分散合意アルゴリズムにおける リーダーを含む全ノード参加型の選挙による ビザンチン故障耐性とクラスター内の合意

岡田 大輝[1,a)]　串田 高幸[1]

概要：分散合意アルゴリズムでの Byzantine fault tolerance の達成は，各ノードとの密な通信によって実現される．PBFT(Practical Byzantine Fault Tolerance) アルゴリズムは実用的ではあるが，合意形成時の通信コストは $O(N^2)$ であり，ノードが増加するごとにその通信コストは大幅に増加する．そこで，本研究のアルゴリズムは選挙形式の合意形成を行う．クラスター内の各ノードを leader もしくは follower の 2 つの役割に分ける. そして，提案された値に対して leader を含めた全ノード参加型の選挙を行い，合意を形成する．これにより，各ノードが自身以外の全ノードと通信を行う必要がなくなり，通信コストを抑えることができるものである．

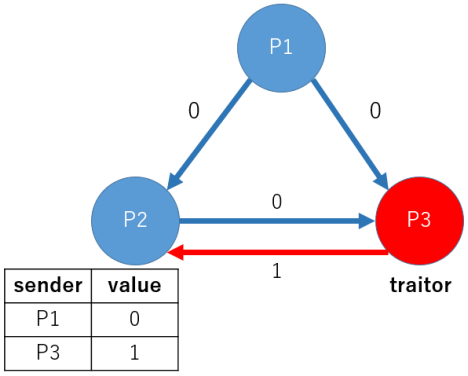
1. はじめに

1.1 背景

分散システムの構築では，システム全体でのデータの一貫性の確保や各マシンに同じ命令を実行するために，どのようにデータや命令を各マシン間で一つに合意し複製するかという点を設計，実装する必要がある．そして，一般的に分散合意アルゴリズムを実装することによりこの問題を解決する．分散合意アルゴリズムは，システムが一貫したグループとして機能するようサポートする．これにより，クラスターを構成している一部のノードに障害が発生してもシステムが動作できるようになり，大規模なシステムの構築において重要な役割を果たしている．[1].

分散合意アルゴリズムとは，分散アルゴリズムの一つに分類されるアルゴリズムである．これは，分散している各ノードで共通の決定に合意するためのアルゴリズムであり，主にレプリケーションの部分を担っている．このアルゴリズムの動作により，分散システムを構成しているクラスター内でのデータの一貫性や整合性を確保する．

そして，このアルゴリズムはフォールトトレランスの問題を解決するために使用される．つまり，データの一貫性と整合性が守られるようにされていると同時に，アルゴリズムそのものが分散システムや分散処理において想定される様々な故障や障害に対して耐性を持つように設計されている必要がある．



sender	value
P1	0
P3	1

図 1　Byzantine 将軍問題時の提案状況と P2 が把握する値

1.2 Byzantine 将軍問題

Byzantine 将軍問題とは，東ローマ帝国の将軍達の間で起きた作戦遂行に関する合意の話を元にした問題である．そして，Leslie Lamport らによって分散コンピューティング上の合意問題として定義された [2].

この問題は，図 1 のような状況を想定する．図 1 は，異なるノード間 (P1,P2,P3) で一つの提案に合意する場面を表している．P2 下の表は P2 に対して送られた値とその送信者を示している．まず，P1 が値 0 を提案する．P1 はその提案した値 0 を P2 と P3 にそれぞれ送信される．P2 と P3 は受け取った値を別のノードへ転送する．この時，P3 が合意を妨げる動作をする裏切り者だったと仮定する．そ

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[a)] C0117071

© 2020 Cloud and Distributed Systems Laboratory

の P3 によって P2 には，提案された値 0 とは違う値 1 が送られる．この場合，P2 は P1 から提案された値 0 と P3 から提案された値 1 の二つの値を持っている事になる．これにより，P2 からは 0 と 1 のどちらが正しい値かを判断することが出来ない．よって，この 3 つのノード間で 1 つの値に合意することが出来ない状況になっている．このような，一定数の裏切り者と呼ばれる存在がいる場合に正常な合意形成がおこなえるかを問う問題なのである

そしてこの問題は，分散システムのクラスター内でのデータの一貫性や耐故障性をサポートする分散合意アルゴリズムでも影響を及ぼす．故障ノードは，異なる値の送信，容認されていない任意のタイミングで任意のメッセージ送信といったアルゴリズムの設計では想定されない誤作動を起こして合意形成を妨げてくる可能性がある．分散合意アルゴリズムに BFT を実装する場合，1.2 節や本節で述べたような作為的な障害に晒された場合でもシステムが正常に動作できるように分散合意アルゴリズムも設計されなくてはならない．

