Vol.2017-IOT-36 No.30
2017/3/4

ブロックチェーン基盤 Hyperledger Fabric の 性能評価と課題整理

佐藤竜也[†1, a)]　長沼健[†2]　根本潤[†1]　福地開帆[†1]　山田仁志夫[†2]

概要：ブロックチェーン技術は破壊的イノベーションとして，金融をはじめとした様々な分野で実用可能性が検討されはじめている．ブロックチェーン技術はトランザクションシステムへの適用が期待されているが，その実用化に向けてはトランザクション性能を中心とした現在の実力と残課題の把握が必要である．そこで報告者はブロックチェーン基盤のオープンソースソフトウェアプロジェクトである Hyperledger の基盤実装の一つである Fabric について，モニタリングを含めた検証環境を構築し，性能を中心とした評価を行った．さらに，評価結果に基づいて，実用化に向けた技術課題を抽出した．

キーワード：ブロックチェーン，ブロックチェーン基盤，Hyperledger，Hyperledger Fabric，性能評価

Defining Technical Challenges of Blockchain Platform "Hyperledger Fabric" through Quantitative Evaluation

TATSUYA SATO[†1]　KEN NAGANUMA[†2]　JUN NEMOTO[†1]
KAIHO FUKUCHI[†1]　NISHIO YAMADA[†2]

Abstract: Blockchain technology has been attracting attention as a disruptive innovation. However, the current blockchain technology has many problems concerning the security and system performance to apply enterprise systems. Therefore, towards applying the blockchain technology to enterprise systems, it is necessary to investigate the technical problems of the current blockchain platform and the open source implementations. In this research, we perform quantitative evaluation of Hyperledger Fabric which is an open source implementation of blockchain platform to reveal the ability and limitations of current blochchain platform and clarify the technical problems.

Keywords: Blockchain, Blockchain platform, Hyperledger, Hyperledger Fabric, Performance evaluation

1. はじめに

ブロックチェーン（Blockchain, BC）技術は，破壊的イノベーションとして金融や Internet of Things（IoT）等の非常に幅広い分野への応用が期待され，注目を集めている．例えば，金融分野では，従来は第三者機関を経由して実施されてきた取引を，BC 技術を用いて利用者間（P2P）の直接取引で代替することで，取引コストの削減ができると期待されている．文献[1]では，様々な分野に BC 技術を応用することで，将来的に日本国内の 67 兆円もの市場が影響を受ける可能性があるという試算結果が示されている．

BC 技術に大きな期待が集まっている一方で，現状ではセキュリティ面やシステム性能面をはじめ，エンタープライズでの実適用には課題が多いといわれている．そのため，BC の実適用に向けては，BC 基盤やそのオープンソースソフトウェア（Open Source Software, OSS）実装における現時点での技術課題の明確化が必要である．

そこで本研究では，BC 基盤の OSS プロジェクトである Hyperledger[8]の基盤実装の一つである Fabric[11]について性能を中心とした評価を行い，BC 基盤および Fabric の現時点での実力を明らかにするとともに，技術的な課題を抽出にすることを目的とする．

なお，Fabric の性能に関しては，既にいくつかの評価報告[2][3]がなされているが，Fabric は発展途上段階にあるため，評価環境や測定条件の異なるデータや知見の蓄積が重要である．本報告では，基盤構成の更新による再測定や将来的な運用も考慮し，モニタリングも含めたシステム構成を検討した点や，評価結果に基づいて技術課題を包括的に整理した点で， Fabric や BC 基盤の実用化に向けた知見の蓄積に寄与する．

2. BC 技術と実適用に向けた課題

2.1 BC 技術の概要

BC 技術は，暗号通貨である Bitcoin[4]の実装技術として注目されている．現状では，Bitcoin の BC が持つ以下 (1) 〜 (3) の設計思想をベースに，さまざまな派生技術が提案され，進化を続けている．

(1) BC ネットワーク上の利用者間（P2P）取引において第

†1 (株)日立製作所　情報通信イノベーションセンタ
Hitachi Ltd., Center for Technology Innovation – Information and Telecommunications
†2 (株)日立製作所　システムイノベーションセンタ
Hitachi Ltd., Center for Technology Innovation – System Engineering
a) tatsuya.sato.so@hitachi.com

