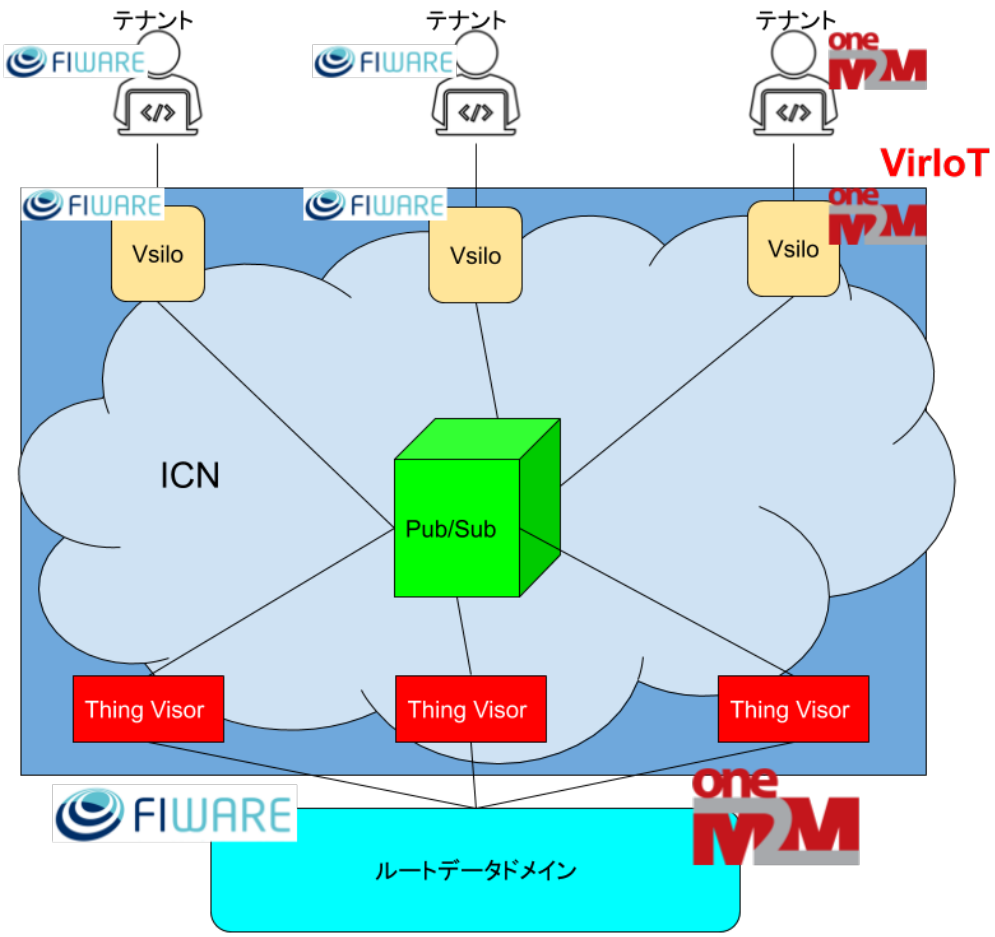

図 2.1: 仮想 IoT システム VirIoT の全体図 [35]

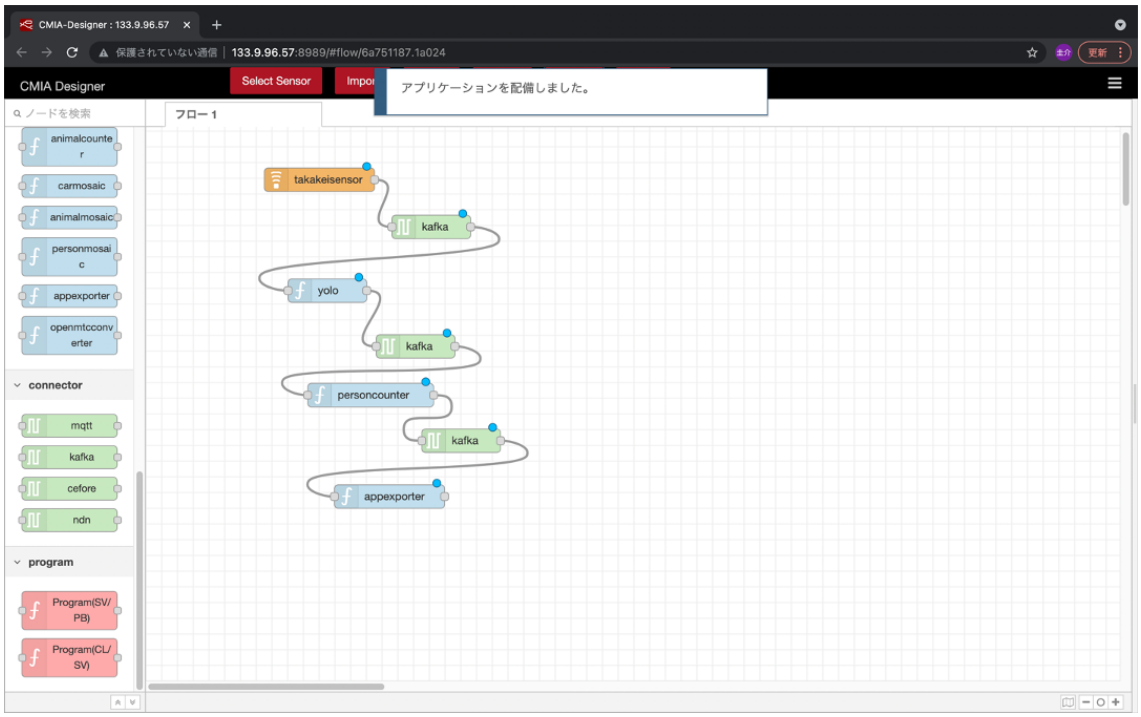

図 2.2: ThingVisor Factory における GUI プログラミング

ことでトランザクションが確立される [37]。ブロックチェーンのブロックにはトランザクションの他に、その直前のブロックのハッシュ値が保存されている。そのため、過去のブロック内のデータに対して改ざんを試みると、後に続くブロックのハッシュ値も変更しなければ改ざんしたことが発覚してしまう。従って、ブロックチェーンはデータを改ざんすることは非常に困難となっており、セキリュティを担保した台帳技術となっている [22]。

2.2.2 Hyperledger Fabric

4 章では、ブロックチェーン技術の一つである Hyperledgr Fabric を用いたデータ共有システムを構築し、性能評価を行なっている。そこで、Hyperledgr Fabric [9]、[2] について簡単に紹介する。

ブロックチェーン技術は許可型ブロックチェーンと非許可型ブロックチェーンの 2 種類に分類されている。HyperledgerFabric は許可型ブロックチェーンに該当する。許可型ブロックチェーンは、特定の参加者のみが参加できるブロックチェーン技術である。一方、非許可型ブロックチェーンは誰でも参加できるため、ノードの数が多く、コンセンサスに大きな遅延がかかる [37]。

Hyperledger Fabric はオープンソース実装であり、金融や産業などの企業向けのブロックチェーン技術である。本節では、Hyperledger Fabric の構成要素について概説し、Hyperledger Fabric の特徴となっている取引の流れを示すトランザクションフロー [9] について紹介する。

Hyperledger Fabric の構成要素の概要

例として図 2.3 のようなブロックチェーンネットワークを想定し、Hyperledger Fabric の公式ドキュメント [2] と論文 [9] を参照し、各要素について簡単に説明する。Hyperledger Fabric のクライアントアプリケーションは Go,JAVA,Node.js などのプログラミング言語によって記述されている。Certification Authority (CA) は主に利用者の ID を登録し、登録証明書を発行する機能を持つ。

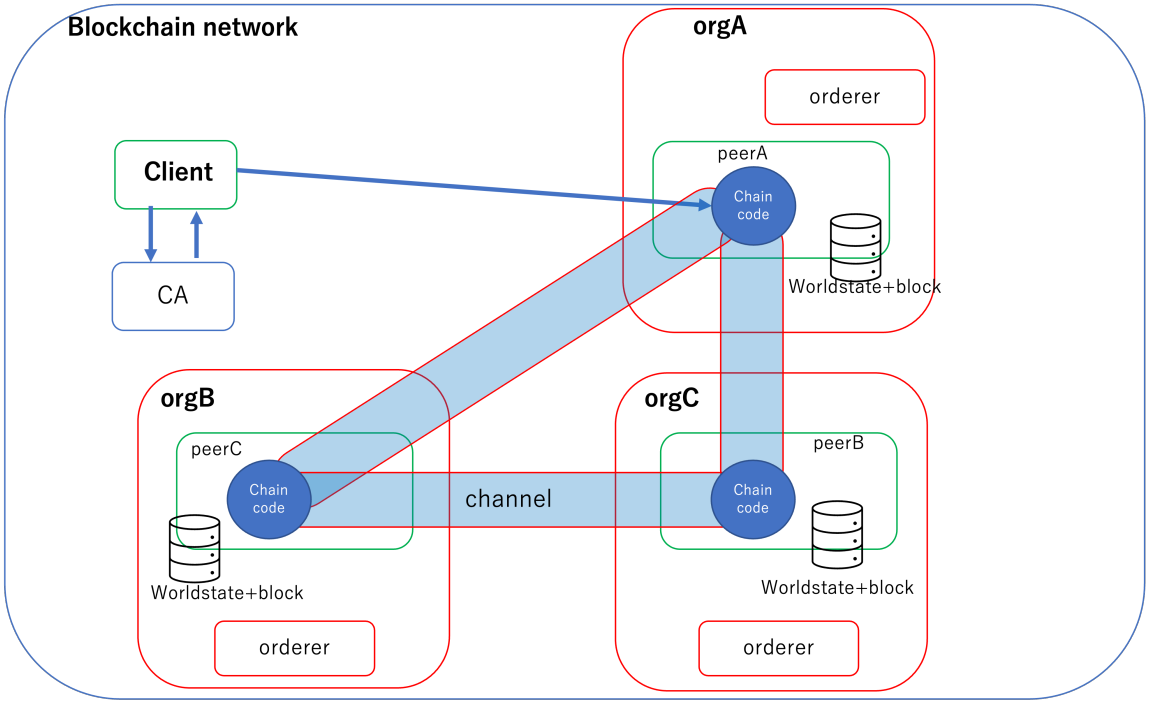

図 2.3: Hyperledger Fabric の構成要素

Hyperledger Fabric の台帳は、トランザクションやトランザクションの履歴などの情報を保存しているブロックチェーンと台帳の状態の現在値を保管する World State によって構成されている。

図 2.3 のように、ピアがチャンネルに参加することによって、ピア同士で通信することができ、台帳を共有することができる。

チェーンコードには取引のロジックが定義されている。また、図 2.3 のようにチェーンコードはピアにインストールされることによって機能する。

オーダラーはクライアントから複数のトランザクションを受け取り、複数オーダラー間でトランザクションの実行順序に対してコンセンサスを行い、トランザクションの実行順序を確立する。そして、ブロックを生成する。オーダラー間でのコンセンサスの実装は、Raft[24]、Kafka[3] など複数挙げられている。なお、Hyperledger Fabric のバージョン 2.X 以降では Raft が推奨されている。

Hyperledger Fabric のトランザクションフロー

Hyperledger Fabric のトランザクションフローは、以下の 3 つのフェーズによって実現されている。

- 実行フェーズ
- オーダリングフェーズ
- 検証フェーズ

最初の実行フェーズは複数の組織が所有するピアでチェーンコードを実行し 、クライアントが実行結果を集約して同じ結果になることを確認し、トランザクションの内容について承認するフェーズである。

図 2.4 に示すように、トランザクションの初めに、クライアントアプリケーションは、エンドーサーピアに transaction proposal と呼ばれるチェーンコードの実行依頼を送信する。エンドーサーピアは transaction proposal を受け取ると、チェーンコードを実行する。この実行結果に対し、署名を行ったものをエンドースメントとしてクライアントに送信する。

表 2.1: エンドースメントポリシーの例 [43]

エンドースメントポリシー
OR(’ 組織 A’,’ 組織 B’,’ 組織 C’)
AND(’ 組織 A’,’ 組織 B’,’ 組織 C’)
OutOf(’ 組織 A’,’ 組織 B’,’ 組織 C’)

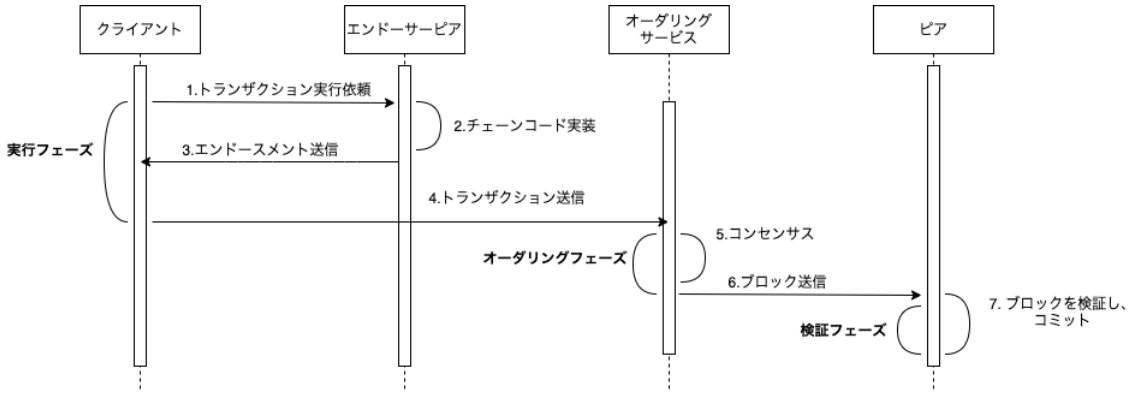

図 2.4: Hyperledger Fabric のトランザクションフロー

この時、エンドースポリシーと呼ばれる、複数ピアからエンドースメントを集める方法が定義されたポリシーに従って、複数のエンドースメントを集約する。クライアントは複数のエンドースメントを比較し、チェーンコードの実行結果が同じであることを確認することで、チェーンコードの実行結果に対して整合性を確保する。また、エンドースメントポリシーの構文は AND、OR、OutOf をサポートしており、3 つのうちのどれかを選択することでエンドースメントの集め方を変更することができる。3 つの組織で構成されるブロックチェーンネットワークを想定した場合の、エンドースメントポリシーの例を表 2.1[43] に示す。表 2.1 のエンドースポリシーが OR の時、 2 つの組織のうち、一つの組織と組織 C のピアからエンドースメントを集めることを意味する。表 2.1 のエンドースポリシーが AND の時、3 つの組織のピアからエンドースメントを集めることを意味する。表 2.1 のエンドースポリシーが OutOf の時、3 つの組織のうち、一つの組織のピアからエンドースメントを集めることを意味する。このエンドースメントポリシーによって定義されている組織のピアがエンドーサーピアである。

次に、オーダリングフェーズは、クライアントから送信される様々なトランザクションの順序付けを行い、そのトランザクションおよびその実行順序に係わる情報を含むブロックを作成し、ピアにブロックを送信するフェーズである。実行フェーズが完了すると、クライアントはエンドースメントをもとにトランザクションを作成しオーダリングサービスに送信する。オーダリングサービスは単一または複数のオーダラーから構成されており、オーダリングサービスがトランザクションを受け取ると、オーダラー間でコンセンサスを取りトランザクションの順序が確立する。コンセンサスを取り終えると、オーダリングサービスにおけるリーダーのノードがブロックを生成し、ブロックを各ピアに送信する。

最後に、検証フェーズは、ブロックにあるトランザクションの正当性を検証し、台帳に追加するフェーズである。オーダリングサービスから受け取ったブロックのトランザクションのキーと World State のキーを比較し、トランザクションの正当性を検証する。検証が終わると、台帳にブロックが追加され、World State も更新される。

2.2.3 ブロックチェーン技術の性能評価

[10] では Hyperledger Fabric v1.0 の性能評価を行なっている。評価シナリオとして、Hyperledger Fabric v1.0 のスループットと遅延時間、ベンチマークツールを用いたトランザクションとチェーンコードのパラメータを変化させた時の遅延時間を測定し、評価している。[10] における環境下では、Hyperledge Fabric の key-value ストアの値を読み込むトランザクションの遅延時間は、1 秒間に送信するトランザクション数に対して線形になっている。key-value ストアの値を更新するトランザクションを 1000tx/s より大きな値で送信すると、スループットは下がり、トランザクションの待機時間が大幅に大きくなっていることを示している。

[21] では Hyperledger Fabric の二つのバージョン、v0.6 及び v1.0 についてパフォーマンスを比較している。1 秒間に送信するトランザクションの数、ピアの数を変化させた時のトランザクションの実行時間、遅延、スループットを計測して評価している。これらの結果は、全体として v1.0 が優れていることを示している。また双方のバージョンでも送信されるトランザクション数が増えるとトランザクションの平均実行時間は大きくなることを示している。ピアの数が増えると v0.6 ではトランザクションの実行時間が大きくなる一方で、v1.0 では特定の範囲内で実行時間が収まっていることを示している。