この，Byzantine 将軍問題に帰着する障害は，Byzantine 故障や Byzantine 障害と呼ばれている．また，この障害に対する耐性を Byzantine fault tolerance(以下，BFT) と呼ぶ．これは，分散システム内に一定数の Byzantine 故障ノードが発生した場合でも，システムが正常に動作する性質を指している．

1.3 課題

Lamport らが定義したように，この合意問題は分散システムでも起こりうる問題である [2]．オリジナルの問題で定義されている裏切り者は，システム上でも故障ノードとして分散システムの信頼性やデータの一貫性，整合性を侵害する．そのために，正常の動作とは異なる様々な動作を行う可能性がある．そのような任意とも言える障害の発生が想定される環境でシステムの信頼性を守る事は難しい問題である．

研究 [3] は，既存の研究は実用性に欠けると述べ，実用的な Byzantine fault tolerance として PBFT(Practical Byzantine Fault Tolerance) アルゴリズムを提案した．PBFT では各ノード間が全ノードに対して通信を行って合意形成を行う物である．これは，実用的なレベルに達した初めてのアルゴリズムとして，今日の合意アルゴリズムの BFT 実装の基本となっている．しかし，全ノードが他ノードに対して密に，通信を行う必要があるため通信コストが高く，合意形成を行うグループの大規模化が難しい．PBFT が合意形成にかかる通信コストは N をノード数とした場合，$O(N^2)$ となる．これは，ノードが増加するごとにその中で合意を行うための通信量が大幅に増加していくことを意味している．

本研究では，BFT においての高通信コストによる大規模化が困難である点を課題に設定する．2 章では，分散合意アルゴリズムの関連研究を提示する．3 章では，本研究で示すアルゴリズムの具体的な説明を行う．4 章では，実装の環境や方法について記述し，それで行った実験の評価を示す．5 章では提案するアルゴリズムに対して議論されるべき点と，合意アルゴリズムにおいて考慮，解決すべき点の提示と考察を行う．

2. 関連研究

BFT プロトコルに関する研究については, 低コスト化を目的としたものがいくつか存在する.

1 つに，クォーラムベースの BFT プロトコルがある [4]. これは, 各ノードが client と直接通信を行う手法である．これにより通信の低コスト化を実現し，かつ優れたパフォーマンスを維持した状態で BFT を達成できるものであるとした．しかし，この手法は client に複製処理に関わる権限を持たせることになる. これは，client を交えてしまうがために固有の新たな問題を引き起こす可能性がある (ユーザーはクラスターに侵入して改ざんする必要はなく，自身の client 側のデータを改ざんして，合意形成を妨害できる)．複製と合意の処理はクラスター内のノードで完結されており，client はマシンへの命令と合意後の結果の取得のみができる状態であることが好ましいと考える．

また，クォーラムベースのアプローチとレプリカベースのアプローチを組み合わせ，これら 2 つの問題を克服するハイブリッド型の BFT レプリケーションプロトコルを提案した研究がある [5]．これは，競合が少ない状況であれば，3f+1 ノード (f は故障ノード) で高効率で動作することが出来る．しかし，競合が多い状況では，BFT の方がスループットが優れているという結論を出した．本研究のアルゴリズムは BFT の改良と位置付けることができ，競合が多い状況においても安定した合意形成が実現できる．

別の研究では，合意形成の通信を 2 ステップにする事により，一般的な BFT プロトコルよりも低い通信コストを実現し処理の高速化する手法が提案されている [6]．これにより，1 回の要求ごとに全体で 3 ステップで合意を実現できるものである．この手法は高速な合意を実現しているが，必要なノード数は 5f+1(f は故障ノード) に増加している．本研究のアルゴリズムは，最小 4 ノード (3f+1) でありシステム設計者にノード数の決定にににおいて研究 [6] よりも多くの選択肢を提供できる．

3. 提案

本研究の合意アルゴリズムは，単一の leader を擁立して行うリーダーベースの合意アルゴリズムのである．システムを構成している各マシンに Consensus Module() が 1 つずつ配置され，それぞれに leader もしくは follower の役割が与えられる．合意形成は，leader から命令の複製を受け

© 2020 Cloud and Distributed Systems Laboratory

取った follower がその命令の提案を行い，leader を含めた全ノード参加型の合意選挙によって行われる．

client からの命令は，その分散システム構成しているマシンに対する操作であり，ソフトウェアの基本機能である CRUD の内の Create(作成)，Update(更新)，Delete(削除) である．システムに対する Read(読み取り) の命令はデータの変更ではなく単純な参照であるため，合意アルゴリズムの動作は必要ない．

3.1 役割

本研究のアルゴリズムはリーダベースの分散合意アルゴリズムである．従って，合意形成の際には，必ず各ノードに以下の役割を付与する．そして，全てのノードは役割を兼任する事はない (leader であり follower という事は起こらない)．

