［ブロックチェーン技術の最新動向］

基般

④ 分散台帳技術における コンセンサス・メカニズム

齋藤　新　日本アイ・ビー・エム（株）

コンセンサスとは

本稿では，分散台帳技術（以下，DLT）で使用されるコンセンサス・メカニズムについて解説する．人口に膾炙（かいしゃ）しているのは「ブロックチェーン」という用語であるが，厳密にいうと特定のデータ構造を指す用語であるため，本稿では分散台帳技術，略して DLT，の用語を用いる．コンセンサスとは，分散・並列アーキテクチャを採用する DLT において，データを保持するノードそれぞれの状態を同期する仕組みの総称である☆1．攻撃者やノードの故障からデータを守るため，また，同時アクセスやネットワーク分断のもとでもデータの整合性を担保するため，さまざまなコンセンサスが使用されている．それぞれは前提とする障害モデルおよび想定する攻撃が異なり，性能についても一長一短がある．

本稿では，まずコンセンサスの研究の歴史について概説し，コンセンサスに求められる性質について紹介する．次に，各 DLT 実装で使われるコンセンサスのそれぞれについて解説する．最後に，それぞれのコンセンサスにおける耐障害性・耐改竄性を比較して議論する．

コンセンサス研究の歴史

はじめに，コンセンサスの研究・開発の歴史について簡単に触れる．DLT には大きく分けて 2 つのタイプがある：不特定の参加ノードを想定するパブリック型，参加ノードが特定されているプライベート型☆2 である．1980 年代，はじめにプライベート型の実装に対する研究が始まり，その後 2000 年前後になってパブリック型 DLT のベースとなる技術の研究が進んだ．

プライベート型：ビザンチン将軍問題

1982 年，Lamport らが命名した「ビザンチン将軍問題」[1] がコンセンサスに関する著名な問題である：

> ビザンチン帝国の将軍たちがある都市を包囲していると仮定する．将軍たちはその都市を攻撃するか，撤退するかを合意したい．攻撃が成功するためには，全員で攻撃する必要がある．将軍たちは互いに遠く離れており直接通信ができないため，使者を送ってメッセージを届けるものとする．
>
> ただし，将軍の中には裏切り者がいて，合意を妨害しようとするかもしれない．また，使者がメッセージを届けるのに失敗する可能性がある．メッセージが改竄される可能性もある．
>
> この状況のもとで，もし忠実な将軍たちの意見が（たとえば攻撃に）一致している場合に，彼らは正しい合意（攻撃）に至ることが可能であるか．

裏切り者の将軍はメッセージを意図的に送らない

☆1　分散合意，分散コンセンサス，などさまざまな呼びがあるが，本稿では「コンセンサス」で統一する．また本稿では，コンセンサス・アルゴリズムをしばしば省略してコンセンサスと呼ぶ．

☆2　ここでは，パーミッション型，コンソーシアム型などもこれに含める．

こともできるし，矛盾する複数のメッセージも送れることに注意されたい．このような攻撃者の設定は，この問題にちなみ「ビザンチン障害」と呼ばれる．ビザンチン障害に対して耐性があることを，ビザンチン耐性（BFT；Byzantine Fault Tolerance）を持つ，と呼ぶ．

この問題は1980年Peaseらが提案したものである[2]．彼らは，将軍の総数をN人，そのうち裏切り者の数を高々f人とした場合に，メッセージの送受信に関する条件をいくつか仮定すれば，$N \geq 3f+1$のとき合意に至ることが可能であること，またそれが下限であることを示した．Lamportらはこの問題をビザンチン将軍問題と命名し，手続き的な解法を与えた（図-1）．

これらのアルゴリズムはノード間でメッセージのブロードキャストを繰り返すため性能が芳しくなかった☆3．しかし，1988年にCastroらが効率の良いPBFT（Practical BFT）アルゴリズム[3]を提案し，この分野での研究が加速した．現在では，非常に大きなノード数に対して動作するもの，ノードのオンライン／オフライン状態の変更が激しい場合にでも効率が良いもの，DoS攻撃に耐性を持ちつつスケーラビリティを確保するもの，などの変種が提案されている．

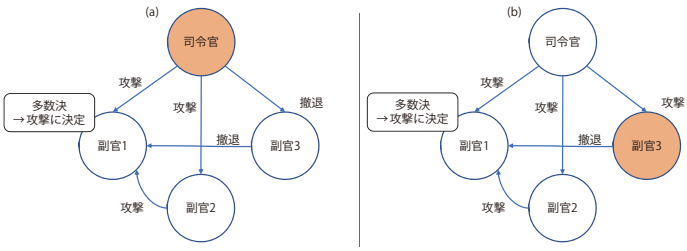

図-1　$N=4$，$f=1$の場合のビザンチン将軍問題の解法
1人の司令官の命令を$N-1$人の副官に伝える，という問題に帰着している．矢印はメッセージを表す．ここでは副官1に関係するメッセージのみ表示している．副官はほかの副官に受け取ったメッセージを転送，各副官は届いたメッセージの多数決を自身の合意結果とする．このとき，(a) 司令官が裏切る場合，(b) 副官が裏切る場合，のいずれも正しく「攻撃」に合意できている．なお，$f>1$の場合は再帰的にさらなるメッセージのやりとりが必要になる．

