情報セキュリティ

～公開台帳と分散計算～

安永 憲司

2017.7.4

公開台帳

- 公開台帳 = 誰でも読み書きできる追記専用ログ

- 技術的要件
  - 非許可性 (permissionless)：誰でも参加可能
  - 一貫性 (consistency)：任意時刻における正直ノード同士のログを比較すると、一方が他方の先頭に含まれる
  - 生存性 (liveness)：正直ノードが入力を行うとその内容が正直ノードのログに記録される

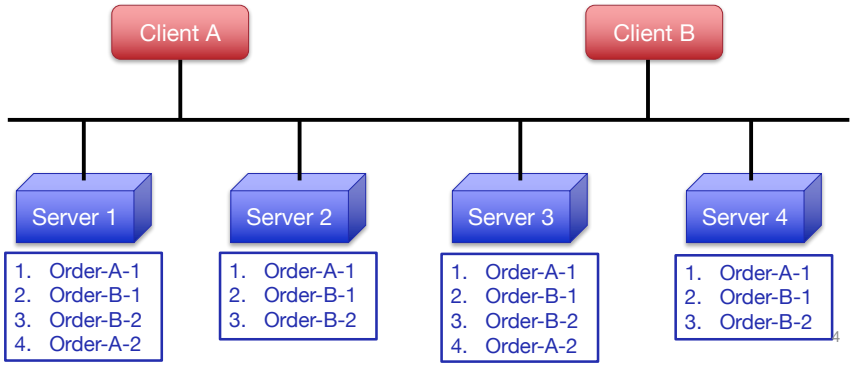
分散計算 (Distributed Computing)

- 複数の計算機がネットワークを介して通信を行い、処理を進める計算
  - 故障計算機がいる場合でも処理を進めたい

- 合意問題 (consensus problem) は基本かつ重要
  - 複数計算機で同じ情報で合意すること
- 様々なモデルが存在
  - 通信タイミング：同期・非同期・部分的同期
  - ネットワーク形状：P2P、リング型
  - 故障の箇所：ノード・リンク
  - 故障の種類：停止・ビザンチン

分散計算における台帳

- State Machine Replication
  - 複数サーバでクライアントの命令を処理し、全サーバで「同じ順番」で処理を行う手法



分散計算における台帳管理

- State Machine Replication を実現する分散計算プロトコルで公開台帳は実現可能（？）
- 分散計算では、誰もが参加可能 (permissionless) な設定は考えられてこなかった
- ただし、参加者を特定した設定（つまり、許可型分散台帳）では様々なプロトコルが存在
  - Paxos
  - Viewstamped Replication (VR)
  - Practical Byzantine Fault Tolerance (PBFT)

Hyperledger プロジェクト

- Linux Foundation がオープンソースソフトウェアによるブロックチェーン技術の整備を目指したもの
  - IBM, Intel, Fujitsu, Hitachi, NTT Data, NEC 等参加
- 現在 5 つのフレームワーク
  - Burrow, Fabric, Iroha, Sawtooth, Indy
- （おそらく）いずれもプライベート・コンソーシアム型ブロックチェーンであり、パブリック型でない
  - Byzantine fault-tolerant プロトコルベース（非許可型ではない分散計算プロトコル）
- ビットコインの思想には反するが、企業受けがよさそう

ブロックチェーン・公開台帳等の呼び方（余談）

- ブロックチェーン (blockchain)
  - Bitcoin（ナカモトプロトコル）の実現方法からの名前
  - ブロックがチェーン状に連なってないとダメ（なはず）
    - ただし、実現したい機能は公開台帳
  - パブリック型・コンソーシアム型・プライベート型ブロックチェーンという呼び方はナンセンス
  - 言葉としてはこれが流行り
- 公開台帳 (public ledger)
  - 台帳技術に対して誰でも参加可能 (permissionless) という意味を持たせている
  - Bitcoin の思想で実現したかった技術はこれ
- 分散台帳 (distributed ledger)
  - 分散計算の延長としての台帳技術
  - 許可型 (permissioned) 技術も公開台帳も含んでいる意味で適した名前であるが、目新しさがない