- **leader**
  client との通信，各 follower への命令の複製と適用の指示を行う．
- **follower**
  leader から受け取った複製された client からの要求の正当性チェックのために選挙を発生させる．合意後，要求を適用する．

leader は，クラスターを形成するノードから擁立される．合意形成時には，必ず 1 つのノードが leader の役割を担う．leader は前述の通り，アルゴリズムの動作を主導する．client からの命令を受け取るのは leader であり，受け取った命令の複製，処理結果を client に返すのも leader である．leader は leader 選挙によって選出が行われる．一度選出された leader は特定の条件を満たすまで leader であり続ける．leader は，follower に対して以下の 3 種類の通信を行う事が出来る．

- **send**
  client からの命令を，follower へ複製して送信するための通信である．これには，命令の複製を転送を示す send の文字列と leader 自身の識別子と client からの命令内容を送信する
- **vote**
  合意形成において follower から命令の提案がされた時，その提案が自身の持つ命令と一致した際に，提案した follower に送る通信である．これには投票することを示す vote の文字列と自身の識別子を送信する．違う場合は投票を拒否する (返信を送らない)
- **apply**
  合意形成の終了後に命令をマシンへ適用することを指示するために行う通信である．これには，適用を指示する apply の文字列と自身の識別子を送信する．

follower は，クラスター内の leader 以外のノードがこの役割を担う．leader を通して送られた命令について 3.3 章で示す手法で合意を行う．そして，合意に成功した場合には client からの要求をマシンに適用する．follower は，他ノードに対して以下の 3 種類の通信を行う事が出来る．

- **proposal**
  leader から命令が複製された際に follower が合意選挙を開始するために行う通信である．これには，提案を示す proposal の文字列と自身の識別子，そして，提案する命令内容を送信する．
- **vote**
  合意形成において他 follower から命令の提案がされた時，その提案が自身の持つ命令と一致した際に，提案した follower に送る通信である．これには投票することを示す vote の文字列と自身の識別子を送信する．違う場合は投票を拒否する (返信を送らない)
- **win**
  proposal を行った follower が，3.2 節で示した勝利条件を満たした際に leader へ送る通信である．これには，勝利したことを示す win の文字列と自身の識別子，そして選挙に勝利した際の命令を送信する．

3.2 合意選挙の勝利条件

合意選挙は，クラスター内の全ノード間で，同じ命令を保持しているかを確認する選挙である．leader から命令の複製を受け取った follower は，提案する形で自分が受け取った命令を全ノードに送る．提案が送られたノードは自身の持つ命令と照合し，同じであれば投票する．そして，本節に示す勝利条件を満たす投票が得られた場合には，その命令は大多数に正しく複製された命令という事になる．勝利条件は以下の 2 つに設定する．2 つの勝利条件は両方同時に達成されなければならない．そのため，どちらか片方のみの達成はそのノードが敗北，もしくは選挙そのものの失敗である．

- 自身を含めた過半数のノードから投票を得る
- 過半数の投票の中に **leader** からの投票がある

一つ目は，選挙という方式をとるうえでは当然の勝利条件であると考える．過半数から投票を得られない命令はクラスター内に複製された命令とは違う内容の可能性が高い．少数派の命令が選挙に勝利できるようにすると，大多数へ複製が出来た正常な命令が上書きされてしまう事態が発生する．正常な leader から正常に命令を複製されたのであれば，3.4 節で示したように 3 分の 1 以下の Byzantine 故障ノードが出現した場合でも，残りの 3 分の 2 以上のノードで正常な命令に投票が可能である．この際，Byzantine 故障ノードが値を他 follower より早く提案し，投票を求めたとしても，leader や他 follower から投票を得る事は出来ない．3.3 節で示した投票する条件によってこれが実現されている．

