PBFT (Practical Byzantine Fault Tolerance) の仕組み

清家大嗣

平成 30 年 9 月 19 日

1　本文書の序論

L. Lamport らは、1982 年にビザンチン将軍問題の定義に成功した。ビザンチン将軍問題を解く初期のアルゴリズムは、接続されたコンピュータが同期していることを仮定[1]し、かつ許容される故障台数 f と同じだけの手続きの入れ子が実装する際に必要であり、現実的に使用するのは非常に困難なアルゴリズム [1] である。

この同期と入れ子による計算量の問題を解決する手法として 1999 年に PBFT (Practical Byzantine Fault Tolerance) [2] は論文としてまとめられた (分散システムは、実装より後に論文が出る傾向がある, Paxos もそのようである)。PBFT は、ビザンチン将軍問題を、非同期かつ比較的効率的に解く[2]。この PBFT アルゴリズムでは、ビザンチン将軍問題において、リーダーが正しいふるまい (non-faulty, not malicious behavior) を行うと、非常に簡単な多数決問題に帰結できるという前提に立つ。そのため、新たに view changes というリーダー再選出の仕組みを加えている。view changes では、リーダーが正しいふるまいを行っているかどうかを多数決で判断し、その結果リーダーへの反乱が成功するとリーダーが変更される。公開鍵暗号のディジタル署名の仕組みによって検閲可能である。

また、現実的になぜビザンチン将軍問題を解く必要があるかについて言及しておきたい。例えば 2008 年に、Amazon は、システムが使えなくなったことがあった (可用性 (availability) が損なわれたということ)[3]。また、Nasa の複数のコンピュータやセンサーの合意の下で、判断を行う話は有名である。

最後に、この文章作成のきっかけは、Alberto Montresor 氏[4]の資料[5] の P. 25 の Proof が示されていないページ (海外の講義では、重要な所を口頭で説明する傾向がある？) から PBFT で示すべき最も重要な性質についてはっきりとした証明が、[2] の論文においても省略されていたことに気づいたためである。

2　ビザンチン将軍問題について

まず、[1] によるビザンチン将軍問題の定義をみる。[1] の表記に倣うため、合意をするコンピュータをプロセスと呼ぶ。ビザンチン将軍問題を解くとは、n 個のプロセス $P_0, P_1, ..., P_{n-1}$ の内、送

[1]通信路の故障により通信が到着しない場合、プロセスが故障したと見做す
[2]非同期システムであるため、FLP 不可能性により、合意が有限時間で保証されることはないことに注意する
[3]http://status.aws.amazon.com/s3-20080720.html
[4]http://cricca.disi.unitn.it/montresor/
[5]http://disi.unitn.it/ montreso/ds/slides17/10-pbft.pdf

り手 P_0 が値 $b_0 \in \{0,1\}$ を持ち、アルゴリズム終了時に各プロセス P_i が値 $b_i = \{0,1\}$ を持ち、以下の 2 条件を満たすようなアルゴリズムを実行することである。

- (条件 1) 故障していない (non-faulty) な全てのプロセスにおいて, b_i の値は一致した値 $b \in \{0,1\}$ となる。
- (条件 2) P_0 が故障していない場合、$b = b_0$ となる。

現実では、b は 0, 1 のような 2 値ではなく、オペレーション (作用) のダイジェスト (ハッシュ値) などについて合意したりする (ブロックハッシュで合意を取るならまさにブロックチェーン)。注意しておきたいのは、条件 2 が存在しない場合、常に $b = 0$ とすれば合意が形成されてしまうことに注意したい。また、[1] では、次のような仮定を置いている。

- 同期システムである
- 通信路に故障がない (通信路の故障は、プロセスの故障となる)

この前提の下、ビザンチン将軍問題一般に成立する次の重要な定理を証明したい。

Th. 1. プロセスの数を n, 故障プロセスを f とした場合, $n = 3$, $f = 1$ において、ビザンチン将軍問題を解くアルゴリズムは存在しない