[27] では、Hyperledger Fabric v1.4.3 に対してのパフォーマンス評価を行なっている。具体的には、2.2.2 節で述べたトランザクションフローにおける 3 つのフェーズ、実行フェーズ、オーダリングフェーズ、検証フェーズについて、エンドースメントポリシーの設定 (AND、OR) の 2 種類を変えた場合のパフォーマンスへの影響を測定している。また、オーダリングフェーズにおけるコンセンサスの実装について、Kafka、Solo、Raft それぞれのパフォーマンスの比較を行なっている。実験結果は、エンドースメントポリシーが AND の時と比べ、エンドースメントポリシーを OR にすることによって、実行フェーズでは高いスループットを達成することとなり、その結果、スケーラビリティの向上に繋がることを示している。上記に加えて、実験結果より、3 つのフェーズのうち、検証フェーズがボトルネックになることも示している。これは、ブロックの検証の処理に時間がかかるためとしている。また、オーダリングフェーズのコンセンサスにおける実装については、Kafka、Solo、Raft それぞれのパフォーマンスの比較の結果から、大きな違いはないことを示している。

[28] では、遅延に敏感な IoT アプリケーション向けの新しいブロックチェーンシステム CATP-Fabric を提案している。IoT サービスや IoT デバイスを扱う IoT 環境下では、高頻度でデータのやりとりが行われる。これにより、トランザクションが競合する可能性がある。トランザクションが競合すると、トランザクションが中止され、リソースの利用効率が低下する。この課題を解決するために提案されたシステムがブロックチェーンシステム CATP-Fabric である。CATP-Fabric と Hyperledger Fabric などのブロックチェーンシステムと比較して、トランザクションの高スループットを達成することを示している。

2.2.4 ブロックチェーン技術の利用例

[11] では、IoT 向けのブロックチェーンシステム Tikiri について提案している。また、CPU 使用率、メモリ使用量、台帳に書き込むトランザクション、読み込むトランザクション、ピアノードの数を増やした時のスケーラビリティについて性能評価を行なっている。、CPU 使用率、メモリ使用量、台帳に書き込むトランザクション、読み込むトランザクションの評価においては、ブロックチェーン技術である Hyperledger Fabric、Mystiko、BigchainDB と比較して評価を行なっている。CPU 使用率、メモリ使用量では、Tikiri は他のブロックチェーン技術より良い性能になっていることを示している。また、ピアノードの数を増やした時のスケーラビリティの評価では、ノードの数が増えた時、線形的に実行されるスループットの割合が増加している。

[16] では、ブロックチェーンを利用した、IoT データの信頼性を担保する IoT プラットフォームを提案している。具体的には、IoT システムに IoT デバイスのデータを登録する際に、周辺の IoT デ

バイスが生成するデータと比較し検証することによって、IoT データの信頼性を確保している。[41] では、この検証方法を改善し、さらにセキュリティを担保した検証方法を提案している。

[30] では、Hyperledger Fabric を用いたマネージド型ブロックチェーン基盤サービスにおけるサービス間での業務システム移行方法について紹介している。具体的に、新しい業務システムに移行する場合、以前使用していた業務システムの台帳のコピーをブロックチェーンネットワークに参加している組織間でコンセンサスを行うことにより移行している。

[25] では、科学論文の出版におけるプロセスの効率性向上を図る、ブロックチェーンに基づいた科学論文の出版システム EUREKA を提案している。EUREKA では出版プロセスに関わる人に報酬を分配する報酬分配システムを保持している。

EUREKA の報酬分配システムは、レビューの品質やレビューする人の評判を考慮せずに報酬を分配しているため、レビューする人たちに対して、不公平に報酬が分配されることがある。そこで、[19] では、発表論文の数やレビューの数などを指標に、査読者の評判を考慮した報酬分配システムを実現している。

[31] では、クラウドソーシングにおける評価の改ざんを防ぐために、ブロックチェーン技術を用いた評価を管理するシステムを提案している。

第3章　VirIoT、ThingVisor Factoryの利便性向上に向けた取り組み[2]

本章では、Fed4IoT が提案する仮想 IoT システム (VirIoT) および ThingVisor Factory の利便性を向上するための提案プロトコルおよび認証サービスについて説明する。

3.1　提案するIoTメタ情報管理プロトコル

本節では、物理 IoT デバイスや ThingVisor が持つメタ情報の管理機能を持つ Catalog Server と Catalog Server に係わる各種プロトコルについて紹介する。

2 章で述べたように、VirIoT の主な役割は、ThingVisor と vSilo を柔軟かつ効率的にネットワーク内に展開し、ThingVisor と vSilo を HTTP や MQTT といったプロトコルで結び、データのやり取りを行うことである。また、ThingVisor Factory は、テナントや ThingVisor 開発者が独自の ThingVisor を動的に設計しネットワーク内に自動配備する仕組みである。しかし、VirIoT および ThingVisor Factory はスマートシティにて利用できる物理 IoT デバイス、IoT サービスや ThingVisor は前もって登録されているものに対して機能しており、動的に構築される ThingVisor や vThings の収集・管理方法は検討されておらず、課題となっている。そこで、上記の課題の下、VirIoT および ThingVisor Factory の利便性を上げるため、物理 IoT デバイスや ThingVisor が持つメタ情報の管理サービスである Catalog Server およびその各種プロトコルを提案する。

Catalog Server は物理 IoT デバイスや ThingVisor が持つメタ情報の管理サービスであるとともに、テナントと VirIoT とのインターフェース機能を提供する。Catalog Server の機能を実現する 3 つのプロトコルを図 3.1 に示す。図 3.1 の緑線のやりとりが物理 IoT デバイスのメタ情報を Catalog Server へ登録するプロトコル、赤線のやりとりが ThingVisor 作成者が作成した ThingVisor のメタ情報を Catalog Server に登録するプロトコル、黒線のやりとりが ThingVisor 作成者が作成した ThingVisor のメタ情報を Catalog Server に登録するプロトコルを示す。

また、提案した ThingVisor の自動配備および vThings 取得に関するプロトコルでは特に処理時間がかかることが予測されるため、ThingVisor 自動配備から vThings 取得までにかかる遅延時間を初期評価したので報告する。

3.1.1　Catalog Server

2 章で述べたように、VirIoT および ThingVisor Factory は前もって登録されている物理 IoT デバイス、IoT サービスや ThingVisor に対して機能しており、動的に構築される ThingVisor や vThings の収集・管理方法は検討されていない。そこで、ThingVisor Factory の利便性向上を図るため、これらメタ情報の管理機能 (保存、検索) やテナントと VirIoT とのインターフェース機能を Catalog Server が提供する。また、Catalog Server は図 3.1 に示すように、ThingVisor Factory の構成要素の一つである。

以降の節で紹介するプロトコルにおいて、Catalog Server に関わるアクターとしてテナント、ThingVisor 作成者、ルートデータドメインの管理者、VirIoT の管理者、を想定している。また、

[2]本章は電子情報通信学会にて発表した「効率的な IoT データ共有および IoT サービス配備のための IoT メタ情報管理に関する検討」[35] と「IoT デバイスを IoT サービス間で柔軟に共用する機構の開発」[36] に基づいている。

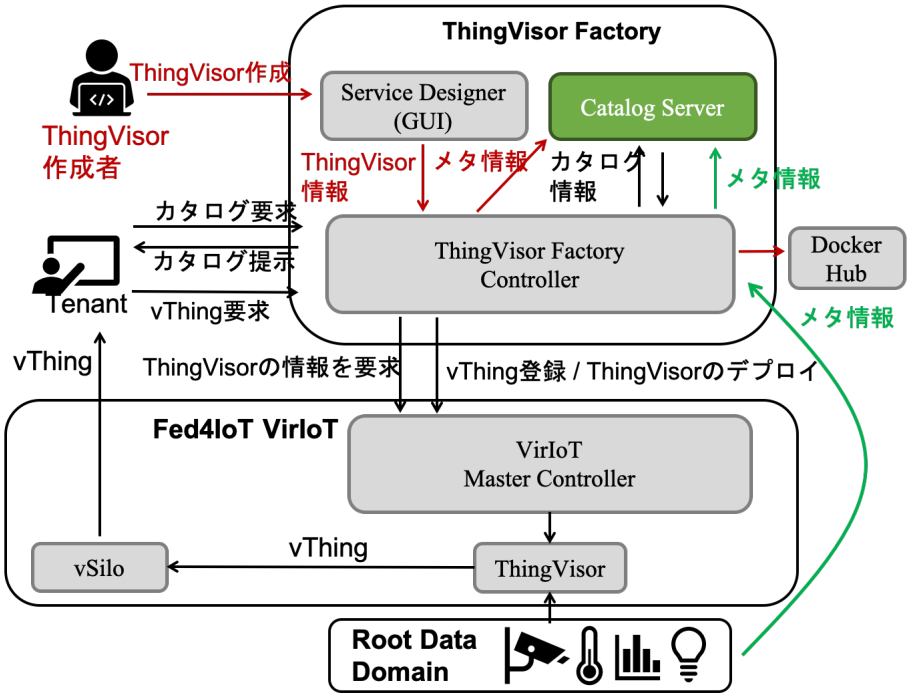

図 3.1: 提案する Catalog Server の位置づけ [35]

VirIoT の管理者、ルートデータドメインの管理者はそれぞれ、IoT サービスの実行環境を提供するプラットフォーマー、スマートシティの運営者を想定する。テナントと ThingVisor 作成者は同一で IoT サービスの提供者を想定しており、サービスの提供者が持つセンサデバイスやその他のセンサデバイスを使って、サービス提供者が、独自のサービスを設計し、提供することを想定している。

3.1.2 物理 IoT デバイスのメタ情報を Catalog Server へ登録するプロトコル

テナントが様々な ThingVisor を容易に開発するためには、ルートデータドメインが所持する物理 IoT デバイスのメタ情報を共有することが重要である。そのため、Catalog Server は物理 IoT デバイスなどのメタ情報の保存、検索機能を提供する。

図 3.1、図 3.2 に示すように、ルートデータドメインが物理 IoT デバイスのメタ情報を ThingVisor Factory を経て Catalog Server に登録する。この時、VirIoT の管理者とルートデータドメインの管理者は異なるため、ルートデータドメインの管理者が物理 IoT デバイスのメタ情報を Catalog Server に登録するには、VirIoT 管理者から登録の許可をもらうことが求められる。ルートデータドメインが VirIoT 管理者に物理 IoT デバイスのメタ情報を送信し、VirIoT 管理者が物理 IoT デバイスのメタ情報に対して登録の許可を与え、ThingVisor Factory に物理 IoT デバイスのメタ情報を送信する。許可された物理 IoT デバイスのメタ情報を ThingVisor Factory が受け取り、Catalog Server に登録することができる。

3.1.3 ThingVisor 作成者が作成した ThingVisor のメタ情報を Catalog Server に登録するプロトコル

次に、ThingVisor 作成者が作成した ThingVisor のメタ情報を Catalog Server に登録するプロトコルについて紹介する。図 3.3[35] に示すように、まず ThingVisor 作成者は、Catalog Server から ThingVisor Factory Controller 経由で使用できる IoT デバイスや ThingVisor のメタ情報を取得す

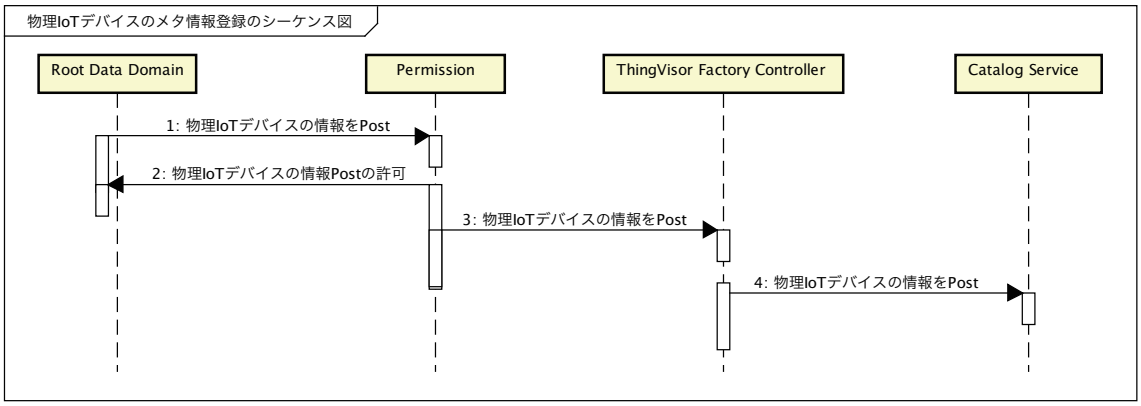

図 3.2: 物理 IoT デバイスのメタ情報登録のシーケンス図 [35]

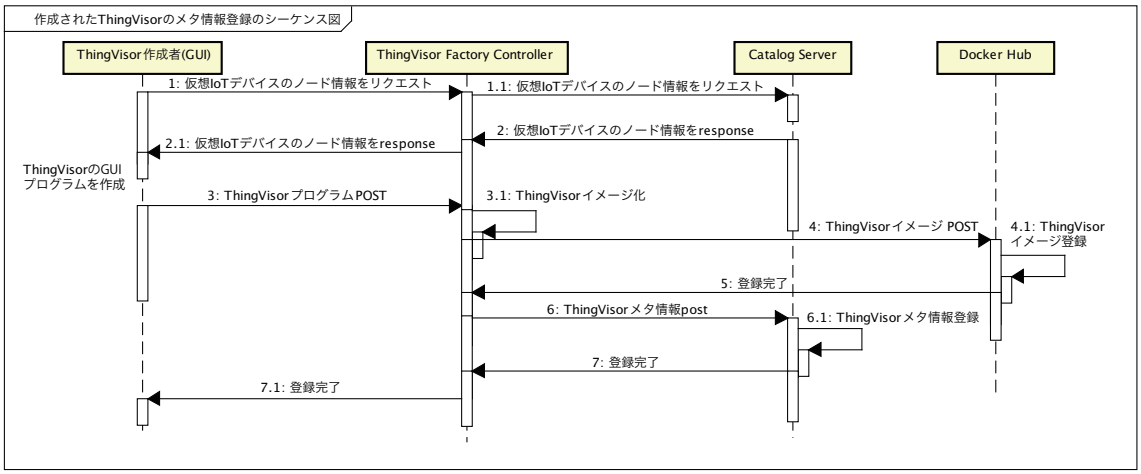

図 3.3: 作成された ThingVisor のメタ情報登録のシーケンス図 [35]

る。このメタ情報を図 2.2 で示すようにコンポーネント (ノード) として表示することで、ThingVisor 作成者は、GUI プログラミングの概念に基づき、ノードを GUI 上にドラッグアンドドロップすることで ThingVisor を作成する。この時、作成した ThingVisor を ThingVisor Factory Controller に送信することによって、ThingVisor Factory 側で ThingVisor の Docker イメージを生成し Docker Hub へ登録を行う。そして、Docker イメージの登録が完了すると、作成した ThingVisor のメタ情報を Catalog Server へ登録する。

3.1.4 ThingVisor の自動配備および vThings 取得に関するプロトコル

最後に、ThingVisor の自動配備および vThings 取得に関するプロトコルについて紹介する。本プロトコルによって、テナントが VirIoT 上で ThingVisor を配置することが可能となることから、Catalog Server は VirIoT へのインターフェースという側面も持つこととなる。

図 3.1、3.4[35] に示すように、まずはじめにテナントは Catalog Server に登録されている vThings の一覧を取得する。その後、テナント固有の vSilo に追加したい vThings を選択することでダッシュボードから ThingVisor Factory Controller へと vThings が要求される。ThingVisor Factory Controller は vThings を要求されると、要求された vThings を出力する ThingVisor が起動しているか否かを VirIoT の Master Controller へ問い合わせる。もし起動しているならば、図 3.4 より、ThingVisor Factory Controller は vSilo に vThings を追加するよう Master Controller に命令する。起動してい

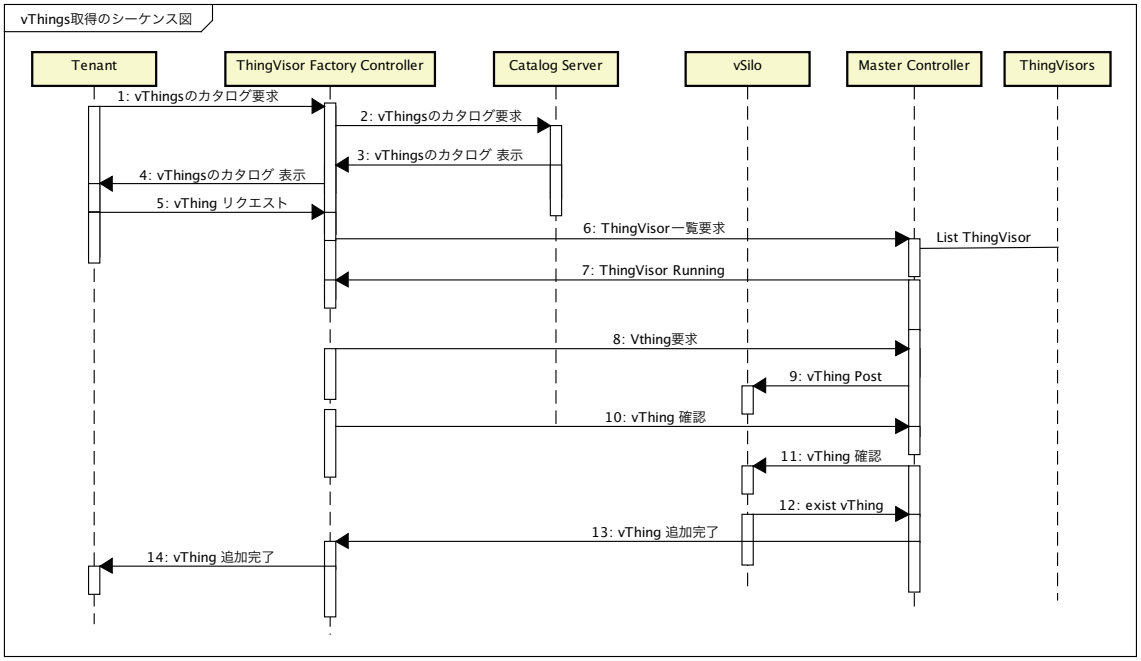

図 3.4: ThingVisor 起動時の vThings 取得のシーケンス図 [35]