許可型分散台帳の設定

- 接続ノード情報 ：接続者は ID・アドレスを共有
  - 接続数 n も共有
- ネットワーク形状：P2P 型
  - 任意のノードから任意のノードに通信可能
  - 一般に通信路は認証されておらず、誰からのメッセージか不明
    - 送信者が各メッセージに署名を添付すれば確認可能
- 通信タイミング：部分的同期ネットワーク
  - 送信すれば一定時間内に相手に届くことだけ保証
    - 意図しないメッセージの欠落・改変はない。遅延は発生
  - インターネット通信のモデル化
- ノードの故障：クラッシュ（停止）、ビザンチン（任意の行動・悪意のある行動）

許可型分散台帳プロトコル

- Viewstamped Replication (VR)
  - Oki, Liskov (1988)
  - クラッシュ故障への耐性
  - 故障ノード数 f に対し、$n \geq 2f + 1$ で実現
    - $f < n/2$ は最適
- Practical Byzantine Fault Tolerance (PBFT)
  - Castro, Liskov (1999)
  - ビザンチン故障への耐性
  - 故障ノード数 f に対し、$n \geq 3f + 1$ で実現
    - $f < n/3$ は最適

Viewstamped Replication (VR) のアプローチ

- 要件：
  - f 個のノードがクラッシュしても、ログへの追記ができ、ログの情報は失われない
  - 同時並行に入力が起きても全順序を作る
- 実現のアイディア：
  - リーダーノードを設定
  - 入力はリーダーへ要請し、リーダーが順を決定
  - ログへの追記は f+1 個ノードが確認してから
    - f 個がクラッシュしても 1 個は引き継ぎ可能に
    - $n \geq 2f + 1$ が必要（$n = 2f+1$ を仮定）
  - リーダーがクラッシュすると順次変更

ノードの状態

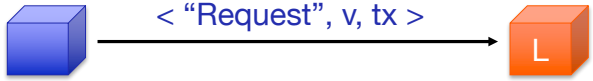
- st $\in$ {normal, view-change, recover} : 現在状態
- v : 最新のビュー番号
- k : 最新の列番号
- LOG : 列番号とその内容の系列（＝台帳）
- リーダーはビュー番号から定まる
  - ノード番号 : 1, 2, …, n
  - リーダー $L = v \pmod n + 1$

VR プロトコル

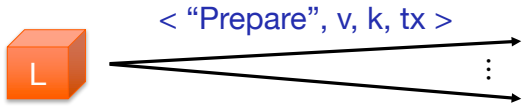
- Normal, ViewChange, Recover の 3 プロトコル
- 各メッセージにビュー番号 v を必ず付加
  - 受け取ったビュー番号 v' が自身のビュー番号 v と同じ時だけ処理
  - 異なる場合、同じにするための手続きを行う
    - $v' < v$ なら v であることを相手に知らせ、$v' > v$ なら最新状態を転送してもらう
- ペア (v, k) は viewstamp と呼ばれる
  - 入力 tx に対しリーダーが (v, k) を割り当て
  - 各 (v, k) に対し、入力は 1 つだけ対応

VR の Normal プロトコル

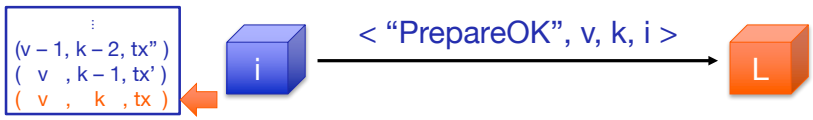
1. 各ノードはリーダー L に入力 tx を要請



2. リーダーは k を増やし Prepare を全員へ送付



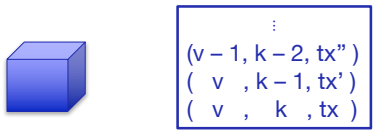
3. 各ノード i は、すべての k’ < k に対し受領すれば ログに (v, k, tx) を追記し、PrepareOK を返送



