社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コンセンサスアルゴリズムに対するラウンドモデルに基づいた簡易的なテスト・検証手法の提案

土屋　達弘†

† 大阪大学　〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †t-tutiya@ist.osaka-u.ac.jp

あらまし　コンセンサスアルゴリズムは，代表的な分散アルゴリズムのクラスであり，分散システム上で信頼性の高いサービスを実現する上で中核となる．このようなサービスには，プロセスレプリケーションやブロックチェーン等が含まれる．本研究では，まず，現在研究を行っているコンセンサスアルゴリズムを簡易的にテスト，検証することで，含まれるバグを効率よく検出する方法を説明する．この方法では，ラウンドモデルというシステムでのイベントの生起する実時間を捨象したモデルを前提に，故障も含めたアルゴリズムの動作を通常の逐次プログラムとして表現する．その結果，プログラムに対する一般のテスト，検証手法を適用して，アルゴリズムのデバッグが可能となる．本稿では更に，既存のアルゴリズムを例に，実際にバグの検出と特定が可能なことを示す．
キーワード　コンセンサスアルゴリズム，検証，テスト，ブロックチェーン

Tatsuhiro TSUCHIYA†

† Osaka University　1–5 Yamadaoka, Suita-shi, 565–0871 Japan
E-mail: †t-tutiya@ist.osaka-u.ac.jp

1. はじめに

分散システムは，ネットワーク上の複数のプロセスが通信を通じて協調することでサービスを提供するシステムである．分散システム上でディペンダブルなサービスを実現するためには，サービスに関与するプロセス間で情報を正しく共有し，プロセスの障害や通信の遅延に対しても，システムとしてサービスの提供を継続する必要がある．コンセンサスアルゴリズムは異なるプロセスが同じ決定を行うことを実現するアルゴリズムであり，分散システムにおける情報共有で中心的な役割を果たす．

一方，分散アルゴリズムの設計は逐次アルゴリズムに比べて非常に難しい．これは，プロセスや通信が非同期的に実行されることに加え，故障のシナリオも無数にあり，これらすべての可能性のある実行に対して，アルゴリズムが正しく動作することが求められるためである．コンセンサスアルゴリズム関しては，権威ある国際会議で発表された Zyzzyva [1] や FaB [2] においても誤りが存在し，アルゴリズムを実装して運用した段階で発見されている [3].

そこで本稿では，現在研究を行っているアルゴリズムの誤りを検出するための方法を説明する．この方法では，コンセンサスアルゴリズの設計の初期の段階で，簡易的に効率よく誤りを検出することを目的としている．本方法では，アルゴリズムの正しさを示すのではなく，少ないプロセス数を想定し，かつ，初期状態からの少ない状態遷移のみを考慮して誤り検出を行う．この方針は以下の 2 つの考えに基づく．

カットオフバウンド　文献 [4] では，コンセンサスアルゴリズムの正しさの検証を任意のプロセス数ついてモデル検査という手法で実現している．モデル検査は，通常，有限状態機械としてアルゴリズムの動作を表現する必要があるが，この手法では個々のコンセンサスアルゴリズムに，ある一定のプロセス数まで正しさを示せば，それ以上のプロセス数についても正しさが保証されることを示しており，検証が必要なプロセス数の上限をカットオフバウンドと呼んでいる．このような，カットオフバウンドが通常存在することから，アルゴリズムに誤りがある場合，少数のプロセス数であっても顕在化することが期待できる．

小スコープ仮説　経験的に，アルゴリズムの誤りは短い実行においても顕在化することが多いことが知られている．これは，小スコープ仮説と呼ばれ，Zyzzyva と FaB の誤りについても当てはまる．また，分散アルゴリズムに留まらず，一般的なソフトウェアについても小スコープ仮説が当てはまることが多く，たとえば，ソフトウェアのモデル化と検証を行うツールである Alloy Analyzer では，小スコープ仮説に基づいて，仕様に合致する小さい実例を対象として自動検証を実施する方針を採用し

ている．

更に，手間をかけずにテストを実施するために，本手法では分散アルゴリズムを，一般的な逐次プログラムとして表現するアプローチを採用する．分散アルゴリズムを実際に分散環境で実行できるようにプログラミングすることは，それ自体非常に負荷の高い作業である．分散アルゴリズムを表現可能なモデル化言語，例えば，TLA++ [], Promela [5], Ivy を用いて記述する場合でも，記述するためにその言語を理解する必要があるだけでなく，アルゴリズムの抽象レベルに合致した表現が得られるとも限らない．