二つ目は，client からの命令は leader からのみクラスター

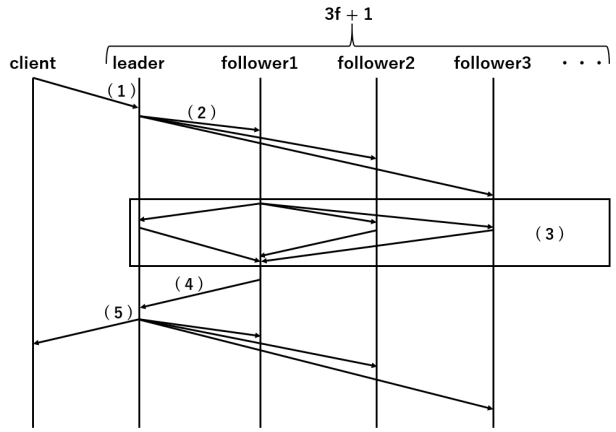


図 2　合意形成の通信フロー

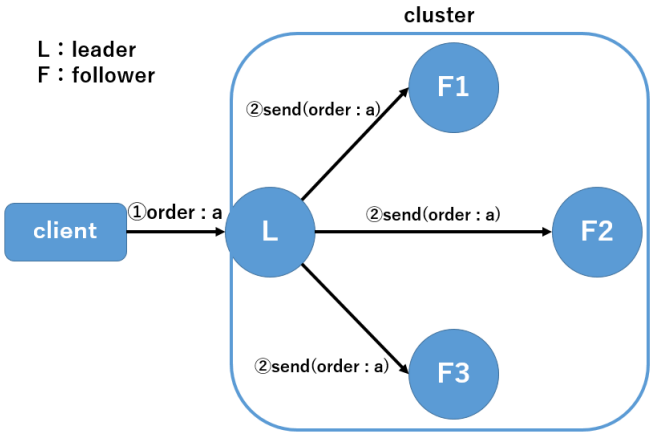


図 3　合意形成 (1)(2) の動き

内のノードへ流れてくるという事を考慮した勝利条件である．follower が client からの命令を把握する方法は leader からその複製を送信してもらうことのみである．follower は自身が持つ命令に正当性がある場合はその命令は leader と同一であるはずである．つまり，その合意選挙において follower が leader とは異なる命令を把握していた場合，それは，アルゴリズムの正常な動作で生み出される事のない命令である可能性が高い．

3.3　合意形成

本研究のアルゴリズムは leader を含めた全ノード参加型の選挙によって合意を行う．概要として，leader が client からの命令を受け取り前述した形式での合意選挙を行う．選挙が正常に終了した場合，その命令を適用する．選挙の勝利者が現れなかった場合は選挙は失敗として，その命令を破棄し，次の命令の合意選挙を行う．本節では，この合意形成の詳細な処理の流れと制約を示す．図 2 は，順序立てを行い示した合意形成においての通信の流れである．

図 2 の (1) で，leader は client から命令を受け取る．なお，client からの命令を受け取りは leader のみの特権である．なので，follower に client からの通信があった場合は，

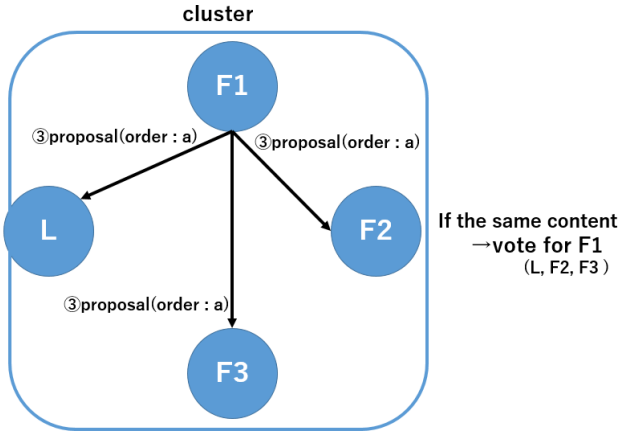


図 4　合意形成 (3) の処理

leader へリダイレクトするような仕組みを導入して，必ず leader を通して各 follower へ命令が送られるようにする必要がある．この段階は，図 3 中の①order:a である．図中では client からの命令である order:a を leader(L) が受け取っている．

図 2 の (2) では，leader は全ての follower へ client から送られた命令を転送する．この際，follower が leader へ通信を行えるように，leader ノードの識別子も通信に含める必要がある．3.5 章では，この場面において leader と follower 間でメッセージ異常が発生した場合を想定する．この段階は，図 3 中の②send(order:a) である．図中では leader(L) は client からの受け取った order:a をクラスター内の follower(F1，F2，F3) に send(order:a) として転送している．

図 2 の (3) では，leader から通信を受け取った命令について合意選挙を開始する．これは，命令を受け取った follower から開始される全ノード参加型の選挙である．follower は，受け取った命令内容を送り主である leader を含めたクラスター内の全ノードに自身に送られた命令の内容を提案する．そして，自分が受信した命令の正当性についての投票を促す．この際，投票を促すノードも自身に投票する．他 follower から投票を促された follower は，自身が leader から受け取った命令内容と同じ内容であれば投票する (vote)．もし，内容に相違があった場合は，投票を拒否する (proposal に対して返信しない)．各ノードは，1 回の合意選挙で投票できるノード数は 1 である．leader は提案を行った follower の命令内容と自身が client から受け取った命令内容が同一であった場合にその follower へ投票する．そうでない場合は投票を拒否する．この段階は，図 4 中の③proposal(order:a) である．図中では，follower ノードの 1 つである F1 が leader から受け取った order:a を提案して各ノードに投票を促している．他ノード (L，F2，F3) は，自身が持つ命令と同一であった場合 F1 の提案に投票する．