三者機関を介することなく，参加者間で合意形成，承認することによって取引を確定させる．

(2) 複数の取引をブロックとしてまとめ，数珠つなぎに分散台帳に記録する．連続するブロックにハッシュ計算を施すことにより，改ざんを実質不可能にする．

(3) すべての参加ノードが同一の台帳データを共有することにより，参加者全員での取引の確認を可能とする．

2.1.1 BC ネットワーク構成の分類

BC 技術は多様化してきているが，BC ネットワークの構成という観点では，以下のとおりパブリック型とプライベート/コンソーシアム型に分類することができる．

(a) パブリック型 BC: 不特定多数のノードが自由に参加する BC である．Bitcoin 等がパブリック型に分類される．パブリック型の場合には，トランザクションに誰でもアクセス/分析可能である．また，不特定多数の信用できない参加者間でトランザクションを承認する必要があるため，一部の悪意ある参加者による過去のデータの不正改ざんを防止できる厳格な合意形成アルゴリズムを利用する必要がある．例えば，Bitcoin では，Proof of Work（PoW）と呼ばれるアルゴリズムが利用される．PoW では承認を行うノードに大量の計算処理を課すことにより不正改ざんを防止する．

(b) プライベート/コンソーシアム型 BC: 特定のノードのみが参加する BC である．通常，トランザクションの承認処理は複数の限定的なノードによって実施される．そのため，PoW などのように厳格な合意形成アルゴリズムは不要である．BC の読み取りは，ネットワーク内で限定される場合と公開する場合の両方が考えられる．また，単一の組織に利用が限定される場合をプライベート型，特定複数の組織（例えば，複数の金融機関とそのパートナー企業）が参加する場合をコンソーシアム型と呼ぶ．

上記のうち，プライベート/コンソーシアム型が，現在，金融分野をはじめとしたエンタープライズでの利用が有力視されるモデルとなっている．

2.1.2 BC 技術の機能に関する進化状況

BC の応用範囲が従来の仮想通貨から，様々なアセット/価値の管理に拡大していく中で，機能面の進化としてスマートコントラクトを実行可能とした BC が登場してきている．スマートコントラクトとは，プログラムの形式で記述した契約情報である．スマートコントラクトにより，単純な取引だけでなくルールベースの取引を実行可能である．スマートコントラクトを BC 上で管理することで，契約の改ざんの心配がなくなり，確実かつ自動的に取引を執行できるため，契約の信頼性を高めることができる利点がある．スマートコントラクトは，金融取引だけでなく IoT を組み合わせた契約自動執行等，幅広い分野での応用が期待されている．このスマートコントラクト機能を実装した BC 基盤の代表例としては Ethereum[7]が挙げられる．

2.1.3 BC 基盤の実装状況

BC 基盤の実装は既にいくつも OSS として公開されている．例えば，前述の Bitcoin や Ethereum も OSS であり，今回の評価対象である Hyperledger Fabric もその一つである．

2.2 BC 基盤 Hyperledger Fabric

Hyperledger プロジェクトは，Linux Foundation が設立したエンタープライズで利用可能な OSS の BC 基盤の開発を目的としたプロジェクトである[8]．同プロジェクトは，2016 年 2 月に設立され，金融機関をはじめとしたユーザ企業や IT ベンダー等, 計 100 社以上が参画している. 弊社も，同プロジェクトの設立時からプレミアメンバーとして参画し，コミュニティ活動に参加している．

Hyperledger プロジェクトでは，BC のユースケース，基盤の機能要件およびアーキテクチャがホワイトペーパーにまとめられている[9]．また，その実現に向けて，複数のベンダーから基盤実装が提案されている．IBM 社と DAH（Digital Asset Holdings）社の共同提案である Fabric[11]はその基盤実装の一つである．Fabric は 2016 年 4 月に公開され，2017 年 1 月現在で開発者プレビュー版 v0.6 までリリースされている．Fabric は前述のホワイトペーパーに対応したアーキテクチャを実装している．

Fabric は，様々な分野でのユースケースに対応可能とするために汎用性の高い BC 基盤機能を提供する．また，現在は，コンソーシアムあるいはプライベート型での利用を想定した BC 基盤となっている．Fabric の主な機能的特徴として，具体的には以下が挙げられる．

- 台帳上では，通貨に限らず様々なアセット/価値の管理に対応可能（状態を管理するアカウントモデル，Bitcoin のような状態を持たない Unspent Transaction Output（UTXO）モデルの両方に対応可能）
- スマートコントラクト実行機能を提供（Fabric ではスマートコントラクトはチェーンコードと呼ばれる）
- 合意形成アルゴリズムはプラグインで切り替え可能
- BC ネットワーク参加者を管理し認証や証明書発行を行うメンバーシップサービスを BC 外部の機構として提供

図 1 は, 公式ドキュメントに示される Fabric のアーキテクチャである．公式ドキュメントの記載内容に従って，Fabric の主要な構成要素を説明する．

- **メンバーシップサービス:** BC ネットワークへの参加者，スマートコントラクト，合意形成を行う検証ノード等，ネットワーク上の全オブジェクトの ID を管理する．
- **BC サービス:** P2P プロトコル，分散台帳，コンセンサスマネージャといった要素によって構成される. P2P プロトコルにより，P2P での双方向ストリーミング，フロー制御，リクエストの多重化といった機能を提供する．既存ネットワークと連携して動作する．分散台帳により，BC と，台帳の（最新）状態を管理する．コンセンサス

