Crossing makes the Future.

まじわる力で みらいを創る

ETHTerakoya

スケーリングワーキンググループ（2021/7/7 第3回）

NTTテクノクロス株式会社

デジタルトランスフォーメーション事業部

第一事業ユニット

兼松 和広

※ 本資料の見解は発表者個人の見解であり、所属組織を代表するものではありません。

NTTテクノクロス

合意形成モデル

Consensus protocol client used

・ブロックチェーンにおける系全体の合意を行うための方式を記載する項目。
・通常、登場するのは以下のモデルのいずれかとなる。(太字はプロダクトとしてのデフォルト設定)

系列	製品	合意形成モデル
Ethereum (含むEEA)	Go-Ethereum	**Proof of Work (Ethash)**
		Proof of Authority (Clique)
	Parity	**Proof of Work (Ethash)**
		Proof of Authority (Aura)
	Hyperledger BESU (Consensys Quorum)	Proof of Work (Ethash)
		Proof of Authority (Clique)
		Proof of Authority (IBFT 2.0)
	GoQuorum (Consensys Quorum)	Proof of Authority (Clique)
		Proof of Authority (IBFT)
		Raft
Hyperledger Fabric		Endorsement Ordering + Solo (v2.x以降非推奨)
		Endorsement Ordering + Kafka (v2.x以降非推奨)
		Endorsement Ordering + **Raft**
Corda		系全体の合意形成なし

・合意形成のモデルによって性能が変化するのは当然であるが、
同一の合意形成モデルを採用していても、パラメータの指定やノード配置によっても性能は変化する。

合意形成モデル

Proof of Work

・ブロックチェーンにおいてもっとも有名な合意形成モデル。
代表事例はもちろんBitcoinとなる。

・性能特性としては以下の特徴を有する。

・Authorityノードを持たないため、すべてのノードがブロック生成者になる可能性がある。
・ナンス計算難易度の自動調整があるため、ノード数が増えても性能向上には寄与しない。
・ナンス計算難易度

・PoWを用いたブロックチェーンの性能を提示する場合、以下の値は明示する必要がある。

・**系全体のブロック生成者になりうるノード数**
・**難易度の計測期間中の推移**
→ 採掘ノードを1ノードで構成(その他はすべて非採掘ノード)し、かつ難易度を低い値で設定する、
難易度が高くならない期間のみ測定する事で一時的なブーストが可能になってしまうため。

なお、ASIC/GPUの利用は最終的に難易度調整により吸収されるため、
系全体の恒久的な速度向上には寄与しない。
(ASIC/GPUの利用は採掘報酬を自身のノードで得るためであり、系の性能向上を目的としたものではない)

・直接性能に影響するわけではないが、最終決済性のないプロトコルであるため、
何ブロックで「更新が完了した」と判定するか(コンファメーション数)も測定上影響を与える。

合意形成モデル

Proof of Authority

・ブロックの生成者がブロックごとに系全体の合意において一定の範囲で決められているタイプの合意形成モデル。
・PoAという単語だけではどの合意形成を示すかあいまいである。

・性能特性としては以下の特徴を有する。

  ・ブロックの生成までに一定の段階を踏み、ノード間の通信が発生する。
    →　ノード間の通信時間が合意形成までの時間に大きな影響を与える。

  ・ブロックの生成までのフェーズに処理コストがかかるため、
    通信を受け付けてから実際にブロック生成に入るまでの待ち時間が存在する。

・PoAを用いたブロックチェーンの性能を提示する場合、以下の値は明示する必要がある。

  ・ブロック生成待ち時間
  ・ブロックの承認に必要なノード数
  ・ブロックの承認に必要なノードのNW的な距離

  →　ブロック承認ノードを最小限とすることで「本来の」合意形成としての位置づけを崩して
    性能測定値を稼ぐことが出来てしまうため。

・実証実験の性質などにより、ブロックの承認に必要なノードのNW的な距離は制約が生まれることもある。
(ブロック承認者が世界中のいくつかのデータセンターに分散している場合など)