この提案・投票の場面においては，複数 follower が同時に提案を全ノードに送信し，分割投票が発生する場合があ

© 2020 Cloud and Distributed Systems Laboratory

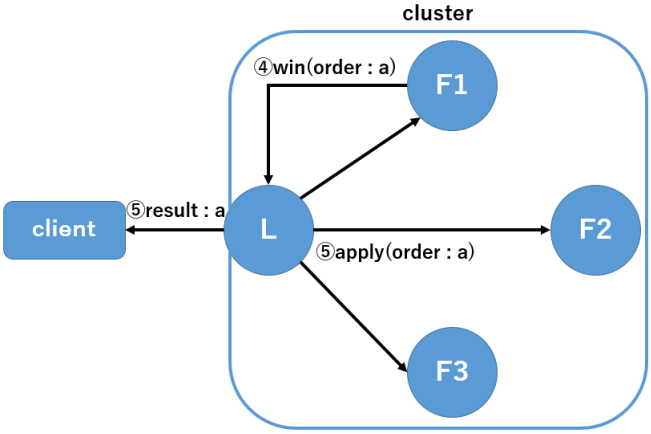


図 5　合意形成 (4)(5) の処理

る．分割投票が発生した場合，正常に命令が複製されたにもかかわらず，選挙が失敗してしまう可能性がある．これを防ぐためには，Raft のランダムアプローチを適用して，各 follower が提案をするタイミングをランダムにすることにより，複数ノードによる同時提案を防ぐという手法がある [1]．

図 2 の (4) では，(3) で行われた合意選挙において勝利条件を満たした follower が出現した．合意選挙に勝利した follower は図??leader へ勝利条件を満たしたことを勝利した際の命令内容と自身の識別子を通信する．3.2 章で示した，leader からの投票を含む過半数ノードからの投票を得るという合意選挙の勝利条件により，選挙に勝利した follower とその時の leader は同一の命令を持っているようになっており，leader と違う命令を持った follower は選挙に勝利できないようになっている．もし，合意選挙において，勝利者が現れなかった場合，この選挙は失敗したと扱う．合意に失敗した命令は破棄され，次の命令の合意選挙に移行する．この段階は，図 5 中の④win(order:a) である．図中では F1 が order:a を提案し，無事選挙に勝利した．勝利した F1 は order:a で選挙に勝利したことを leader(L) に報告する．

ここで起こりうる問題として，Byzantine 故障ノードの動作により，一つは正常に勝利した follower，もう一つは勝利したことを偽装する follower の 2 つ以上の follower から勝利報告が届くことである．しかし，前述した通り，各ノードの投票権は 1 回のみであり，leader もこのルールに従う．よって，勝利の偽装は leader からの投票を得ていなければならず，leader から投票を得られるのならそれは正しく命令を複製している正常なノードである．leader からの投票を偽装する可能性も考えられるが，leader が自分が投票したノードの識別子を保持していれば勝利の偽装は避けられる．

図 2 の (5) の段階に入った時点で，命令の複製と合意選挙については正常に終了したと言える．leader は合意選挙に勝利した follower が出現したことを把握した後，再度全ての follower へ合意選挙に勝利した命令内容と適用を指示を許可するメッセージを送り，follower はそのメッセージを受信した際に，送られてきた命令の適応を開始する．この段階は，図 5 中の⑤apply(order:a) である．図中では，F1 が order:a で選挙に勝利したことを把握した leader(L) は，全 follower(F1,F2,F3) へ選挙に勝利した命令内容である order:a を適用するよう指示している．

ここで送られた命令内容が，これがこの合意選挙の最終結果となる．ここで共に送られてきた命令を follower はマシンへ適応できる．これにより，仮に合意選挙中に違った命令が保持されていた場合でも正常な follower はここで送られてきた命令を適用する事により．他ノードと同じマシンの適用状態になる．

3.4　全体ノード数と故障ノードの許容数

本研究のアルゴリズムは最小 4 ノードで動作することを想定している．クラスター内のノード数 (N) の決定は，どのくらいの個数の故障ノード (f) を許容するかを考慮して決定されるべきである．研究 [2] は，正常なプロセス 2f+1 個存在しなければならず，さらに，Byzantine 故障以外の不作為な障害を起こすノードの発生も想定したノード数にしなければならない事を示した．これらの事から，クラスターのノード数を BFT を考慮して決定する場合は数式 1 の計算式になる事が示されている [7][8]．

$$N = 3f + 1 \tag{1}$$

以上の事から，BFT を考慮して決定するノード数の最小値は 4 であり，最低限 4 ノード以上でクラスターを構成しないと BFT を達成できないという事になる．本研究のアルゴリズムもこの研究に準拠し，最小 4 ノードで動作するとした．