マネージャにより，プラグイン可能な合意形成アルゴリズム用のインタフェースを提供する．

・ **チェーンコードサービス：** スマートコントラクトを実行する軽量でセキュアな実行環境を提供する．

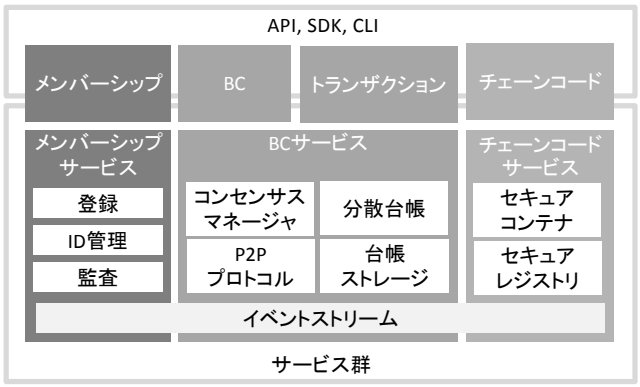

図 1　Hyperledger Fabric のアーキテクチャ（[11]に基づく）
Figure 1　Architecture of Hyperledger Fabric.

現在 Fabric では，上記の BC サービスとチェーンコードサービスが同一のノード上の機能として提供される．以降では，このノード，すなわち，合意形成を行い，チェーンコードを実行し，その結果をローカルの台帳上に保持するノードのことを「検証ノード（Validating Peer）」と呼ぶ．

Fabric ではコンセンサス方式はプラグイン可能となっているが，その標準実装として Practical Byzantine Fault Tolerance （PBFT）を提供している．PBFT は，Castro らによって提案された分散システムの研究分野で著名なアルゴリズムである[6]．PBFT は，コンセンサスを実施する検証ノードの総数 n 台に対して，(n-1)/3 台までのノードが故障あるいは悪意を持ったノードであっても正常に合意形成することができる．Bitcoin のコンセンサス方式である Proof of Work（PoW）は時間の経過とともに取引の確定確率が高まる方式であるため，厳密には取引が確定しないという，いわゆる処理のファイナリティの課題があったが，PBFT では合意をもって取引が確定するため，この課題を解決することができる．また，PoW と PBFT はノードのスケーラビリティとスループットのスケーラビリティで対極的な関係にある[5]．PBFT は，基本的に固定台数（の少数の）ノード間での合意形成を前提とするが，PoW ほどは大量の計算処理を必要としない．そのため，PoW に比べて高速であり，BC 基盤のスループット向上に将来性のある一手法と考えられている．

2.3 本研究の課題

BC の実適用に向けては，BC 基盤およびその実装における現時点での技術課題の明確化が必要である．特に，金融業務への適用に向けては，一般的にトランザクションのスループットが性能面の最重要指標である．必要となるスループットは，オンライン決済サービスで平均数百 tx/s （transaction per second），VISA システムで平均数千 tx/s となり[13]．さらに株取引システムでは数万 tx/s のオーダー[2]となり，これが BC 基盤でどのようなシステムまで適用可能であるかを把握するための一つの目安となる．

そこで本研究では，Hyperledger Fabric に関して性能を中心とした評価を行い，BC 基盤および Fabric の現時点での実力や制約事項等を明らかにするとともに，技術的な課題を抽出することを目的とする．

3. Hyperledger Fabric の評価

3.1 評価の目的

Fabric および BC 基盤の課題抽出に向けて，以下を主な目的として評価を行う．

目的 1: Fabric の現実装における実力（主に性能）の把握

Fabric は公開されて間もないため，その品質（特に非機能面）が未知数であった．そのため，実機上に環境を構築して動作検証を行い，さらには性能を測定することで，Fabric の現実装における実力を把握する必要がある．性能評価においては，より本格的な BC ネットワークを構築することが望ましい．

目的 2: Fabric/BC 基盤の性能ボトルネック抽出方法の検討

性能限界や傾向を把握するためには，そのボトルネック箇所を特定することが重要である．しかし，Fabric では，そのような調査や分析を行う手段が整備されていない．そのため，性能ボトルネックの抽出方法の検討が必要である．Fabric 自体のバージョンアップ時等には，再度性能評価を行うことが予想されるため，ボトルネック抽出は繰り返し実行できるようにシステム化することが望ましい．

3.2 評価方法

3.2.1 概要

先に示した評価の目的 1 と 2 を達成するために，以下の評価方針を採用した．

目的 1 の達成に向けた方針

性能評価に適した本格的な評価環境として，マルチホスト上にまたがった環境上に Fabric による BC ネットワークを構築し，その上で性能測定を行う．

目的 2 の達成に向けた方針

Fabric（および BC 基盤）の性能ボトルネックを容易に抽出可能とするためのモニタリング環境も合わせて整備する．

目的 1 の達成に向けた方針に従って，マルチホスト（厳密には VM）上に，コンテナ基盤 Docker[14]のコンテナによる BC ネットワークを構築した．Docker コンテナ上にネットワークを構築した理由は以下のとおりである．