Proof. $f = 1$ であるため、送り手以外で必ず故障していないプロセスが 1 つ存在する。そのプロセスを P_1 としても一般性を失わないので P_1 を故障していないプロセスとする。そして、次の 2 通りのパターンを考える。送り手 P_0 が故障してる場合、受け手 P_2 が故障している場合である。

P_1 は、P_0 からの判断 $b \in \{0,1\}$ を受け取るが、P_1 が故障しているか判断できないため、その値が正しいか判断できない。そこで、唯一の手掛かりとなりうる P_2 の受け取った値を P_2 から受信する。つまり、これ以上の情報は得られないことと合わせて、P_1, P_2 の値の組み合わせは 4 通りで、この組み合わせに対して必ず 1 通りの出力を出すアルゴリズムを考える (アルゴリズムは決定的に入力から出力が決まるものである)。しかし、P_0, P_2 の出力が異なっている場合、どのようにアルゴリズムを組んでも P_0 と P_2 のどちらが故障しているか判断できず、これに対して 1 通りの出力を解としてしまうと、その選ばれなかったほうが故障していないプロセスの場合、(条件 2) が満たされなくなってしまうケースが発生する。従って、アルゴリズム的に P_0, P_2 から得た情報を下に出力を決めてしまうと、必ず反例が出てしまう。よって $n = 3, f = 1$ では、ビザンチン将軍問題を解くアルゴリズムは存在しない。 □

このことをベースに次の定理が示される。

Th. 2. プロセスの数を n、故障プロセスを f とした場合, $n \leq 3f$ を満たす場合、ビザンチン将軍問題を解くアルゴリズムは存在しない

Proof. 定理が示すビザンチン将軍問題を解くアルゴリズム A が存在すると仮定する。また、$n \geq 4$, $n \leq 3f$ を満たす n に対して、n 個のプロセスを 3 つの集合に分割する。つまり、

- $V_0 = \{V_0, V_3, ...\}$
- $V_1 = \{V_1, V_4, ...\}$
- $V_2 = \{V_2, V_5, ...\}$

となる。$V_0, V_1, V_2 \leq |f|$ であることに注意すると、$n = 3$, $f = 1$ となる問題を解くアルゴリズム B が、アルゴリズム A より作れることから背理法で示す。B の各プロセス P'_0, P'_1, P'_2 の内部に、V_0, V_1, V_2 の子プロセスを生成する。B において、故障しているプロセスを P'_i と置くと、その子プロセス $V_i, V_{i+3}, ...$ は故障している可能性があることに注意する。仮に、P'_0 に与えられた b を P_0 に与え、アルゴリズム A を実行すると、前提から故障していない 2 プロセス (P_0, P_1, P_2 の内 2 つ) は合意が取れる (高々 f 個のプロセスしか故障していないため, アルゴリズム A により honest な子プロセスが全て同じ合意をしている)。つまり、$\{P_0, P_1, P_2\} - \{P_i\}$ のプロセスに関しては、合意が取れているということである。まとめると、仮に定理を否定するアルゴリズム A が存在すると、その子プロセス群を使って、作られた 3 プロセス P_0, P_1, P_2 で $n = 3, f = 1$ の合意形成が取れてしまうので矛盾 ($n = 3f + 1$ の場合に解けるアルゴリズム A に対して証明と同じことをすると、どんなに均等に分割しても $|V_i| = f + 1$ となるプロセス集合が存在し、この場合、$n = 3, f = 1$ のビザンチン将軍問題を A で解くには $n' = 3f + 4 > n = 3f + 1$ 個のプロセス必要なケースが発生するため、矛盾は生じない)。 □

この 2 つの定理は、非同期、同期システムは関係せずに成立する。それは、非同期システムで実現可能なアルゴリズムは同期システムでも実現可能なアルゴリズムという考えに基づている。従って、この定理 2 は、あらゆるビザンチン将軍問題を考える上で重要な定理である。また、[1] の本では、古典的ビザンチン将軍問題を解くアルゴリズムも証明されているが、非常に関数の入れ子の多くなるアルゴリズム (f に比例する) のため、ここでは取り扱わない。