3.5　Byzantine 故障の発生

本節では，実際に 3 分の 1 以下のノードが Byzantine 故

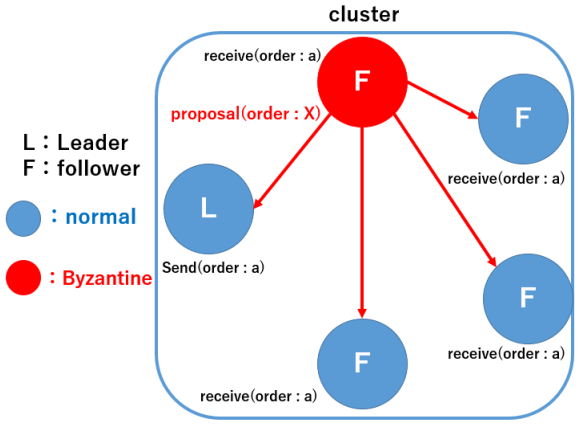


図 6　follower に Byzantine 故障ノードが出現した場合

障ノードであった場合に本研究のアルゴリズムがどのような状態になるかを示す．

最初に，図 6 のような状況を考える．これは，follower の一つが Byzantine 故障ノードとして出現している状況である．この Byzantine 故障ノードは，合意を妨げるために，正しい leader から送られた命令とは全く違う命令を提案して投票するように他 follower に要求するような動作をすることもある．そして，提案タイミングにランダム化アプローチを採用していた場合は，Byzantine 故障ノードの提案がこの合意選挙において一番最初の提案になることもある．しかし，3.3 節で述べた投票条件により，これは誰からも投票される事無く，この提案は失敗に終わる．そしてこれ以降の合意選挙においても提案に失敗し続けるため，結果的に全体の合意形成に影響を及ぼすことはない．

Byzantine 故障ノードが発生する状況での合意形成において，一番発生してはいけないのは，Byzantine 故障ノードが leader になってしまうことである．leader に Byzantine 故障ノードが選出された場合，その状況で正しい合意形成を行う事は非常に困難である．client と直接通信する権利や follower に命令を複製する権利は leader のみが持つ．そのため，Byzantine 故障ノードに強権を与える事になってしまうからである．leader となった Byzantine 故障ノードの動作や命令は，follower には不正なものとして捉える事は難しいだろう．follower の基本動作は leader からの命令に従うのみだからである．

4. 実装と評価

4.1 実装

実装を図 7 に図示する．本研究のアルゴリズムを Consensus Module というソフトウェアとして実装する．Consensus Module は図 7 に示している 3 つのコンポーネントで構成されている．

controller は，各 ConsensusModule を制御する役割を担っている．各 Module とシステムを利用する client は controller を通して通信を行う．そして，controller が自身の現在の状態 (leader か follower) や識別子 (ID) を管理する．さらに，現在の leader を把握するために leader の識別子を保有する．これは，leader としてハートビートを送ってきたノードの識別子である．そして，controller は自身の役割と他ノードからの通信に含まれる通信の目的を示しす文字列を参照して，follower function もしくは leader function を呼び出して処理を行う．そしてその結果を他ノード，もしくは client へ送信する．

leader function は，自身の役割が leader の際に controller から呼び出されるコンポーネントである．leader function には，以下の leader が持つべき機能と行うべき動作が定義されている．

- ハートビート

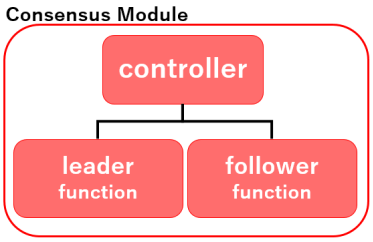


図 7　実装構成図

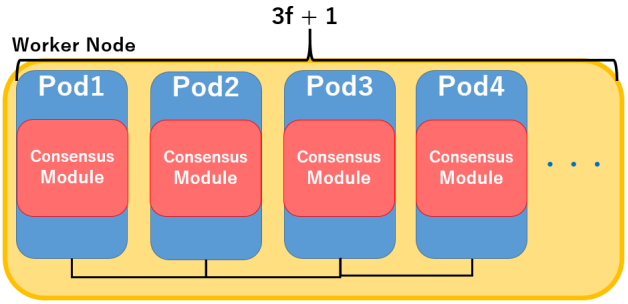


図 8　実験環境構成図

表 1　VM のスペック

	内容
OS	Ubuntu18.04
CPU	2 コア
メモリ	2048MB
ストレージ	20GB

- client との通信機能
- follower への通信機能 (send, vote, apply)