ない場合は、図 3.5[35] より、ThingVisor Factory Controller が Master Controller に ThingVisor を起動させる命令を送信し、ThingVisor の起動を促す。その後、上記で示したように vSilo に vThings を追加する命令を Master Controller に送信する。最後に、Master Controller が ThingVisor Factory Controller へ vThing 追加完了を示すメッセージを送信する。

3.1.5 VirIoT 上における単一のテナントが vThing 取得にかかる遅延時間の評価

実験環境及び実験シナリオ

本節で提案しているプロトコルの性能評価として、特に処理時間が大きくなると予想される ThingVisor の自動配備および vThings 取得に関するプロトコルについて、実際に VirIoT 上に ThingVisor が自動配備され vThing の取得までにかかる遅延時間について評価を行う。図 3.4 および図 3.5 に示されているように、ThingVisor が配備されているか否かによって大きく遅延時間が変わると予想されるため、ThingVisor の配備の有無の違いに注目し遅延評価を行う。本評価実験の実験環境を図 3.6 に示す。図 3.6 より、本実験環境は仮想マシン VitualBox 上で、ゲスト OS は Ubuntu 18.04.5 で行なった。また仮想マシンは一台で行っており、VirIoT、Catalog Server、ThingVisor Factory Controller は全て同一のマシンで起動している。

ThingVisor は WeatherAPI から気象情報を取得する ThingVisor である weather-thingvisor を用いており、これは世界の都市の気象情報を vThings として出力する。vSilo については、IoT プラットフォームである oneM2M のオープンソース実装の一つである Mobius ブローカーを利用しているものを使用している。ThingVisor と vSilo 間のデータ接続は MQTT ブローカーを利用する。

また、図 3.5 のシーケンス番号 1 から 20 までの処理を行えるよう ThingVisor Factory Controller と Catalog Server を含めプロトコル実装し、シーケンス番号 5 から 20 までの vThing 取得にかかる遅延時間を二つの条件下で測定し、遅延評価を行なう。なお、本実験の試行回数は以下の条件それぞれに対して 10 回行い、平均値をとることとする。先に述べたように、ThingVisor の配備の有無の違いに注目するため、以下に示す二つの条件下において遅延時間の評価を行う。

- ThingVisor が起動しているときの vThing 取得にかかる遅延時間

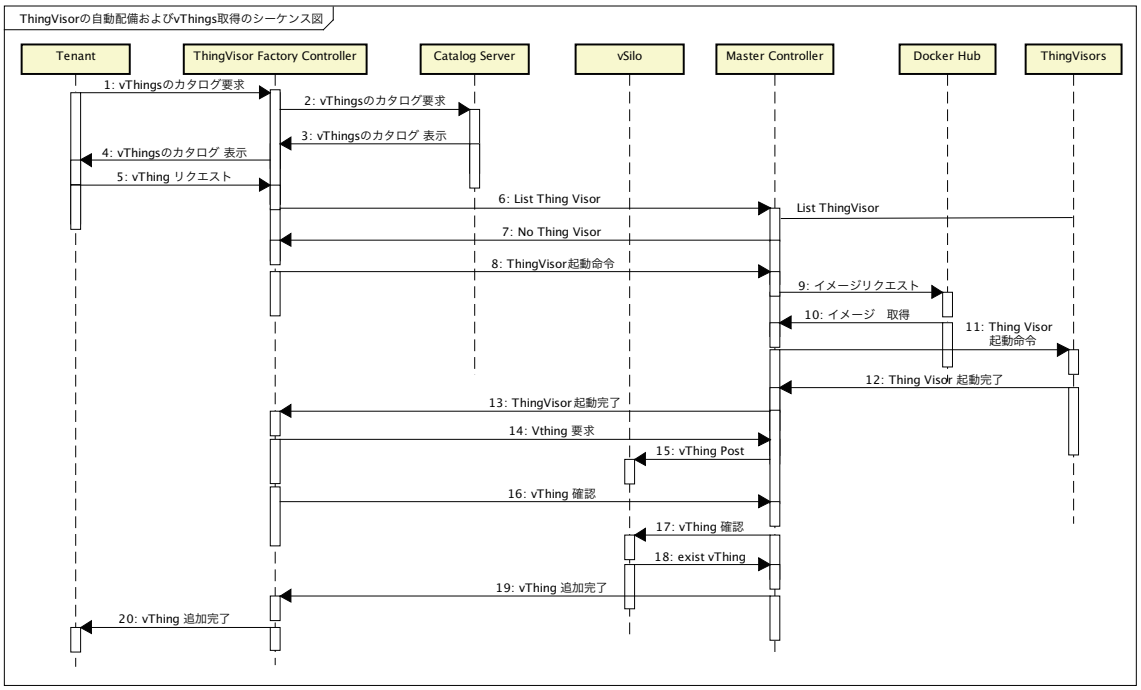

図 3.5: ThingVisor の自動配備および vThings 取得のシーケンス図 [35]

- ThingVisor が起動していないときの vThing 取得にかかる遅延時間

評価結果

表 3.1[35] より、ThingVisor が起動している時よりも ThingVisor が起動していないときの方が IoT サービス配備遅延時間が大きくなっており、ThingVisor を起動させるのに大きな遅延が生じることがわかる。

表 3.1: 単一のテナントによる vThing 取得における遅延時間 [35]

TV の条件	TV の作成時間 [S]	IoT サービス配備の遅延時間 [S]
TV が起動している	ー	1.619681621
TV が起動していない	8.347873259	9.589337516

3.1.6 VirIoT 上における複数のテナントが vThing 取得にかかる遅延時間の評価

実験環境及び実験シナリオ

先に記載した評価実験では、テナントが一人であることを想定していたが、本評価実験では、テナントを複数人いるものと想定する。このとき、複数のテナントが同じ ThingVisor を要求する場合において、ThingVisor が出力する vThing をそれぞれのテナントに追加したときの vThings 取得時間をそれぞれ測定した。本評価実験の実験環境を図 3.7 に示す。図 3.7 より、本実験環境は、仮想マシン VMware 上で、仮想マシンは一台で行っており、VirIoT、Catalog Server、 ThingVisor Factory Controller は先ほどと同様全て同一のマシンで起動している。また、ThingVisor も weather-thingvisor を、vSilo も Mobius ブローカーのものを利用している。ThingVisor と vSilo 間を結ぶ通信プロトコルも MQTT ブローカーを利用している。

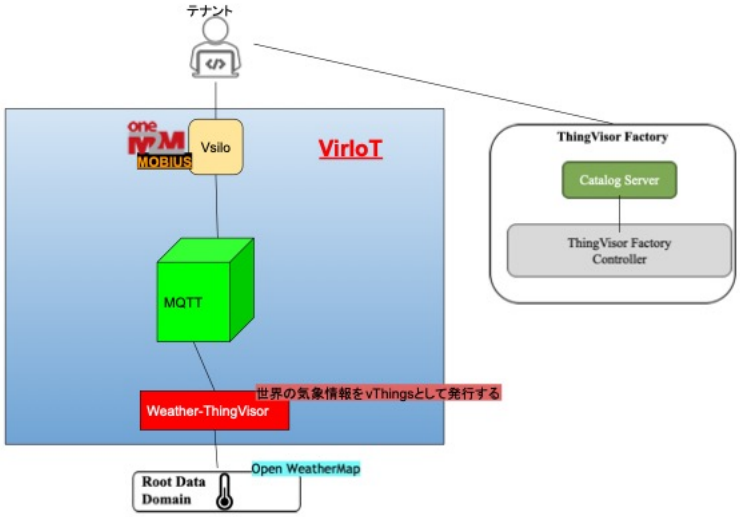

図 3.6: 実験環境

テナントの人数として 5 人と設定し、それぞれが同じ ThingVisor をリクエストする。リクエストする vThings は東京の温度を出力するものとする。このとき、ThingVisor が起動していない状態で、テナント 1 からテナント 5 まで用意し、テナント 2 からテナント 5 まで、それぞれ、テナント 1 がリクエストを送信してから、3 秒間隔で ThingVisor にリクエストを送信するようにして測定した。

評価結果

図 3.8 より、テナントが複数人存在する場合、ThingVisor を共有することで、時系列的に後から処理結果のデータ（vThing）を要求してくるテナントに対しては、ThingVisor を配備する処理を省くことができるため、表のように vThing の取得時間が実際に短く抑えられることを示す結果となる。

3.2 提案する認証サービス

本節では、3.1 節で提案したプロトコルに対して、セキュリティを担保する認証サービスについて検討し、認証サービスと提案プロトコルを登録サービスとして実装したことを紹介する。

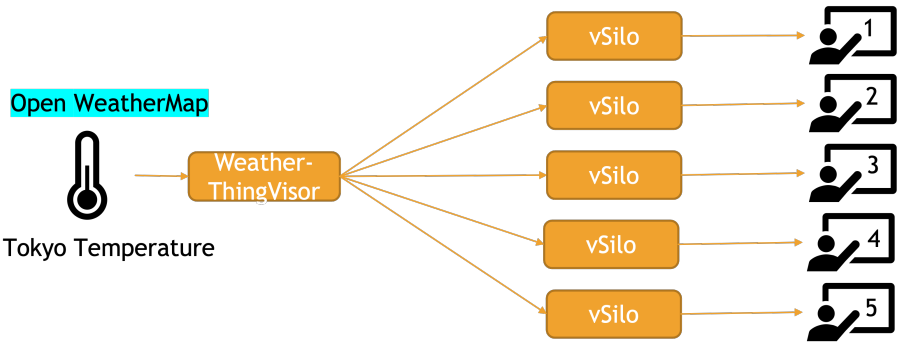

図 3.7: 実験環境

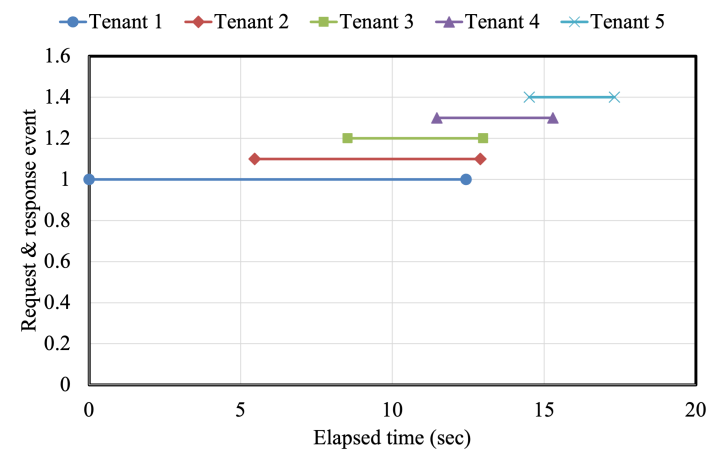

図 3.8: 複数のテナントによる vThing 取得における遅延時間

3.2.1 IoT デバイスの認証および登録サービス

ThingVisor Factory に対して、利用可能な IoT デバイスや IoT サービスを提供する際、不正な IoT デバイスおよびサービスの登録を防ぐ必要がある。本節では、そのようなセキュリティを考慮し、IoT デバイスやサービスを登録する登録者の属性情報を用いた属性認証について紹介する。紹介する認証サーバは、国境を越えた人見守りサービスにおける認証サービス [14] を参考にしている。本認証サービスでは、見守りサービス利用者の属性情報（例えば、性別、家族構成等）を利用し、サービスの利用要求や IoT デバイス（カメラ）へのアクセス要求について、認証および認可に利用している。

この認証サービスを、ThingVisor Factory に対して IoT デバイスや IoT サービスを登録する際の認証および認可へと再利用することとする。ユースケースとして、本節では、スマートシティの市民自身が、所有する IoT デバイスを利用したサービスを設計し、利用することとする。まず、市民自身が保持している IoT デバイスを直接 ThingVisor Factory に登録する場合を考える。ThingVisor Factory へ IoT デバイスやサービスを登録する場合の処理やデータの流れを、図 3.9[36] および図 3.10[36] に示す。

図 3.9 に示すように、登録者は登録するセンサー情報と認証で扱う登録者のユーザ属性情報を登録サーバに送信する。このとき、登録サーバでは、ユーザ属性情報のみを選択し、認証サーバへ送信することで認証を行う。認証サーバでは、各スマートシティに配置されているブロックチェーンのピアノードに対してリクエストを送信する。この時、各ピアの分散台帳には、前もってユーザ属性情報に乱数値をかけたハッシュ値が登録されており、このハッシュ値と認証のために送信されてきたユーザ属性情報のハッシュ値を比較することで認証している。

もし、認証することができれば図 3.9 のシーケンス番号 3 から 5 のように、認証できたことを示すメッセージとアクセストークンが発行され、Catalog Server にてアクセストークンを検証し、登録の認可がされる。しかし、認証に失敗すると図 3.9 のシーケンス番号 4.1 のように拒否メッセージが登録者に送信され、認証に失敗したことが報告される。

3.2.2 認証及び登録サービスの実装

ここでは、3.2.1 節にて紹介した IoT デバイス、サービスの認証および登録サービスの実装例を説明する。各種サービスは、マイクロサービスアーキテクチャに従っており、各種機能に分割して Docker コンテナを利用しマイクロサービスとして実装している。なお、各機能は、Python を利用して実装している。

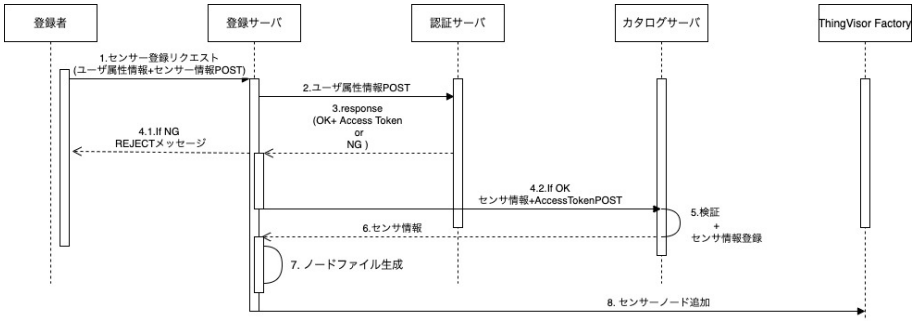

図 3.9: IoT デバイスおよび IoT サービスの認証および登録のシーケンス図 [36]

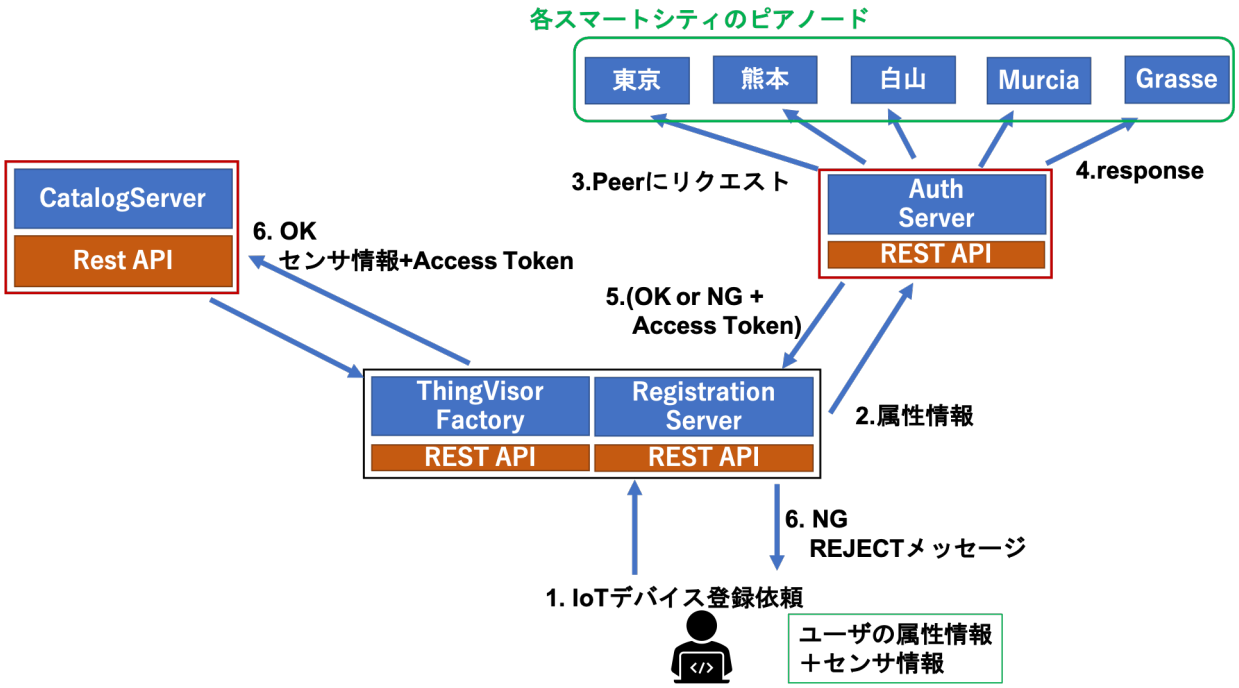

図 3.10: IoT デバイスの認証、登録サービスの構成図 [36]