3 PBFT の条件 $N > 3f$ と各用語説明

本節ではプロセスのことをレプリカと言い換える。これは [2, 3] の流儀に則ったものである。PBFT の論文 [2] では前節で述べた同期システムを非同期システムに変え、合意形成の定義を $n - f$ 台の honest なレプリカの内、少なくとも過半数が同意しているといった風に条件を緩くしている。また、ディジタル署名が前提となり、各レプリカが送信する内容がそのレプリカ自身にしか送信できないものというのが保証されるようになっている。

最初に PBFT における以下の定理を証明する。

Th. 3. 最大で f 個の悪意のあるノードに耐えるには、N は $N \geq 3f + 1$ を満たす必要がある

Proof. $N - f$ 個のレプリカとの通信が終了したのちに、各レプリカは自身の判断を行うとする。この場合、$N - f$ としたのは、障害のある (faulty な) レプリカは、通信自体を行わない可能性があり、そのケースに対応するためである (honest なレプリカによる通信が有限時間内に終了するという条件でアルゴリズムは確実に実行されなければならないため)。
しかし、$N - f$ で判断を下すのはリスクもある。これは faulty (malicious ともいう) なレプリカは、誤った応答を素早く伝送する可能性があるためである (悪意あるレプリカによる投票数が占める割合が最大のケースでも honest なレプリカの数が上回るように定足数 (quorum) を決定する必要があるため)。
従って、$(N - f) - f = N - 2f > f$ となるように定足数を決定すれば、多数決実行時に正しい振る舞いをするレプリカの票が必ず faulty なレプリカの票より多くなる。そして、最も小さい N は、$N = 3f + 1$ の時である。これは、節 2 で証明した、ビザンチン将軍問題を解くアルゴリズムが存在しない N を 1 だけ上回る値である。 □

以下、$N \geq 3f+1$ を満たすとして議論を進める。この条件の下、PBFT がどのように実行されていくか説明する。最初に、PBFT で用いる記号を次のように定義する。

- レプリカ (Replica) の ID を $0, 1, 2, ..., N-1$ とする。
- view と呼ばれる、ある時点でのリーダーとなるレプリカ (プライマリーレプリカと呼ばれる) を定義する。リーダーが faulty, malicious なふるまいをして、その後合意形成が成功すると、すぐに 1 インクリメントした ID を持つレプリカがリーダーになるように honest ノードの view が変更される (このプロセスを view changes と呼ぶ)。従って、プライマリーレプリカ i は、$i = v \mod n$ のように定義される ($i = N-1$ の場合に view changes が起きた時、$i = 0$ となるように合意形成するため)
- プライマリーレプリカ (Primary replica) 以外のレプリカは、バックアップ (backups) とも呼ばれる。
- client、プライマリーレプリカにデータベースの状態を変化させるトランザクションを送信する。このトランザクションを処理した結果、何らかの作用 (operation) を働かせることを、オペレーションをインボーク (Invoke an operation: 作用を呼び出す) するなどと言ったりする。

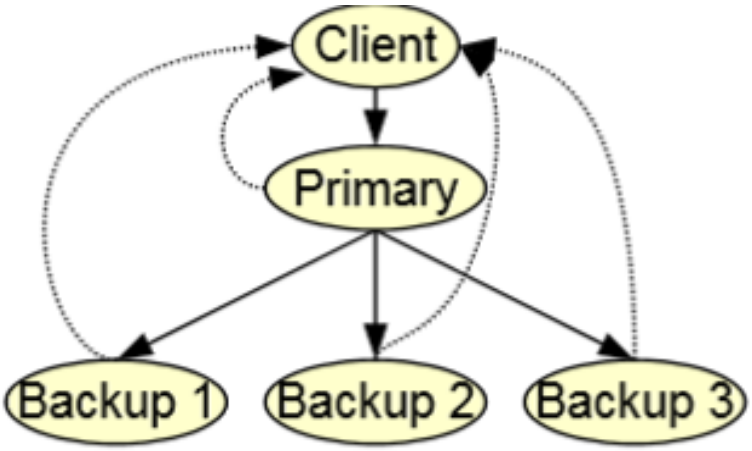

図 1: [2] より引用。client からの要請を受けプライマリーからの指令が backups に伝搬されている

図 1 が、PBFT システムの概要である。プライマリーレプリカが、クライアントのトランザクションを保持 (withholding) しても faulty であることがわからない問題などがあるが、そのことについていったん目を瞑る (2012 年に論文として報告された Aardvark などのシステムがこの問題を解決している。

