EUROPEAN CENTRAL BANK
EUROSYSTEM

BANK OF JAPAN

Project Stella

日本銀行・欧州中央銀行による分散型台帳技術に関する共同調査

― 分散型台帳技術による資金決済システムの流動性節約機能の実現 ―

2017年９月

目　次

※本稿は、欧州中央銀行および日本銀行による報告書「Payment systems: liquidity saving mechanisms in a distributed ledger environment」の日本銀行決済機構局による仮訳である。

Project Stella
分散型台帳技術による資金決済システムの流動性節約機能の実現

１．背景

分散型台帳技術（distributed ledger technology、DLT）は、金融資産や金融取引等のデータを記録するための技術であり、単一の中央管理システムを置くことなく、ネットワークでつながれた複数のコンピュータがデータの検証・更新を行うことを可能とする。2016 年 12 月、日本銀行と欧州中央銀行は、金融市場インフラへの DLT の応用可能性を調査するための共同調査プロジェクト「Project Stella」を立ち上げることを公表した。本報告書は、同共同調査の第一段階における調査結果を取り纏めたものである[1]。

Project Stella は、金融市場インフラへの DLT の応用可能性に関する最近の幅広い議論に貢献するものである。今回の共同調査は、より安全、迅速かつ安価な金融取引を可能にするような技術革新を促していくとの中央銀行の関心に沿って実施されてきた。

今次調査では、以下で述べるように、対象範囲を絞って分析を行った。すなわち、現行の資金決済システムの主要な機能が、DLT を用いることで安全かつ効率的に実行できるかについて、手触り感を持って確認することに比重を置いた。このため、DLT の導入に伴うコスト削減効果、市場統合への影響、中央銀行のオーバーサイト活動への含意といった点については、検討の対象外と位置付けられる[2]。

本報告書の構成は、以下のとおりである。第 2 章では、今次調査における主な調査結果を概観する。第 3 章では、両中央銀行が運営する資金決済システムの

---

[1] 今次調査には、日本銀行から小早川周司、河田雄次、渡邉明彦、小林亜紀子、欧州中央銀行から Dirk Bullman、Frederick Chorley、Cedric Humbert、Thomas Leach、Andrea Pinna が参加した。

[2] DLT にかかる効率性と安全性は、応用事例のデザイン、機能、必要となるリソースを含む幅広い概念である（詳しくは、決済・市場インフラ委員会報告書「支払・清算・決済における分散型台帳技術―分析的枠組み―」（2017 年 2 月）を参照）。しかし、本報告書は、DLT を評価するための第一歩として、限られた論点、すなわち、①処理スピードや②耐障害性に焦点を当てている。さらに、本報告書の分析結果は、Hyperledger Fabric バージョン 0.6.1 に基づくものである。当該バージョンは、「開発者が試行することによって、プログラムのロジックや安定性を確認することを意図した開発段階のものである」（2016 年 9 月 16 日付の Hyperledger Fabric のリリースを参照）。

概要、第４章では、実験環境の概要を記載する。第５章では、DLT を用いることで既存の中央銀行資金決済システムのサービス水準を満たすことができるかについて、効率性の観点から、流動性節約機能の導入に焦点を当てて評価する。第６章では、同様に、安全性の観点から評価を行う。最後に、第７章では、今次調査を通じて導き出された結論を記載する。

２．今次調査における主な調査結果

今次調査を通じて明らかになった点を予め纏めると、以下のとおりである。

（１）DLT に基づくシステムは、即時グロス決済（real-time gross settlement、RTGS）システムとほぼ同等のパフォーマンスを示し得る。

今次調査では、DLT を応用した環境下でも、ユーロ圏および日本における RTGS システムの処理速度とほぼ同等のスピードで、取引指図を処理できることがわかった。両中央銀行の資金決済システムにおける標準的なトラフィック量（秒間取引指図件数＜Request Per Second、RPS＞が約 10～70 件）の下では、DLT 環境下で処理に要する時間は、平均して１秒を下回った。この間、RPS を 250 件まで増加させたところ、こうしたトラフィック量の増嵩と処理に要する時間との間には、トレードオフの関係があることが確認された。さらに、待ち行列と二者間同時決済を含む標準的な流動性節約機能を、DLT 環境下で実行できることも確認された。

（２）DLT のパフォーマンスは、取引内容の検証作業を行う参加者（検証ノード[3]）の多寡――すなわち、DLT を構成するネットワークの規模――や、これらのノード間の物理的な距離に影響を受ける。

今次調査では、ネットワークの規模とパフォーマンスとの間にある、よく知られたトレードオフ関係についても定量的に確認された。取引検証作業に携わるノードの数を徐々に増やしたところ、取引処理にかかる時間は増大した。また、これらのノード間の距離がパフォーマンスに及ぼす影響は、DLT ネットワークの構成に依ることもわかった。すなわち、検証を行うノードのうち、コンセンサスの形成に必要な数のノードが近傍にある場合には、その他のノードが離れていても、処理時間への影響は限定的であった（ただし、この場合、コンセン

[3] 「検証ノード」（ないし「ノード」）とは、ネットワークに参加するノードのうち、台帳に記録される取引の共有・処理に責任を有するものを表す（別添２を参照）。

サス形成に関わるノードとその他のノードとで台帳の内容に不一致が生じ得る)。その一方で、コンセンサスの形成に必要な数のノードが離れている場合には、処理時間への影響が大きくなることが明らかとなった。

（３）DLT は決済システム全体としての耐障害性や信頼性を高められる可能性がある。

今次調査では、限られた例ではあるものの、①取引検証作業に携わるノードで障害が発生したり、②取引指図のフォーマットが誤っていたりするような場合でも、ネットワーク全体として機能を維持し得ることが確認された。このうち、ノード障害を巡っては、コンセンサスの形成に必要な数のノードが正常に機能している限り、システム全体として継続運行が可能であることがわかった。また、障害が発生したノードは、障害により機能停止していた時間の長さに関わらず、比較的短時間で台帳を更新し得ることがわかった。ただし、今回用いた DLT 基盤では、単一の認証局（certificate authority）が存在しており、DLT によって得られるメリットを減殺するような単一障害点（single point of failure）となり得る可能性には留意する必要がある。このほか、誤ったフォーマットで記載された取引指図が送信された場合でも、全体の処理時間を遅らせることなく、そうした指図を検知できることが確認された。

３．資金決済システムにおける流動性節約機能

欧州中央銀行と日本銀行は、RTGS システムを運営する責務がある。これらのシステムは、それぞれの金融市場において、安全かつ効率的な決済フローを実現するものであり、金融政策や金融システムの安定にかかる中央銀行の責務とも深く関わっている。

このうち、TARGET2 は、ユーロシステムによって運営される RTGS システムであり、金融調節にかかる取引のほか、インターバンク取引や顧客取引、さらには、すべての大口ネット決済システムおよびその他の金融市場インフラ（例えば、証券決済システムやクリアリングハウス等）とのユーロ建て取引も取り扱っている。2016 年の実績をみると、TARGET2 では 1 営業日あたり、金額ベースでは 1.8 兆ユーロ（約 217 兆円）、件数ベースでは 343,729 件の取引を中央銀行マネーを使って決済している。TARGET2 の運営拠点は２つの異なる地域に設置されており、半年ごとにメインサイトを切り替える形となっている。

次に、日銀ネットは、日本銀行によって運営されるRTGSシステムである。日銀ネットでは、短期金融市場での取引や国債取引にかかる資金決済のほか、顧客取引や金融調節にかかる取引、民間ネット決済システムやその他金融市場インフラにかかる資金決済などが処理されている。2016年には、1営業日あたり、金額ベースでは137兆円（約1.1兆ユーロ）、件数ベースでは67,326件の取引を決済している。日銀ネットでは、メインサイトのほかに遠隔地のバックアップサイトが設置されている。

RTGSシステムでは、個々の取引は即時かつグロス・ベースで決済される。また、流動性節約機能を活用することで、RTGSシステムの資金効率を向上させることができる。TARGET2と日銀ネットは、いずれも流動性節約機能を実装しており、具体的には、待ち行列、二者間での取引指図の同時決済（オフセッティング）、多者間での取引指図の同時決済等の機能を備えている。もっとも、その詳細は、2つのRTGSシステムの間で異なる（詳しくは別添1を参照）。