具体な実装として、図 3.10 に示すように、各サービスへのアクセスについては RESTful API で定義し実装を行っている。ThingVisor Factory、Catalog Server、登録サーバは早稲田大学の研究室内のサーバ上にインストールし、認証サーバとピアノードは、IBM ブロックチェーンプラットフォーム上にインストールし動作検証を行った。ユーザ属性情報とセンサー情報は JSON 形式で記述しており、図 3.9 のシーケンス番号 5 を除いたシーケンス番号 1 から 8 までを実装した。

カメラと Raspberry Pi によって IoT デバイスを模擬し、プラグアンドプレイで認証および ThingVisor Factory への登録を実装したデモを紹介する。ラズパイの電源を入れると、ラズパイからカメラのセンサー情報と認証で扱うユーザ属性情報を登録サーバに送信する。本デモで扱う認証情報は人の名前と出生名とした。これらの情報を登録サーバで受信後、登録サーバは認証サーバへとこれらの情報を転送する。その後、認証サーバにて認証が行われ、無事に認証が完了すると、登録サーバにてセンサー情報の登録が行われる。全ての処理が完了すると、図 3.11 のように ThingVisor Factory 上にて「登録したセンサー名（ここでは takakeisensor）」が現れることとなり、利用することが可能となる。実際にカメラのデータが流れているかを確認するため、人体検知をする YOLO サービスと人数を数える person counter サービスを使った IoT アプリケーションを図 3.12 のようにコンポーネ

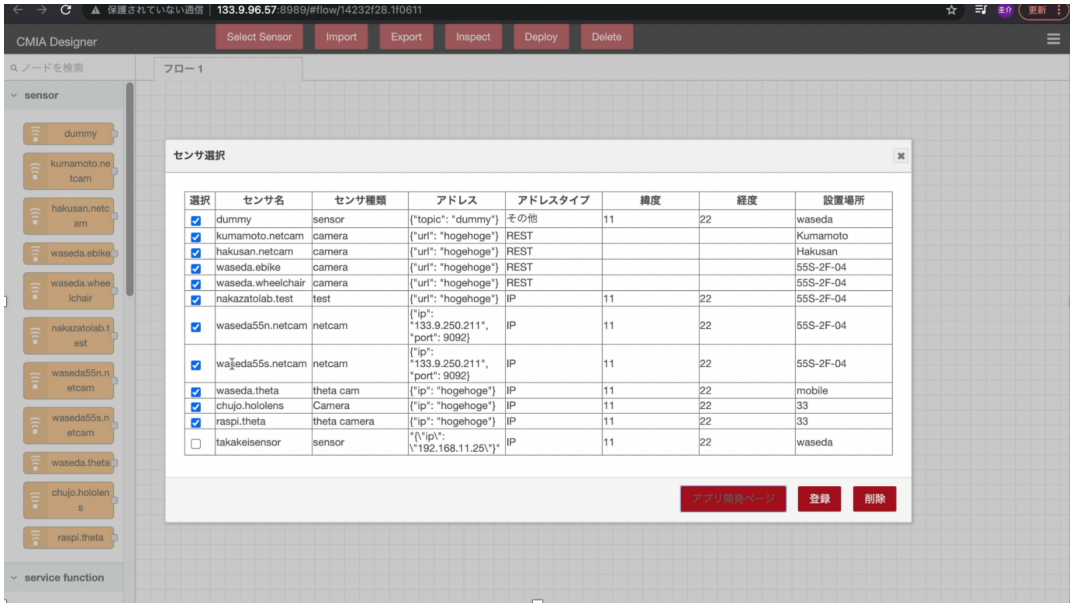

図 3.11: ThingVisor Factory に登録されている IoT デバイスおよびサービス一覧

ントをドラッグして作成する。そして、作成した IoT アプリケーションを実行すると、カメラからデータが送信され、図 3.13 のように人の画像と、左側のグラフに人数が表示され、IoT アプリケーションが正常に動作していることを確認した。

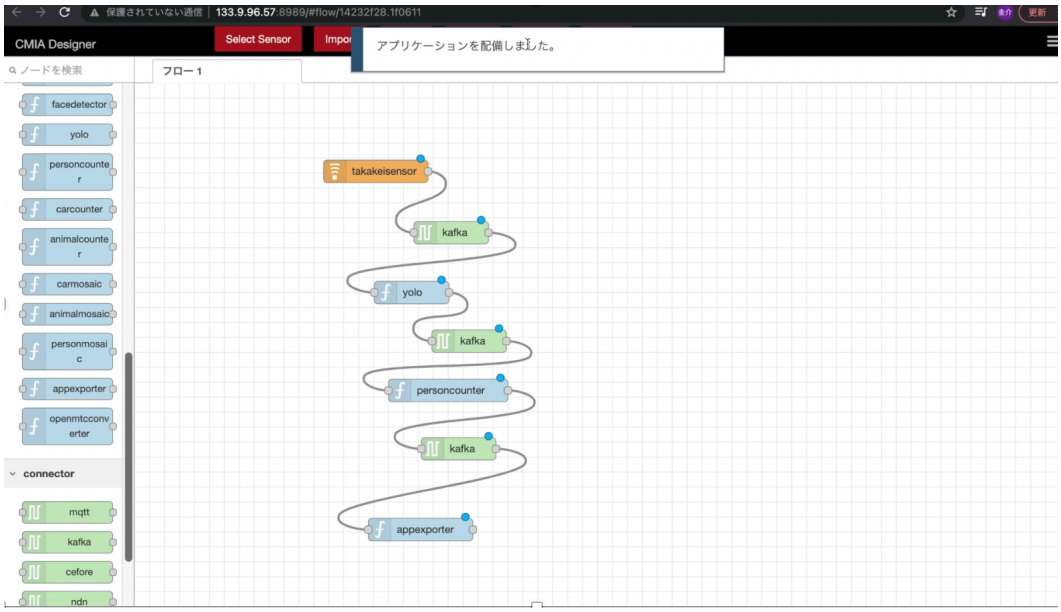

図 3.12: IoT アプリケーションの作成および実行の様子

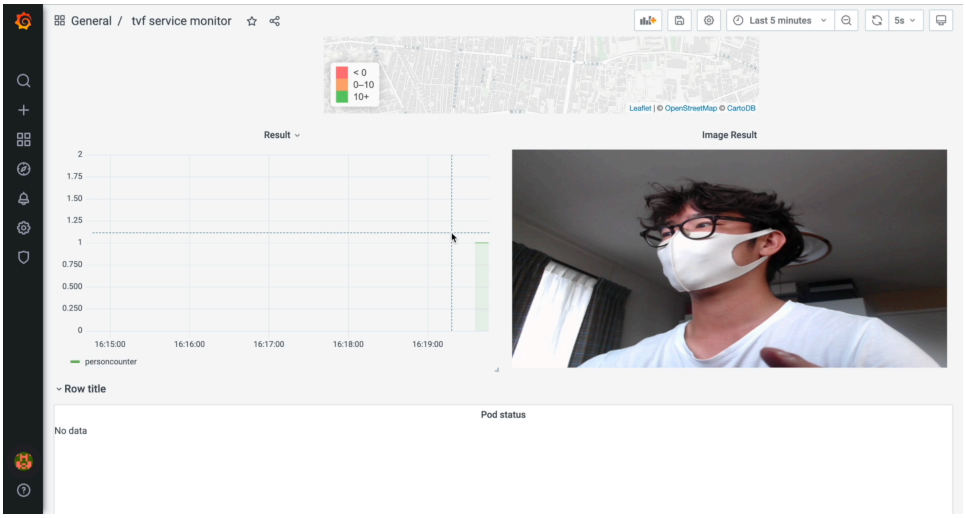

図 3.13: IoT アプリケーションが生成するデータ

第4章　ブロックチェーンを活用したデータ共有システムの検討

本章では、スマートシティを実現するための主要な技術の一つであるデータ共有技術について、ユースケースを示すとともに、ブロックチェーンの活用を検討する。その際、各ユースケースにおけるデータ共有システムにおけるブロックチェーンの適用可能性について、初期実験の性能評価結果から議論する。

4.1　データ共有システムのユースケース

本節では、スマートシティにおけるデータ共有システムのユースケースとして、代表的な 2 つのユースケースについて紹介する。具体的には、システムへの認証時に必要となる人の属性情報やセンサデバイスから生成される IoT データそのものを登録する例と、システムに登録されているデータをシステムから呼び出し利用する例である。

4.1.1　人の属性情報や IoT データをデータ共有システムへ登録するユースケース

まず、図 4.1 に示すような、人の属性情報や IoT データをデータ共有システムへ登録するユースケースについて説明する。具体的には、ユーザが認証を行うためにユーザの属性情報を登録する場合やセンサデバイスから生成される IoT データそのものをデータ共有システムへ登録する利用例である。

本ユースケースでは、データ共有システムに登録されるデータに対して改ざんされているか否かといったデータの信頼性を担保することが重要となる。これは、悪意ある攻撃者によって人の属性情報が改ざんされてしまうと、本来利用権を持つユーザが正しく認証されず、システムを利用することができなくなってしまうためである。また、データ共有システムで扱われる IoT データそのものが悪意ある攻撃者によって改ざんされてしまうと、IoT サービス提供者が想定したサービスを正常な状態でかつ適切に利用者へ提供することが困難になる可能性がある。さらに、データ共有システムに対して不正にアクセスされ、人の属性情報や IoT データが盗難される可能性もある。このように、上記に示す問題から、本ユースケースでは、登録されるデータに対して改ざんされていないかといった信頼性を担保する機能をデータ共有システムには要求される。

また、データ共有システムは、IoT データなどの高頻度でやり取りされる環境下において、システムダウンを起こさないために遅延時間やスループットなどのスケーラビリティも担保することが重要となる。

4.1.2　データ共有システムに登録されている IoT データを活用するユースケース

次に、図 4.2 のように、データ共有システムに登録されている IoT データを呼び出す利用例について説明する。具体的には、ユーザ (ここでは、IoT サービス提供者を想定する) が、データ共有システムに登録されている IoT データを呼び出し、自身のサービスに利用する例緒である。

本ユースケースにおいても、前節で示したように、データ共有システムに登録されているデータが改ざんされているかといったデータの信頼性が重要と言える。これは、先に述べたように、改ざんさ

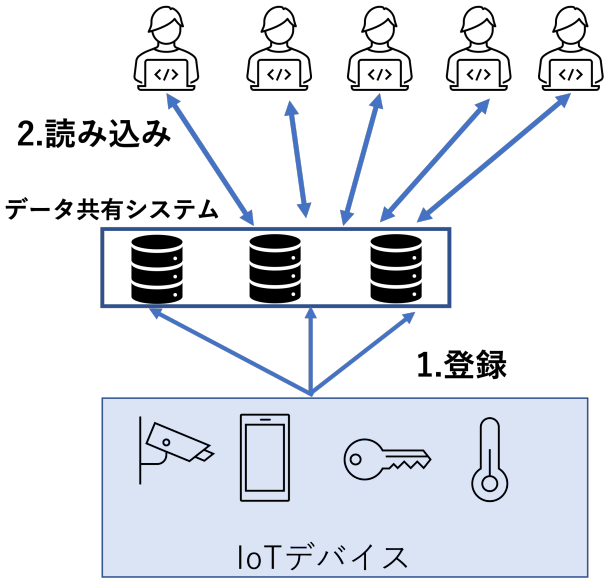

図 4.1: データ共有システムを用いたデータの登録および活用のユースケース

れた IoT データを IoT サービス提供者が、改ざんされていることを把握せず無自覚に利用することで、IoT サービス提供者が想定するサービスを適切に提供することが困難になる可能性があるためである。上記に示す問題から、本ユースケースにおいても、登録されているデータに対して改ざんされていないかといった信頼性を担保することが、データ共有システムの機能要件として挙げられる。また、前節で示したように、データ共有システムは、IoT データなどの高頻度でやり取りされる環境下において、システムダウンを起こさないために遅延時間やスループットなどのスケーラビリティも担保することが重要となる。

4.2　ブロックチェーンを活用したデータ共有システム

4.1 節で挙げたユースケースにおけるデータ共有システムの要求に対応するために、ブロックチェーンを適用したデータ共有システムについて検討する。ブロックチェーンを適用したデータ共有システムでは、登録されるデータに対して改ざんされていないかといった、データの信頼性を担保することが期待できる。

ブロックチェーンは、2 章で述べたように、各ピアで台帳を共有しており、各ピアでトランザクションの要求に対してコンセンサスを行うことによって、トランザクションを検証する。検証されたトランザクションはブロックとして、ブロックを一つなぎに追加していく。このブロックにはその直前のブロックのハッシュ値も保有されているため、ブロックチェーンはデータを追跡し、改ざんすることを困難とする、セキリュティを担保した台帳技術となっている [22]

4.1 節で述べた二つのユースケースにおいて、図 4.2[26] に示すように、ブロックチェーン技術を用いたデータ共有システムについて検討する。このデータ共有システムでは、ブロックチェーン技術の一つである Hyperledger Fabric を適用することとし、スマートシティ領域において、複数のスマートシティ運営者によってデータ共有システムが管理されることを想定する。

2 で述べたように、Hyperledger Fabric は許可型ブロックチェーンと非許可型ブロックチェーンのうち、許可型ブロックチェーンに該当する。非許可型ブロックチェーンでは、ブロックチェーンネットワークに誰でも参加することができる。そのため、ブロックチェーンネットワークの参加者間でトランザクションを承認する際に、悪意あるユーザがアクセスする可能性があり、複雑なコンセンサスアルゴリズムを採用する必要がある。[37]

表 4.1: 想定する属性情報 [14]

属性情報
Family Name
Given Name
Date of Birth
Date of Issue
Date of Expiry
Issuing Country
Facial Biometric template
Normal Pplace of residence
Resident City
Age attestation: 18+
Age attestation: 20+
Age attestation: 60+
Family Name at birth
Given Name at birth
Place of Birth
Gender
Police
Medical doctor
Government officer
Rescue

一方で、許可型ブロックチェーンはトランザクションに対する承認プロセスは複数の限られた組織が実施する。よって、非許可型ブロックチェーンのような負荷の大きいコンセンサスプロセスを使用せず、軽量なコンセンサスプロセスを採用している。したがって、許可型ブロックチェーンはトランザクションに対する承認プロセスの遅延時間やスループットなどのシステム性能が優れている。以上の理由により、本論文では、データ共有システムに許可型ブロックチェーンの代表とされる Hyperledger Fabric の適用を検討する。

4.3 ブロックチェーンを活用したデータ共有システムの性能評価

ブロックチェーンを IoT やスマートシティ領域にて活用するためには、データ量の大きさや高頻度で発生するトランザクションの場合において、スケーラビリティの問題が言われている。

そこで本稿では、4.1 節で述べた二つのユースケースについて、ブロックチェーン技術の一つである Hyperledger Fabric をデータ共有システムとして利用した場合、ユースケースに耐えうる十分な性能が得られるか、まずは初期実験からその Hyperledger Fabric の適用の妥当性について、性能評価実験から議論する。

4.3.1 データ共有システムへ様々なデータを登録した場合の性能評価

Hyperledger Fabric をデータ共有システムとして利用した場合において、様々なデータサイズのデータを利用し、Hyperledger Fabric の性能評価を実機実験から検証する。そして、この実験結果から、4.1.1 節で述べた人の属性情報や IoT データをデータ共有システムへ登録する利用例において、データ共有システムの限界性能について議論する。