(1) 現 Fabric は（チェーンコードの実行基盤が）Docker に依存しているため
(2) BC ネットワークをリセットしながら繰り返して性能

評価を行う上で Docker 環境は自動化しやすいため

なお，Docker コンテナ管理および Docker 上のマルチホストネットワーク構築のために，複数コンテナ構築ツール Docker Compose とクラスタ管理ツール Docker Swarm を導入した．これらは Docker 純正のツール群となっている．さらに，目的 2 の達成に向けた方針に従って，BC ネットワークの性能ボトルネック分析および障害検出/分析のために OSS ベースのモニタリング環境を整備した．具体的に利用した OSS は以下のとおりである．

・ 性能メトリックス収集/分析/可視化 OSS:
Telegraf [16] + InfluxDB [17] + Grafana [18]
・ ログ収集/分析/可視化 OSS:
Fluentd [19] + ElasticSearch [20] + Kibana [21]

以上の環境を用いて，実験条件を変えつつ性能測定を行い，性能値やログ分析によってボトルネックを調査した．なお，ソースコードは評価時点で最新版であった 2016 年 5 月 31 日のものを利用した．2017 年 1 月 17 日の時点では，開発者プレビュー版 v0.6 までリリースが進んでいるが，Fabric のアーキテクチャは評価時点のものと大きくは変わっていない．そのため，以降で示す評価結果は現在においても基本的に有効であると考えられる．ただし，2017 年 3 月末にリリース予定の v1.0 では，スケーラビリティとセキュリティの改善を主目的として，アーキテクチャの大幅な変更が行われ，v0.6 までの技術的な課題が一部解決される予定である．その点については後述する．

3.2.2 評価環境

実験に用いた評価環境のシステム構成図を図 2 に示す．以下にシステムを構成する各 VM について説明する．

・ 検証ノード用 VM 4 台（#0, 1, 2, 3）
- 役割: 合意形成とチェーンコード実行をするノードである．Fabric が提供する PBFT ベースの合意形成アルゴリズムを利用し，4 台の間で合意形成を行う．
- 1 台あたりの VM スペック: CPU 2 コア, メモリ 4GB, ディスク 20GB

・ メンバーシップサービス用 VM 1 台
- 役割: BC ネットワークに参加する検証ノードやクライアントの認証や証明書を管理するノードである．
- 1 台あたりの VM スペック: CPU 2 コア, メモリ 4GB, ディスク 20GB

・ クライアント兼モニタリング VM 1 台
- 役割: クライアントとして，内製の負荷生成ツールによって，REST 経由で BC ネットワークに対してトランザクションを発行する．また，前述の性能メトリックス収集/分析/可視化およびログ収集/分析/可視化 OSS が動作し，BC ネットワーク上の各 VM やその上の Docker コンテナをモニタリングする．
- 1 台あたりの VM スペック: CPU 2 コア, メモリ 4GB, ディスク 20GB

その他のシステム構成に関する情報は以下のとおりである．

・ VM 間は 1GbE ネットワークで接続される．
・ 複数ホストにまたがった Docker コンテナのクラスタ管理には Docker Swarm を用いる．Docker Swarm を用いた管理をするため，BC ネットワークを構成するノード上では，Swarm マネージャ（Master）やエージェント（Agent）をそれぞれ Docker コンテナとして稼働させる．なお Swarm マネージャと同じホスト上ではバックエンドにて OSS のサーバクラスタ管理ツールである Consul[15]が稼働しており，Swarm によって束ねられるホストの管理に利用されている．
・ BC ネットワークを構成するメンバーシップサービスと検証ノードは Docker コンテナとして稼働する．
・ その他ソフトウェアは OS 上のプロセスとして稼働する．

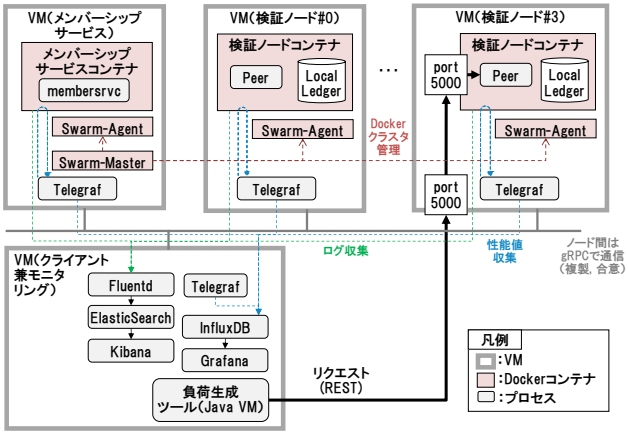

図 2　評価環境のシステム構成図

Figure 2　System architecture of evaluation environment.