４．実験環境

DLT基盤

DLTは、ネットワーク参加者が何らかのコンセンサス形成メカニズムを通じて台帳を更新することを可能にする技術である。すなわち、複数の主体が取引毎にコンセンサスを形成するため、各取引の正当性やその内容の説明について、十分な制御をきかせることが可能となる。DLT基盤開発の進捗度合いは区々であるほか、各基盤は、参加者の管理、コンセンサス形成アルゴリズム、情報の秘匿性の確保、プログラム可能なスマートコントラクトの内容といった面で異なる。今次調査で利用したHyperledger Fabric（バージョン0.6.1、以下「Fabric」）をはじめとする一部のDLT基盤では、台帳は各ノードが保管するものの、台帳間の整合性については、ビザンチン障害耐性を有する実用的なアルゴリズム（Practical Byzantine Fault Tolerance、PBFT）によって担保されている。

スマートコントラクト

DLTにおいて、取引処理の内容は、スマートコントラクト[4]として実装される。今次調査において、欧州中央銀行と日本銀行は、2種類のスマートコントラク

[4] 本報告書において、「スマートコントラクト」とは、各検証ノード上で実行されるプログラムのソースコードの集合を指しており、法律上の契約（コントラクト）という概念とは異なる。

トを作成した。すなわち、待ち行列への待機や同時決済を行わず、取引指図の振替のみを実行する単純なスマートコントラクトと、流動性節約機能を実行するスマートコントラクトである。両中央銀行が開発した流動性節約機能スマートコントラクトは、それぞれの RTGS システムにおける待ち行列や二者間同時決済の仕組みに基づいて設計されている[5]。

実験手法

欧州中央銀行と日本銀行は、それぞれの RTGS システムで用いられている流動性節約機能の一部をスマートコントラクトとして再現した。その効率性を評価するにあたり、まずベンチマークとして、DLT を一切用いない場合を想定した。次に、分散ネットワークの影響を受けることなく DLT に移行した場合の影響を評価するため、ノード間でのコンセンサス形成を必要としない、検証ノード 1 台の場合を想定した。最後に、コンセンサス形成を必要とする分散的な環境の下で、実験を行った。

実験環境

欧州中央銀行は、行内における専用の仮想環境[6]を使って実証実験を行った。これに対して、日本銀行ではクラウドサービス[7]を利用した。両中央銀行は、同じ実験を、それぞれの環境下で別々に行い、得られた結果が両者で共通である（再現性がある）ことを確認した。

テストデータ

実証実験は、実際の取引明細に基づく仮想データを用いて行われた。仮想の参加者毎に口座が割り当てられ、その口座に関する全ての情報（例えば、口座残高、待ち行列に待機している取引指図）は台帳上に保存された[8]。また、テスト内容に応じて、①事前に定めた一定のトラフィック量で取引指図を投入する場合と②より現実に即した環境下でのパフォーマンスを評価するために、実際の

---

[5] 今次調査で開発した流動性節約機能のアルゴリズムは、TARGET2 および日銀ネットのアルゴリズムを再現するよう設計されているが、必ずしも同一のものではない。

[6] 実験では、仮想コア数 16、メモリ 8GB、ディスク容量 50GB の Red Hat Enterprise Linux 7（RHEL）2 台を用いた。これらのマシン上で、検証ノードの起動やコードの実行を行った。

[7] 検証ノード 1 台につき、メモリ 7.5GB、ディスク容量 8GB の Ubuntu（16.04.1 LTS 64bit）サーバを割り当てて実験を行った。検証ノード数やノード間の距離は、実験シナリオに応じて適宜変更した。

[8] テストデータは日中ピーク時における実際の取引件数に基づく。欧州中央銀行では、口座数は約 200、約 700、約 1,000 口座、取引件数は約 11,000 件/時間であり、日本銀行では、口座数は約 200 口座、取引件数は約 38,000 件/時間である。

トラフィックパターン（例えば、日中ピーク時のパターン）[9]に応じて取引指図を投入する場合を用いた。

パフォーマンスの検証

パフォーマンスは処理時間（レイテンシ）として計測した。ここで、トラフィック量（秒間取引指図件数）は、実際のトラフィック量に応じた値を用いたほか、最大 250 件/秒の場合も用いた。処理時間の計測にあたっては、ある取引指図が送信された時刻と当該取引指図にかかる処理が実行され処理結果がブロックに記録された時刻[10]の差分を、当該取引指図の処理時間としてノード毎に記録した上で、全てのノード、またはコンセンサス形成に必要な数ノード（注：PBFT では総ノード数の約 2/3）における全取引指図の処理時間の統計量を算出した。

安全性の検証

今次調査では、以下の 3 つのシナリオに基づいて、システムの耐障害性を計測することに焦点を当てた[11]。すなわち、①1 台ないし複数の検証ノードで一時的に障害が発生するシナリオ、②参加者や取引指図を認証する役割を担う特別なノードで一時的に障害が発生するシナリオ、③システムに投入される取引指図のうち、一部が誤ったフォーマットで送信されるシナリオ、の 3 つである。安全性の評価にあたっては、これらのシナリオの下での処理時間の遅延度合いや、システムが復旧して機能を回復するまでにかかる時間などを計測した。

[9] テストデータの作成にあたっては以下の論文を参考にした。
Soramäki, K. and Cook, S. (2013), "SinkRank: An Algorithm for Identifying Systemically Important Banks in Payment Systems", *Economics: The Open-Access, Open-Assessment E-Journal*, Vol. 7.

[10] 各取引指図は、決済が完了した場合、もしくは待ち行列に待機した場合にブロックに記録される。

[11] 決済システムは時に予見不可能な様々な脅威に晒されるため、全ての論点を網羅的にカバーする形で安全性を評価することは困難である。また、本実験を行った時点では、DLT 基盤の各機能についての詳細な説明が抜けており、そのテスト範囲や信頼性などにもやや難がみられた点には留意が必要である。

５．効率性の面からみた検証結果

５．１　ネットワークの規模の影響

今次調査では、まず、取引検証作業に携わるノードの数を増やしていった場合にパフォーマンスに与える影響を検証した。これらの作業は、①取引指図の振替のみを実行する単純なスマートコントラクトと②流動性節約機能を実行するスマートコントラクトの両方を用いて行った。

５．１．１　単純なスマートコントラクトの結果

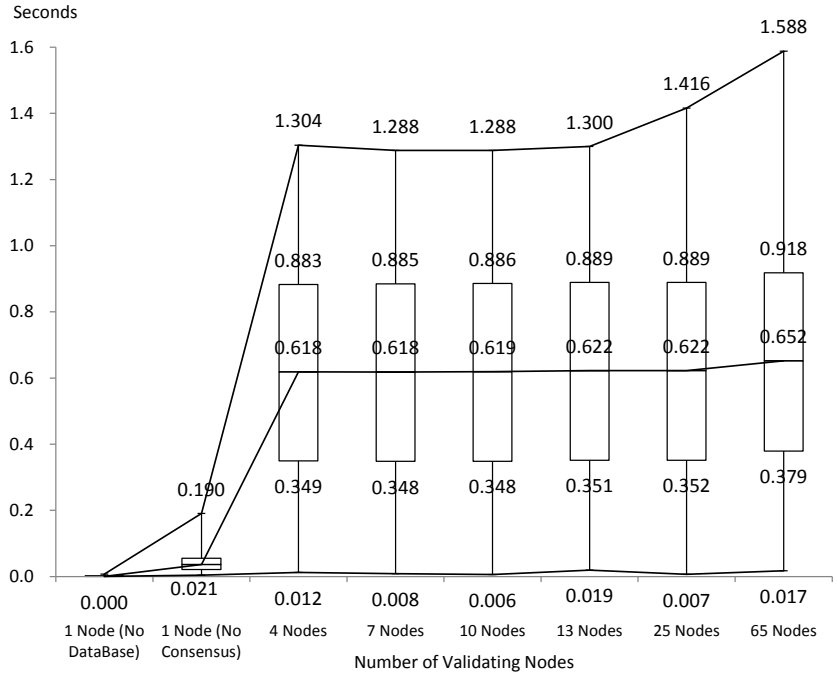
流動性節約機能を用いることなく取引指図を実行する単純なスマートコントラクトの結果をみると、検証ノード数と処理時間はトレードオフの関係にあること――すなわち、ノード数が増加すると、取引指図が処理されブロック上に記録されるまでの時間が長くなること――が確認された。処理時間の中央値は、ノード数が 4～65 台まで変化した場合でも、0.6 秒近辺でほぼ横ばいに推移している一方で、ノード数が増えるにつれて、より長い処理時間を要する取引指図も存在することが明らかとなった（図表 1）。処理時間の最大値は、ノード数が 65 台の場合には 1.6 秒まで達している[12]。