基本的にはこれらのアルゴリズムは多数決の性質を利用したものである．つまり，1ノード1票と見なして投票させ，多数派の意見を正当なものとして信じる，ということである．したがって，参加ノード数が特定できないパブリック型の実装には適用できない．

Hashcashと Proof of Work

一方，参加ノード数が特定できないパブリック型の実装においては，計算リソース，つまり現実における時間や電気代というコストをユーザに払わせることにより信頼性を担保する仕組みが考えられた．

これらのベースになっているのが，1997年にBackによって提案されたHashcashというスパムメール防止システムである☆4．当時，電子メールの送信コストがほぼ0であることからスパムメールが問題になっていた．Hashcashは，メールを送信する際にはある条件を満たすハッシュ値を与えるような文字列を見つけることを要求するものである（図-2）．そのためには多数のハッシュ計算が必要になる．これにより送信コストに見合わないスパム送信の抑制につながると考えられた．受信者の立場からすると，送信にコストがかかっているメールほど読むに値する，と判断することができた．

Hashcashと，データ同期を促進するようなインセンティブ設計をもとにして作られたのが，2008年にNakamotoにより提案されたビットコインである．台帳にブロックを作成するためにはHashcashとほぼ同様の計算を行う必要がある．

近年では，大量のハッシュ計算に伴う計算リソースの浪費などが指摘され，総称してProof of Stake

```
X-Hashcash: 1:20:1303030600:adam@cypherspace.org::McMybZIhxKXu57jd:ckvi
```

図-2　Hashcashにおいて，メールヘッダの`X-Hashcash`フィールドに格納する値の例（Wikipediaより）．送信するメールにより定まっている情報に加えて，送信者が選んだ`McMybZIhxKXu57jd`という nonce を含んでいる．このフィールド値全体のSHA-1ハッシュ値は16進で`00:00:0b:7c:...`となり，先頭20ビットが0である．この20という値が3～4文字目に含まれている．

☆3　たとえばLamportらのアルゴリズムの通信回数は$O(N^f)$である．

☆4　http://www.hashcash.org/papers/announce.txt

と呼ばれる種類のコンセンサスが多数提案されている．詳細については後述する．

障害モデルおよび通信環境の前提

コンセンサス・アルゴリズムについて比較・議論する際によく用いられる 2 つの障害モデルについて述べる．

クラッシュ障害（不作為障害）は一般的な「障害」の意味に近く，ノードの停止・クラッシュなどが含まれる．一方，ビザンチン障害（作為障害）はメッセージの改竄，矛盾するメッセージの送信，意図的にメッセージを無視するなど，コンセンサス・プロトコルに従わないあらゆる振る舞いを示すような障害，もしくは攻撃を指す．たとえば Paxos や Raft はクラッシュ耐性を持ち，PBFT はビザンチン耐性を持つコンセンサス・アルゴリズムである．

また，ほとんどの場合，通信に関しては非同期，つまり，あるメッセージの送信と受信は異なるタイミングで起こることが仮定される．そして，送信されたメッセージが欠落する，もしくは，送信順と受信順が異なるなどの可能性を想定することがほとんどである．一方で，電子署名技術など暗号技術の発達に鑑みて，送信途中のメッセージは改竄されない・送信者は偽造できない，との仮定を置くことがほとんどである．

コンセンサスに求められる性質

コンセンサスには，前述のような障害モデルなどを仮定した上で，以下の性質が成り立つことが求められる [4]：

1. Liveness/Termination：合意プロセスが開始したらいずれは完了すること，すなわち各ノードが「ある値に合意できた」と判断した状態になること．
2. Agreement：各ノードが思っているその「合意した値」はノード間で一致すること．
3. Validity：ノード間で一致するその「合意した値」は誰かが実際に提案した値であること．これは，常にある一定の値に合意したものと見なすような自明なアルゴリズムを排除するためである．
4. Finality：各ノードの「合意に至ったという判断」「合意した値」がいったん確定すると覆らない．

1 ～ 3 は古典的な分散合意においてよく議論された性質である一方，4 は成り立つことが前提であった．近年の分散台帳では 4 が完全には成り立たないことが多い．他には頑健性，すなわち攻撃に対してシステムが安定であること，および，処理性能（レイテンシ，スループット）などがアルゴリズム評価の対象となる．

コンセンサス・メカニズムの限界

残念ながら，コンセンサスに関する完全なアルゴリズムというものは存在しない．これまでにいくつかの結果が知られている．

- CAP 定理：ネットワーク分断（P）を想定したデータ同期システムにおいて，完全な整合性（C）および可用性（A）を同時に満たすコンセンサス・メカニズムは存在しない．2000 年に Brewer によって提唱され☆5，2002 年 Gilbert らによって証明された☆6．現在のコンセンサス・アルゴリズムは障害ノード数に関して一定の上限を持つという事実はこの定理に裏付けられている．
- FLP 定理：少なくとも 1 台のクラッシュ耐性を持ち，決定的☆7 なアルゴリズムを持つコンセンサスは存在しない．1985 年に Fisher らによっ

☆5 Brewer, E. A. : Towards Robust Distributed Systems (Abstract), In : Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '00, Portland, Oregon, USA: ACM, p.7, ISBN: 1-58113-183-6, doi: 10.1145/ 343477.343502 (2000), http://doi.acm.org/10.1145/343477.343502