本題に入る。PBFT の最も重要なアイデアは次の 2 つである。

- アルゴリズムが、ディジタル署名を利用した証明書 (certificates) に大きく依存している。ディジタル署名による証明書を利用する考えは、1982 年の Lamport らの証明には含まれていない。将軍による署名があれば、将軍 (プライマリーレプリカ) が異なる命令を honest なレプリカに送ったことは、honest なレプリカ間の通信によりはっきり証明される。つまり、秘匿する以外の malicious なふるまい (異なった更新命令を出す) は、すぐさまプライマリーレプ

リカの異常行為として検知される。このことは、この後で述べる malicious なふるまいをするプライマリーレプリカを交代させる view changes アルゴリズムにおいて役に立つかもしれない (ディジタル署名技術により将軍の異なった命令を出す異常検知は非常に簡単なものとなるように思われる。

- もう一つの PBFT の重要なアイデアは、投票結果の判定を実行する定足数 (quorum) が $N - f(\geq 2f + 1)$ なことである。この定足数 $N - f$ は、後で説明する PREPARE フェイズ、COMMIT フェイズでの各レプリカが行う 2 回の行動決定をするときの多数決に用いられる。この定足数で多数決を実行すると、同じプライマリーレプリカに対してフォークが発生しないことが証明される。どんなに最悪のケースでも $N - 2f(= f + 1)$ の honest ノード間での合意形成が成立し、その合意形成が成立した場合、他の honest ノードはそれと両立しないメッセージを受け入れるだけの票を集めることができない。PBFT において非常に重要な定理なのだが、[2] のスライドの 25 ページで定理では証明を記述していない。

4 PBFT プロトコルの概要 (Protocol Schema)

PBFT のプロトコルは以下の 4 つに分解される。

- Normal Operations: プライマリーレプリカに障害がなく、backups も最大で f $(N > 3f)$ 台しか故障しない。望ましい状態である。
- View changes: 障害のあるプライマリーを除外し、新しいプライマリーを選択する。
- Garbage collection: 証明書を保管するために使用されたストレージを再利用する方法 (全てのレプリカで合意形成が取れた後の署名は捨ててもよい (データが消えることのない前提でレプリカは動くので)。が、大した容量ではないので保存しててもよさそうである。)
- Recovery: 障害のあったレプリカが復帰する方法: (本文章では省略する)

ここでは、1 つ目と 2 つ目 3 つ目に着目する。

4.1 Normal Operations 手続き

client からのリクエストは <REQUEST, $o, t, c>_{\sigma_c}$ のように記述される。o は、作用 (operation) であり、t はタイムスタンプ、c はクライアントの ID であり、σ_c はリクエストへのクライアントによる署名である (ダイジェスト (ハッシュ) した後の署名でもよい)。図 2 にプライマリーレプリカがリクエストを受信している様子を示す。

次にプライマリー p は、<<PRE-PREPARE, $v, n, d(m)>_{\sigma_p}, m>$ を各レプリカに送信する。v は誰がプライマリーであるか, n はシークエンス番号 (シークエンス番号の順に処理される)、$d(m)$ はメッセージ m のダイジェストであり、σ_p はプライマリーの署名であることを示す[6]。レプリカの v とプライマリーが一致しない場合、シークエンス番号 m やメッセージダイジェスト $d(m)$ がおかしい場合、<<PRE-PREPARE, $v, n, d(m)>_{\sigma_p}, m>$ は捨てられる。ここまでの流れが図 3 である。

[6]ここで、クライアントからのリクエストが全て Pre-Prepare に含まれていないことに注意したい。これは、view changes で用いる保存しておくべき、プライマリーの Pre-Prepare メッセージのデータサイズを少しでも減らすためである (プライマリーはこのフェイズで自身の票を投じているので、次の Prepare フェイズで署名データを送信しない。つまり、これを丸ごと保存する必要があるのでハッシュを取っている (しかし、あくまで全ての更新手続きデータを保存するブロックチェーンでは、このようなことはしないことにも注意したい))