５．１．２　流動性節約機能スマートコントラクトの結果

次に、流動性節約機能を実行するスマートコントラクトの結果をみると、検証ノード数と処理時間の間のトレードオフ関係が改めて確認された（図表 2）。また、流動性節約機能スマートコントラクトを用いた処理時間と、単純なスマートコントラクトを用いた処理時間の差は、0.01～0.02 秒にとどまった。こうしたことから、流動性節約機能を活用しても、処理時間でみたパフォーマンスには、然程大きな影響がみられないことがわかった。

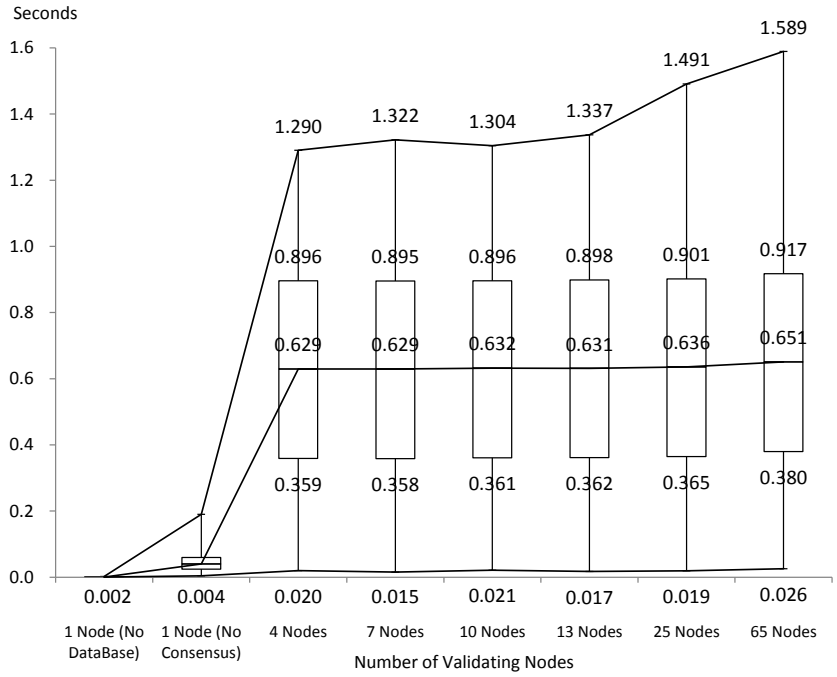
[12] Fabric では、同じネットワークに参加し得る検証ノード数に限界があることが確認された。すなわち、今次調査では、最大 65 台まで同時に参加できることを確認した。

【図表１】単純なスマートコントラクトにおける処理時間の推移



（注）横軸はノード台数、縦軸は処理時間（秒）。グラフの値は下から最小値、25%点、中央値、75%点、最大値を示す。

【図表 2】流動性節約機能スマートコントラクトにおける処理時間の推移



（注）横軸はノード台数、縦軸は処理時間（秒）。グラフの値は下から最小値、25%点、中央値、75%点、最大値を示す。

５．１．３　処理時間の内訳

検証ノードの増加によって処理時間が遅延する要因をみるため、Fabric の処理フローに沿って、処理時間の内訳を確認した（Fabric の概要は別添 2、処理時間の内訳に関する補足情報は別添 3 を参照）。

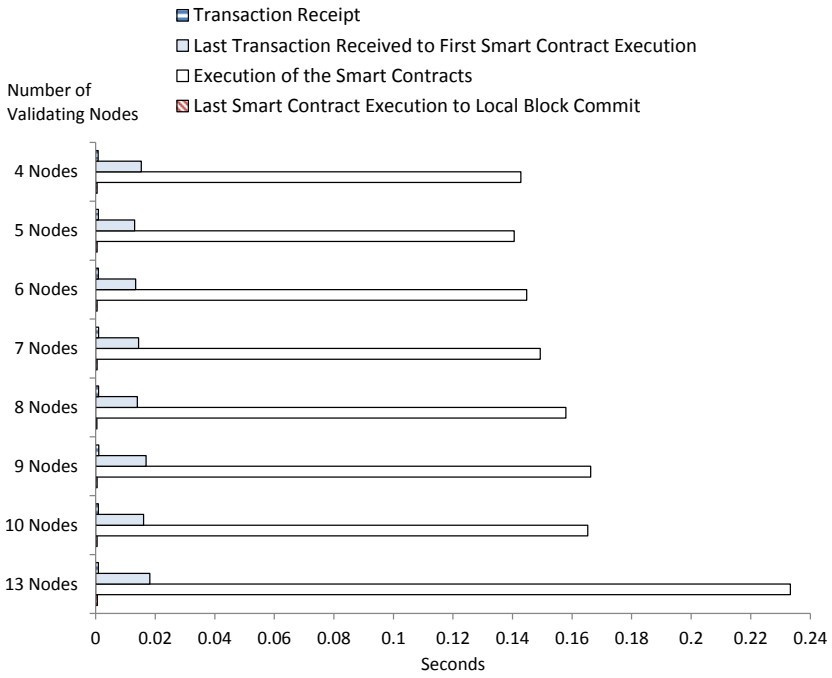
Fabric では、複数の取引指図を一纏めにして一括処理する方式（バッチ処理方式）を採用していることから、処理時間には、取引指図が溜め置かれる時間が含まれている[13]。この時間は平均約 0.5 秒でほぼ一定となっており、ノード数を増やしても、増加する傾向はみられなかった[14]。

図表３は、バッチ組成に要する時間を除いたうえで、処理時間の内訳を示したものである。スマートコントラクトの実行時間（Execution of the Smart Contracts）は、複数の取引指図にかかる処理が直列に行われることもあって、処理時間の大部分を占めている[15]。また、取引指図間の処理順番（時間的な順序性）を決め、検証ノード間でその情報をやり取りする時間（Last Transaction Received to First Smart Contract Execution）は、ノード数が 4 台から 13 台に増加する中で、約 20%増加した。この間、各ノードがブロックへの書き込みに要する時間（Last Smart Contract Execution to Local Block Commit）は、ほぼ無視し得る水準にとどまっている。

なお、ブロックのサイズとRPSとの間には、強い相関関係がみられる（図表４）。

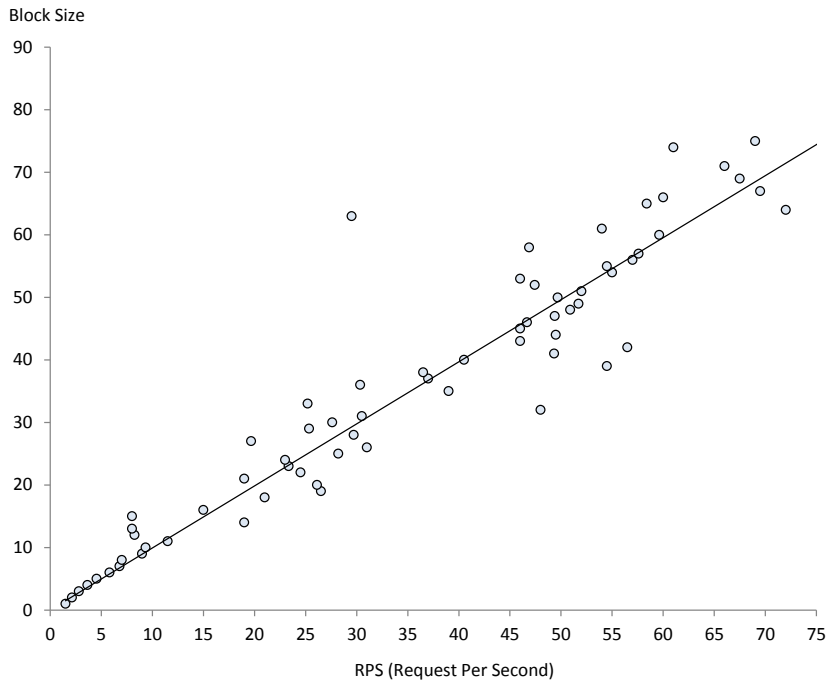
[13] 取引指図の実行およびブロックへの記録は、複数の取引指図を取りまとめた上で一括して行われる（別添２参照）。今次調査で用いたパラメータの下では、未処理の取引指図件数が 500 件に達した場合、もしくは 1 秒が経過した場合に、未処理の取引指図が一括して処理される。
[14] テストデータでは、RPS は 500 件を下回ることから、各ブロックは 1 秒が経過した段階で作成される。このため、ブロックの作成が開始されるまでに、各取引指図は平均して約 0.5 秒待機することになる。
[15] システムが十分な処理能力を有していれば、個々の取引指図にかかるスマートコントラクトの実行時間は概ね一定と考えられる。また、Fabric では取引指図が直列に処理される。結果として、１つのブロックに含まれる取引指図が増加するほど、その処理時間は長くなることになる。