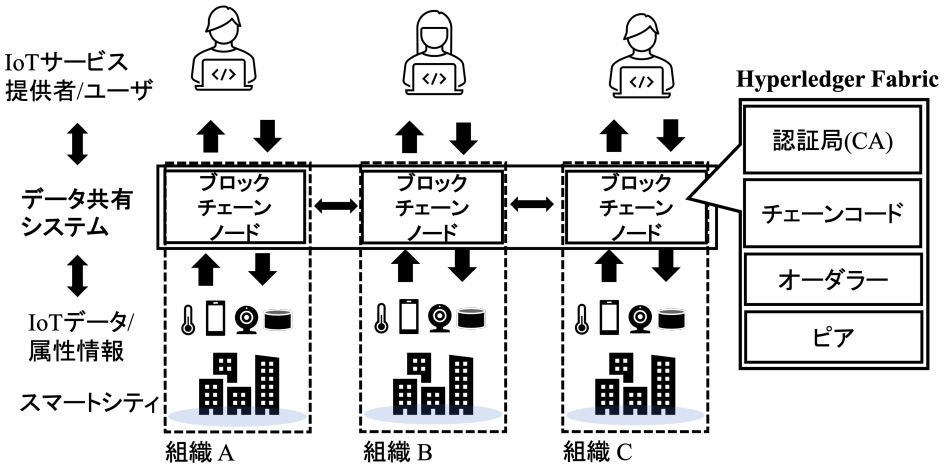

図 4.2: ブロックチェーンを用いたデータ共有システム [26]

実験環境及び実験シナリオ

本実験は、図 4.3 に示すように、3 台の小型 PC（Intel NUC）によってブロックチェーンネットワークを実現しており、このネットワークは Docker Swarm によって構築されている。その内の 2 台は、組織 (組織 A と組織 B) を模擬しており、残りの一台を Hyperledger Caliper とクライアントアプリケーションとして実験を行なった。ピア、オーダーラーと Hyperledger Caliper は Docker コンテナ上で実行している。

本実験環境は、[18] のサイトを参考に構築した。[18] では、図 4.4 のように 4 つのホストでブロックチェーンネットワークを Docker Swarm によって構築している。4 つのホストにて、ピア、オーダラーをそれぞれのホストで yaml ファイルからドッカーコンテナ上で起動している。

これに対して、本実験環境は、[18] での実行環境よりホストが一台少なく、CLI を起動しているホストでは、オーダラーが一台少なく、ピアは起動していない。従って、本実験は 3 つのホストで [18] での yaml ファイルを図 4.5 に示すように、起動しないコンテナを削除し、修正することによってブロックチェーンネットワークを構築した。

本実験では、事前にユーザーの認証を行なっていることを想定しているため、CA は起動せずに実験している。そのため、証明書と秘密鍵は予め持っている状態としており、トランザクションに証明書と秘密鍵をハードコーディングしている。Hyperledger Caliper に登録するデータは、180B の人の属性情報 (人の名前と苗字)[14]、250KB と 419KB の PNG 画像データ、1.3MB と 3.7MB の 3 次元の点群データを用いた。

登録する人の属性情報は 3.2 節で利用しているものを想定する。想定する属性情報を表 4.1[14] にまとめた。表 4.1 のように、属性情報には人の名前や苗字だけでなく、生年月日や居住地、顔の生体認証情報などがある。これら属性情報は、例えば、「keisuke」という名前の値そのものを登録しているのではなく、「keisuke」という値に対して乱数をかけたハッシュ値を属性情報と定義し、データ共有システムに登録する。3 次元の点群データは RealSense L515 で撮影した研究室内の空間データと椅子のオブジェクトデータである。

Hypeledger Fabric の公式のサンプルのアプリケーションである fabcar アプリを修正して、人の属性情報、画像データと 3 次元の点群データを Hyperledger Caliper に登録した。fabcar アプリでは、車の車種、持ち主、色などを扱うアプリケーションとなっており、Node.js で記述されているクライアントアプリケーションとチェーンコードを図 4.6 と図 4.7 のように修正した。図 4.6 に示すようにクライアントアプリケーションでは、車の情報を定義している箇所に対し、画像データや 3 次元の点群データを base64 にエンコードするよう修正した。チェーンコードはクライアントから受け取った

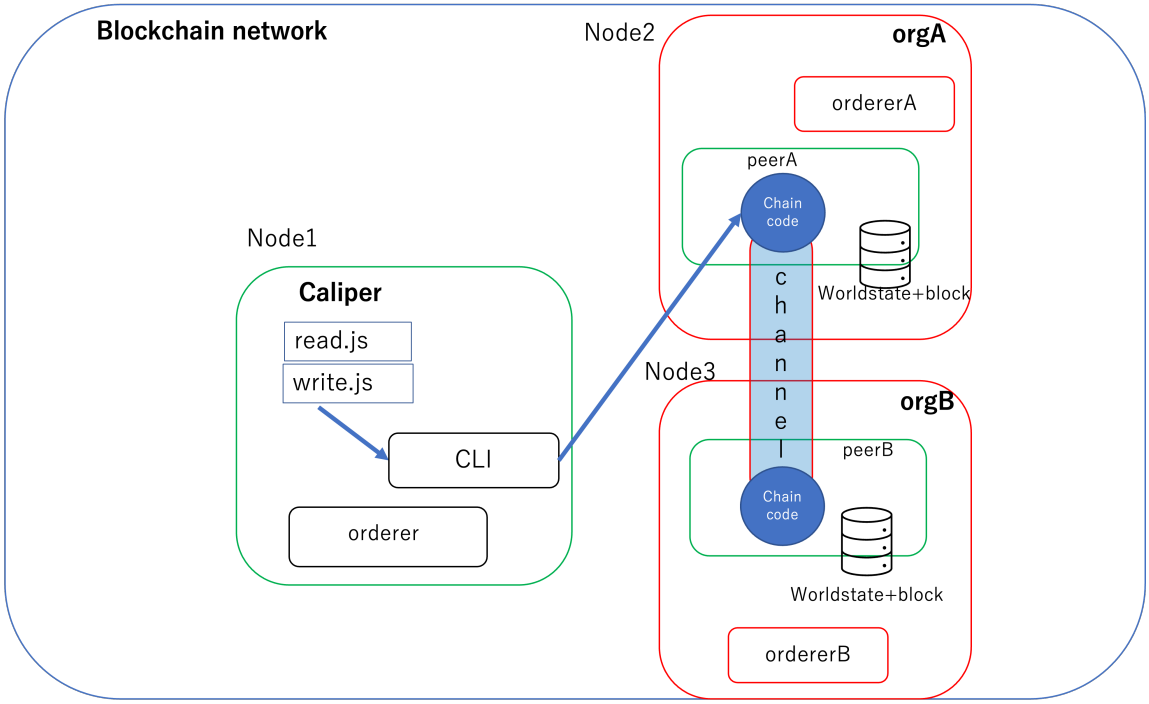

図 4.3: 本実験環境

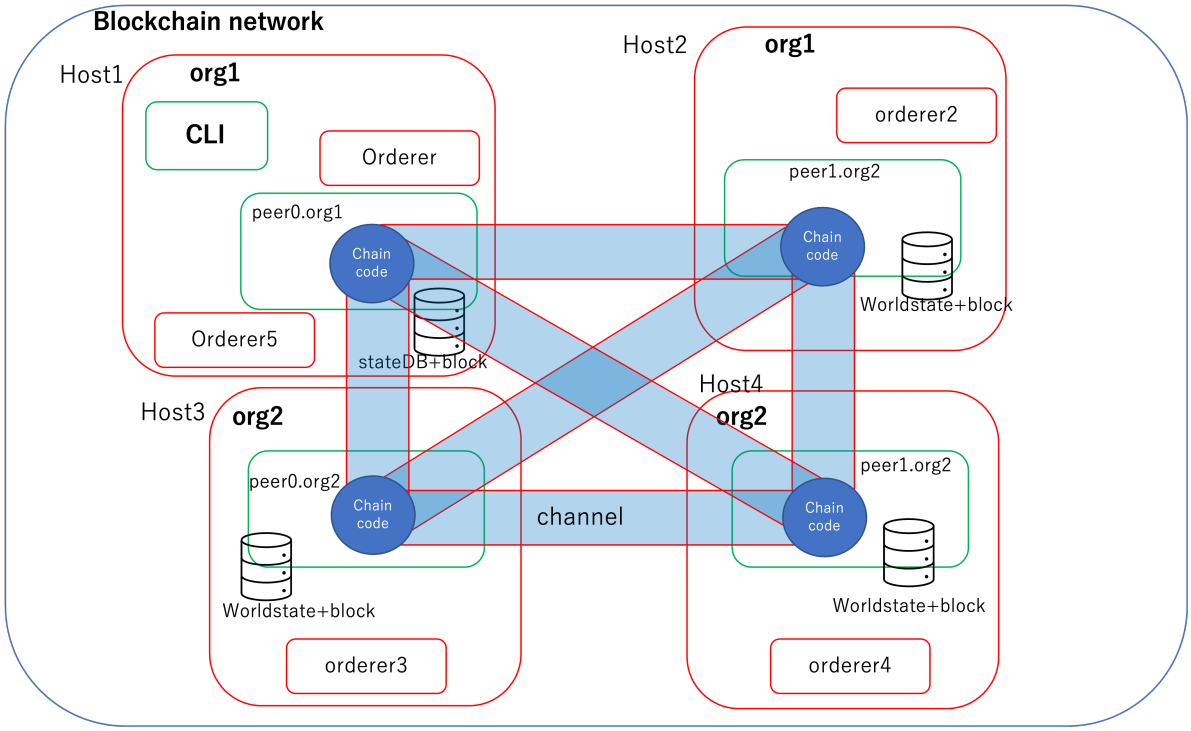

図 4.4: 参考にした実験環境

```
# Copyright IBM Corp. All Rights Reserved.
#
# SPDX-License-Identifier: Apache-2.0
#

version: '2'

volumes:
  orderer.example.com:

networks:
  byfn:
    external:
      name: first-network

services:

  orderer.example.com:
    extends:
      file:   base/docker-compose-base.yaml
      service: orderer.example.com
    container_name: orderer.example.com
    networks:
      - byfn


  cli:
    container_name: cli
    image: hyperledger/fabric-tools:$IMAGE_TAG
    tty: true
    stdin_open: true
    environment:
      - SYS_CHANNEL=$SYS_CHANNEL
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      #- FABRIC_LOGGING_SPEC=DEBUG
      - FABRIC_LOGGING_SPEC=INFO
      - CORE_PEER_ID=cli
      - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
      - CORE_PEER_LOCALMSPID=Org1MSP
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
      - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: /bin/bash
    volumes:
        - /var/run/:/host/var/run/
        - ./../chaincode/:/opt/gopath/src/github.com/chaincode
        - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
        - ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/
        - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts
    depends_on:
      - orderer.example.com
            #- peer0.org1.example.com
    networks:
      - byfn
```

図 4.5: 修正した yaml ファイル

```
async submitTransaction() {
    this.txIndex++;
    let dataNumber = 'Client' + this.workerIndex + '_IoT' + this.txIndex.toString();
    //let carColor = colors[Math.floor(Math.random() * colors.length)];
    //let carMake = makes[Math.floor(Math.random() * makes.length)];
    //let carModel = models[Math.floor(Math.random() * models.length)];
    //let carOwner = owners[Math.floor(Math.random() * owners.length)];
    const filePath = "benchmarks/samples/fabric/fabcar/test8.png";
    //const filePath = "benchmarks/samples/fabric/fabcar/2021_10_22__14_27_40.ply"
    const fs = require('fs');
    const data = fs.readFileSync(filePath, { encoding: "base64" });

    let args = {
        contractId: 'fabcar',
        contractVersion: 'v1',
        contractFunction: 'createData',
        contractArguments: [dataNumber,data],
        timeout: 30
    };

    await this.sutAdapter.sendRequests(args);
}
```

図 4.6: 修正したクライアントアプリケーション

```
async createDara(ctx, dataNumber, data) {
    const { performance } = require('perf_hooks');
    var startTime = performance.now();
    console.info('============= START : Create Data  ===========');

    const tx = {
        "device": "sensor",
        "docType": "IoTdevice",
        "image": data,
    };

    await ctx.stub.putState(dataNumber, Buffer.from(JSON.stringify(tx)));
    console.info('============= END : Create Data ===========');
    var endTime = performance.now();
    console.log(endTime - startTime);
}
```

図 4.7: 修正したチェーンコード

データを台帳に登録する処理が記述されている。図 4.7 に示すようにクライアントから受け取った車の情報を定義している箇所を画像データや 3 次元の点群データであることを示すように修正した。

本実験では、2 章で述べた 3 つのフェーズである、実行フェーズ、オーダリングフェーズ、バリデーションフェーズを含む一つのトランザクション応答時間を測定する。また、クライアントがピア A にトランザクションを送信してから結果が帰ってくるまでの応答時間を Hyperledger Caliper を使用して測定している。なお、本実験の試行回数は 5 回行い、平均値をとることとする。

Hyperledger Caliper では、トランザクションを送信してから帰ってくるまでの応答時間、スループットの測定のみしかサポートしておらず、3 つのフェーズについて応答時間やスループットを測定し、分析することはできない。そのため、これら 3 つのフェーズの応答時間はピアとオーダラーが生成するログメッセージを参照することで計測した。ピア A のログメッセージ、ピア B のログメッセージ、オーダラーのログメッセージ、リーダーになっているオーダラーのログメッセージをそれぞれ図 4.8 から図 4.11 に示す。また、図 4.8 から図 4.11 のログメッセージを参考にトランザクションフローの様子をシーケンス図で表したものを図 4.12 に示す。

本実験では、図 4.12 のシーケンス番号 1 から 4.2 までのフローを実行フェーズの応答時間、図 4.12 のシーケンス番号 6.1 から 8.2 までをオーダリングフェーズの応答時間、図 4.12 のシーケンス番号 9.1 から 10.2 までを検証フェーズの応答時間としている。

まず初めに、図 4.12 のシーケンス番号 1 から 4.2 までの実行フェーズの応答時間について説明する。図 4.8 の「INFO 100」と記述されている箇所で grpc サーバーによってクライアントとピア A で通信が行われ、エンドースメントポリシーが確認されている。これは図 4.12 のシーケンス番号 1 に該当する。図 4.8 の「INFO 101」と「INFO 102」、図 4.9 の「INFO 077」と「INFO 078」において、クライアントから各ピアにチェーンコード実行依頼を送信し、チェーンコードを実行したことを

```
2021-12-09 12:53:45.138 UTC [comm.grpc.server] 1 -> INFO 100 unary call completed grpc.service=discovery.Dis
covery grpc.method=Discover grpc.peer_address=10.0.10.131:42086 grpc.peer_subject="CN=fabric-common" grpc.co
de=OK grpc.call_duration=5.424707ms
2021-12-09 12:53:45.254 UTC [comm.grpc.server] 1 -> INFO 101 unary call completed grpc.service=discovery.Dis
covery grpc.method=Discover grpc.peer_address=10.0.10.131:42086 grpc.peer_subject="CN=fabric-common" grpc.co
de=OK grpc.call_duration=6.657248ms
2021-12-09 12:53:45.356 UTC [endorser] callChaincode -> INFO 102 finished chaincode: fabcar duration: 40ms c
hannel=mychannel txID=0c8d09db
2021-12-09 12:53:45.362 UTC [comm.grpc.server] 1 -> INFO 103 unary call completed grpc.service=protos.Endors
er grpc.method=ProcessProposal grpc.peer_address=10.0.1.29:41070 grpc.peer_subject="CN=fabric-common" grpc.c
ode=OK grpc.call_duration=62.838235ms
2021-12-09 12:53:45.760 UTC [gossip.privdata] StoreBlock -> INFO 104 Received block [10] from buffer channel
=mychannel
2021-12-09 12:53:45.797 UTC [committer.txvalidator] Validate -> INFO 105 [mychannel] Validated block [10] in
 36ms
2021-12-09 12:53:45.843 UTC [kvledger] CommitLegacy -> INFO 106 [mychannel] Committed block [10] with 1 tran
saction(s) in 41ms (state_validation=3ms block_and_pvtdata_commit=25ms state_commit=8ms) commitHash=[1b4e8a0
b8a59cb5b071a57307bc80c9a32a9f3099574d63ab89672d18ae1f0b3]
```

図 4.8: ピア A のログメッセージ

```
2022-01-07 07:38:38.947 UTC [comm.grpc.server] 1 -> INFO 077 unary call completed grpc.service=discovery.Discov
ery grpc.method=Discover grpc.peer_address=10.0.10.131:40694 grpc.peer_subject="CN=fabric-common" grpc.code=OK
grpc.call_duration=2.520481ms
2022-01-07 07:38:38.993 UTC [endorser] callChaincode -> INFO 078 finished chaincode: fabcar duration: 28ms chan
nel=mychannel txID=a88d92ef
2022-01-07 07:38:38.993 UTC [comm.grpc.server] 1 -> INFO 079 unary call completed grpc.service=protos.Endorser
grpc.method=ProcessProposal grpc.peer_address=10.0.1.91:37628 grpc.peer_subject="CN=fabric-common" grpc.code=OK
 grpc.call_duration=30.056618ms
2022-01-07 07:38:39.070 UTC [gossip.privdata] StoreBlock -> INFO 07a Received block [6] from buffer channel=myc
hannel
2022-01-07 07:38:39.073 UTC [committer.txvalidator] Validate -> INFO 07b [mychannel] Validated block [6] in 2ms
2022-01-07 07:38:39.150 UTC [kvledger] CommitLegacy -> INFO 07c [mychannel] Committed block [6] with 1 transact
ion(s) in 76ms (state_validation=0ms block_and_pvtdata_commit=71ms state_commit=2ms) commitHash=[a7496acec676d5
7608f73510372a7560293daa50bf6808825589649d3f877978]
```