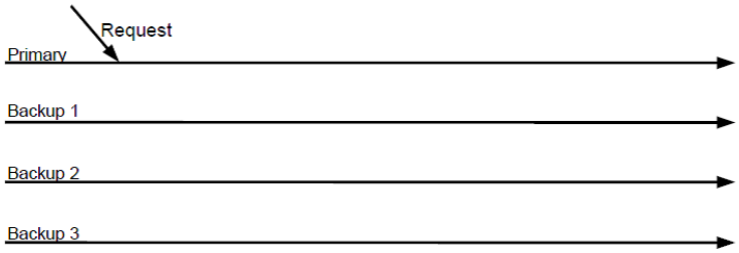

図 2: [2] より引用。client からの要請をプライマリーが受ける

図 3: [2] より引用。client からの要請を受けプライマリーからの指令が backups に伝搬されている

次にプライマリーから受信した Pre-Prepare をベースにして、各レプリカが自身の票を投じる Phase である (これにより、悪意あるリーダーのふるまいに耐性ができる)。レプリカ i は、<PREPARE, $v, n, d(m), i>_{\sigma_i}$ をマルチキャストする。v は自分にとってのプライマリーが誰かを表し、n はシークエンス番号、$d(m)$ はメッセージのハッシュであり、i は自身の ID、σ_i は i によるディジタル署名である。図 4 に、PREPARE フェイズの流れを示す。

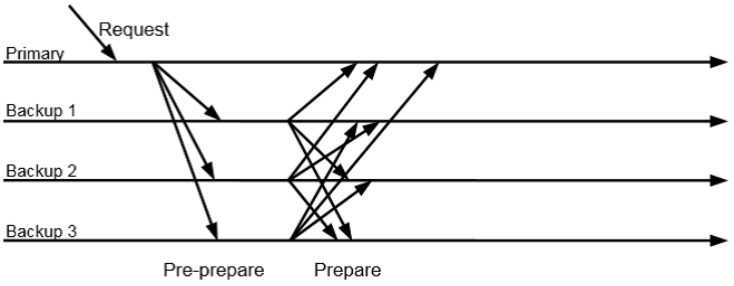

図 4: [2] より引用。PREPARE プロセスまでの実行の流れである

次に PREPARE フェイズで $N-f(N=3f+1$ で, $N-f=2f+1)$ 個の投票を集めた honest なレプリカは Prepare Certificate (P-Certificate) と呼ばれる証明書群について説明する。ここでこの証明書群を、レプリカ i が、他のプライマリーも含むレプリカからマルチキャストされた、<PREPARE, $v, n, d(m), i>_{\sigma_i}$ を集めた集合として prepared(m, v, n, i) とする。prepared(m, v, n, i) は、$N-f$ 個のレプリカが、プライマリーが $v \mod n$ の場合、シークエンス番号が n で処理はメッセージ m で処理することに合意したという証明をしている。そして、このような合意 prepared(m, v, n, i) は、v, n が等しい場合、仮にレプリカ i が異なっても、ただ一つの m についてしか合意できないことを示す。つまり、次の定理が成立する。

Th. 4. *PREPARE* フェイズで合意形成に必要な定足数 *(quorum)* を $N-f$ とした場合、ある特定のシークエンス番号 n で合意形成を行う際に、任意の *2* つの *honest* なレプリカ i,j が、異なるメッセージ m,m' を *Commit* する準備ができるケースは存在しない。つまり、$m \neq m'$, $i \neq j$ において *prepared*(m,v,n,i) と *prepared*(m',v,n,j) という結果が得られることはない *([2] の P. 25 の定理。このスライドには証明はない)*