☆6 Gilbert, S. and Lynch, N. : Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services, In: SIGACT News 33.2, pp.51-59, ISSN: 0163-5700. doi: 10.1145/564585.564601 (June 2002), http://doi.acm.org/10.1145/564585.564601

☆7 決定的とは，ノードの状態と受け取ったメッセージから次の状態が一意に決まる，という意味である．

て証明された[☆8]．実際，これまでに知られているクラッシュ／ビザンチン耐性を持つアルゴリズムは非決定的な動作を行う．特に liveness を担保するために乱択アルゴリズムやタイムアウトを使用している．

主要な DLT 実装におけるコンセンサス

パブリック型 DLT のコンセンサス・アルゴリズム

パブリック型 DLT の典型的なコンセンサス・アルゴリズムでは，以下の処理が並行して行われる（図 -3）：(1) 各クライアントはトランザクション（送金など，台帳を変更するための指示）をいずれかのノードに送信する．(2) それを受け取ったノードは他のノードに配布する．(3) ノードは未処理のトランザクションをまとめて 1 つのブロックを作成し，それを自己の台帳に追加して他のノードにも配布する．(4) 他ノードからブロックを受け取ったノードはそれを自己の台帳に追加する (5) ノードは台帳に存在するブロックのうち正当であると判断するものを承認し，承認情報を他のノードに配布する．(6) 他ノードから承認情報を受け取ったノードは，自己の台帳の正当な最新状態を決定する．承認は台帳の状態が分岐した場合に，正当な状態（ブロックチェーンの場合は正当なブランチ）を決定するために必要な処理である．

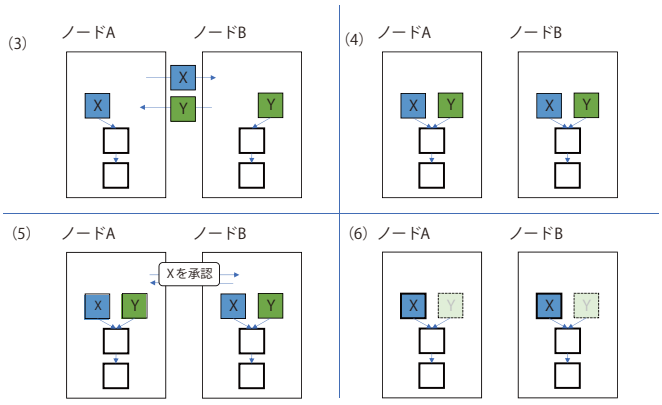

図 -3　パブリック型で標準的に使われるコンセンサス．各四角形がブロックを表す．ブロックは上に追加されるものとする．太字の枠線は承認されたブロックを表す．(6) においてブロック X を含むブランチがその時点でのメインブランチである．

処理 (3)，(5)，(6) は実装による差異が大きい．以下の解説では特筆しない限り，上記のアルゴリズムもしくはそれを微修正したものが用いられているものとする．

ビットコイン

ビットコイン（通貨記号 BTC）は 2009 年に稼働を開始した，最も古い DLT 実装の 1 つである．コンセンサスとしては PoW と Nakamoto コンセンサスを採用している．トランザクションとはアドレス A から B への送金であり，ブロックは 1 個以上のトランザクションの列である．

前述の処理 (3) において，ブロック生成は任意のノードが行うことができる．正当なブロック生成のためには，作成しようとするブロックのヘッダのハッシュ値が，その時点で決められているターゲット値より小さくなるように，ヘッダに含める nonce を探索する必要がある[☆9]．ターゲット値が小さくなるほど nonce を見つける難易度が増す．ビットコインでは，ネットワーク全体で 10 分に一度ハッシュ生成が成功するような難易度に設定されている．生成するブロックには自分に報酬（採掘報酬[☆10]）を支払うようなトランザクションを含めることができる．これがトランザクションを処理しネットワークを維持するインセンティブになっている．

ビットコインでは，処理 (5) および (6) におけるブロックの承認と承認の処理は，ブロック生成時に行われる．ブロック生成においては直前のブロックを指定する（正確には直前のブロックのハッシュ値をブロックヘッダに含める）必要があり，これがそのブロックを承認している処理にあたる．複数の

☆8 Fischer, M. J., Lynch, N. A. and Paterson, M. S. : Impossibility of Distributed Consensus with One Faulty Process, In: Journal of the ACM 32.2, pp.374-382 (1985).

☆9 このブロック生成処理は鉱石の発掘にちなんでマイニングと呼ばれる．マイニングを行うノードはマイナー（miner）と呼ばれる．

☆10 ビットコインでは一定期間ごとに半減し，最終的には 0 になるように設計されている．

ノードが並列にブロックの作成を行うため，時にはブロックのチェーンが分岐することがあり得る．その際にはメインとなるチェーン（メインブランチ）が決められ[☆11]，その上にあるブロックだけが有効になる．前述の採掘報酬は，作成したブロックがメインブランチに将来的に含まれ続けないと使用することができないため，直前のブロックを選択するということは，メインブランチに含まれるべきブロックを指定することに相当し，ひいてはそのブロックを承認することに相当する．まとめると，ビットコインではブロックを作成できたということは，報酬を受け取る権利およびブロックを承認する権利を得たことにあたる．