本評価環境を実現する上で，コンテナ管理やモニタリングに利用した OSS 一覧を表 1 に示す．

表 1　コンテナ管理やモニタリングに利用した OSS 一式

Table 1　OSS for container management and monitoring.

分類	OSS	用途
コンテナ管理	Docker Compose	複数コンテナ環境を管理する
	Docker Swarm	複数ホストによる Docker コンテナクラスタ環境を管理する
	Consul	サーバクラスタを管理する
性能監視	Telegraf	（Docker コンテナや VM から出力された）性能値を収集する
	InfluxDB	性能値を蓄積する DB
	Grafana	性能値を表示するダッシュボードを提供する
ログ監視	Fluentd	（Docker コンテナから出力された）ログを収集する
	ElasticSearch	ログを蓄積する DB
	Kibana	ログを表示するダッシュボードを提供する

今回の実験で取得した主なモニタリング項目を表 2 に示す.

表 2　主なモニタリング項目

Table 2　Major monitoring metrics.

分類	項目名	説明
性能指標	BC 長	トランザクション実行期間中の BC 長
	CPU 利用率	各コンテナもしくは VM の CPU 利用率（VM はクライアント CPU ネックの発生有無を確認するために取得）
	メモリ利用率	各コンテナのメモリ利用率
	ディスク I/O	各コンテナの単位時間あたりのディスク読み書きデータ量
	ネットワーク I/O	各コンテナの単位時間あたりのネットワーク送受信データ量
	ディスク容量利用率	各 VM のディスク容量（ディスク溢れの発生有無を確認するために取得）
ログ	検証ノード	検証ノードプロセスが出力するエラー，インフォ，デバッグログ
	メンバーシップサービス	メンバーシップサービスプロセスが出力するエラー，インフォ，デバッグログ

3.2.3 測定方法

クライアントから REST 経由で BC ネットワークにアクセスし，チェーンコードを実行して負荷をかけることで性能測定を行った．チェーンコードには，Fabric の公式プロジェクトに付属のサンプルチェーンコード「map」を利用する．本サンプルチェーンコードは簡易キーバリューストアとして動作する．なお，負荷をかける際には，複数のクライアントからの同時アクセスを模擬するため，マルチスレッドでトランザクションを並列実行した．

クライアントによる負荷生成ツールプログラムの処理の流れは以下のとおりである．

1. ユーザログイン（セキュリティ機能 ON 時のみ）
2. map チェーンコードを BC 上にデプロイ
3. スレッド毎に実行トランザクション（invoke）を指定した回数繰り返す（並列実行）
4. 全スレッドの実行トランザクションが完了するまで（レスポンスがかえってくるまで）待つ
5. スレッド毎に参照トランザクション（query）を指定した回数繰り返す（並列実行）
6. 全スレッドの参照トランザクションが完了するまで（レスポンスがかえってくるまで）待つ

ここで，今回の測定におけるトランザクションのスループット計算方法は以下のとおりである．

$$\text{スループット(tx/s)} = \frac{\text{全スレッドによる合計リクエスト件数}}{\text{全レスポンスが返ってきた時刻}-\text{リクエスト処理を開始した時刻}}$$

3.2.4 実験パラメータ

今回の測定で利用した実験パラメータは表 3 のとおりである．これらは性能に与える影響が特に大きいと想定したパラメータである．セキュリティ機能 OFF と ON 時のそれぞれの場合で測定した．ON 時には，メンバーシップサービスを利用して，ネットワークへの参加ノードの認証やトランザクションの秘匿化が行われる．一方，OFF 時にはメンバーシップサービスは利用されず，認証や秘匿化は行われない．合意形成アルゴリズムとしては，提供されている PBFT ベースのアルゴリズムである Batch PBFT を測定対象とした．Batch PBFT では，複数のトランザクションを 1 つのブロックにまとめて PBFT による合意形成をバッチ処理する．合意バッチサイズは測定当時のデフォルト値である 2 とした．なお，Fabric では合意形成を行わずにトランザクションを素通しする Noops という設定も存在するが，これはテストや開発時の利用を想定したものであり，本番での利用可能性が低いため，今回は測定対象外とした．その他のパラメータについては，(Fabric の動作が不安定になることがあったため）性能等をみながら調整した．

表 3　実験パラメータ

Table 3　Parameters of experiments.