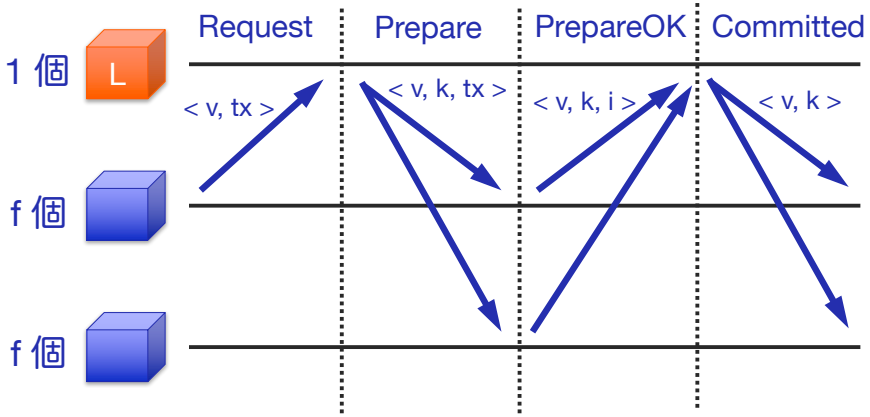
VR の Normal プロトコル

4. リーダーは異なる f ノードから PrepareOK を受け取ると、 committed(v, k ,tx) = 1 として Committed を全員へ送付

5. 各ノードは Committed を受け取ると、 すべての k’ < k を受領するまで待つ



VR の Normal プロトコル（まとめ）

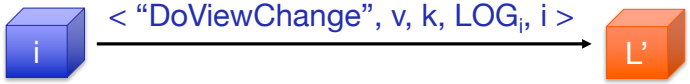


VR の ViewChange プロトコル

- リーダークラッシュ時にリーダーを変える手続き
- どのようなことが問題？
  - 本当にクラッシュしたのか、 遅延が発生しているだけか、区別がつかない
  - リーダーが (v, k, tx) に Prepare を送付後クラッシュし、次のリーダーが (v+1, k, tx’) に Prepare を送付すると、同じ k に別の tx’ が割り当たる
- しかし、ビュー番号が異なるため問題にならない
  - (v, k) と (v+1, k) がある場合、 (v+1, k) を優先
- committed = 1 であるものをすべて引き継ぐ

VR の ViewChange プロトコル

1. ノード i はリーダーが故障したと考えたとき、ビュー番号 v を増やし、状態を view-change にし、次のリーダーへ DoViewChange を送付



2. 次のリーダーは、異なる f+1 ノードから DoViewChange を受け取ると、最新入力として最大の (v, k) をもつログを採用し、状態を normal にし、StartView を全員へ送付

VR の ViewChange プロトコル

3. 各ノードは StartView を受け取ると、(v, k) とログを、送付された情報に更新

■ 注意点

- 状態が view-change のときは、古いリーダーからの Prepare を受け付けない

分散台帳の満たすべき性質

■ 一貫性 : 全ノードに同じ順で入力が記録される

■ 生存性 : 各ノードの入力が記録される

- VR プロトコルの場合、k ノードがクラッシュしても上記性質を満たす必要がある

VR プロトコルが満たす性質

A) committed(v, k, tx) = 1 かつ committed(v, k, tx’) = 1 のとき、tx = tx’

- ほぼ自明

B) committed(v, k, tx) = 1 である tx はすべての非故障ノードに記録される

- committed = 1 であれば、f+1 以上のノードに記録済み
- そのうち 1 つ以上は、ViewChange 実行時に提出される

C) 非故障ノードの入力 tx に対し、Request 受信後から f 個の PrepareOK 受信までリーダーが非故障ならば、ある (v, k) に対し committed(v, k, tx) = 1

- ほぼ自明

VR プロトコルの一貫性・生存性

- 一貫性 : 全ノードに同じ順で入力が記録される
  - (A) より、各 (v, k) に対して committed となる入力は１つだけ
  - プロトコルの性質より、列番号 k の入力は１つ
  - (B) より、committed(v, k, tx) = 1 となれば、すべての非故障ノードで入力 tx が記録

- 生存性 : 各ノードの入力が記録される
  - (C) より、リーダーが一定期間非故障ならば、入力 tx は committed = 1 となる
  - (B) より、committed = 1 の入力は、すべての非故障ノードのログに記録される