なお，ビットコインには小さなプログラム（スクリプト）を実行する仕組みはあるものの，署名やハッシュ値のチェックなどに用途が限られている．したがって基本的には，スマートコントラクトを自由に記述・実行する仕組みはないと見なしてよい．

Ethereum

Ethereum（通貨記号 ETH）は 2015 年，Buterin らによって提案，開発された DLT 実装である．

ビットコインと比較したときの大きな特徴は，チューリング完全なスマートコントラクトを実行する仕組みである．スマートコントラクトの実行基盤としてスタックベースの Etherum VM（EVM）を搭載する．高級言語として Solidity などが使用されている．EVM でのコード実行にはステップごとに一定の gas（手数料）が必要である．Ethereum ではアドレスにスマートコントラクトを紐付けることができ，そのアドレスにコインを送ることによりスマートコントラクトの実行が行える．送られたコインはトランザクション作成者が指定したレートで gas に変換され，その gas 量を上限として EVM で実行される．

なお，Ethereum は PoW ベースのアルゴリズムを採用する．ブロック生成間隔は 30 秒程度である．

Tezos

Tezos（通貨コード XTZ）は Liquid PoS（LPoS）と呼ばれる PoS の変種を採用する DLT 実装である．Tezos の最大の特徴は処理性能は目指さず，形式検証を用いて動作の正しさを追求するとしていることである．そのために関数型言語 OCaml で実装されており，一部のモジュールの正しさは Coq[☆12] などを使って証明されている．スマートコントラクトを実行する Michaelson VM についても，型付きのものが採用されている．

もう 1 つの特徴として，コンセンサス・プロトコルを自己更新する機能を持つ．これによりプロトコル変更に伴うハードフォーク問題を解決するとしている．コンセンサスのコア部分はブロックのフォークをどう解決するかということ，トランザクションを処理することにより台帳の状態がどう変更されるかを決定することである．Tezos ではこの部分のプロトコルがプラガブルになっている．プロトコル更新提案トランザクションが提出可能な時期が決まっており，提出されたプロトコルに対してコイン保持者からの投票，テストネット上での動作確認を行い，最終投票で更新が決定すると新しいプロトコルに差し替えられる．

LPoS コンセンサスでは 10,000XTZ を 1 単位（コインロールと呼ばれる）としてブロック作成・承認の権利が与えられる[☆13]．各コインロールは ID を持つ．以前のブロックに格納された乱数から生成権利を持つコインロール ID が生成される．なお，ブロック生成の最短間隔は 1 分であり，204 × 84 ブロック単位で作成・承認権利の割当が行われる．実際にブロックを作成・承認すると報酬が与えられる．また，これらの権利は他者に委譲できる．それに対しても報酬が得られるので，1 ロールに満たな

☆11 なお，参照実装では，それぞれのチェーンに含まれるブロックの難易度（ブロック作成時のターゲット値の逆数）の和を積算し，最も大きいものをメインブランチと見なすようになっている．

☆12 OCaml ベースの定理証明系．

☆13 前述のプロトコル変更に対する投票も同様である．

い保持者については委譲により資産を増やすことができる．現在はコインを保持し続けると報酬が年率5.5% 程度になるように設計されている．

IOTA

IOTA は台帳のデータ構造として DAG を採用する DLT 実装である．IoT ネットワーク向けのスケーラブルな DLT 実装を目指しており，特徴的な設計がなされている．コンセンサスとしては PoW でも PoS でもないと言える．

典型的な分散台帳では，トランザクションは時系列順に並べられる．ブロックチェーン[☆14]と DAG（有向非巡回グラフ）が，トランザクションの依存関係を表現する主要な方法である．ほとんどの実装で，複数のトランザクションがまとめられ，ブロックの単位で扱われる．

ブロックチェーンではジェネシス・ブロックを先頭として，すべてのブロックが直列に並べられる．ほぼ同時にブロックが作成されたなどの理由により，あるブロックの直後ブロックが複数になった場合には，どちらか一方だけが正当なブロックとして承認される．

一方で，DAG ではトランザクションの依存関係を半順序として表現する．たとえば，IOTA の DAG ではブロックではなく，個々のトランザクションを直接台帳に追加する．その際，あるトランザクションの直後トランザクションが複数になること，および，トランザクションの直前トランザクションが複数になること，のどちらも，状態の不整合を引き起こさない限り許される．

IOTA では，トランザクション追加時に，クライアント（IoT 機器などを想定している）は簡単な PoW[☆15] を行う必要がある．さらに，直前トランザクションを 2 つ指定する必要がある．これはトランザクションの承認にあたる．参照実装における指定方法は，より承認されているトランザクションの子孫が多く選ばれるようなランダムウォークで選ぶことになっているが，次に示すように，互いにコンフリクトするトランザクションを先祖に含むような指定はできない．

これを図 -4 の例で説明する．トランザクション B_i が追加された直後の状態 S_i において A が 100 コイン，B と C が 0 コインを持っているとする．この後にトランザクション B_{i+1} : $[A \rightarrow B : 100]$，および，トランザクション B_{i+2} : $[A \rightarrow C : 100]$ が追加されたとする．そうすると 2 つの状態が同時に存在する：前者のトランザクションを追加後の A と C が 0 コイン，B が 100 コイン持っている状態 S_{i+1} と，後者を追加後の A と B が 0 コイン，C が 100 コイン持っている状態 S_{i+2} である．DAG を採用する場合，どちらもこの時点では有効な状態として見なされる．