・PoWと異なり最終決済性を持つ合意形成モデルであることが多いが、例外もある（Clique）ため注意。

合意形成モデル

State Machine Replication

・ブロックの生成者が系全体の合意において一定の範囲で決められているタイプの合意形成モデル。
ただし、ブロック単位でブロックの生成者が選出されず、一定期間（条件）を満たす限り同一のブロック生成者を採用する。

・現状、例示したブロックチェーンプロダクトで採用されているのはRaftのみ。

・性能特性としては以下のような特徴を有する。

　・PBFT系プロトコルのような「投票」の機会がリーダー選出のタイミングのみと少ないため、
　ノード間のネットワーク通信量が少ない。
　（ハートビートを除けばリーダー→フォロワーの方向のブロック配信の通信のみ）

・SMRを用いたブロックチェーンの性能を提示する場合、以下の値は明示する必要がある。

　・リーダーの持続期間（交代間隔）

・Hyperledger Fabricの場合、Raftを指定するのはOrdererを決めるロジックの部分のみであり、
Transactionの正しさを検証するEndorsment-Ordering部分とは別になるため、
そちらはそちらで別途性能条件を明示する必要がある。

・合意形成アルゴリズムとしては**ビザンチン耐性がない**ことには注意が必要。

合意形成モデル

Endorsement Ordering（Hyperledger Fabric）

・トランザクションをブロック生成ノードに投げる前に、
関連するノード（Endorsement Peer）にトランザクションの内容を検証してもらい、
署名を返却してもらうモデル。

・系全体の合意に利用していない点でやや性質が異なるものの、Cordaも本モデルに近い合意形成を行う。

・性能特性としては以下の特徴を有する。

・トランザクションの実行を各Endorsement Peer側で実施するため、CPU/メモリ/NW負荷がかかる。
・トランザクションに署名をつける必要がある範囲はチェーンコードごとに個別でポリシーを設定可能。
→　署名ポリシーの設定、Endorsement Peerの処理性能、NW的な配置などにより、性能測定の結果が変わることがある。

・Endorsement Orderingの性能を提示する場合、以下の値は明示する必要がある。

・各ワークロードごとの署名ポリシーの設定
・トランザクションの承認に必要なノード数
・トランザクションの承認に必要なノードのNW的な距離

・トランザクションの承認だけでは最終決済性に到達せず、
OrdererによってTx順列の整理を行われて初めてトランザクションの最終決済が成立する。

→ Read/Write Setの重複により、トランザクションがOrderer側からINVALIDとされることがある。
→ 上記を避けるためのロジックはブロックチェーンプロダクト側というよりは、
実装するアプリケーション側のロジック（とデータ定義）に依存する。
→ 上記の衝突を最小限にして性能向上させるための中間プロダクト（ Nexledger Accelerator など）も存在する。

合意形成モデル

要求値

・プロダクト自体で選択できる合意形成モデルの数がさほど多くないため、
プロダクトが決定した時点でほぼ択一に近い状態となる。

・その他、Ethereumにおいては
「パブリックメインネット」「パブリックテストネット」「プライベートネット」が存在するため、
特に「パブリックを利用した」とある場合、いずれを採用しているのかは明示の必要がある。

→　パブリックテストネットを用いて商用サービスを展開しているケースもある。

・性能向上のみを目的とした場合、「冗長性」「ビザンチン耐性」が置き去りとなるケースもあり、
顧客側が想定するブロックチェーンシステムの要件や目的と合わせて検討する必要がある。

トランザクション処理

State transition method that changes data in the blockchain from one value to another
ブロックチェーンのデータをある値から別の値に変更する状態遷移法

・性能測定の対象となる処理の内容。

・各アプリケーションによって実装している処理の内容は当然異なるため、
「参照」や「更新」といった単語ではトランザクションの内容を適切に判断することは難しい。

・トランザクションの実行を含むか（ call/query or send/invoke ）、
処理の内部で台帳の参照/更新がどのくらい行われるかが性能上は大きなポイントとなる。

→　OPコードやgolang/pprofの集計結果などがあればおおよそ処理の重さは客観的に提示可能。

台帳の読み込み：SLOAD/getState()/getStateByRange() etc…
台帳の書き込み：SSTORE/putState() etc…
イベントの発呼：emit/putEvent() etc…
暗号化系処理　：golang/crypto呼出 etc…