【図表３】処理時間の内訳（バッチ組成の時間を除く）



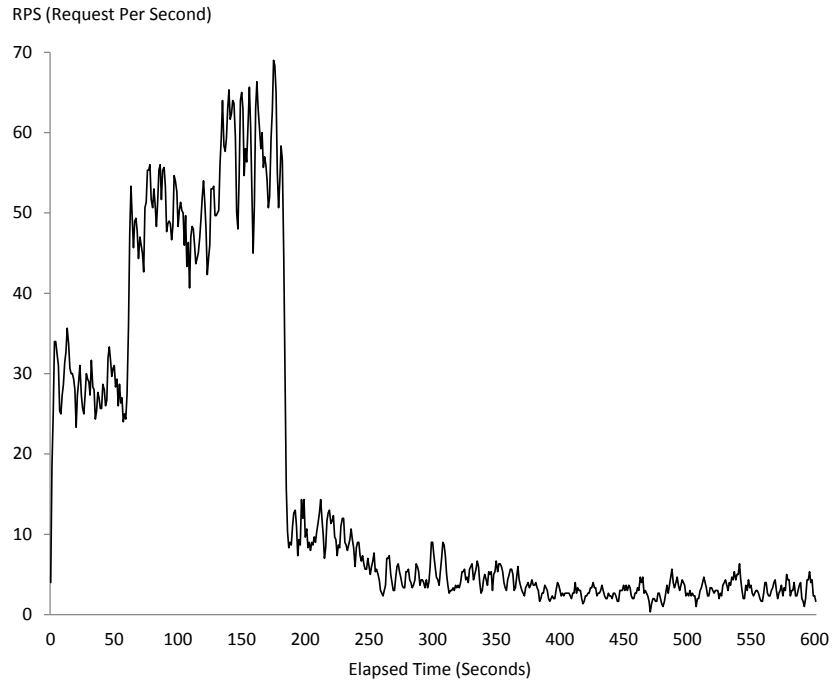
（注）横軸は処理時間（秒）、縦軸はノード台数。凡例の詳細は別添３参照。

【図表４】RPS とブロックサイズの関係



（注）横軸は秒間取引指図件数（RPS）、縦軸はブロックサイズ（1 ブロックに含まれる取引指図の件数）。

【図表５】ピーク時間帯における RPS の例



（注）横軸は経過時間（秒）、縦軸は秒間取引指図件数（RPS）。

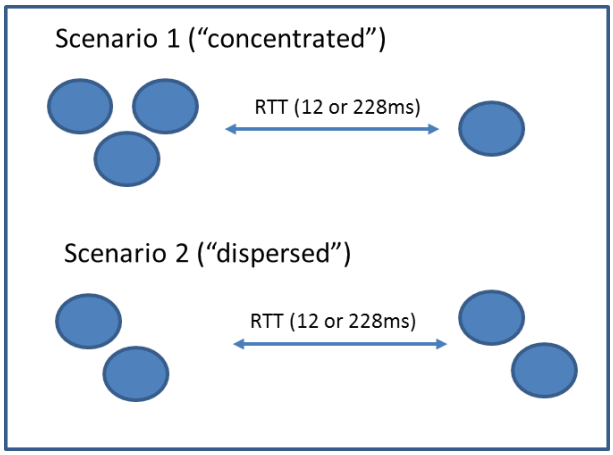
５．２　ノード間の距離がパフォーマンスに及ぼす影響

次に、一部のノードがその他のノードから離れた場所にある場合（すなわち、ノード間での情報伝達に時間を要する場合）に、それがパフォーマンスにどのような影響を及ぼすかについて検証した。

実証実験では、総ノード数を 4 台として 2 つのテスト・シナリオを想定した。第一は、「集中型シナリオ（concentrated）」であり、4 台のノードのうち、3 台のノードが同じ場所にある一方、残り 1 台のノードはそれらのノードとは離れた場所にある場合を想定した。第二は、「拡散型シナリオ（dispersed）」であり、2 つの離れた場所にそれぞれ 2 台のノードが位置する場合を想定した。いずれのシナリオについても、ノード間の距離が、往復遅延時間（round-trip time、RTT）でみて、①12 ミリ秒のケース（フランクフルト・ローマ間、あるいは東京・大阪間の通信に要する時間を想定）、および②228 ミリ秒のケース（フランクフルト・東京間の通信に要する時間を想定）について検証作業を行った[16]。

[16] 欧州中央銀行では、traffic control コマンドを用いて、ネットワークの通信遅延を再現した。日本銀行では、クラウドサービスを用いて、物理的に異なる地域に実際に検証ノードを立てて、実験を行った。

２つのシナリオのイメージ

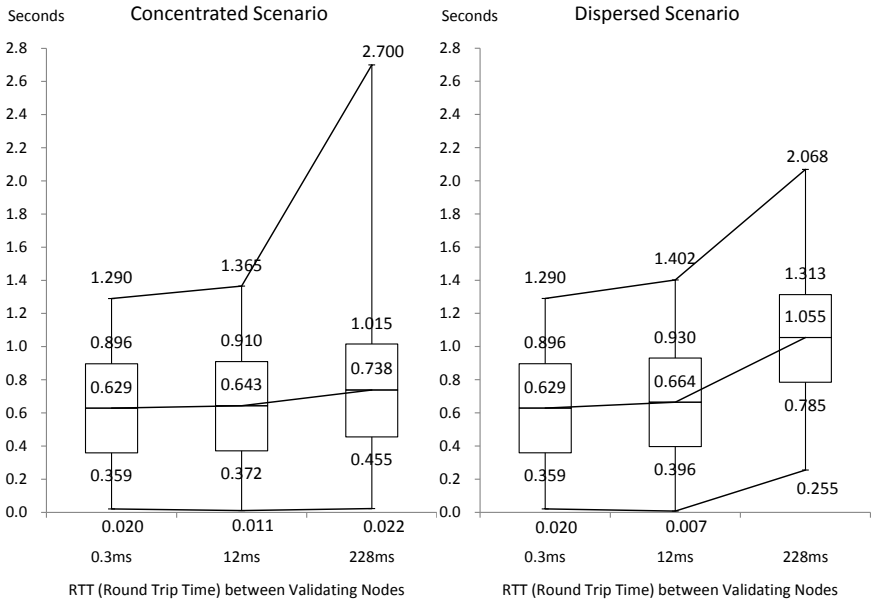


実験結果をみると、集中型シナリオでは、ノード間の距離を変えても大きな影響はみられないことがわかった（図表６、７）。処理時間は、12 ミリ秒、228 ミリ秒のいずれのケースでも、すべてのノードが同じ場所にあるベースライン・シナリオと比べて、ほとんど変わらなかった。もっとも、他のノードから離れているノードにおいては、処理時間が増加（ベースライン・シナリオと比べると、処理時間が最大 112%増加）する場合や、コンセンサス形成に関与することなく他のノードの台帳の状態にキャッチアップしているような証左がみられた。

これに対して、拡散型シナリオでは、ノード間の距離が長い場合には、処理時間がベースライン・シナリオと比べて最大 67%増加することが明らかとなった。

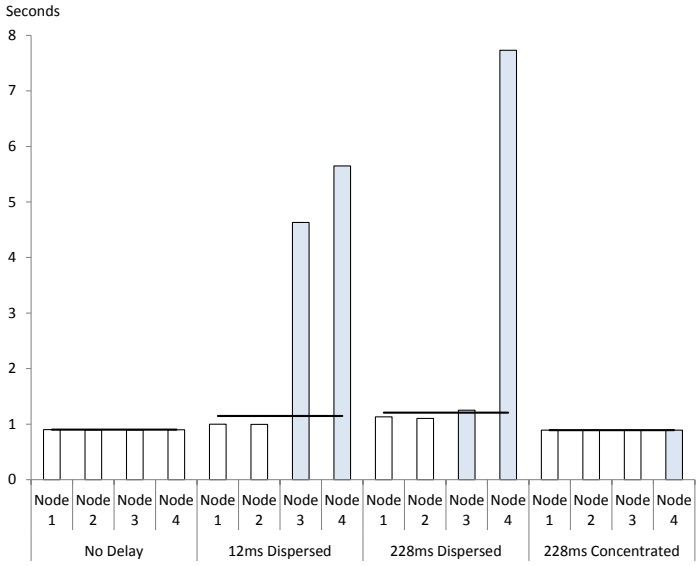
これら２つのシナリオの結果を比較すると、全体として、コンセンサスの形成に必要な数のノード（ノードが４台の場合には３台）が近接している場合では、比較的短時間でコンセンサスに到達することがわかった。これに対して、遠距離に配置されているノードの同意がなければコンセンサスが形成できないような場合には、より時間がかかることがわかった。