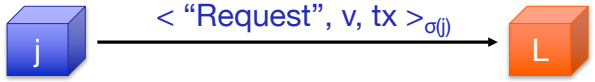
Practical Byzantine Fault Tolerance (PBFT) のアプローチ

- 要件： f 個のノードがビザンチン故障であっても、ログへの追記ができ、ログの情報は失われない
  - リーダーがビザンチンの可能性あり
  - ビザンチンノードが偽情報を送る可能性あり

- 実現のアイディア：
  - 各メッセージに電子署名を付け偽造不可に
  - ビザンチンリーダーに備え、Propose フェーズ導入
    - 「準備をした証拠」を生成してからログに記録
  - 「準備証拠」は 2f+1 ノードの署名付きで構成
    - ノード数 n ≥ 3f + 1 が必要（n = 3f+1 と仮定）
    - 2f ノードでは矛盾した準備証拠が生成される可能性

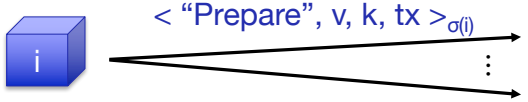
PBFT の Normal プロトコル

1. 各ノード j はリーダー L に入力 tx を要請



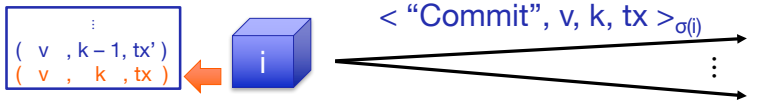
2. Request が有効かつ新しい場合、リーダーは k を増やし Propose を全員へ送付

3. 各ノード i は、すべての k’ < k に対し有効な Propose を受領すれば、Prepare を全員へ送付



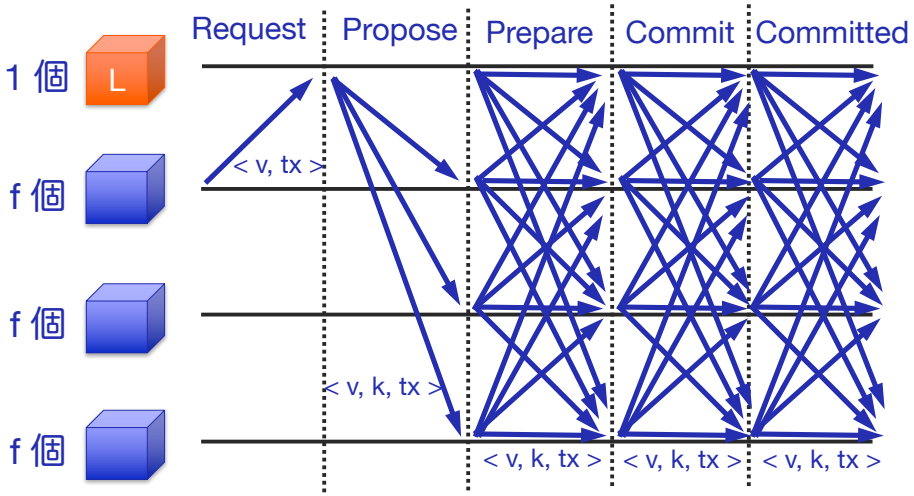
PBFT の Normal プロトコル

4. 各ノード i は、受領した Propose に対応する Prepare を（自身を含む）異なる 2f+1 ノードから受け取ると、ログに (v, k, tx) を追記、Commit を全員へ送付し、prepared(v, k, tx, i) = 1 とする
   - Propose と対応する 2f+1 Prepare ＝「準備証拠」



5. 各ノード i は（自身を含む）異なる 2f+1 ノードから Commit を受け取る、または f+1 ノードから Committed を受け取ると、Committed を全員へ送付し、l-committed(v, k, tx, i) = 1 とする
   - 2f+1 Commit / f+1 Committed = 「記録証拠」