図 4.9: ピア B のログメッセージ

```
2021-12-02 11:26:33.450 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 062 Writing block [12] (Raft index: 16) to le
dger channel=mychannel node=3
2021-12-02 11:31:03.993 UTC [comm.grpc.server] 1 -> INFO 063 streaming call completed grpc.service=orderer.AtomicBroadcas
t grpc.method=Broadcast grpc.peer_address=10.0.1.70:58488 grpc.peer_subject="CN=fabric-common" grpc.code=OK grpc.call_dur
ation=35.850082ms
2021-12-02 11:31:04.176 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 064 Writing block [13] (Raft index: 17) to le
dger channel=mychannel node=3
2021-12-02 11:34:36.138 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 065 Writing block [14] (Raft index: 18) to le
dger channel=mychannel node=3
2021-12-02 11:38:04.856 UTC [comm.grpc.server] 1 -> INFO 066 streaming call completed grpc.service=orderer.AtomicBroadcas
t grpc.method=Broadcast grpc.peer_address=10.0.1.72:38564 grpc.peer_subject="CN=fabric-common" grpc.code=OK grpc.call_dur
ation=17.580932ms
2021-12-02 11:38:04.989 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 067 Writing block [15] (Raft index: 19) to le
dger channel=mychannel node=3
2021-12-02 11:40:56.012 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 068 Writing block [16] (Raft index: 20) to le
dger channel=mychannel node=3
2021-12-02 11:44:44.981 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 069 Writing block [17] (Raft index: 21) to le
dger channel=mychannel node=3
```

図 4.10: オーダラーのログメッセージ

```
2022-01-07 07:31:25.618 UTC [orderer.consensus.etcdraft] propose -> INFO 05b Created block [5], ther
e are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:31:25.629 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 05c Writing block [5] (R
aft index: 9) to ledger channel=mychannel node=2
2022-01-07 07:38:39.055 UTC [orderer.consensus.etcdraft] propose -> INFO 05d Created block [6], ther
e are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:38:39.067 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 05e Writing block [6] (R
aft index: 10) to ledger channel=mychannel node=2
2022-01-07 07:44:27.290 UTC [orderer.consensus.etcdraft] propose -> INFO 05f Created block [7], ther
e are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:44:27.303 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 060 Writing block [7] (R
aft index: 11) to ledger channel=mychannel node=2
2022-01-07 07:46:33.627 UTC [orderer.consensus.etcdraft] propose -> INFO 061 Created block [8], ther
e are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:46:33.640 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 062 Writing block [8] (R
aft index: 12) to ledger channel=mychannel node=2
2022-01-07 07:49:23.696 UTC [orderer.consensus.etcdraft] propose -> INFO 063 Created block [9], ther
e are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:49:23.708 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 064 Writing block [9] (R
aft index: 13) to ledger channel=mychannel node=2
2022-01-07 07:51:33.074 UTC [orderer.consensus.etcdraft] propose -> INFO 065 Created block [10], the
re are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:51:33.087 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 066 Writing block [10] (
Raft index: 14) to ledger channel=mychannel node=2
2022-01-07 07:54:41.277 UTC [orderer.consensus.etcdraft] propose -> INFO 067 Created block [11], the
re are 0 blocks in flight channel=mychannel node=2
2022-01-07 07:54:41.308 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 068 Writing block [11] (
Raft index: 15) to ledger channel=mychannel node=2
```

図 4.11: リーダーとなっているオーダラーのログメッセージ

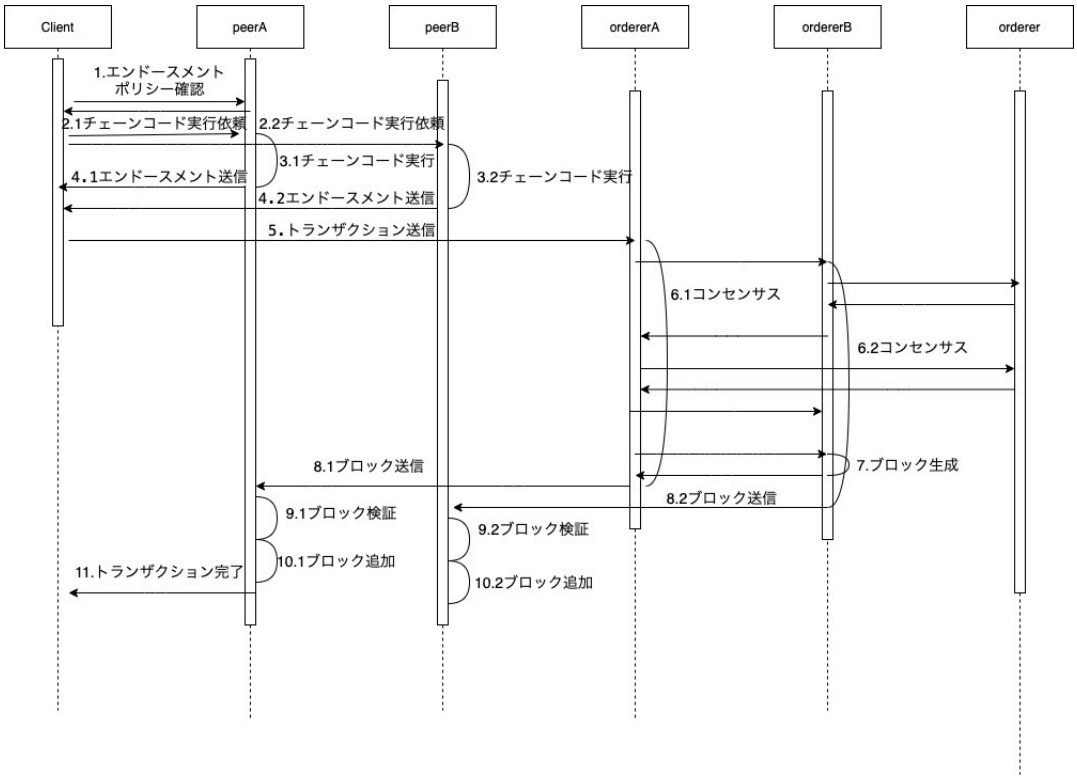

図 4.12: ピアとオーダラーのログメッセージを参照にしたトランザクションフロー

表すため、図 4.12 のシーケンス番号 2.1 から 4.2 に該当する。ここまでが検証フェーズの応答時間である。

次に図 4.12 のシーケンス番号 6.1 から 8.2 までのオーダリングフェーズの応答時間について説明する。図 4.10 の「INFO 063」はオーダラーがトランザクションを受け取ることを示しており、このトランザクションを受け取るオーダラーはランダムになっている。トランザクションのコンセンサスが行われ、オーダラー間でトランザクションのやりとりが行われる。そして、オーダラーのリーダーが図 4.11 の「INFO 05d」のようにブロックを生成すると、図 4.10 の「INFO 063」と図 4.11 の「INFO 05e」のように各オーダラーはブロックをピアに送信する。トランザクションを受け取ってからブロックを送信するまでが図 4.12 のシーケンス番号 6.1 から 7 までのトランザクションのコンセンサスを示す。図 4.12 のシーケンス番号 8.1 と 8.2 のブロックの送信時間は、図 4.10 の「INFO 063」のようにブロックを送信した時の時間と図 4.9 の「INFO 07a」のようにピアが受け取った時間の差分としている。ここまでがオーダリングフェーズの応答時間である。

最後に、図 4.12 のシーケンス番号 9.1 から 10.2 までの検証フェーズの応答時間について説明する。ピアはブロックを受け取ると図 4.9 の「INFO 07b」のようにブロックを検証する。そして、図 4.9 の「INFO 07c」のようにブロックをコミットしてブロックチェーンと World State を更新する。ここまでを実行フェーズの応答時間として定義する。

評価結果

図 4.13[26] は、各データサイズを Hyperledger Fabric へ登録する際の応答時間の結果である。

図 4.13 より、データサイズが大きくなると、全体の応答時間、各フェーズ応答時間が大きくなっていることがわかる。この原因は、図 4.12 に示すように、全体の応答時間、各フェーズの応答時間にはトランザクションやブロックの伝送遅延も含んでいるためである。特にオーダリングフェーズは、コンセンサス時に orderer 間でトランザクションのやりとりが行われ、ブロックをピアに送信するため、伝送遅延が大きくオーダリングフェーズの応答時間に影響すると考える。

人の属性情報や IoT データをデータ共有システムへ登録するユースケースにおいて、図 4.13 より、本実験環境下では 180B の人の属性情報を登録するトランザクションの応答時間は約 200ms となっている。そのため、もしユーザが属性情報を登録した後、認証を行う必要がある場合、大きな待ち時間は発生せずに認証することができる。図 4.13 より、本環境下では 1MB を超える IoT データをデータ共有システムに登録する場合、約 1 秒の時間がかかる。そのため、登録する発生頻度が増えるとスループットが低下し、遅延時間が大きくなることが予測され、1MB を超える IoT データ扱うのは困難であると考える。したがって、想定するデータ共有システムでは最大で 419KB ぐらいの IoT データを扱うことを推奨する。

4.3.2　データ共有システムからのデータを呼び出す場合の性能評価

次に、Hyperledger Fabric を用いたデータ共有システムからデータを呼び出す場合について、実機評価を行う。そして、この実験結果から、データ共有システムに登録されている IoT データを活用する利用例において、データ共有システムが持つ限界性能について、議論する。

実験環境及び実験シナリオ

実験環境は 4.3.1 節で述べたものとおおよそ同じである。違う点としては、Hyperledger Fabric に対して登録されているデータを呼び出すため、あらかじめデータを登録していることである。あらかじめ登録しているデータは 4.3.1 節と同じ 180B の人の属性情報、250KB と 419KB の PNG 画像データ、1.3MB と 3.7MB の 3 次元の点群データである。

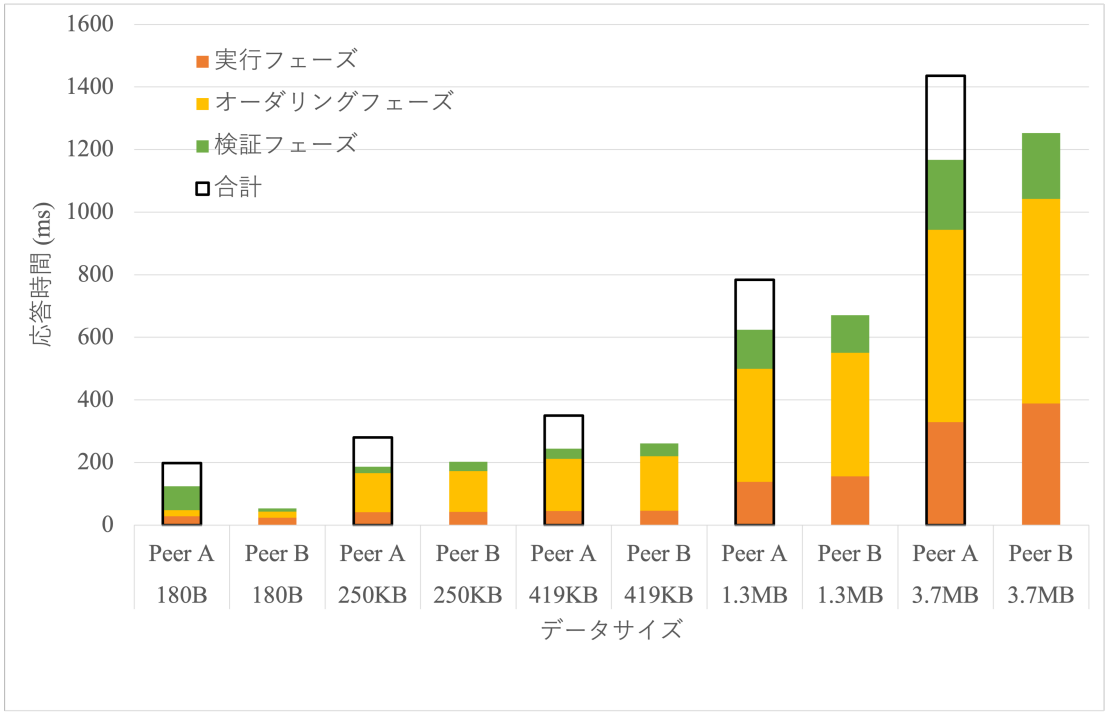

図 4.13: Hyperledger Fabric へ登録する場合の平均応答時間 [26]

Hyperledger Fabric からデータを呼び出す際の性能評価実験では、異なるデータサイズに対して、トランザクションの発生頻度を変化させて、Hyperledger Caliper を用いてトランザクションの平均応答時間およびスループットを測定する。本実験では、トランザクションに対してオーダリングフェーズとバリデーションフェーズは実行されず、Hyperledger Fabric の World State から読み込む処理のみとなっている。なお、本実験の試行回数は 5 回行い、平均値をとることとする。

評価結果

図 4.14 から図 4.18 に各データサイズにおける Hyperledger Fabric からデータを呼び出す場合の平均応答時間とスループットのグラフを示す。また、表 4.2 に平均応答時間と最大スループットをまとめたものを示す。図 4.14 から図 4.18 より、データ共有システムからデータを呼び出す場合は、オーダリングフェーズとバリデーションフェーズが省かれているため、データ共有システムへ登録する場合に比べて、スケーラビリティが大きく緩和されていることがわかる。各データサイズにて、スループットの最大値を超えると、トランザクションがキューに溜まり始めるため、トランザクションの応答時間が大幅に大きくなっていることがわかる。表 4.2 より、大きいデータサイズを呼び出す程、スループットは小さくなっていることがわかる。これは、データが大きくなるほど World State にあるデータの検索時における負荷と、ピアからクライアントに結果を送信する負荷が大きくなるからだと考える。

表 4.2 のスループットは、一秒間に処理できるトランザクション数を示している。そのため、データ共有システムに登録されている IoT データを活用するユースケースでは、表 4.2 のスループットの小数を切り捨てた値は、一秒間にデータ共有システムに 1 回のトランザクションを送信できるユーザの数と置き換えることができる。

データ共有システムに登録されているデータを読み込む場合の、トランザクションを送信してから結果が帰ってくるまでの平均応答時間と一秒間にデータ共有システムに 1 回のトランザクションを送信できるユーザの数を 4.3 に示す。表 4.3 より、データ共有システムに登録されている人の属性情