しかし，ここで両者を直前トランザクションとして持つ B_{i+3} が台帳に追加されたとする．この場合，B_{i+3} 追加後の状態は直前トランザクションを推移的にたどったすべてのトランザクションを反映したものになるべきであるが，負の残高を許さない場合，これは不可能である．したがって，B_{i+3} は B_{i+1} か B_{i+2} のどちらか一方しか直前トランザクションとして指定，つまり承認できない．これにより，いずれか一方のトランザクションのみが正当なものとして承

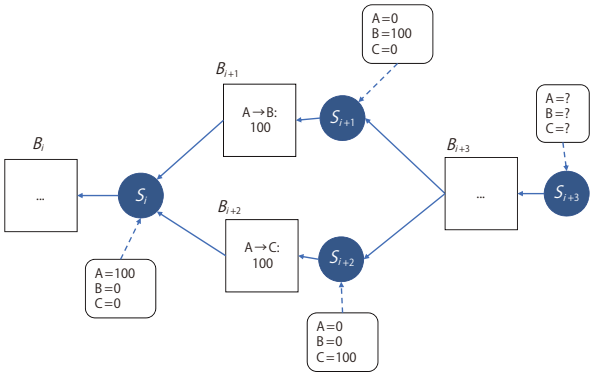

図 -4　DAG においてトランザクションが追加できない例．トランザクション B_i を処理した直後の状態を S_i とする．S_{i+3} は B_{i+1} と B_{i+2} の結果を反映させる必要があるが，負の残高を許さない限り不可能である．

☆14 ここではデータ構造の名称としてブロックチェーンという用語を使用する．

☆15 Nonce を 3^8 個の候補から探す必要がある．

認されるので，台帳全体の整合性は保たれる．

DAG 方式はトランザクション作成・承認作業が並行して行えるので処理性能のスケーラビリティに寄与すると考えられているが，そのためには前述したような非整合性が早期に解消されるようなトランザクション追加アルゴリズムになっていることが求められる．

IOTA は採掘報酬およびトランザクション手数料を持たないため，ノード保持者および資産保持者に対する直接的なインセンティブがないことについては議論がある．

プライベート型 DLT のコンセンサス・アルゴリズム

前述したように，プライベート型 DLT では finality を持つ，多数決ベースのコンセンサスが用いられる（次節参照）．計算能力が票数に相当するパブリック型と異なり，それぞれのノードの重みを対等に 1 票と見て投票していることになる．クラッシュ耐性のためには参加ノード数の約半数まではクラッシュしてもよい．その場合でも過半数の投票を集めることができれば合意に至ることができる．ビザンチン耐性のためには約 1/3 まで（このしきい値を f とする）はビザンチン障害ノードでもよい．この場合には一致する $f + 1$ 個の投票を集めることが必要である．

このアルゴリズムが正しく働くためには，TX の実行結果が決定的であることが求められる．ある 1 ノードが作成した（TX とその実行結果を含む）ブロックを他ノードが承認さえすればよい，パブリック型のアルゴリズムとは対照的である．

性能を追求する実装が多いのもプライベート型 DLT の特徴である．初期のビットコインが 7TPS（transactions per second）程度であったのに対し，数百～数千 TPS に到達するようなものも存在する．

プライベート型の実装はほぼすべてがスマートコントラクトをサポートすることもあり，台帳のデータ構造としてアカウントベースを採用することがほとんどである．

ビザンチン耐性 vs. クラッシュ耐性

ここではビザンチン耐性を持つアルゴリズムとして PBFT，クラッシュ耐性を持つアルゴリズムとして 2014 年に提案された Raft[5] を取り上げる．これらはさまざまなコンセンサス・アルゴリズムのベースとなっているので，ここで両者のアルゴリズムを比較しつつ紹介する（図 -5）．以下，トランザクションを TX と略記する．

どちらも参加ノードのうち 1 台をリーダー

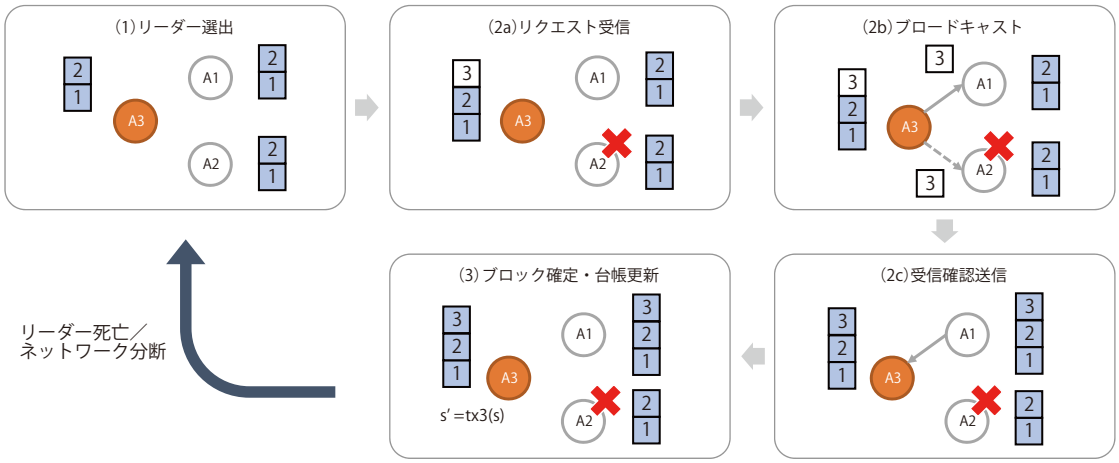

図 -5　全部で A1 ～ A3 の 3 台のノードがいて，A2 は一時的に停止しており，ステップ（1）でノード A3 がリーダーとして選ばれた場合における，Raft コンセンサスの処理概要．