このような問題を避けるため，本手法では，アルゴリズムを，その動作をシミュレーションする通常の逐次プログラムとして表現する．そして，そのプログラムに通常のテスト，検証手法を適用することで，バグの検出を行う．ここで逐次プログラムとして分散アルゴリズムを記述するために，分散アルゴリズムの動作をラウンドモデルによって表現する．ラウンドモデルでは，分散システム上で発生する事象，たとえば，メッセージの送受信や故障等について，それらが発生した実時間を捨象し，システム全体が非同期的なラウンドを繰り返すものとして，アルゴリズムの動作を把握する．これにより，システムの動作をラウンド毎の状態遷移と見なすことが可能となり，逐次的に動作を表現することが可能となる．

本論文の構成は以下の通りである．節 2. では，コンセンサスアルゴリズムと想定する分散システムのモデルについて説明する．節**??**では，C 言語の逐次プログラムによって，コンセンサスアルゴリズムの動作を表現する方法を示す．節 4. では，得られた C 言語のプログラムに対し，有界モデル検査を適用する方法を述べる．最後に節 5. でまとめと今後の研究について述べる．

2. システムモデルとコンセンサス

2.1 ラウンドモデル

分散システムは，n 個のプロセス $p_1, p_2, \ldots, p_n$ からなるものとする．システムの状態は非同期ラウンドを繰り返すことで変化する．ラウンドの番号は 1 から始まるものとする．

各ラウンドにおいて，各プロセスはメッセージの送信，受信，そして状態遷移を，この順序で実行する．ラウンドで送信されたメッセージが受信されなかった場合，そのメッセージは喪失する．つまり，メッセージの送受信は各ラウンド内で閉じて実行される．この性質を，communication closedness と呼ぶ [6].

システムにおける故障は，メッセージの喪失によってモデル化する．本研究では，任意のメッセージが喪失する可能性があるものとする．したがって，あるプロセスから送信したメッセージがすべて喪失する可能性もある．このようなプロセスはクラッシュしているプロセスを表現している．したがって，ここで仮定している故障モデルは，従来のオミッション故障モデル（メッセージの送受信の失敗と，プロセスのクラッシュ）に対応する．

2.2 コンセンサス

上記のシステムモデルにおけるコンセンサス問題を以下のように定義する．初期状態において，各プロセス p_i は提案値 v_i をもつ．また，プロセスは値 x を決定するプリミティブ decide(x) を実行できる．このとき以下の条件を実現する問題をコンセンサス問題とする．

停止性：プロセスはいずれ決定を行う．

合意性：異なるプロセスが決定した値は同一である．

妥当性：決定された値はいずれかの提案値である．

プロセスやメッセージに同期性をまったく仮定しない場合，コンセンサス問題は，故障を 1 プロセスのクラッシュ故障に限定した場合でも決定性のアルゴリズムで解くことが不可能なことが知られている．そこで，多くのコンセンサスアルゴリズムは，同期性に関する何らかの仮定を導入した上で，完全な非同期性の下では停止性は保証しないが，合意性と妥当性については保証するように設計されている．また，妥当性に関しては，提案値が書き換えられたり，アルゴリズムの実行の途中で提案値とは異なる値を作り出さない限りは，容易に満たすことが可能である．本研究では，同期性に関する仮定を置かず，また，ビザンチン故障など提案値を変化させる状況は取り扱わないので，合意性に注目し，アルゴリズムの正しさを合意性の観点で判定する．

3. コンセンサスアルゴリズムの例

本節では本稿で検証を行うコンセンサスアルゴリズムについて説明する．ここでは Challon-Bost と Schiper による FastPaxos の変種であるアルゴリズムを取り上げる．Paxos は Lamport による代表的なコンセンサスアルゴリズムであり，Fast Paxos は最適化された Paxos の 1 つである．Fast Paxos では，ほとんどのプロセスが同じ提案値を有する場合，1 ラウンド目で決定を行うことができる．その条件が成り立たない場合は，通常の Paxos を利用してより遅いラウンドで決定を行う．アルゴリズム 1 は文献のアルゴリズムを表しており，ラウンドモデルを前提にしている．

例として，単純なコンセンサスアルゴリズである One-third-rule アルゴリズム [7] を取り上げる (Algorithm 1). このアルゴリズムでは，プロセスの状態は決定する値の候補を表している．各ラウンドでプロセスは自分の状態をブロードキャストしたのち，ブロードキャストされたメッセージを受信した結果，3 分の 2 より多くのプロセスが同じ値を提案値の候補と考えているのが分かれば，その値で決定を行う．そうでない場合は，最も多くのプロセスが候補として考えている値を，新たな候補値として選択する．もし，そのような値が複数ある場合は，それらにおける最小値を選択する．