PBFT の Normal プロトコル（まとめ）

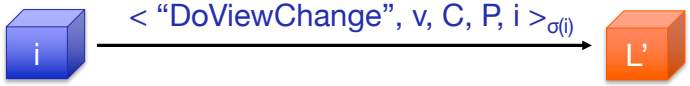


PBFT の ViewChange プロトコル

- リーダー故障時にリーダーを変える手続き
- どのようなことが問題？
  - ビザンチン設定では、不正メッセージも含まれ、どの入力を引き継ぐべきかを正しく判断する必要
  - (v, k, tx) を記録した後、 ViewChange においてビザンチンが (v, k, tx') と主張するかも。ビザンチンの場合、2f ノード分は主張が矛盾する可能性
    - リーダーを含むビザンチン f 個が、ある f 個には tx として、別の f 個には tx' として振る舞うことが可能
- 2f+1 ノード分の署名付き「準備証拠」で解決
  - (v, k, tx) が準備されたことの証明
- prepared = 1 であるものをすべて引き継ぐ

PBFT の ViewChange プロトコル

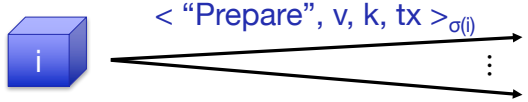
1. ノード i はリーダーが故障したと考えたとき、ビュー番号 v を増やし、状態を view-change にし、次のリーダーへ DoViewChange を送付
   - ノード i の記録証拠集合 C, 準備証拠集合 P を添付



2. 次のリーダーは、（自身を含む）異なる 2f+1 ノードから DoViewChange を受け取ると、その集合を S_1 とし、記録証拠集合・準備証拠集合からログを構成し、準備証拠集合しかない入力に対し Propose を生成し S_2 とし、状態を normal にして、StartView を全員へ送付

PBFT の ViewChange プロトコル

3. 各ノード i は、有効な StartView を受け取ると、状態を normal にし、S_2 に含まれる Propose に対する Prepare 以降の Normal プロトコルを実行



- 注意点
  - 状態が view-change のときは、古いリーダーからの Propose 等は受け付けない
  - StartView を受領後、新しいビュー番号における準備証拠生成のため Prepare 以降を再実行

PBFT プロトコルが満たす性質

A) 非故障ノード i, j において prepared(v, k, tx, i) = 1
かつ prepared(v, k, tx', j) = 1 のとき、tx = tx'

■ 証明：prepared(v,k,tx,i) = 1 → 2f+1 ノードが Prepare 送信
→ tx, tx' 両方に送信したノード数を t とすると、
3f+1 = n ≥ (2f+1) + (2f+1) – t より t ≥ f+1
→ 1 つ以上の非故障ノードが両方に送付 → tx = tx'

B) 非故障ノード i で l-committed(v, k, tx, i) = 1 である
tx はすべての非故障ノードで列番号 k に記録される

■ 証明：l-committed = 1 である
→ ある非故障ノードが 2f+1 個以上の Commit 受領
→ f+1 個以上の非故障ノードで prepared = 1
→ そのうち 1 つ以上は ViewChange 実行時に提出される
→ ViewChange 後の新しいビューのログで引き継がれる

PBFT プロトコルが満たす性質

C) 非故障ノードの入力 tx および非故障ノード i に対し、Request 受信後から prepared(v, k, tx, i) = 1 となるまでリーダーが非故障ならば、l-committed(v, k, tx, i) = 1

- ほぼ自明

PBFT プロトコルの一貫性・生存性

■ 一貫性 : 全ノードに同じ順で入力が記録される
- (A) より、各 (v, k) に対して、非故障ノード i において prepared(v, k, tx, i) = 1 となる入力 tx は 1 つだけ
- したがって、l-committed(v, k, tx, i) = 1 となる入力も 1 つ
- Normal, ViewChangeプロトコルの性質より、列番号 k に対する入力は 1 つだけ
- (B) より、l-committed(v, k, tx, i) = 1 であれば すべての非故障ノードで列番号 k に入力 tx が記録される

■ 生存性 : 各ノードの入力が記録される
- (C) より、リーダーが一定期間非故障ならば、入力 tx は l-committed = 1 となる
- (B) より、l-committed = 1 の入力は、すべての非故障ノードのログに記録される

PBFT プロトコルの効率化

■ PBFT プロトコルは、様々な効率化が可能であることが知られている
- 通信量の削減
- 電子署名の代わりに MAC
- 入力のバッチ処理
- 記憶するデータの削減