(leader) として，その他をバックアップとして扱う．(1) リーダー選挙 (leader election)，(2) リーダーによる TX の受付とレプリケーション，を繰り返すことにより可用性を保ちつつデータの整合性を担保する．この (1) ～ (2) のサイクルは view, term, epoch などと呼ばれる．

ステップ (1)：初期状態，もしくはリーダーと一定時間通信できないノードが現れた場合，システムは (1) のリーダー選挙に入る．Raft では立候補したノード，PBFT ではノード ID 順で次のノード，がリーダー候補になる．候補者以外はいずれかの候補に投票する．リーダー候補は過半数など，一定数の投票を集めるとリーダーとして認められ，上記 (2) のステップが始まる．どの候補も定足数に至らなかった場合は選挙をやり直す．

(2) のステップでは，クライアントはリーダーに TX を送る (図 -5 (2a))．リーダー以外，つまりバックアップノード，に送った場合についてはそのノードがリーダーに転送するか，クライアントにリーダーへの再送を要求する．リーダーは受け取った TX の直列化を行う，つまり TX にシーケンス番号を割り振る．その後，Raft の場合はリーダーはバックアップにそれを送信し，バックアップはそのままコピーする (図 -5 (2b))．PBFT の場合はビザンチン耐性を保つために 3 ラウンドのやりとりを行い[☆16]，最終的にバックアップにコピーされる．いずれのアルゴリズムにおいても，TX を受け取りそれに合意するバックアップは acknowledge を返信する (図 -5 (2c))．一定数の ack がリーダーに集まったら，その TX は合意された (コミットされた) と判定される．各ノードはその TX を実行して台帳の状態を更新する (図 -5 (3))．

まとめると，バックアップがリーダーに従うことにより TX 列のコンセンサスをとるアルゴリズムである．このような仕組みを，強いリーダー (strong-leader) と呼ぶ．

Hyperledger Fabric

Hyperledger Fabric (以下，Fabric) はオープンソース・コンソーシアム Hyperledger のプロジェクト名および実装名である．Fabric は処理性能を追求するために，やや変わったコンセンサスを採用している．アーキテクチャとしては，オーダラーと呼ばれる，TX を整列する特権ノードを設けたことと，TX の仮実行 (シミュレーション) をあらかじめ行っておき，それが決定的であることを事前に確認する，という特徴を持つ．また，台帳としては DLT で典型的な KVS に加えて，CouchDB など JSON データを格納できるドキュメントストアを採用することにより，ビジネスアプリケーションで必要とされる複雑な検索が可能である．

コンセンサス・アルゴリズムは以下の通りである[☆17]：たとえば，ある時点で A が 200 トークン，B が 200 トークン持っていると仮定する．これは台帳に KVS のデータとして格納されていると考えてよい．ただし各エントリはバージョン属性を持つ．バージョンはエントリに書き込みがあるとインクリメントされる．ここでは A, B のバージョンがそれぞれ v1, v3 であるとする．このとき，$[A \rightarrow B : 100]$ をスマートコントラクトとして実行する場合，この TX は以下のように処理される．(1) クライアントは各ノードにこの TX を送信し，シミュレーションを依頼する．各ノードは実際に台帳の状態を変更せずに，これを実行した場合，KVS に対してどのような読み書きを行うかという read set/write set を返信する．これらをまとめてエンドースメントと呼ぶ．Read set にはシミュレーション中に読み出すエント

☆16 複数ラウンドを要する理由は以下の通りである：Raft においてはいずれのノードも嘘をつかないため，リーダーしか送れないメッセージがバックアップに届いた場合，それはたしかにリーダーが各バックアップに同じメッセージを送ったものである．しかし，ビザンチン耐性を持つためには，悪意あるノードがリーダーを騙って送信してきた可能性や，悪意あるリーダーが各バックアップに矛盾するメッセージを送っている可能性を考える必要がある．そのため，他のバックアップに確認をとり，本当にリーダーからそのメッセージが届いたか否かを確かめる必要があるからである．

☆17 バージョン 0.x と 1.x でアーキテクチャが大きく異なる．ここでは 1.x について説明する．

リとそのバージョン，write set には書き込みを行うエントリとその値を記録する．上記の例では，正しいノードは read set として $\{A : \text{v1}, B : \text{v3}\}$，write set として $\{A : 100, B : 300\}$ を返すはずである．(2) もしノードたちが返すエンドースメントに不一致があれば TX の実行失敗となる．これは一般にはスマートコントラクトの非決定性を表す．(3) もし一定数が一致していれば，それらを当該 TX に含めてオーダラーに送る．(4) オーダラーは各ノードから受け取った TX を順序付けした後，複数個をまとめてブロックを作成し，これをノードたちに配布する．(5) ノードは受け取ったブロックに含まれる各 TX に対し，後述する MVCC 検証を行う．検証を通ったものを合意済みと判断し，TX に含まれる write set を台帳の状態に反映する．その際にはバージョンがインクリメントされるため，エントリ A, B のバージョンはそれぞれ v2, v4 となる．