【図表６】ノード間の距離が離れている場合の処理時間の推移



（注）左は集中型シナリオ、右は拡散型シナリオの結果。左右いずれのグラフも、横軸はノード間の往復遅延時間（ms）、縦軸は処理時間（秒）。グラフの値は下から最小値、25%点、中央値、75%点、最大値を示す。

【図表７】ノード間の距離が離れている場合の処理時間の例



（注１）横軸はシナリオ毎の各ノード（全４台）、縦軸は処理時間（秒）。

（注２）コンセンサス形成に必要な数のノードが処理を終えた段階で、システム全体として当該処理は完了していると考えられる。このため、横線は、取引の検証および台帳への記帳が総ノード数の約 2/3（3 台）において完了するまでに要した時間を示している。なお、1 台のノードがその他のノードよりも大幅に遅れ、当該ノードの台帳が他のノードの台帳と異なることがあり得る点には留意が必要。

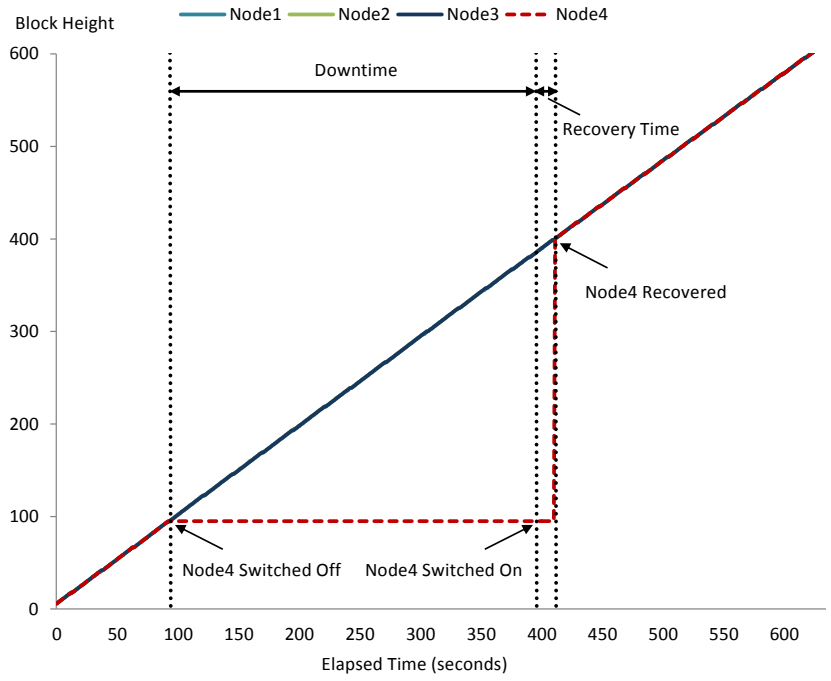
６．安全性の面からみた検証結果

６．１　検証ノードの障害

検証ノードの障害やネットワークの切断などにより、１台ないし複数の検証ノードがコンセンサス形成に関与できなくなった場合、当該ノードが復旧後にネットワークに再接続した上で、正常に機能している他のノードの台帳に合わせて自身の台帳を更新することが必要となる。

今次調査では、検証ノードに障害が発生した場合の影響を評価した。具体的には、４台のノードのうち１台を、一定時間（ダウンタイム）に亘って停止させた後、再起動させ、他のノードの台帳の状態にキャッチアップするまでにかかる時間（リカバリータイム）を計測した（図表８）。

【図表８】ノード障害下でのシステムの可用性



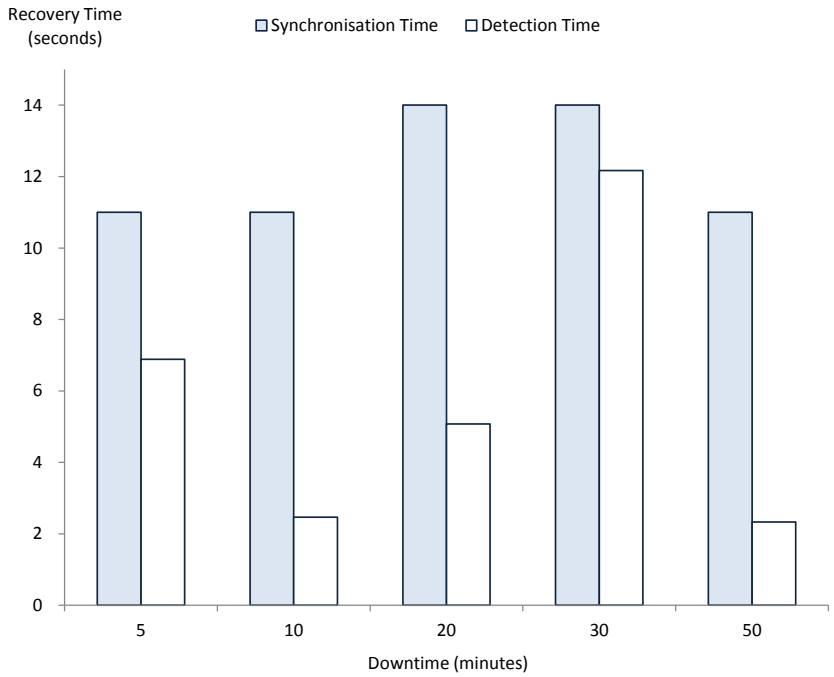
（注）横軸は経過時間（秒）、縦軸はブロック高（累積ブロック数）。

実験結果をみると、コンセンサスの形成に必要な数のノード（ノードが４台の場合には３台）が機能し続ける限り、１台のノードに障害が発生しても、システム全体の可用性（アベイラビリティ）は維持されることが確認された。この間、正常に機能している３台のノードの台帳に記録されるブロックは積み上がり続ける一方、障害が発生したノードの台帳は、当該ノードが復旧するまで更

新されないことも明らかとなった。

全体として、台帳の復旧にかかる時間は、様々なダウンタイムの下でも、比較的短時間（30 秒以内）に収まっていた（図表９）[17]。より詳細にみると、検証ノードは、自身の台帳が他のノードの台帳と一致しているかを定期的に確認している。仮に、台帳間で不一致が検知された場合には、最新の台帳に更新すべく同期が行われる。こうした処理フローに沿って、リカバリータイムを、①台帳間の不一致を検知するのに要する時間（検知タイム）と、②台帳を同期するのに要する時間（同期タイム）に分けたところ、実験で用いた取引パターンの下では、同期タイムは、11～14 秒の間で比較的安定していた一方で、検知タイムは、2～13 秒の間で振れていた。後者については、ノードが復旧した後のトラフィック量の多寡が、台帳の整合性を確認するタイミングに影響したと考えられる。

【図表９】リカバリータイムの内訳



（注）横軸はダウンタイム（分）、縦軸はリカバリータイム（秒）。青の棒グラフは同期タイム、白の棒グラフは検知タイムを指す。

[17] ただし、本実験にあたっては、Fabric の初期パラメータを変更する必要があった。すなわち、障害ノードが復旧後に台帳の同期を行うためには、リーダー検証ノードの交代機能（view change）を停止する必要があった。

６．２　認証局における障害

システムのセキュリティを確保するうえで、ネットワーク参加者や取引指図の登録や真正性の確認は重要である。Fabric では、こうした機能を認証局（certificate authority、CA）が担っている。

Fabric では取引検証作業は分散して行われるように設計されているものの、CA は一台しか存在しないことから、これがシステムにとっての単一障害点になり得る可能性がある。そこで、検証ノードが取引指図の処理を行う中で、CA を停止・再起動させた場合の挙動を検証した。