パラメータ	設定値のパターン
セキュリティ機能	OFF, ON
合意形成アルゴリズム	Batch PBFT (バッチサイズ=2)
クライアントのスレッド数	1, 2, 4, 6, 8, 10, 12, …
スレッドあたりのリクエスト数	1000 (セキュリティ機能 OFF), 200（セキュリティ機能 ON）

3.2.5 その他の設定

性能測定時における検証ノードとメンバーシップサービスのログ出力設定は，正常に稼働しているかを確認する上で必要最小限の情報を出力するように調整した．この理由は，ログ出力が性能のオーバヘッドになり得るためである（コミュニティ内の Issue でもログ出力が性能のオーバヘッドになっていると報告されている）．具体的には，検証ノードのログ出力は「info」，メンバーシップサービスのログ出力は「trace」に設定した．

3.3 結果と考察

図 3 にセキュリティ機能 OFF 時の，図 4 にセキュリティ機能 ON 時の invoke/query スループット測定結果を示す．図に示す通り，セキュリティ機能 OFF/ON 時ともに，invoke と query の両方で，並列スレッド数を増やしていくとスループットも増加する傾向にあった．ただし，ある程度並列度を増やすとスループットは頭打ちとなった．

また，スループットが頭打ちになった後も，それ以上に並列度を増やしていくと，内部的にエラーが発生する等，安定稼働が困難となる場合が見受けられた．つまり，高負荷を与えた場合には，挙動が安定しなくなる場合があるため，フロント側でリクエストの流量制御を行う等の対策が必要となり得る．

さらに，トランザクションは確定までにタイムラグが有った（通常は 1 秒以内だったが，高負荷時に PBFT のプライマリーノードを切り替えるビュー変更が発生することがあり，その場合等には数秒程度かかることがあった）。そのため，ユースケース次第でトランザクション確定の管理が必要になり得る．

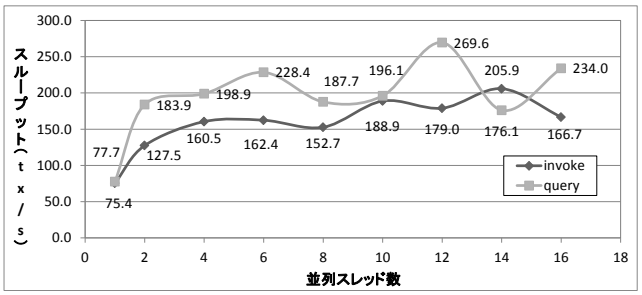

図 3　セキュリティ機能 OFF 時のスループット
Figure 3　Throughput of transactions (Security: OFF).

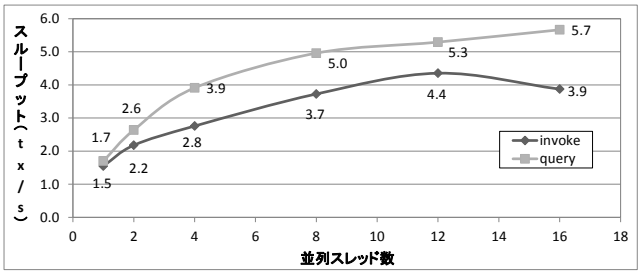

図 4　セキュリティ機能 ON 時のスループット
Figure 4　Throughput of transactions (Security: ON).

図 5 と図 6 に測定時における各種性能メトリックスの典型例を示す．これらは監視ツールである Grafana のダッシュボード上の表示画面をキャプチャしたものである．

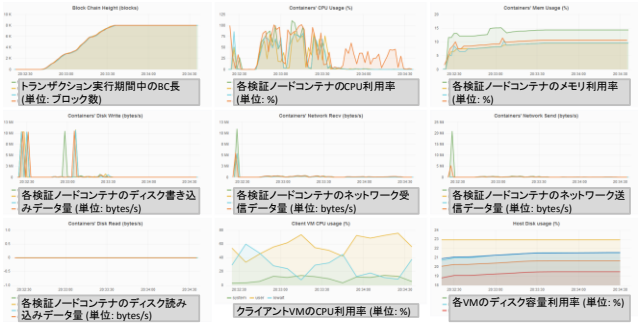

図 5　セキュリティ機能 OFF 時の性能メトリックス例
Figure 5　Example of monitoring metrics (Security: OFF).

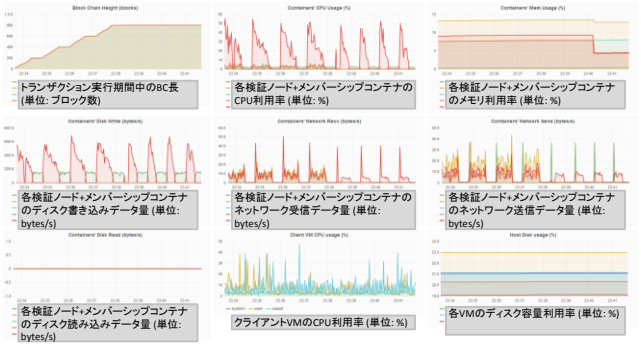

図 6　セキュリティ機能 ON 時の性能メトリックス例
Figure 6　Example of monitoring metrics (Security: ON).