プログラムでは，ラウンド 1 から，記号定数 `MAX_ROUND` で表される既定の回数のラウンドの実行をシミュレーションする．プロセスの個数は記号定数 `NUM_PROC` で指定している．ラウンドの実行の前に，提案値を各プロセスにランダムに設定する．そののち for ループによりラウンドの実行を表現する．

メッセージの送信は，変数 `chnl` へ送信メッセージを書き込むことで表現している．

メッセージは喪失，遅延により宛先に届かない場合がある．

Algorithm 1: One-third-rule アルゴリズム

1: **initialization**
2: x_i := 提案値 v_i
3: **Round** r
4: send x_i to all processes
5: **if** 受信したメッセージ数 $> 2n/3$ **then**
6: x_i := 受信した値の最小値のうち，最も受信数の多いもの
7: **if** 受信した値で $2n/3$ を超える値ものが一致 **then**
8: この一致する値で決定

```
int firstdecision = 0;
void decide(int proc, int value)
{
        decision[proc] = value;
        if (firstdecision == 0)
        {
                firstdecision = value;
        }
        else
        {
                assert(firstdecision == value);
        }
}
```

図 1　合意性のチェック

これは，chnl をランダムに-1 に更新することで表現する．

その結果，chnl[i][j] が-1 であれば，プロセス p_i はプロセス j からのメッセージを受信しなかったことになる．-1 でない場合，chnl[i][j] はプロセス p_j からの送信メッセージの内容を表す．プロセス p_i は chnl[i][j] の内容に基づいて状態を変更し，条件が成り立てば決定を行う．

状態更新の際，最も多く受信した値の中で最小のものを選ぶ必要がある．検証のための専用の言語を用いた場合，このような自然言語で単純に表現できる動作でも，簡潔に正しく表現することは簡単ではない．一方，C 言語のような通常のプログラミング言語であれば，実行して動作を確認しながら容易に表現できる．

合意性が違反されていないことはアサーションによって表現する．図 1 にアサーションを記述した箇所を示す．この関数 decide(proc, value) は，プロセスが決定を行う際にそのプロセス番号と決定値を引数として実行される．決定値がそれまでに決定された値と同じことをアサーションで表しており，もし異なる場合はアサーション違反となる．

4. アルゴリズムの検証

C 言語のプログラムとして表現したアルゴリズムは，提案値やメッセージの喪失，遅延を表す変数の値を乱数で設定することで，ランダムシミュレーションが可能となる．

ランダムシミュレーションを通して意図した通りアルゴリズムが表現できていると確信できた場合，他の体系的なテスト，検証手法を適用してアルゴリズム自体の誤りの検出を行う．逐次的なプログラムを網羅的にテスト，検証する手段としては，モデル検査や記号実行などが存在する．本稿では，これらの手法の中でモデル検査の一種である有界モデル検査を適用する．有界モデル検査は，その名が示す通り，初期状態から一定の長さの動作を検証の範囲とし，検証の問題を論理式の充足可能性問題へ還元する．その結果得られる充足可能性問題は SAT や SMT ソルバによって解くことが可能となる．ここでは有界モデル検査ツールとして CBMC を用いる [8]．CBMC では LTL で表現できる性質を検証可能であるが，合意性の検証はアサーション違反の検出によって行う．また，CBMC では，変数に対し乱数で値を設定する代わりに記号的に扱うことを指定するディレクティブがあり，このような変数が指定された範囲の任意の値を取るものとして検証することができる．

```
int _randint(int first, int last)
{
    int tmp;
    #if defined(CBMC)
    __CPROVER_assume((first <= tmp) && (tmp <= last));
    #else
    tmp = rand() % (last + 1 - first) + first;
    #endif
    return tmp;
}
```

図 2　CBMC の適用

C 言語のプログラムをこのような式に変換して有界モデル検査を実行するツールとして，CBMC が知られている [8]．本研究ではこのツールを用いてアサーション違反の可能性を検出する．

コンセンサスでは提案値が入力値に相当し，これを変数で記号的に扱うことで任意の提案値に対する動作を検証する．さらに，送信メッセージが受信されるかされないかの選択も記号的に扱い，任意のメッセージの受信パターン，任意の故障パターンを検証で考慮する．入力値についてもメッセージ受信の成否も，元のプログラムでは乱数を用いて決定しており，これは関数 _randInt(first, last) によって乱数を得ることで行っている．この関数は，引数 first と last の間の整数をランダムに選択して返す．