Proof. prepared 合意形成時に故障している (または悪意のある) レプリカの数を f_c とし、また最大故障許容数を f を分かりやすくするために $f_{\max}$ とする。また、m と合意してしまう honest レプリカ数を h_α, m' と合意してしまうレプリカ数を h_β とする。その場合、次の 2 不等式が成立する。

$$\begin{cases} h_\alpha + f_c \geq N - f_{\max} \\ h_\beta + f_c \geq N - f_{\max} \end{cases} \tag{1}$$

f_c は任意の投票を行える[7]ため、上不等式が成立する。また、honest なレプリカは v,n が等しい PREPARE メッセージに対して 1 度しか署名しないため $h_\alpha + h_\beta = N - f_c$ を利用すると、次式のように変形できる。

$$\begin{cases} h_\alpha \geq N - f_{\max} - f_c = (h_\alpha + h_\beta + f_c) - f_{\max} - f_c = h_\alpha + h_\beta - f_{\max} \\ h_\beta \geq N - f_{\max} - f_c = (h_\alpha + h_\beta + f_c) - f_{\max} - f_c = h_\alpha + h_\beta - f_{\max} \end{cases} \tag{2}$$

この式は、直ちに次のような不等式に変形できる。

$$\begin{cases} f_{\max} \geq h_\alpha \\ f_{\max} \geq h_\beta \end{cases} \tag{3}$$

上不等式から次の 2 つの両立しない不等式が得られるため、明らかに矛盾。従って、honest なノードは、定足数 (quorum) を $N - f_{\max}$ とした場合に、あるシークエンス番号 n に対して必ず 1 通りのメッセージ m について、Commit を行う合意を形成するに至る。

$$\begin{cases} N \geq 3f_{\max} + 1 > 3f_{\max} \\ N = h_\alpha + h_\beta + f_c \leq 3f_{\max} \end{cases} \tag{4}$$

□

従って、PREPARE フェイズで $N-f$ 個の証明書 <PREPARE, $v,n,d(m),i$>$_{\sigma_i}$ を集めたレプリカは、シークエンス番号 n, view が v の場合、メッセージ m の処理のみが行われる可能性があることが確定される。この際, prepared(m,v,n,i) について、現在の view と一致しているか、COMMIT メッセージが条件を満たしているか (トランザクションの処理要件) や、シークエンス番号に関する制約について確認する。そして、ここで初めて COMMIT フェイズへ移行できる。COMMIT フェイズ では、<Commit, $v,n,d(m),i$>$_{\sigma_i}$ をマルチキャストする。マルチキャストされた <Commit, $v,n,d(m),i$>$_{\sigma_i}$ は、Commit のメッセージダイジェストや署名が問題ないか、受信したレプリカの view が v で、シークエンス番号 n が条件と一致している場合に受け入れられる。この流れを図 5 に示す。

ここで、実行証明書 (Commit Certificate: C-Certificate) を新たに定義する。C-Certificate は、P-Certificate を持っているレプリカが、自身を含んだ異なるレプリカから受信した $N-f$ 個の <Commit, $v,n,d(m),i$>$_{\sigma_i}$ を集合としてまとめることにより, committed(m,v,n,i) として生成される。この集合を形成したレプリカは、view が v におけるシークエンス番号 n までの全てのリクエストを実行しているならば、該当する view v とシークエンス番号 n により決定されるメッセージ m を実行可能となる。

[7]同期システムであれば、当然ディジタル署名による異常検知で悪意あるレプリカ判定も容易である。しかし、ここでは非同期アルゴリズムの効率のため各レプリカの投票の異常検知を行われていないことに注意する。