ハートビートは，leader が全 follower に定期的に送る通信である．これは，leader が正常に稼働している事を follower に伝えるための通信であり，follower はこれについては合意を行わず，返信も送らない．このハートビートが定期的に送られることにより，follower は現在の leader が少なくとも稼働しているのかを知ることが出来る．このハートビートが送られないという事は，現在の leader が故障して動作が出来なくなっている可能性が高い．client との通信機能は，client からの要求の受信と命令適用後の結果を client に返す機能である．client との通信は leader のみが行えるものであり，この機能は leader function のみがもつ．follower との通信機能は 3.1 節で leader が follower に対して実行できる 3 種類 (send, vote, apply) の通信である

follower function は，自身の役割が follower の際に呼び出されるコンポーネントである．follower function には，3.1 節で示した合意選挙時に他ノードに行う通信 (proposal, vote，win) が定義されている．

4.2 実験環境

実験は，図 8 で示した環境で行う．Kubernetes 環境を構築し，Master Node と Worker Node をそれぞれ 1 ノード用意する．それぞれのノードは VM であり，1 に示すス

© 2020 Cloud and Distributed Systems Laboratory

ペックで作成されている．各 Pod は単一コンテナで構成されており，1Pod が 1 ノードとして動作する．Pod には Consensus Module が 1 つずつ配置されている．そして，Pod 間で通信を行って合意形成を行う．実験では，開始時点で 1 つのノードを leader として擁立させる．leader ノードは client から命令を貰う代わりに，そのノード内にクラスター内に適応する命令が記述されているテキストファイル (Order.txt) を保有している．Order.txt から 1 行ずつ命令を読み出して follower へ命令を転送する．実験では，以下の 3 種類の命令を扱う．

- **create: filename**
  filename で指定された名前でテキストファイルを生成する．
- **append: [filename, value]**
  filename のテキストファイルに value を書き込む．
- **delete: filename**
  filename のテキストファイルを削除する．

follower は，3.3 節で示した合意選挙を発生させる．そして，合意選挙にて勝利者が出現した際には勝利した命令を自身のノードで実行する．

follower の中に Byzantine 故障ノードを意図的に出現させる．個数は，3.4 節で示したノード数を参考に決定する．Byzantine 故障ノードとなった follower は leader が持つ Order.txt とは別に命令が記述されたテキストファイルを持つ (ByzantineOrder.txt). 書かれている命令の種類は Order.txt と同じ 3 種類だが，内容が異なるものである．Byzantine 故障ノードは leader から send で命令が送られた際，その命令ではなく ByzantineOrder.txt で記述された命令をクラスター内に提案するように動作させる．

4.3 評価

評価では，Byzantine 故障ノードの出現した状況でも大多数に命令の複製と適用が出来るかという BFT の点と，合意形成を行う際の通信コストの点で評価を行う．

まず、BFT の点を評価するために，各 follower に自分が適用した命令を記録させるようにする．これは，Order.txt と同じ形式になるように記録する．その各 follower が持つ記録ファイルと leader が持つ Order.txt とを比較し，どの程度差分が発生するかを評価する．差分がなければ，その follower は leader からの命令を全て複製、適用できたことになる．

通信コストの点は，1 命令において通信を行った回数をカウントするように各ノードの機能に一部変更を加える．そして leader の send から最後の leader の apply までを 1 セットとして各セットの通信回数を集計し，その平均を算出する．算出された値が 1.3 で示した BFT の概ねの通信コストである $O(N^2)$ とどの程度の差が生じるかで評価する．

5. 議論

リーダーベースのアルゴリズムにおいては，リーダー選挙の仕組みの導入は不可欠である，これは，本研究のアルゴリズムにおいても議論されるべき点である．もし BFT を考慮しないのであれば，Raft のように，選挙タイムアウトを設けて，立候補のタイミングがプロセス間で衝突することを防ぎ，リーダー選挙を leader になれるノードの条件付けを行った多数決で決定すれば良いだろう [1]．BFT の考慮する場合，leader になろうとする follower は，Byzantine 故障ノードかもしれない．そして，現 leader が正常に動作している事も関係なく，自身の情報を更新して leader 選挙を開始するかもしれない．考えられる対策として，各ノード共通の選挙許可タイムアウトを導入し現 leader が動作中はこのタイムアウトが作動しないようにする (leader のハートビートが届くたびにタイムアウトをリセットする)．そして，この選挙許可タイムアウトがタイムアウトしない限り投票を拒否する．これにより，Byzantine 故障ノードが一人でに選挙を始めても，すぐにそのノードが leader になる事は無くなるだろう．