その結果、CA が停止している間は、新たな取引指図の送信はエラーとなり、こうした指図の送信元に対しては、Fabric が機能を停止しているとの警告メッセージが表示された。その後、CA が復旧すると、指図の処理が自動的に再開されることを確認した。

６．３　誤ったフォーマットの取引指図に対する耐障害性

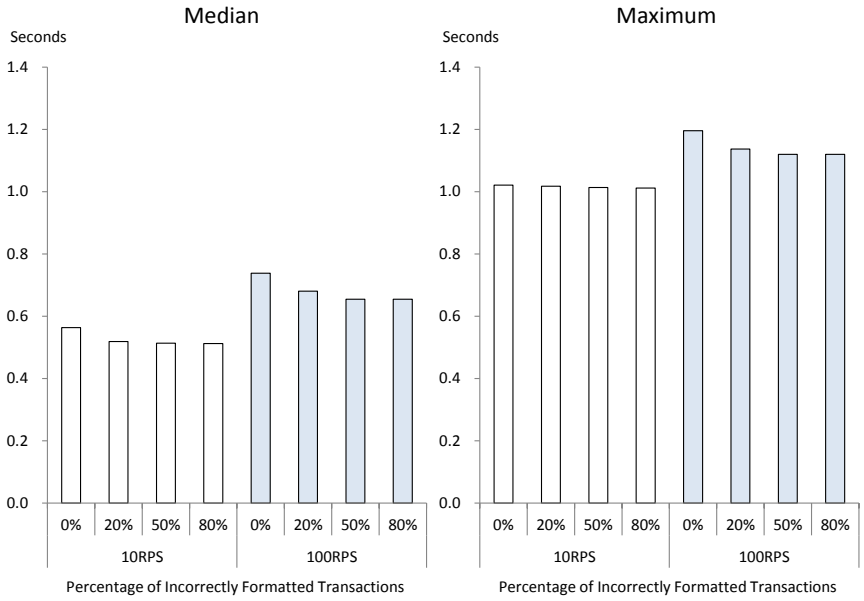
DLT システムの耐障害性を確保する上での課題の 1 つに、誤ったフォーマットで記載された取引指図（不正指図）が大量に送られてきた場合に、システム全体として機能し続けることができるか、という点がある。こうした事例は、例えば、参加者の意図しない行動の結果として起きることが考えられる[18]。

実証実験では、取引指図の一部が誤ったフォーマットで送られてきたと想定し、そうした不正指図の占める割合を 0%から 80%まで増加させた時に、どのような影響を及ぼすかを検証した。こうした不正指図は、（DLT 基盤側ではなく、アプリケーション側の）スマートコントラクト内で定義されたエラー検知機能で対処される。実験結果をみると、システム全体としては、不正指図の割合に関わらず、正しいフォーマットの取引指図を遅滞なく処理できることがわかった。すなわち、RPS を 10 件と 100 件に固定したシナリオの下では、不正指図の割合を徐々に増やしても、処理時間の中央値と最大値は、それぞれ 0.5～1 秒、1～1.3 秒の間でほぼ横ばいで推移することが確認された（図表 10）。この

[18] 今次調査では中央銀行が運営する市場インフラを対象に DLT の応用可能性を検討していることから、参加者が銀行間決済を行う先に限定されるような環境を念頭に置いた。こうした環境下では、インターネット上で匿名の第三者が意図的に大量のトラフィックを送りつけるリスクよりも、銀行間決済の参加者がフォーマットに誤りのある取引指図を意図せずして送信してしまうリスクの方が高いように窺われる。

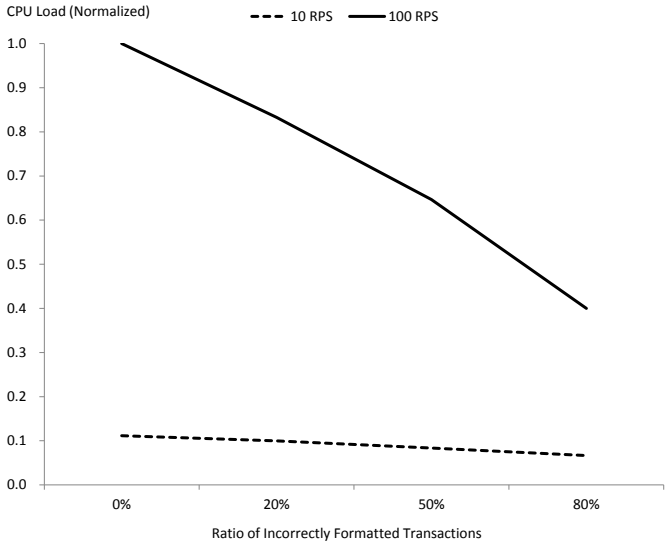
間、トラフィック量が増えるにつれて、システムリソースもより多く必要となるという点についても確認された。すなわち、不正指図の割合が増えるに従って（正しいフォーマットの取引指図の割合が減るため）、システム負荷は低減した（図表 11）。

【図表 10】不正指図が処理時間に与える影響の推移



（注）左は処理時間の中央値、右は処理時間の最大値を指す。左右どちらも横軸は不正指図の占める割合（%）、縦軸は処理時間（秒）。

【図表 11】不正指図がシステムリソースに与える影響の推移



（注）横軸は不正指図の占める割合（%）、縦軸は CPU 負荷（%）。CPU 負荷は、100RPS の下で不正指図の占める割合が 0%の場合を 1 として指標化されている。

７．おわりに

欧州中央銀行と日本銀行は、市場インフラ・サービスの重要な担い手として、それぞれの資金決済システムにおける既存の機能の一部が DLT を用いた環境の下で再現できるかどうかを巡って、掘り下げた実証実験を行うこととした。

まず、効率性の面について言えば、RTGS サービスのうち実験の対象となった部分については、DLT によって、現行の大口資金決済システムが求めるパフォーマンスとほぼ同等の水準を満たす可能性があることがわかった。取引指図を互いに検証しコンセンサスを形成するという DLT の仕組みは中央管理型のシステムと比べて複雑であることを考えると、こうした結果については前向きに評価される。また、実証実験を通じて、ネットワークの規模とパフォーマンスの間のトレードオフ関係を定量的に確認することができた。すなわち、検証ノードの数を増やすと、取引指図の実行にかかる時間が増加した。さらに、検証ノード間の距離がパフォーマンスに影響を及ぼすことも明らかとなった。すなわち、取引指図を処理するのに必要な時間は、検証ノード間の距離に応じて変化する傾向がみられた。

今回の一連の実験を通して、DLT のシステムを設計する際には、ノードの配置やシステムに関する様々なパラメータを適切に調整していくことが重要であることも示唆された。本報告書でも触れているように、ノードの数やノード間の距離によって、パフォーマンスは大きく左右される。さらに、1 つのブロックに含まれる取引件数（一括処理のためのバッチサイズ）や、新規ブロックを組成するまでの待ち時間（タイムアウト）といったパラメータが、パフォーマンスに影響を与えることもわかった。こうしたノードの配置やパラメータ設定については、ユースケースに応じて、十分に検討を重ねていくことが必要である。

次に、耐障害性や信頼性の面では、今次調査は、定量的な結果に裏付けられた新たな視点を提示することができた。すなわち、DLT では、検証ノードにおける障害や、フォーマットの誤った取引指図といった課題に対処する可能性を持ち合わせていることが確認された。このうち、ノード障害については、検証ノードが、障害によって機能を停止していた時間の多寡に関わらず、比較的短時間で台帳を最新の状態に更新できる可能性があることがわかった。また、単一障害点となり得る認証局が障害に晒されている間は新たな取引は行えないものの、認証局の復旧後は、取引処理を自動的に再開することができた。最後に、誤ったフォーマットで記載された取引指図については、スマートコントラクトで検知し排除することが可能であり、システム全体として、正しいフォーマッ

トの取引指図の処理を進める上で問題は生じなかった。

結論として、今次調査では、DLT を資金決済システムに応用する可能性について、前向きに捉えるに値する検証結果が示された。もっとも、今次調査の検証結果は、あくまで実験環境で行われたものであり、本番環境における DLT の実用可能性について評価するものではない点には留意が必要である。