MVCC（multi-version concurrency control）検証は TX の台帳への反映時，シミュレーション時の write set をそのまま適用できるか否かを判定する方法である．具体的には，TX の反映時において，write set に含まれるそれぞれのエントリのバージョンが，現在の台帳（KVS）の対応するエントリのバージョンと等しくなければならない．これにより，シミュレーション時の TX の実行結果と，TX 反映時点のそれとが一致することが保証される．

Corda

Corda は R3 コンソーシアムにより開発されているプライベート型 DLT 実装である．金融取引に使用されることを念頭においた実装であるとしており，たとえば，契約文書を DLT 内のデータに紐付けることができるようになっている．Corda は自らの実装をブロックチェーンでないとしている．

Corda で扱う資産を作成する際には利害関係者全員の署名が必要である．たとえば借用証書（IOU）であれば，少なくとも債権者と債務者の署名を必要とする．作られた IOU は不変である，つまり，IOU の内容を変更する際には既存の IOU を廃棄して新しく作成する必要がある．これは UTXO 的なデータの管理方法である．利害関係者間で資産の複雑なやりとりを行うスマートコントラクトを実装することが可能であり，これは flow と呼ばれる．

Corda の特徴の 1 つは，台帳上に記録される各資産を利害関係者の間でしか共有しないことである．Fabric や Quorum など他のプライベート型 DLT でも部分的には見られるが，全データに関してこのような設計をとる実装は珍しい．しかし，これはシステムがビザンチン耐性を持たないということである．なお，ある資産が第三者に譲渡されるなど，二重支払いのチェックが必要になる場合には，ノータリー（notary）と呼ばれる特権ノードがチェックを行う．

各アルゴリズムの比較

最後に，コンセンサス・アルゴリズムによる性能の違いを比較する例として，耐障害性・頑健性の比較を行う．

PoW vs. PoS

パブリック型 DLT におけるコンセンサスには参加者（台帳を維持する者，コインを保持する者）へのインセンティブ，ならびに，同一人物が多数のアカウントを作成して多数派を装うシビル攻撃への耐性が求められる．これらは，誰が（優先して）ブロックを作成できるのか，誰がブロックを承認できるのか，ブロック作成や承認への報酬はどう与えられるか，という観点から分類される．

それを実現する主要なアルゴリズムが PoW（Proof of Work）と PoS（Proof of Stake）である．

PoW では Hashcash と同様に，計算能力を使用してブロックを作成させることによりシビル攻撃を防止する．作成したブロックが承認されると作成者に報酬が与えられる．ただし報酬が使用できるのはそのブロックがメインブランチに乗っている場合の

みである．一般にはブロック列が分岐した場合，より大きな計算能力が費やされている方をメインブランチとして採用する．

PoW において指摘される問題点の 1 つは，計算能力の無駄遣いである．Krause らによると，4 大暗号資産のマイニングに使用される電力量はスロベニアのそれに匹敵する[☆18]．マイニングで行っている計算は，基本的には（ランダム値に見えるような）ハッシュ値の計算であり，意味のある計算をさせることはできない．

また，PoW のインセンティブ設計上の問題点も挙げられている．一般的には，報酬はその暗号資産のネットワークおよび台帳の維持に貢献する参加者に与えられるのが望ましい．PoW では報酬はブロックの作成者（マイナー）に与えられる．マイナーは報酬を得たらそれをただちに売却してリアルマネーが得られる．ところが，暗号資産の価値を信じてそれを保持し，使用し続ける利用者にはまったく報酬が与えられないことが問題視されている．

一方，PoS では，その暗号資産を保持している，または，持ち続ける参加者に報酬を与えるような設計をとっている．ブロックの作成権利が現在の所持金額に比例してランダムに決められている実装が多い．また，ブロックの作成には計算コストがほぼかからないような設計が典型的である．また，どちらのブロックを正当なものとして認めるかという承認（投票ともいう）を，ブロック作成者以外でも行えるようになっている．ただし，ブロックの作成などに計算コストがかからないため，チェーンの分岐を促進したり二重投票を行うことが容易である．その対策として供託金制度を採用し，不正と見られる行動に関しては供託金を没収するなどの設計がよく見られる．ただし，PoS の有効性については現在でもいろいろな議論がある．

PoW ではネットワーク全体の計算能力の 51% を独占することにより，好きなブランチをメインブランチにすることができる（51%攻撃）．コインを持っている攻撃者は，あるブランチで取引所にコインを譲渡し現金を受け取る．その後で他のブランチをメインブランチに切り替えることにより，現金は攻撃者にあるまま，コインを譲渡した歴史を消すことができる．実際に，規模の小さい暗号資産のいくつかでは実際に 51% 攻撃が発生している．