しかし，Byzantine 故障ノードが leader にならないように完全に対策を施すことは困難である (leader 選出時は正常でも動作中に Byzantine 故障ノードと化す可能性もある)．なので，Byzantine 故障ノードが leader になってしまう可能性を予め想定し，leader に Byzantine 故障の疑いがある際には follower の側から新しい leader を選出するように動作を追加する方が現実的であると考える．例として，リーダーがフォロワーに送るハートビートに何かしらの情報を入れ，不整合であった場合に，すぐリーダー選挙を始める．合意選挙が失敗したり，制限時間を設けてそれ以内に選挙が終わらなかった場合には異常正常関わらず新しくリーダー選挙を始めるといった対策が挙げられる．また，研究 [9] では，合意形成の貢献に応じて変動するスコアを導入して，その数値に応じてノードの除外をするといった手法を提案している．leader をどのように制御するかはリーダーベースの合意アルゴリズムを設計する上での問題の 1 つである．いくつかの関連研究では，リーダーを擁立せずに Byzantine 状況下でも合意出来る可能性についての研究や leader を擁立しない合意アルゴリズムが提案されている [7][10][11].

3.3 節で示した合意選挙は，勝利条件を満たした follower が出現するまで続けられることになる．選挙回数に制限を加えない場合最悪のケースとして follower の数だけ提案と全ノードへの通信が行われることになる．正しく命令が follower へ転送できたにも関わらず，合意できないケースも考える事は不可能ではない (ネットワークの遅延により投票が届かないなど)．しかし，正しく転送出来ているので

© 2020 Cloud and Distributed Systems Laboratory

あれば，全 follower が提案するよりも早く合意選挙の勝利条件を満たした follower が出現すると考える方が現実的であり，合意できないほどの故障，または故障ノード数に侵されている場合は，どんな，合意アルゴリズムでさえ正常に動作することは不可能である．最後に提案した follower が勝利条件を満たす可能性もあるが，そこまで合意できなかった命令の正当性があるかは疑問である．よって，一回の合意選挙において実行できる提案回数に制限も設けるなどして，システムのボトルネックになる可能性を防ぐ必要がある．

6. おわりに

本研究では，BFT が実装された比較的簡略化されたリーダーベースの分散合意アルゴリズムを提案した．本研究のアルゴリズムは，値や命令の合意の際に，leader を含めた全ノード参加型の合意選挙の手法をとっている．そして，これを Kubernetes 環境で実装と実験を行う事を示した．

参考文献

[1] Ongaro, D. and Ousterhout, J.: In Search of an Understandable Consensus Algorithm, *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, Berkeley, CA, USA, USENIX Association, pp. 305–320 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2643634.2643666⟩ (2014).

[2] Lamport, L., Shostak, R. and Pease, M.: *The Byzantine Generals Problem*, p. 203–226 (online), available from ⟨https://doi.org/10.1145/3335772.3335936⟩, Association for Computing Machinery (2019).

[3] Castro, M. and Loskov, B.: Practical Byzantine Fault Tolerance (1999).

[4] Abd-El-Malek, M., Ganger, G. R., Goodson, G. R., Reiter, M. K. and Wylie, J. J.: Fault-Scalable Byzantine Fault-Tolerant Services, *SIGOPS Oper. Syst. Rev.*, Vol. 39, No. 5, p. 59–74 (online), DOI: 10.1145/1095809.1095817 (2005).

[5] Cowling, J., Myers, D., Liskov, B., Rodrigues, R. and Shrira, L.: HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance, pp. 177–190 (2006).

[6] Martin, J. . and Alvisi, L.: Fast Byzantine Consensus, *IEEE Transactions on Dependable and Secure Computing*, Vol. 3, No. 3, pp. 202–215 (2006).

[7] Lim, J., Suh, T., Gil, J. and Yu, H.: Scalable and leaderless Byzantine consensus in cloud computing environments, *Information Systems Frontiers*, Vol. 16, No. 1, pp. 19–34 (2014).

[8] Aguilera, M. K., Delporte-Gallet, C., Fauconnier, H. and Toueg, S.: Consensus with Byzantine Failures and Little System Synchrony, *International Conference on Dependable Systems and Networks (DSN'06)*, pp. 147–155 (2006).

[9] Jiang, Y. and Lian, Z.: High Performance and Scalable Byzantine Fault Tolerance, *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, pp. 1195–1202 (2019).

[10] Borran, F. and Schiper, A.: A leader-free byzantine consensus algorithm, *International Conference on Distributed Computing and Networking*, Springer, pp. 67–78 (2010).

[11] Crain, T., Gramoli, V., Larrea, M. and Raynal, M.: Dbft: Efficient leaderless byzantine consensus and its application to blockchains, *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, IEEE, pp. 1–8 (2018).

© 2020 Cloud and Distributed Systems Laboratory