以　　上

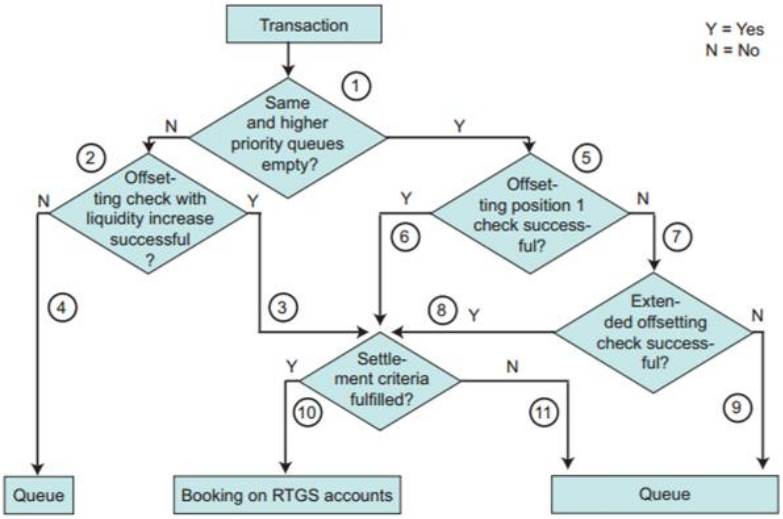
【別添 1】即時グロス決済システムにおける流動性節約機能

TARGET2

TARGET2 では、振替依頼が送信されると、基本的には以下の受付処理が行われる。

- 各振替依頼には、参加者によって、優先順位（「通常」、「優先」、「最優先」の 3 区分）が設定されている。優先順位が設定されていない場合は「通常」扱いとなる。
- 振替依頼の処理を直ちに試みる。ただし、Earliest Debit Time Indicator により、一定時刻以降に決済を行うよう設定されている場合もある[19]。
- 直ちに決済を行うことができない振替依頼については、設定された優先順位に基づき、当該参加者の待ち行列に待機させる扱いとなる。

【図表 A】TARGET2 における受付処理のアルゴリズム



参加者は、待ち行列に待機している未決済の振替依頼について、次の処理を行うことができる。

- 待機振替依頼の優先順位の変更。
- 待機振替依頼の順番の変更。
- 待機振替依頼の決済処理開始・終了希望時刻の変更（送信時に設定されている場合）。
- 待機振替依頼の取消し。

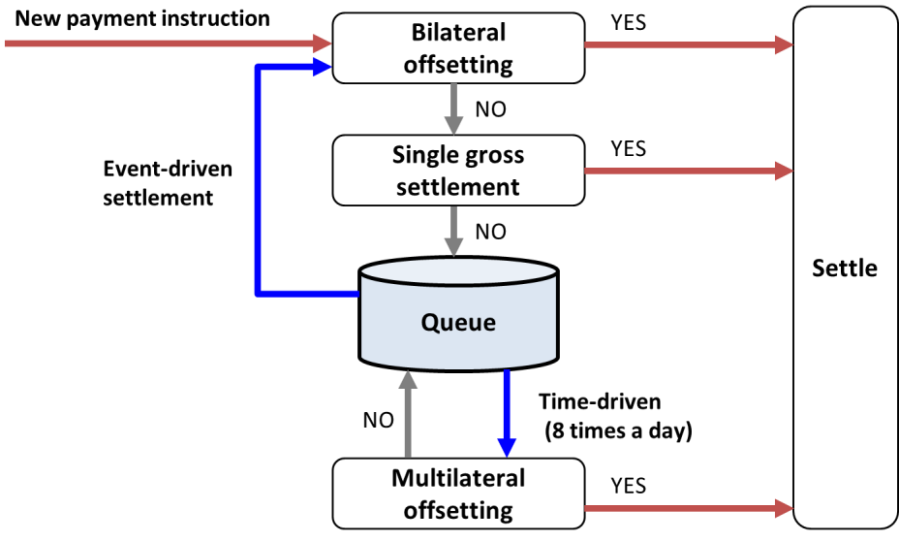
[19] Early Debit Time Indicator が設定された振替依頼は、指定された時刻に初回の決済が試みられる。

待ち行列の解消は、２通りの方法で行われる。第一に、特定のイベントが生じると、参加者の待ち行列に待機している「最優先」ないし「優先」が指定された待機振替依頼の決済が試みられる。第二に、日中を通して、参加者の待ち行列に待機している全ての待機振替依頼を対象として、最適化アルゴリズムが逐次実行される。最適化アルゴリズムについては、今次調査では対象外としている[20]。

日銀ネット

日銀ネットの参加者は、２種類の当座預金口座――単純な振替を行うための「通常口」と、流動性節約機能を利用するための「同時決済口」――を保有することができる。

【図表Ｂ】日銀ネット「同時決済口」における決済アルゴリズム



同時決済口では、振替依頼が受け付けられると、二者間同時決済にかかるアルゴリズムが起動する。例えば、Ａ銀行からＢ銀行宛ての振替依頼が送信された場合、システムはＢ銀行の待ち行列をみて、当該振替依頼とＢ銀行からＡ銀行宛ての待機振替依頼を同時に決済できるかを確認する。同時決済可能な振替依頼が存在しない場合は、当該振替依頼を単独で決済できるかを確認する。即時

[20] 最適化アルゴリズムの詳細については、User Detailed Functional Specifications の 157～170 ページを参照。
https://www.ecb.europa.eu/paym/t2/shared/pdf/professionals/release_10/UDFS_book_1_v10.0_20160712.pdf?2a6a2ac1bbb113e551b563a6a547188f

に決済できない振替依頼は、その優先度（「通常」ないし「優先」）に基づき、当該参加者の待ち行列に待機させる扱いとなる。

二者間同時決済アルゴリズムは、参加者の口座残高が増加した場合ないし待ち行列の最上位にある待機振替依頼に異動が生じた場合にも起動する。この間、参加者は、待ち行列内の待機振替依頼について、その待機順序の変更、取消し、優先順位の変更を通して、自らの待ち行列の管理を行うことができる仕組みとなっている。

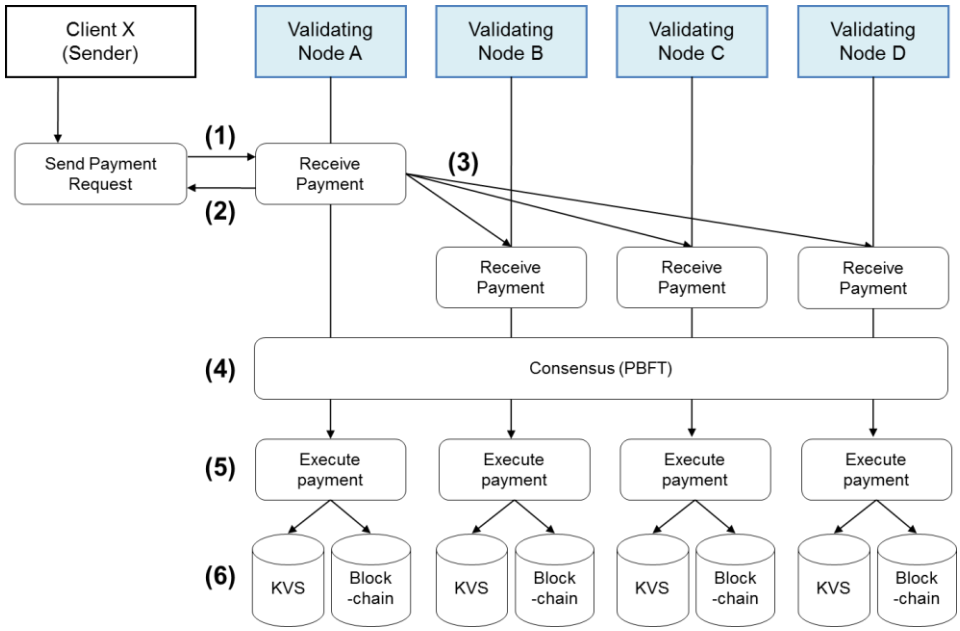
多者間同時決済アルゴリズムは、日中の特定の時刻に起動し、待ち行列の待機振替依頼のうち、同時に決済できる最大の組み合わせを探索する。基本的な考え方は、まず、全ての参加者の待機振替依頼を同時に決済したと仮定し、その場合に資金不足に陥る参加者の待ち行列から、支払額が最大の待機振替依頼を一件ずつ除外していくことによって、全ての参加者が資金不足にならないような待機振替依頼の組み合わせを探すというものである。