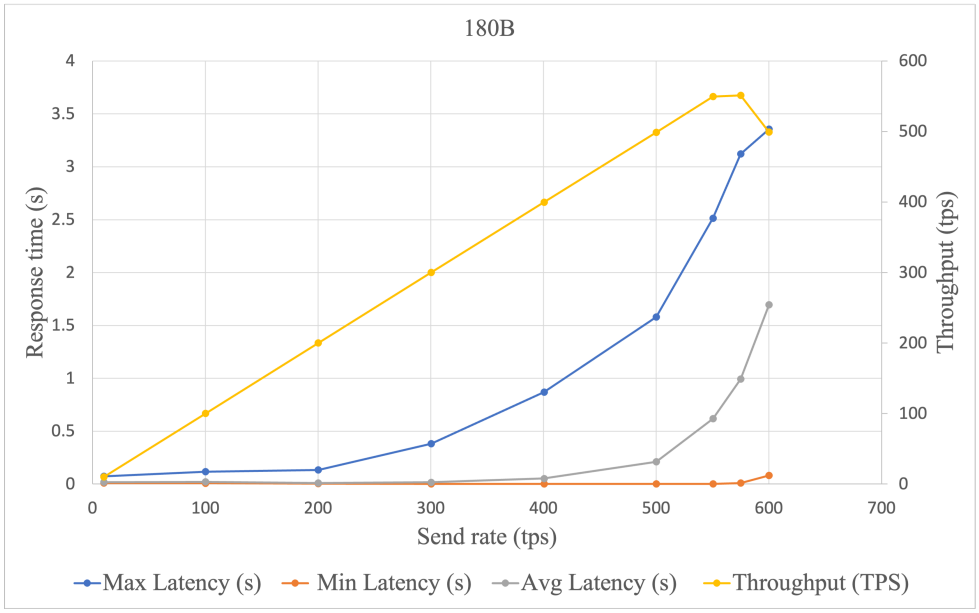

図 4.14: 想定するデータ共有システムから 180B のデータを呼び出す場合の平均応答時間とスループット

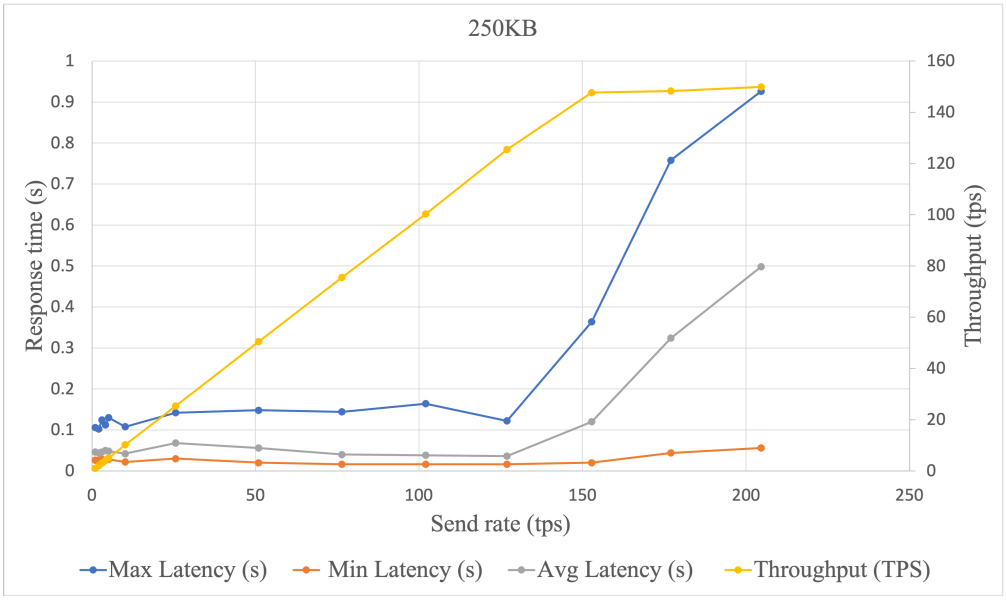

図 4.15: Hyperledger Fabric から 250KB のデータを呼び出す場合の平均応答時間とスループット

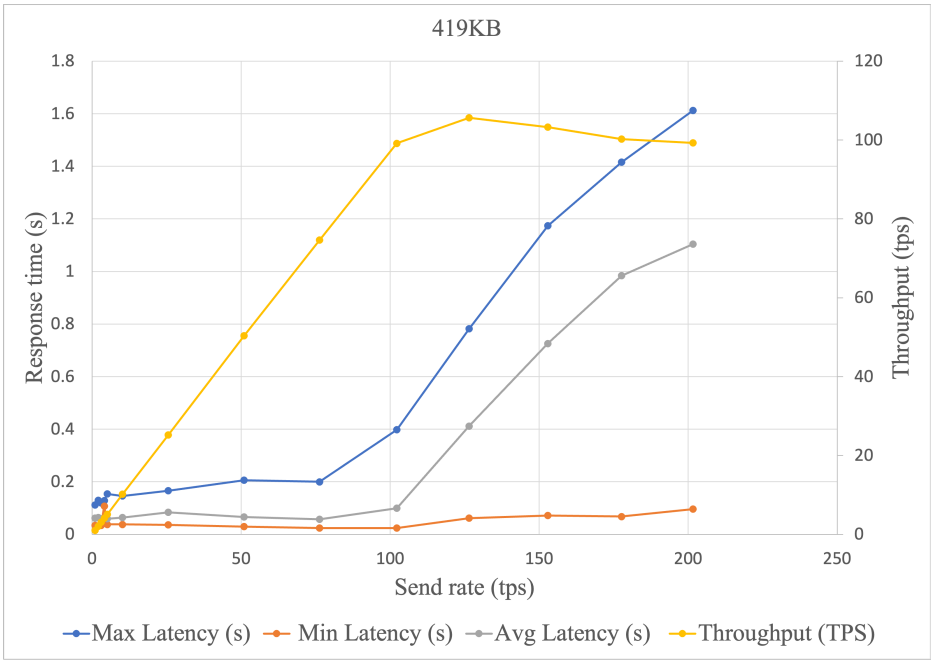

図 4.16: Hyperledger Fabric から 419KB のデータを呼び出す場合の平均応答時間とスループット

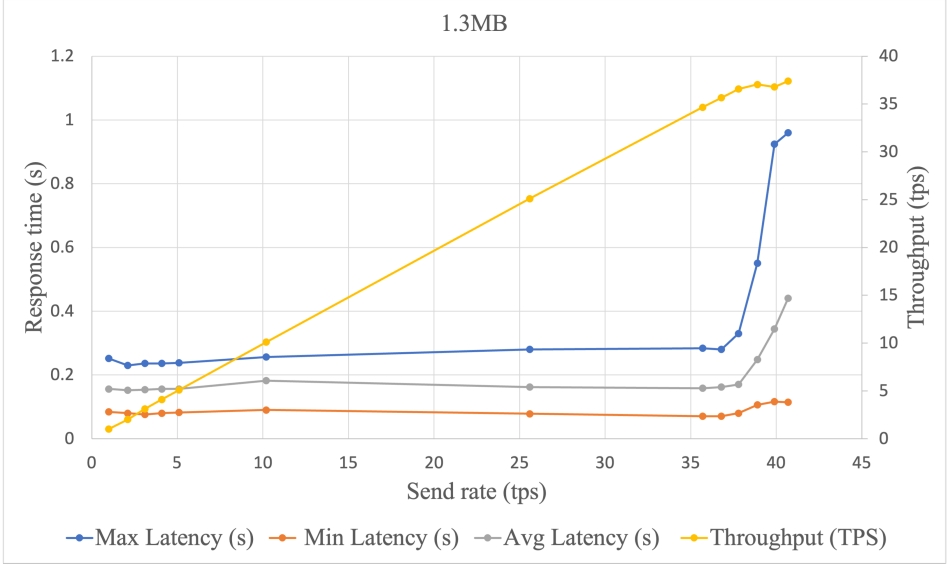

図 4.17: Hyperledger Fabric から 1.3MB のデータを呼び出す場合の平均応答時間とスループット

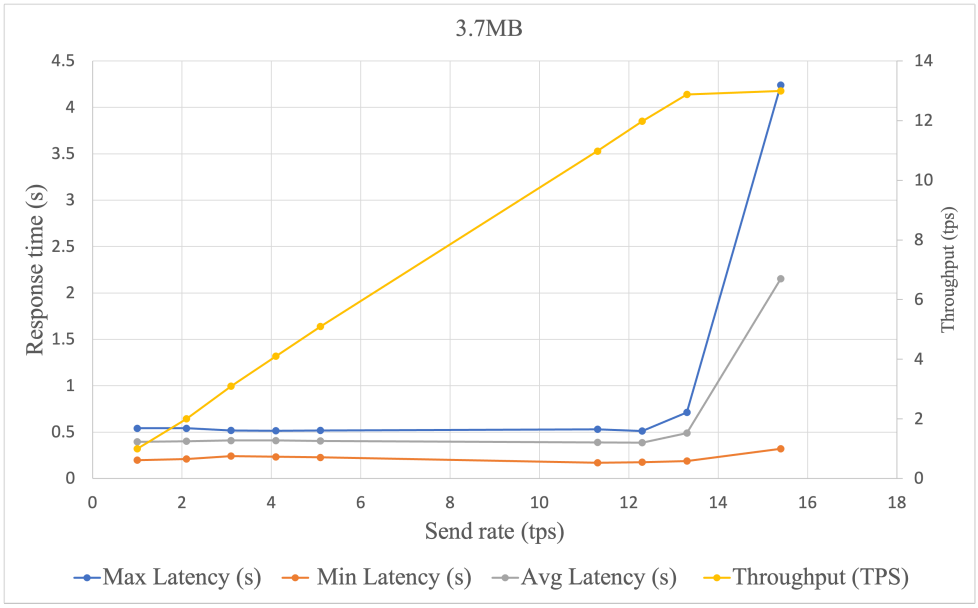

図 4.18: Hyperledger Fabric から 3.7MB のデータを呼び出す場合の平均応答時間とスループット

表 4.2: データの呼出しにおける Hyperledger Fabric のパフォーマンス [26]

Data size	Average response time (s)	Throughput (tps)
180B	0.618	549.52
250KB	0.12	147.68
419KB	0.1	99.16
1.7MB	0.17	36.56
3.7MB	0.49	12.88

報や IoT データを読み込む場合、データサイズが大きくなるほど、データ共有システムに一秒間にアクセスできる人数が小さくなっていることがわかる。したがって、データ共有システムに役 200B の IoT データが登録されている場合、一秒間に役 550 人のユーザがデータ共有システムにトランザクションを送信することができる。

表 4.3: 想定するデータ共有システムからデータを呼び出す場合の性能

Data size	Average response time (s)	User
180B	0.618	549
250KB	0.12	147
419KB	0.1	99
1.7MB	0.17	36
3.7MB	0.49	12

4.4　データ共有システムを用いた動画のライブ配信に関する検討

これまでの節において、ブロックチェーンをデータ共有システムへ適用する場合について、主に議論し、データ共有システムの性能評価を行ってきた。これらの性能評価の結果を踏まえて、IoT とは

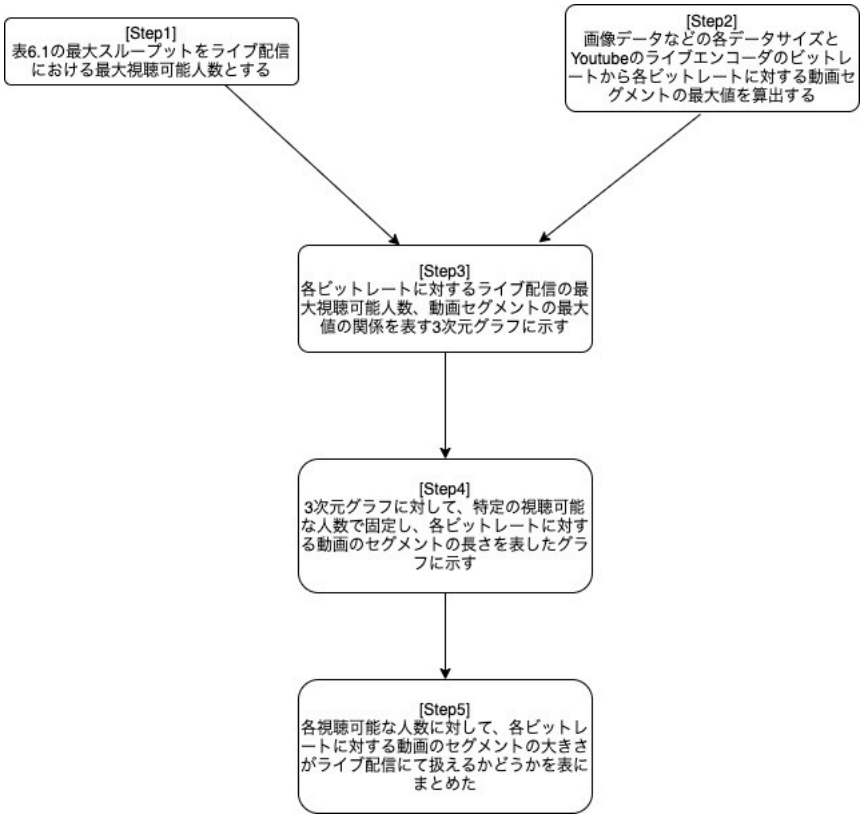

図 4.19: ライブ配信におけるデータ共有システムの性能評価方法

別のユースケースとして、映像配信への適用について、本節で追加検討する。

具体的に、図 4.20 のようなブロックチェーンによるデータ共有システムを用いた動画のライブ配信を想定する。図 4.20 のように、ライブ配信では、ライブ配信者は動画をセグメントにしてデータ共有システムに送信している。このライブ配信で、ライブ視聴者が滞りなく視聴するには、ライブ配信者が動画をデータ共有システムにアップロードする時間とライブ視聴者がデータ共有システムから動画をダウンロードする時間を足した時間が動画のセグメントの大きさより小さい値になる必要がある。想定するデータ共有システムでライブ視聴者が滞りなく視聴するために、4.3 節の実験結果と YouTube[8] のライブエンコーダのビットレートを参考に、ライブ配信で扱うことができる動画のセグメントの大きさ、ライブの最大視聴可能人数について求め、ライブ配信におけるデータ共有システムの評価を行う。

ライブ配信の最大視聴人数、ライブ配信で扱うことができる動画のセグメントの長さを求めるフローチャートについて図 4.19 に示す。このフローチャートに従い、ライブ配信におけるデータ共有システムの性能評価を行う。

表 4.2 のスループットは各データサイズにおける 1 秒間あたりに処理できる最大トランザクション数を示しており、図 4.19 のステップ 1 に示すように、小数を切り捨てた値を各データサイズにおける最大ライブ配信視聴人数とする。前節で評価してきたデータサイズをビットレートに変換した値、各ビットレートにおけるライブ配信の最大視聴人数を表 4.4 に示す。また、参考にする YouTube のライブエンコーダの各解像度におけるビットレートを表 4.5 に示す。

次に、図 4.19 のステップ 2 に示すように、各ビットレートサイズにおける動画セグメントの最大値を求める。250KB 以上の画像データや 3 次元の点群データそれぞれの単位をビットレートに変換した値を D_{movie}、表 4.5 の YouTube のライブエンコーダの各ビットレーを $D_{youtube}$、各ビットレートに対する動画セグメントの最大値を $T_{maxlive}$ と定義すると、$T_{maxlive}$ は式 (6.1) で表わせる。

表 4.4: 各ビットレートにおけるライブ配信の最大視聴人数

データサイズ	ビットレート (Kbps)	最大視聴人数 (人)
250KB	2000	147
419KB	3352	99
1.7MB	10400	36
3.7MB	29600	12

表 4.5: YouTube のライブエンコーダの各解像度におけるビットレート

解像度	Youtube の動画ビットレート (Kbps)
720P	1500
1080p	3000
1080p 60fps	4500
1440p 30fps	6000
1440p 60fps	9000
2160p(4k) 30fps	13000
2160p(4k) 60fps	20000

$$T_{maxlive} = D_{movie}/B_{youtube} \tag{4.1}$$

式 (6.1) により、各ビットレートに対する動画セグメントの最大値を求めた。図 4.19 のステップ 3 に示すように、求めた各ビットレートに対するライブ配信の最大視聴人数、動画セグメントの最大値の関係を表す 3 次元グラフを図 4.21 に示す。図 4.21 より、ビットレートおよびライブ配信の最大視聴人数が小さくなるほど動画セグメントの値は大きくなっていることがわかる。また、図 4.21 の青点がライブ配信可能、赤点はライブ配信ができないビットレート、最大視聴人数、動画セグメントの大きさを示している。

図 4.19 のステップ 4 に示すように、図 4.21 に対し、例えば、視聴可能な人数を 36 人で固定し、各ビットレートに対する動画のセグメントの大きさを表したグラフを図 4.22 に示す。図 4.13 に示す 250KB 以上のデータを Hyperledger Fabric に登録する応答時間を、想定するデータ共有システムに動画のデータをアップロード時間と仮定する。また、表 4.2 の Hyperledger Fabric からデータを読み込む際にかかる応答時間を、想定するデータ共有システムから動画をダウンロードする応答時間と仮定する。上記で示したアップロードする時間を T_{upload}、動画をダウンロードする時間を $T_{download}$ と定義する。また、ライブ配信で必要な最小の動画のセグメントの大きさを $T_{minimumlive}$ と定義すると、以下の式 (6.2) で $T_{minimumlive}$ が表される。

$$T_{minimumlive} = T_{upload} + T_{download} \tag{4.2}$$

図 4.22 の閾値は $T_{minimumlive}$ を示している。図 4.22 より、この閾値より大きい値がライブ配信にて扱える動画のセグメントの大きさである。そのため、閾値と閾値より大きい各ビットレートに対する動画セグメントの大きさとの間の領域がライブ配信にて扱える動画のセグメントの大きさを示している。図 4.22 の場合、1500 から 9000 までのビットレートであれば、動画セグメントの大きさが閾値より大きくなっているため、ライブ配信に扱うことができることがわかる。

図 4.19 のステップ 5 に示すように、各視聴可能な人数に対して、各ビットレートに対する動画のセグメントの大きさがライブ配信にて扱えるかどうかをまとめたものを表 4.6 に示す。表 4.6 より、高品質な動画を扱うライブ配信を行う場合は、視聴人数が 10 人程でなければ視聴できない。また、

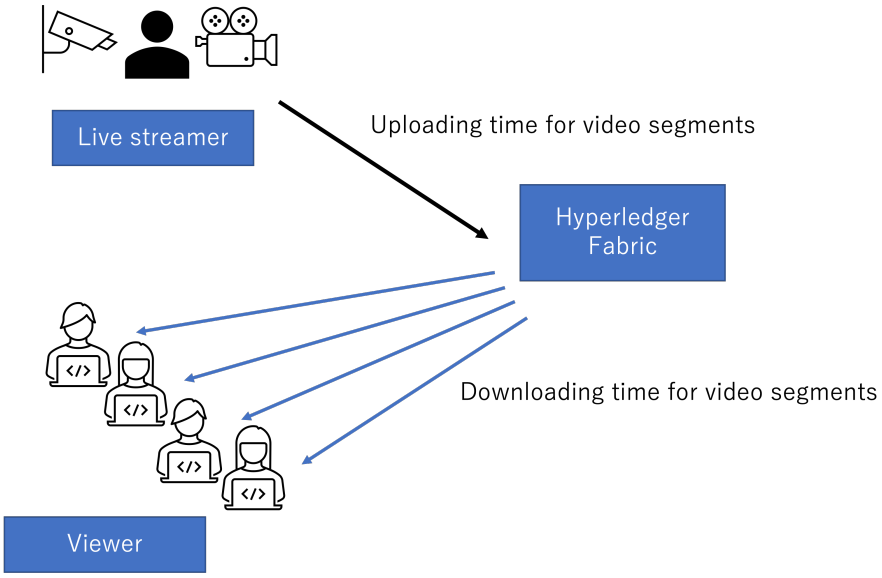

図 4.20: データ共有システムを用いたライブ配信

1080p の解像度で行う場合は、一つのピアに対して最大視聴可能人数は 147 人でライブ配信を行えることがわかる。

表 4.6: 各ビットレートおよび各最大視聴人数におけるライブ配信の可不可

最大視聴人数 [人] / ビットレート [Kbps]	147	99	36	12
1500	○	○	○	○
3000	○	○	○	○
4500	○	○	○	○
6000	×	○	○	○
9000	×	×	○	○
13000	×	×	×	○