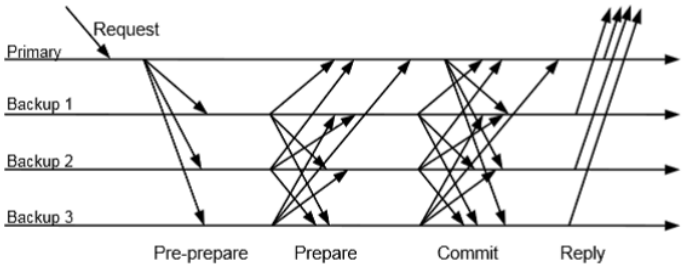

図 5: [2] より引用。Commit フェイズ実行の流れ r

ここで、PREPARE と COMMIT フェイズを分けている理由について説明したい ([4] の GitHub に詳細な説明あり)。COMMIT フェイズは、VIEW-CHANGE の際に、$2f+1$ 個のレプリカによる合意を集めた場合、コミットしている最長のシークエンス番号をそれらのレプリカが保有していることを保証する。仮に、COMMIT フェイズがない場合、各 non-faulty なレプリカは全て異なるシークエンス番号までのメッセージの処理を行っている可能性がある。そのシークエンス番号の長さが長い f 個のレプリカが VIEW-CHANGE の合意形成に参加しない場合、$2f+1$ 個の最長のシークエンス番号 $n_{\max}$ を下に、そこからの処理を新たなリーダー v' が行うようになる。しかし、この状況に至るとブロックチェーンで言うフォークが発生した状況になる (ロールバックが発生する)。従って、COMMIT フェイズを設けて、合意が取れている最長のシークエンス番号が必ず、VIEW-CHANGE の NEW-VIEW の P に含める必要があるのである (少なくとも $(2f+1)-f=f+1$ が最長シークエンス番号の $P-Certificates$ を知っているため、必ず最長シークエンス番号の P-Certificates が含まれることになる)。

最後にトランザクションが処理されたかどうかは、client に <Reply(v,t,c,i,r)>$_{\sigma_i}$ という形で伝えられる。r はリクエストの結果である。この同一な結果が $f+1$ 個以上集まれば commit が少なくとも 1 つの honest ノードで実行されたことが示される。

4.1.1 Normal Operations のまとめ

以下の 5 つの処理から成立する。Client から、リクエストをリーダーが受けて、そのリクエストに基づいた処理を記述した署名ありのメッセージを PRE-PREPARE として伝搬。その後、PRE-PREPARE を受け取った全レプリカとリーダーの間で受け取った処理を署名付きで PREPARE を伝搬させることで共有 (リーダーは既に PRE-PREPARE で自身の判断を送っているので受け取るだけ)。最後に、署名 PREPARE が $N-f$ $(N>3f)$ だけ集まったら faulty なレプリカがいくら票を分散させても、honest なレプリカはシークエンス番号 n とビュー v について、1 つの票しか投じないため、その合意しか実行可能なものはなくなる。

- <REQUEST, o,t,c>$_{\sigma_c}$
- <<PRE-PREPARE, $v,n,d(m)$>$_{\sigma_p}$, m>
- <PREPARE, $v,n,d(m),i$>$_{\sigma_i}$
- <COMMIT, $v,n,d(m),i$>$_{\sigma_i}$
- <REPLY, v,t,c,i,r>$_{\sigma_i}$

4.2 view changes 手続き

処理の条件が満たされないレプリカ i が反乱を起こすことができるという仕組みである。反乱を起こすことを決めたレプリカ i は VIEW-CHANGE と NEW-VIEW 以外の全てのメッセージの受信を止める。そして、<VIEW-CHANGE, $v+1, P, i$>$_{\sigma_{(i)}}$ をマルチキャストする。$v+1$ は ID を 1 インクリメントしたリーダーへの変更を申請することを意味する。P は Garbage Collection で stable な合意が取れているシークエンス番号 n 以降の、レプリカ i がローカルに合意している全てのメッセージ m の P-Certificate である P_m をすべて集めたものである。

$2f+1$ 個の VIEW-CHANGE を受け取った場合、リーダーを $p = v \mod N$ から $p' = v+1 \mod N$ に変更するという合意が取れたことになる。

新しいリーダー $v+1$ は、その後自身が VIEW-CHANGE で受信した P-Certificate の集まりで最も高いシークエンス番号までの合意が取れている一連のメッセージを調べる。そのシークエンス番号を h とする。そして、その高さになるまで次式によって j を $n+1$ から h までインクリメントして集合 $\mathcal{O}$ を新たに生成する (リーダーが異なるので署名が違うが、同じ状態を持つシークエンスの列が生成される)。

$$\mathcal{O} \leftarrow \mathcal{O} \cup < \text{PRE} - \text{PREPARE}, v+1, j, d(m_j) >_{\sigma_{p'}} \tag{5}$$