性能メトリックスに基づいてボトルネックや性能傾向を考察したところ，今回の実験では，セキュリティ機能 OFF 時の性能のボトルネックは，各検証ノードのコンテナの CPU であった．これは合意形成処理およびスマートコントラクト実行処理が重く，CPU を大きく消費しているためと考えられる．

一方，セキュリティ機能 ON 時は，検証ノード 1 台とメンバーシップサービスのディスク書込み処理の負荷が高くなっており，これが性能のボトルネックになっているものと考えられる．コンテナのディスク書き込みに着目すると，特定の検証ノード（リクエストを受け取っているノード）とメンバーシップサービスの書き込み処理負荷が交互に大きくなっていた．セキュリティ機能 ON 時には，Fabric はトランザクション毎に証明書（TCert と呼ぶ）を発行することで，取引の匿名性を確保する仕様となっているが，ログを確認したところ，メンバーシップサービスが TCert を発行/保持する処理に長い時間を要していた．証明書は 200 件ずつバッチ的に発行しているため，これが性能劣化に影響している可能性がある．なお，セキュリティ機能については上記のように各トランザクションに対する証明書の発行が本当に必要かどうかはユースケースに応じると考える．

今回の評価結果を総括すると，Fabric は現時点の実装でもオンライン決済サービスの平均処理と同等レベルの性能が得られそうな感触を得た反面，まだ正式版リリース前の発展途上にあるため，機能面と性能面の両方でまだ課題が残っている状況であることがわかった．ただし，プロジェクト上のソースコードが日々更新されている等，コミュニティ活動は非常に活発であるため，近い将来の進化や実用化が期待できる．

4.　技術課題の整理

4.1　評価結果に基づく技術課題の整理

前章に示した Fabric の評価結果に基づいて抽出した技術課題を表 4 に示す．#1~#4 は性能とセキュリティの課題，#5~#7 は安定稼働に向けた運用管理に関する課題である．今回の評価によって，一般的によく言われている BC の性

能とセキュリティに関する課題を事実ベースの裏付けをもって確認でき，問題点や課題をより具体化することができた．また，BC をシステムとして安定稼働させるための仕組みが足りていないことがわかったため，実適用にむけてはこれらに対する対応/改善が必要であることがわかった．

なお，今回抽出した技術課題の中には，Hyperledger のコミュニティの中でも課題として認識されており，既に対応が予定されているものも存在する[10][12]．特に，次期バージョン v1.0（2017 年 3 月末にリリース予定）では，現在，スケーラビリティとセキュリティの改善を主な目的として，アーキテクチャの大幅な変更が行われている最中である．大幅な変更がされた理由としては，特にスマートコントラクト実行処理の CPU 負荷が高くトランザクション性能が出なかったため，スマートコントラクト実行を並列化できるようにノードの役割分割が必要だったためである．v1.0 でのアーキテクチャ変更の中で表 4 に示した v0.6 までの技術的な課題についても一部解決される予定である．表 4 の最右列には，v1.0 のロードマップや現在開発中の実装といった公開情報に基づく v1.0 時点での対応/改善見込みを示す．表に示す通り，v1.0 でも課題が残るため，実用化に向けては，今後も継続的な課題解決が必要である．

5. おわりに

本報告では，BC 基盤およびその実装における技術課題を明らかにするために，Hyperledger プロジェクトの基板実装のひとつである Fabric について，モニタリングを含めた検証環境を構築し，性能を中心とした評価を行い，その結果に基づいて技術課題の整理を行った．評価の結果，Fabric は現時点の実装でもオンライン決済サービスの平均処理と同等レベルの性能が得られそうな感触を得た反面，まだ正式版リリース前の発展途上にあることもあり，性能，セキュリティ，安定稼働に向けた運用管理の各観点における課題が残っていることがわかった．

Hyperledger および Fabric はコミュニティ活動が活発であり，実装も日々進化している．今回示した技術課題の一部は，2017 年 3 月末にリリース予定の次期バージョン v1.0 で解決される予定である．しかしながら，v1.0 でも残課題があり，さらに v1.0 による大幅なアーキテクチャ変更による新たな課題が発生すると想定される．そのため，コミュニティ動向をウォッチしつつ，バージョンアップに応じて，本報告で示したような性能評価・検証を定期的に実施していくことが重要である．今後は，今回示した Fabric の環境構築/性能評価の知見を活かして，さらに自動化を進めた検証環境も適宜整備していきたい．