CBMC を適用する場合，図 2 のようにディレクティブを用いてこの関数を前処理の段階で置換する．記号定数 CBMC が定義されている場合，乱数を用いる代わりに，戻り値を表す変数を記号的に扱うことを指示した文が前処理によって選ばれる．具体的には，tmp は first <= tmp && tmp <= last を満たす任意の整数値を持つ変数として記号的に扱われる．

同様な手法を用いて，別のコンセンサスアルゴリズムである LastVoting [7] についても，C プログラムを作成して CBMC を適用した．用いた計算機は CPU として Xeon E3-1245 (3.5GHz)，メモリ 8GB を有する Windows 10 PC である．

表 1 と表 2 に CBMC の実行に必要だった時間をアルゴリズム毎にまとめる．LastVoting は 4 ラウンドで一つのまとまった処理を行うため，ラウンド数は 4 の倍数としている．また，1 台の故障プロセスに耐えるために One-third-rule アルゴリズ

表 1　One-third-rule アルゴリズムに対する CBMC 実行時間 (単位: 秒)

	$n=4$	$n=5$	$n=6$
$r=1$	0.5	0.7	1.1s
$r=2$	1.8	4.2	8.9s
$r=3$	3.9	9.9	26.0s
$r=4$	6.2	15.8	69.8s

表 2　LastVoting アルゴリズムに対する CBMC 実行時間 (単位:秒)

	$n=3$	$n=4$	$n=5$
$r=4$	2.3	5.4	22.9
$r=8$	4.7	13.9	91.0
$r=12$	9.8	42.9	TO

ムでは最低 4 個，LastVoting では最低 3 個プロセスが必要なため，n の値はそれぞれ 4, 3 以上とした．タイムアウト時間は 5 分に設定した．この時間内に検証が終了しなかった場合は，表において TO と記している．いずれのアルゴリズムについても，指定したラウンド，プロセス数の範囲ではアサーションの違反，すなわち，合意性の違反は検出されなかった．

5. おわりに

本研究では，分散アルゴリズムを C 言語の逐次プログラムで表現し，そのプログラムに対してテストや検証を実行することで，アルゴリズム中のバグを検出する手法を提案した．現時点ではアルゴリズム中のバグの検証には至っていないため，実際にバグ検出が可能であることを示すことも今後の課題である．また，研究ではメッセージ喪失で表現できる故障である，オミッション故障とクラッシュ故障を対象とした．しかし，ブロックチェーン用のコンセンサスアルゴリズムとしては，ビザンチン故障への耐性をもつコンセンサスアルゴリズムの利用が望まれる場合も多く，今後は，このクラスのアルゴリズムもテストできるように提案手法を拡張する．

謝　辞

本研究は JSPS 科研費 18KT0098 の助成を受けたものである．

文　献

[1] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E.L. Wong, “Zyzzyva: Speculative byzantine fault tolerance,” ACM Trans. Comput. Syst., vol.27, no.4, pp.7:1–7:39, 2009. https://doi.org/10.1145/1658357.1658358

[2] J. Martin and L. Alvisi, “Fast byzantine consensus,” 2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan, Proceedings, pp.402–411, IEEE Computer Society, 2005. https://doi.org/10.1109/DSN.2005.48

[3] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, and J. Martin, “Revisiting fast practical byzantine fault tolerance,” CoRR, vol.abs/1712.01367, pp.00–00, 2017.

[4] O. Maric, C. Sprenger, and D.A. Basin, “Cutoff bounds for consensus algorithms,” Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II, eds. by R. Majumdar and V. Kuncak, vol.10427, pp.217–237, Lecture Notes in Computer Science, Springer, 2017. https://doi.org/10.1007/978-3-319-63390-9_12

[5] G.J. Holzmann, The SPIN Model Checker, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2004.

[6] A. Damian, C. Dragoi, A. Militaru, and J. Widder, “Communication-closed asynchronous protocols,” Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II, eds. by I. Dillig and S. Tasiran, vol.11562, pp.344–363, Lecture Notes in Computer Science, Springer, 2019. https://doi.org/10.1007/978-3-030-25543-5_20

[7] B. Charron-Bost and A. Schiper, “The Heard-Of Model: Computing in Distributed Systems with Benign Failures,” Distributed Computing, vol.22, no.1, pp.49–71, April 2009.

[8] E. Clarke, D. Kroening, and F. Lerda, “A tool for checking ANSI-C programs,” Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004), eds. by K. Jensen and A. Podelski, vol.2988, pp.168–176, Lecture Notes in Computer Science, Springer, 2004.