【別添２】Hyperledger Fabric の概要

Fabric バージョン 0.6.1 は、参加者を限定するタイプのブロックチェーン基盤であり、ビザンチン障害耐性のあるコンセンサスアルゴリズム、具体的にはビザンチン障害耐性を有する実用的なアルゴリズム（Practical Byzantine Fault Tolerance、PBFT）を備えている。

基本構造：Fabric のネットワークは、検証ノード、認証局およびクライアントアプリケーションから構成される。検証ノード（もしくは単に「ノード」）はトランザクションの検証と台帳の管理を行う。認証局はユーザの身元やアクセス権限を示す電子証明書の発行・取消を行うが、認証局の存在は必須ではない。クライアントアプリケーションはユーザが利用するものであり、検証ノードへトランザクションを送信する。

【図表 C】Fabric バージョン 0.6.1 の処理フロー



（1） クライアント X が自身の電子署名を付してトランザクションを検証ノード A へ送信する。
（2） 検証ノード A は受付通知をクライアント X へ送信する。
（3） 検証ノード A は、事前に定義したバッチサイズ以上のトランザクションの受領、もしくは一定時間経過後に、その間に受領した一連のトランザクションに、通し番号等を付して、他の検証ノードにブロードキャストする（事前準備メッセージ）。
（4） 他の検証ノードは電子署名や通し番号等の確認を行ったうえで、二回のブロードキャスト（準備メッセージ、コミットメッセージ）のやり取りを行う。
（5） 各検証ノードは、コンセンサスの形成に必要な数（検証ノードが全体で 4 台の場合は、自身を含む 3 台）の検証ノードからコミットメッセージを受信したことを確認したうえで、スマートコントラクトにより一連のトランザクションにかかる処理を実行する。
（6） 各検証ノードは、実行結果に基づきステータス情報を更新するほか、新たなブロックを追加して一連のトランザクションにかかる情報（取引の内容、タイムスタンプ等）を保存する。

処理フロー：トランザクションはクライアントアプリケーションから検証ノードへ送信されると、一旦、リーダー検証ノードへ送信される。リーダー検証ノードは自身が受信した複数のトランザクション間の順番（時間的な順序性）を決めた上で、トランザクションの通し番号についてのコンセンサス、すなわち合意を得るために、トランザクションの情報を他の検証ノードへブロードキャストする。トランザクションの通し番号について合意が得られれば、各検証ノードはそれらのトランザクションを実行し、自身の台帳に記帳する。リーダー検証ノードが機能していないとみられる場合には、他の検証ノードにより新たなリーダー検証ノードが選出される。

検証ノードは、事前に定義された一定数のトランザクションを処理する都度、定期的に、自身の台帳上の最新のブロックが他の検証ノードと一致するかを確認し合う。自身の台帳の状態が他の検証ノードから遅れていることを検知した場合は、他の検証ノードから差分を取得して、台帳の同期を行う。PBFT で許容し得る以上の数の検証ノード間で台帳が異なる場合は、ネットワーク全体で処理が停止することとなる。

流動性節約機能のコード設計に関連する技術的な特徴は、以下の通り。

データ複製：全ての検証ノードは同一内容の台帳を共有しており、台帳には任意の形式のデータ（注：ビットコインで用いられる未使用トランザクションアウトプット形式以外も可能という意味）を記録することが可能である。このため、共有する情報の範囲がノード間で異なるような DLT 基盤と比べると、一部の情報を一か所に集中させること（例えば、流動性節約機能の場合は資金の送り手・受け手双方の残高および待ち行列）を前提とするような既存のアルゴリズムの実装は比較的容易である。その一方で、全ての検証ノードが同じ情報を保有する設計は、データの秘匿性の観点で懸念が生じる可能性がある。

決定性の制約：Fabric では、これから処理すべきトランザクションの時間順序性が PBFT アルゴリズムによって決められた後で、各検証ノードが個別にスマートコントラクトを実行し、その実行結果等を台帳に記録する。このため、各検証ノードで同一の結果を得るために、スマートコントラクトは決定的（注：入力が与えられると出力が一意に決まるという意味）でなくてはならない。このことは、タイムスタンプや乱数などの非決定的な値を扱う際には問題が生じ得ることを意味する。

直列処理：Fabric では、各トランザクションは直列に処理される（トランザクションを跨いで並列に処理されることは無い）。このため、ロックなどを用いて共有リソースへのアクセスを制御する必要は無いものの、その処理性能には限界がある。今次調査で欧州中央銀行と日本銀行が実装した流動性節約機能スマートコントラクトは、TARGET2 および日銀ネットで用いられる実際のアルゴリズムとは異なり、直列処理を前提として設計されている。このような直列処理という特性は、多者間同時決済においても、処理時間に大きな影響を及ぼし得るといった課題を生じさせる。

パフォーマンスの計測に関連する技術的な特徴は、以下の通り。

パフォーマンスに影響を与えるパラメータ：Fabric のデフォルト設定である PBFT バッチモードにおいては、スループットと処理時間（レイテンシ）のトレードオフ関係を最適化させるため、複数のトランザクションを一纏めにしてコンセンサス形成を行う。このため、処理時間は、事前に定義されたトランザクション数（バッチサイズ）または待ち時間（タイムアウト）に大きく影響されることとなる。今次調査では、Fabric のデフォルトパラメータ（バッチサイズは 500、タイムアウトは 1 秒）を用いた。

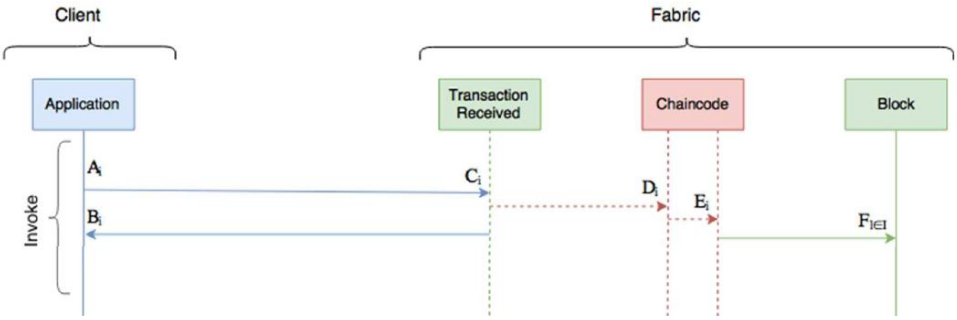
計測方法：Fabric では、照会（query）に対して PBFT は起動せず、照会を受けた検証ノードが自身の台帳に基づいて応答を返す。このため、ネットワーク全体における最新の状態を取得するためには、照会を用いる方法は適当でないと考えられる。今次調査では、クライアントアプリケーション上でトランザクションが送信された時刻と各検証ノード上でブロックが生成された時刻に基づき、処理時間を計測した。

【別添 3】タイムスタンプを用いた処理時間の内訳

以下のフロー図は、トランザクションがクライアントアプリケーションから送信され、一連の処理を経て、ブロックに格納されるまでを示している。この間、以下の 6 つのタイムスタンプが取得できる。

- A：クライアントアプリケーションからトランザクションが送信された時刻。
- B：クライアントアプリケーションに受付通知が送信された時刻。
- C：Fabric のコアプログラムで当該トランザクションを受信した時刻。
- D：スマートコントラクトが起動した時刻。
- E：スマートコントラクトが終了した時刻。
- F：一連のトランザクションがブロックに一括して書き込まれた時刻。

【図表 D】Fabric の処理フローにおける各タイムスタンプのイメージ



本文 5.1.3 における処理時間の内訳にかかる定義は、以下のとおりである。
I はトランザクションの集合、K はブロックの集合、同一ブロックに含まれるトランザクションの添え字を i と定義する。

- トランザクション受信（transaction receipt）：$C_i - A_i$ for $i \in I$ の平均値。
- 最後のトランザクション受信から最初のスマートコントラクト実行まで（last transaction received to first smart contract execution）：
  $D_k - C_k$ for $k \in K$ の平均値。ここで、$D_k = \mathrm{Min}(D_i)$、$C_k = \mathrm{Max}(C_i)$
- スマートコントラクト実行(execution of the smart contracts)：$E_i - D_i$ の合計値。
- 最後のスマートコントラクト実行からブロック生成まで(last smart contract execution to local block commit)：$F_k - E_k$ for $k \in K$ の平均値。ここで、$E_k = \mathrm{Max}(D_i)$。ブロック書込み時刻は同一ブロックに含まれる全てのトランザクションで同一であるため、F_k は当該ブロックに含まれる任意のトランザクションのブロック書込み時刻。