参考文献

[1] 経済産業省，“ブロックチェーン技術を利用したサービスに関する国内外動向調査,” (2016).
[2] 山藤敦史 他, “金融市場インフラに対する分散型台帳技術の適用可能性について,” JPX ワーキング・ペーパー, Vol.15 , (2016).
[3] 戸澤晶彦 他, “Hyperledger/Fabric の性能解析,” 情報処理学会第 111 回プログラミング研究会, 4 pages, (2016).
[4] Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” (2009).
[5] Marko Vukolić, “The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication,” International Workshop on Open Problems in Network Security (2015).
[6] Miguel Castro et al., “Practical Byzantine Fault Tolerance.” OSDI, Vol. 99, (1999).
[7] “Ethereum ホワイトペーパー, ” https://github.com/ethereum/wiki/wiki/White-Paper, (参照 2017-01-17).
[8] “Hyperledger Project ホームページ,” https://www.hyperledger.org/, (参照 2017-01-17).
[9] “Hyperledger Project ホワイトペーパー”, https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8Iir-gqS-ZYe7W-LE9gnE/, (参照 2017-01-17).
[10] “Hyperledger Project Wiki,” https://wiki.hyperledger.org/, (参照 2017-01-17).
[11] “Hyperledger Fabric,” https://gerrit.hyperledger.org/r/#/admin/projects/fabric, (参照 2017-01-17).
[12] Marko Vukolić, “Hyperledger Fabric: Towards Scalable Blockchain for Business,” Trust in Digital Life (2016).
[13] “Scalability,” https://en.bitcoin.it/wiki/Scalability, (参照 2017-01-17).
[14] “Docker,” https://www.docker.com/, (参照 2017-01-17).
[15] “Consul,” https://www.consul.io/, (参照 2017-01-17).
[16] “Telegraf,” https://www.influxdata.com/time-series-platform/telegraf/, (参照 2017-01-17).
[17] “InfluxDB,” https://www.influxdata.com/time-series-platform/influxdb/, (参照 2017-01-17).
[18] “Grafana,” http://grafana.org/, (参照 2017-01-17).
[19] “Fluentd,” http://www.fluentd.org/, (参照 2017-01-17).
[20] “ElasticSearch,” https://www.elastic.co/jp/products/elasticsearch, (参照 2017-01-17).
[21] “Kibana,” https://www.elastic.co/jp/products/kibana, (参照 2017-01-17).

表 4　評価結果に基づいて抽出した技術課題

Table 4　Technical problems based on the evaluation results.

#	項目名	問題点/課題の説明	v1.0 での対応/改善見込み
1	基本スループット	セキュリティ機能 OFF 時にはトランザクション性能は数百 tx/s 程度（CPU 次第では向上可能性あり）で頭打ちとなる．今回の主なボトルネックは合意形成およびスマートコントラクト実行処理の負荷によるものである．	○（v1.0 のアーキテクチャ変更によって性能向上見込み）
2	セキュリティ機能利用時のスループット	セキュリティ機能 ON 時は OFF 時に比べ性能が劣化する．主なボトルネックはトランザクション証明書（TCert）発行/管理によるディスク書き込みの発生によるもの． →ユースケース等に応じたセキュリティ機能の要否の議論が必要である．	△（同上．ただし，TCert を対象にした利用 ON/OFF 設定はできない見込み）
3	トランザクション確定までのタイムラグ発生および管理	検証ノード間での合意形成処理は，クライアントからのリクエスト処理とは非同期的に実施されるため，トランザクション確定までにはタイムラグがある（ラグが長い場合には数秒程度）． →ユースケース次第ではトランザクション確定の管理が必要になり得る．	×
4	ディスク消費容量	BC 長が増えるにつれて，ディスク容量を消費するため，長期データ保存時に容量が肥大化する（今回の評価では 1 トランザクションあたり約 0.4 KB ずつ増加）．	×
5	処理安定性	現在の実装では，高負荷時の動作が安定しない一方，リクエストの流入を制御する機構を備えていない． →実用に向けてフロントで流量制御をする仕組み等が必要である．	ー（処理スループットの向上は見込めるが，フロントで流入制御は行うかどうかは今後の確認要）
6	Service Level Agreement（SLA）およびモニタリング整備	安定稼働と性能問題検出/分析のためには，ログや性能値のモニタリングが必須であるが，現状では Fabric 内ではその機能を提供しない（コンソール上のログ出力のみ）．また BC 基盤やサービスの場合には従来型のシステムとは指標や SLA が異なるため（例えば BC の長さ等），SLA の定義から検討が必要となる．	×（ブロックチェーンの状態を可視化するツール Hyperledger Explorer が提案されているが，現状は開発者向けのブロック/トランザクション情報ビューアに留まる）
7	システム構成変更/メンテナンス対応	ノード追加や更新には対応していない．そのため，ノード数の追加やコンセンサス方式の入替が容易ではなく，少なくとも全台の停止が必要となる．また，一度デプロイしたスマートコントラクトを更新することができない．	○（ノードの動的追加やスマートコントラクトの更新対応見込み）
		データのマイグレーションやバックアップ機能等も整備されていない．	×