ナイーブな PoS はより攻撃を受けやすい．PoW はブランチの切り替えには計算コストがかかり，失敗すると努力は無駄になる．しかし PoS ではブロックの作成が比較的容易である．そのため攻撃者は不正なブロック・ブランチを作成したり二重承認することなどが行いやすい（nothing at stake 攻撃）．対応策としては不正な行いに対してペナルティを設けることである．たとえば Tezos では，ブロック作成者と承認者にデポジットを払わせ，不正が発覚した場合にはそれらを没収する設計になっている．

パブリック型 vs. プライベート型

DLT がパブリック型であるか否かは，コンセンサス・アルゴリズムを含むアーキテクチャ全体に大きな影響を与える．

パブリック型においては台帳の維持のため，ブロックの作成者や承認者にインセンティブを支払う設計がとられている．また，相対的に大きな計算能力を持つ，もしくは，コインを長期保有している参加者が優遇されるような仕組みを持つ．欠点としては，ほとんどのアルゴリズムは厳密な意味でのファイナリティを持っていない．たとえばビットコインでは 51% 以上の計算能力を持っていると，これまでの合意結果を覆すようなブロック列を作成することが可能である．

なお，数は少ないが，EOS のようにパブリック型とプライベート型のハイブリッドなアルゴリズムを採用することにより，参加者を限定しないにもか

[☆18] Krause, M. J. and Tolaymat, T. : Quantication of Energy and Carbon Costs for Mining Cryptocurrencies, In: Nature Sustainabil-ity 1.11, pp.711-718. doi: 10.1038/s41893-018-0152-7 (2018), https://doi.org/10.1038/s41893-018-0152-7

かわらずファイナリティを実現する実装も見られる．

プライベート型 DLT は，主要な特徴として，PBFT など，ファイナリティを持つコンセンサスを採用できることが挙げられる．これは，あるトランザクションを処理する間は参加ノード数が固定であり，そのすべてを把握できていることによる．また，PoW を採用するパブリック型 DLT と異なり，無駄な計算能力を必要としない．これは高い処理性能にもつながる．

一方，パブリック型はその限られた非集中性のために，真の分散台帳でないという指摘も見られる．性能・処理速度・匿名性確保のために特権的なノードを設けることにより，単一障害点が発生したり，厳密な意味でのビザンチン耐性を損なう例が多い．たとえば Fabric の認証局・オーダラー，Corda のノータリー，Quorum の maker などである．これらの完全な解決は難しいが，それらの特権ノードをクラスタリングしそれらのコンセンサスをとることによりある程度解決できる．ただし性能は犠牲になる．

コンセンサス・アルゴリズムの課題と技術開発の方向性

これまでさまざまな DLT 実装ならびにコンセンサス・アルゴリズムが提案されてきた．しかし，DLT は非中央集権型システムにおける銀の弾丸ではなく，すべての要求を満たすことは難しい．今後は，これまでに明らかになった欠点を克服しつつ，それぞれの用途により特化した DLT の研究・開発が進むと著者は考えている．

たとえば，プライベート型 DLT は PBFT に代表されるような多数決ベースのアルゴリズムを使用してファイナリティのあるコンセンサスを実現している．一方で，少なくとも参加者の 2/3 以上がオンライン状態でないと合意プロセスが進まない，など可用性（もしくは liveness）の欠点が存在する．一般に strong leader を採用するアルゴリズムではリーダーの責務が大きく，パフォーマンス・ボトルネックおよび攻撃の対象になり得る．それを解決するため，たとえば，複数リーダーを許容する MirBFT のようなアルゴリズムが提案されている [6]．

参考文献

1) Lamport, L., Shostak, R. and Pease, M. : The Byzantine Generals Problem, In: ACM Trans. Program, Lang. Syst., Vol.4, No.3, pp.382-401, issn: 0164-0925. doi:10.1145/357172.357176 (July 1982), http://doi.acm.org/10.1145/357172.357176
2) Pease, M., Ousterhout, J. and Lamport, L. : Reaching Agreement in the Presence of Faults, In: J. ACM, Vol.27, No.2, pp.228-234. issn: 0004-5411, doi: 10.1145/322186.322188 (April 1980), http://doi.acm.org/10.1145/322186.322188
3) Castro, M. and Liskov, B. : Practical Byzantine Fault Tolerance, In : Proceedings of the Third Symposium on Operating Systems Design and Implementation, OSDI' 99, pp.173-186 (1999).
4) Volzer, H. : A Constructive Proof for FLP, In : Informaion Processing Letters, Vol.92, No.2, pp.83-87 (2004).
5) Ongaro, D. and Ousterhout, J. : In Search of an Understandable Consensus Algorithm, In: 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA: USENIX Association, pp.305-319, IBSN: 978-1-931971-10-2 (June 2014), https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro
6) Stathakopoulou, C., David, T. and Vukolic, M. : Mir-BFT: High-Throughput BFT for Blockchains, In: CoRRabs/1906.05552, arXiv: 1906.05552 (2019), https://arxiv.org/abs/1906.05552

(2019 年 10 月 22 日受付)

齋藤　新　shinsa@jp.ibm.com

2001 年東京大学大学院理学系研究科情報科学専攻修士課程修了，同年日本アイ・ビー・エム（株）入社，現在に至る．分散台帳技術における形式検証の研究に従事．2018 年より同大学院情報理工学系研究科博士後期課程に在学．