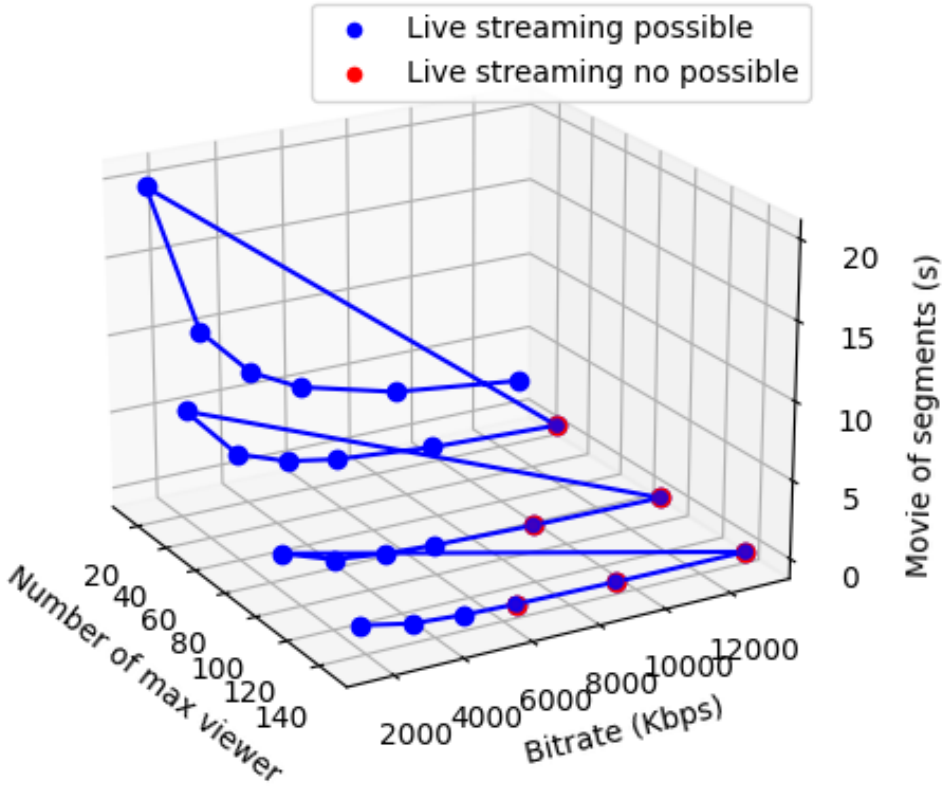

図 4.21: 各ビットレートに対するライブ配信の最大視聴人数、動画セグメントの最大値の関係

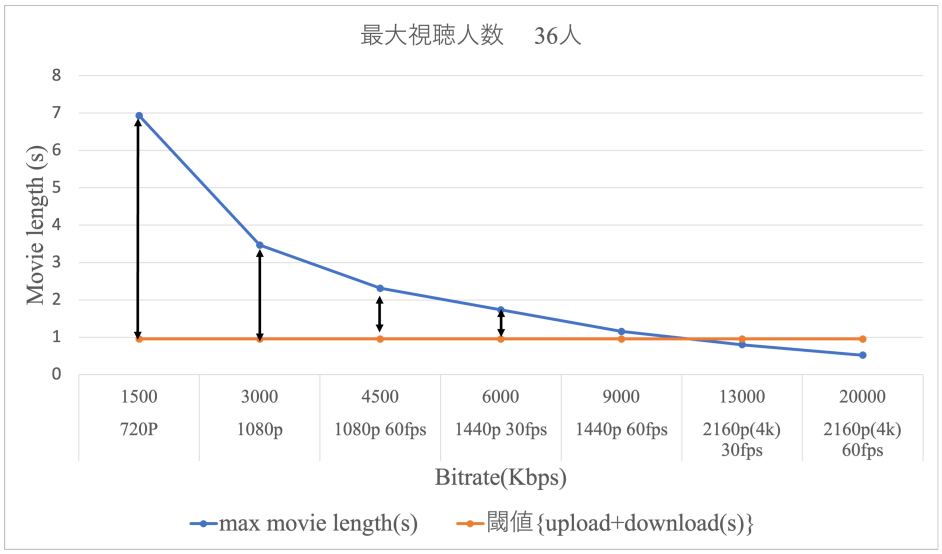

図 4.22: 最大視聴人数 36 人の時、各ビットレートに対する動画のセグメントの大きさ

第5章　今後の課題と結論

本稿では Fed4IoT が提案する仮想 IoT システム (VirIoT) および ThingVisor Factory の利便性を向上するための各種プロトコル、認証サービスについて紹介した。各種プロトコル、認証サービスを実際に実装し、IoT デバイスの電源を入れ、ユーザの属性認証から IoT デバイスのメタ情報の登録、IoT アプリケーションの機能までを、プラグアンドプレイで動作することを確認した。

また、ブロックチェーン技術を用いて IoT データなどを非中央集権的に管理するデータ共有システムを構築し、2 つのユースケースの下、データ共有システムの性能評価を行なった。具体的には、2 つのユースケースは 1) 人の属性情報や IoT データをデータ共有システムへ登録するユースケース、2) データ共有システムに登録されている IoT データを活用するユースケースである。1) では、役 420KB の IoT データであれば許容できる応答時間で登録できることがわかった。2) では、180B の小さなデータに対しては、1 秒間に最大 549 人のトランザクションを処理することができることがわかった。

最後にブロックチェーン技術を用いたデータ共有システムの実験結果から、データ共有システムを用いた動画のライブ配信に関して検討した。ブロックチェーンを用いたデータ共有システムにおいて、1080p の解像度におけるライブ配信では最大視聴可能人数は 147 人で行うことが期待できた。今後の課題はノード (ピアとオーダラー) の数を増やした規模を大きくしたブロックチェーンネットワークを構築し、様々なデータサイズのデータを用いて、HyperledgerFabric を用いたデータ共有システムの性能評価を行うことである。また、複数ピアに対して、データを呼び出すトランザクションを送信した時のデータ共有システムの性能評価を行うことである。本稿ではデータを呼び出すトランザクションを単一のピアに対してのみ送信する実験しか行えていない。動画のライブ配信では、多数ピアに対してデータをダウンロードするトランザクションを送信すれば、最大視聴可能人数が大きくなり大規模なライブ配信をすることが期待できる。

参考文献

[1] FIWARE. [Online]. Available: https://www.fiware.org/.

[2] Hyperledger fabric. [Online]. Available: https://www.hyperledger.org/use/fabric.

[3] Kafka. [Online]. Available: https://kafka.apache.org/.

[4] oneM2M. [Online]. Available: http://www.onem2m.org/.

[5] Open Mobile Alliance (OMA). [Online]. Available: `https://omaspecworks.org/`.

[6] OpenMTC. [Online]. Available: https://www.openmtc.org/.

[7] Power profiling: Orion context broker. [Online]. Available: https://github.com/telefonicaid/fiware-orion.

[8] YouTube. [Online]. Available: `https://support.google.com/youtube/answer/2853702?hl=ja#zippy=%2Cp-fps%2Cp%2Cpkfps%2Cpk-fps`.

[9] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger Fabric: A Distributed Operating System For Permissioned Blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[10] Arati Baliga, Nitesh Solanki, Shubham Verekar, Amol Pednekar, Pandurang Kamat, and Siddhartha Chatterjee. Performance Characterization of Hyperledger Fabric. In *2018 Crypto Valley conference on blockchain technology (CVCBT)*, pages 65–74. IEEE, 2018.

[11] Eranga Bandara, Deepak Tosh, Peter Foytik, Sachin Shetty, Nalin Ranasinghe, and Kasun De Zoysa. Tikiri—Towards a lightweight blockchain for IoT. *Future Generation Computer Systems*, 119:154–165, 2021.

[12] A. Detti, G. Tropea, G. Rossi, J. A. Martinez, A. F. Skarmeta, and H. Nakazato. Virtual IoT systems: Boosting IoT innovation by decoupling things providers and applications developers. In *2019 Global IoT Summit (GIoTS)*, pages 1–6, June 2019.

[13] GSCIM ETSI. ‘ Context information management (CIM): NGSILD API. Technical report, Tech. Rep.[Online]. Available: `https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.01.01_60/gs_cim009v010101p.pdf`, 2019.

[14] Fed4IoT. Deliverable4.2: Smart-city information Sharing Services - Second Release. [Online]. Available: `https://fed4iot.org/index.php/deliverables/`.

[15] The Eclipse Foundation. Eclipse om2m. [Online]. Available: https://www.eclipse.org/om2m.

[16] Yuki Hasegawa and Hiroshi Yamamoto. Highly Reliable IoT Data Management Platform Using Blockchain and Transaction Data Analysis. In *2020 IEEE International Conference on Consumer Electronics (ICCE)*, volume 2020-, pages 1–6. IEEE, 2020.

[17] K. Kanai, H. Nakazato, H. Kanemitsu, and A. Detti. ThingVisor Factory: Thing Virtualization Platform for Things as a Service. In *CCIoT'20*, pages 7–12, Nov 2020.

[18] KC Tam. Multi-Host Deployment for First Network (Hyperledger Fabric v2). [Online]. Available: https://kctheservant.medium.com/multi-host-deployment-for-first-network-hyperledger-fabric-v2-273b794ff3d.

[19] Yoshiro Minamoto, Yuanyu Zhang, Masahiro Sasabe, and Shoji Kasahara. Reputation-Based Reward Distribution Mechanism for Blockchain-Based Scientific Paper Publishing Systems. *IEICE Technical Report; IEICE Tech. Rep.*, 120(413):287–292, 2021.

[20] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Decentralized Business Review*, page 21260, 2008.

[21] Qassim Nasir, Ilham A Qasse, Manar Abu Talib, and Ali Bou Nassif. Performance analysis of hyperledger fabric platforms. *Security and Communication Networks*, 2018, 2018.

[22] NTT データ. ブロックチェーンの仕組み. [Online]. Available: https://www.nttdata.com/jp/ja/services/blockchain/002/.

[23] NTT ファシリーズ.「都市 OS」がスマートシティのデータ利活用を推進する. [Online]. Available: https://www.ntt-f.co.jp/column/0145.html.

[24] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[25] Andreas Schaufelbühl, Sina Rafati Niya, Lucas Pelloni, Severin Wullschleger, Thomas Bocek, Lawrence Rajendran, and Burkhard Stiller. EUREKA–A Minimal Operational Prototype of a Blockchain-based Rating and Publishing System. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 13–14. IEEE, 2019.

[26] Keisuke Takahashi, Kenji Kanai, and Hidenori Nakazato. Performance Evaluation of Blockchains Towards Sharing of Digital Twins. *IEEE LifeTech 2022 (under reviewing)*.

[27] Canhui Wang and Xiaowen Chu. Performance Characterization and Bottleneck Analysis of Hyperledger Fabric. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1281–1286. IEEE, 2020.

[28] Xiaoqiong Xu, Xiaonan Wang, Zonghang Li, Hongfang Yu, Gang Sun, Sabita Maharjan, and Yan Zhang. Mitigating Conflicting Transactions in Hyperledger Fabric Permissioned Blockchain for Delay-sensitive IoT Applications. *IEEE Internet of Things Journal*, 2021.

[29] インターネットイニシアティブ（IIJ）.「IoT プラットフォーム」とは何か、その機能やメリットをくわしく解説. [Online]. Available: https://iotnews.jp/archives/192303.

[30] 小澤洋司 永井崇之. マネージド型ブロックチェーン基盤サービス間での業務システム移行方式の検討. **電子情報通信学会技術研究報告; 信学技報**, 120(413):281–286, 2021.

[31] 許斐健太 and 上山憲昭. ブロックチェーンを用いたクラウドソーシングのワーカ評価システム. **電子情報通信学会技術研究報告; 信学技報**, 120(413):293–298, 2021.

[32] 金井 謙治, 吉田 英聖, 金光 永煥, 中里 秀則, 横谷 哲也, 向井 宏明, 中村 健一, and 上杉 充. Things as a Service を実現する Fed4IoT プラットフォームの研究開発. 信学技報 vol. 119, no. 256, CS2019-69, pp. 39-44, 電子情報通信学会, 10 月 2019.

[33] 金井 謙治, 中里 秀則, and 金光 永煥. Things as a Service を実現する ThingVisor Factory と IP/ICN 間の Things 共有アーキテクチャの提案. 信学技報 vol. 119, no. 424, CS2019-101, pp. 19-24, 電子情報通信学会, 2 月 2020.

[34] 原田恵, 前大道浩之, 山崎育生, et al. 水平統合型 IoT プラットフォーム標準規格 oneM2M の最新動向. *NTT* 技術ジャーナル, 30(2).

[35] 高橋圭介, 中里秀則, and 金井謙治. 効率的な IoT データ共有および IoT サービス配備のための IoT メタ情報管理に関する検討. 電子情報通信学会技術研究報告*;* 信学技報, 120(252):13–18, 2020.

[36] 高橋圭介, 中里秀則, and 金井謙治. IoT デバイスを IoT サービス間で柔軟に共用する機構の開発. 電子情報通信学会技術研究報告*;* 信学技報, 121(113):80–81, July 2021.

[37] 佐藤竜也, 長沼健, 根本潤, 福地開帆, 山田仁志夫, et al. ブロックチェーン基盤 Hyperledger Fabric の性能評価と課題整理. 研究報告インターネットと運用技術 *(IOT)*, 2017(30):1–8, 2017.

[38] 中里 秀則. スマートシティのインターオペラビリティ　—日欧共同研究 Fed4IoT—. *ITU* ジャーナル, 49(9):9 – 12, 9 月 2019.

[39] 竹内 崇 and 寺澤 和幸. データ利活用型都市経営を実現する情報プラットフォーム：FIWARE. *NEC* 技報 *= NEC technical journal / NEC* マネジメントパートナー株式会社 編, 71(1):45–50, 2018.

[40] 戦略的イノベーション創造プログラム. 都市 OS. In スマートシティリファレンスアーキテクチャ, chapter 7. 内閣府, 3 月 2020.

[41] 鎌田 拓朗 and 山本 寛. ブロックチェーンを活用した高信頼 iot データ管理基盤の設計と車両管理を対象としたシステム設計. 電子情報通信学会技術研究報告*;* 信学技報, 121(17):7–12, 2021.

[42] 東京都庁. デジタルツイン実現プロジェクト. [Online]. Available: `https://info.tokyo-digitaltwin.metro.tokyo.lg.jp/#tokyo_3d`.

[43] 中村 岳 日本オラクル株式会社. Blockchain GIG #9 今度こそわかる！Hyperledger Fabric（再）入門. [Online]. Available: `https://gakumura.hatenablog.com/entry/hyperledger-fabric-getting-started`.

研究業績

1. 高橋圭介, 中里秀則, 金井謙治. 効率的な IoT データ共有および IoT サービス配備のための IoT メタ情報管理に関する検討. 電子情報通信学会技術研究報告; 信学技報, 120(252):13–18, 2020.

2. 高橋圭介, 中里秀則, 金井謙治. IoT デバイスを IoT サービス間で柔軟に共用する機構の開発. 電子情報通信学会技術研究報告; 信学技報, 121(113):80–81, July 2021

3. Keisuke Takahashi, Kenji Kanai, and Hidenori Nakazato. Performance Evaluation of Blockchains Towards Sharing of Digital Twins. IEEE LifeTech 2022 (under reviewing).

4. 高橋圭介, 中里秀則, 金井謙治. デジタルツインの共有を目指すブロックチェーンの性能評価. 電子情報通信学会研究会 2022 (執筆中)

謝辞

本研究の成果は，総務省の（平成 30 年度）戦略的情報通信研究開発推進事業（国際標準獲得型）【JPJ000595】「スマートシティアプリケーションに拡張性と相互運用性をもたらす仮想 IoT-クラウド連携基盤の研究開発 (Fed4IoT)」のご支援の下、研究を進めることができました。

本研究を進めるにあたり、中里秀則教授の多くのご指導を賜りました。2 年間ご指導くださった中里秀則教授に心から感謝申し上げます。最後に、本研究に対し、助言してくださった研究員の金井謙治先生、中里秀則研究室の皆様、経済的にも支援してくださった家族に深く感謝致します。

2022 年 1 月 24 日　高橋圭介