V の高さが、stable な合意が取れている n と等しい場合は、次式によって $\mathcal{O}$ を定義する。

$$\mathcal{O} \leftarrow \mathcal{O} \cup < \text{PRE} - \text{PREPARE}, v+1, n, d(\text{null}) >_{\sigma_{p'}} \tag{6}$$

そして、p' は <NEW-VIEW, $v+1, V, \mathcal{O}$>$_{\sigma_{p'}}$ をマルチキャストする。V は、VIEW-CHANGE メッセージの集合 (これも処理が重くなる要因) であり、これを加えてマルチキャストする必要がある。

<NEW-VIEW, $v+1, V, \mathcal{O}$>$_{\sigma_{p'}}$ を受信した各レプリカは、p' によって正式に署名されていること、正しく VIEW-CHANGE が署名されていること、$\mathcal{O}$ により更新される状態に一貫性があることを確認する (プライマリーによる計算と同じことを行う)。そして、$\mathcal{O}$ の全エントリーを p' が実行したものとしてログに保存し、$\mathcal{O}$ に含まれるメッセージに対応する PREPARE を再度署名により作成 ($v+1$ に変わったので必要) しマルチキャストする。そして、ブロードキャストされた結果取得した $2f+1$ 個の PREPAREs をログに書き加える。

$v+1$ は、<NEW-VIEW, $v+1, \mathcal{V}, \mathcal{O}$>$_{\sigma_{p=v+1}}$ を送信する。$\mathcal{V}$ は、全ての受信した $2f+1$ 以上の VIEW-CHANGE メッセージの集合である ($p = v+1$ が正式に多数決を取って、新たなプライマリーになったことを自身が理解していることを証明する)。$\mathcal{O}$ は、新たな PRE-PREPARE メッセージである。これをベースとして Normal Operation がまた実行されていく。

4.2.1 View Changes まとめ

署名や、Garbage Collection で行っている合意で決定されるシークエンス番号 n より先の処理を再度新たなリーダー p' で再現する必要があるため、処理コストが非常に重い。極力、リーダー交代が発生しない設計をする必要がある。

- <VIEW-CHANGE, $v+1, P, i$>$_{\sigma_i}$
- <NEW-VIEW, $v+1, \mathcal{V}, \mathcal{O}$>$_{\sigma_{p=v+1}}$

4.3 Garbage Collection

正しいレプリカが、リクエスト o に関するログのメッセージを、次のケースが成立するまで保持するという考えは、効率が悪い。o が正しいレプリカの多数決によって実行され、この事実が view change によって証明される。従って、安定した checkpoint という仕組みを用いて、ログ (log) の切り捨てを行う仕組みを導入する。各レプリカは、決められた間隔で (例えば k 個のリクエストが処理された後) に、<CHECKPOINT, n, d_s, i>$_{\sigma(i)}$ をマルチキャストする。ここで、n は最後に状態に反映される実行が行われたリクエストのシークエンス番号であり、d_s はシステムが取る状態のダイジェスト (ブロックチェーンでいう所のマークル木やパトリシアトライ木) である。i はレプリカのインデックスである。このようにすることで、CHECKPOINT のブロードキャストを受信した各レプリカは、$2f+1$ 個の合意が取れている最も高いシークエンス番号 n までのリクエストが処理されたものとして取り扱うことができる。

この $2f+1$ 個の集合を S と定義して、安定したチェックポイントの証明書 (stable checkpoint certificate) と呼ぶ。

4.3.1 Garbage Collection まとめ

定期的に、CHECKPOINT の更新を行うことで、保存すべき容量を減らす仕組みである。ブロックチェーンで言うと、ブロック高 n までは合意が取れているとして、全ての処理用メッセージを捨てる (truncate) することである。

- <CHECKPOINT, n, d_s, i>$_{\sigma(i)}$

参考文献

[1] 情報工学レクチャーシリーズ 「分散処理システム」

[2] 「Distributed Algorithms Practical Byzantine Fault Tolerance」 http://disi.unitn.it/~montreso/ds/slides17/10-pbft.pdf

[3] 「Practical Byzantine Fault Tolerance 」 http://pmg.lcs.mit.edu/papers/osdi99.pdf

[4] 「View change explains the need for COMMIT phase」 http://ug93tad.github.io/pbft/