ISSN 2186-5647

モデル検査器を用いたコンセンサスアルゴリズムの検証

阪大・情報科学（院）　○中野 伸哉
阪大・情報科学　小島 英春・土屋 達弘

1. はじめに

分散システムにおけるコンセンサスとは，全てのプロセスで同じ決定を行う問題であり，プロセス多重化に利用される．コンセンサス問題を解くアルゴリズムとしては，Paxosアルゴリズム[1,2]や，Raftアルゴリズム[3]が知られている．Raftアルゴリズムは，Paxosアルゴリズムと比較して，理解と実装が容易なアルゴリズムとして注目されている．本稿では，モデル検査器SPIN[4]を用いて，Raftアルゴリズムの保証されている項目について，いくつか検証を行う．

2. Raftアルゴリズム

Raftアルゴリズムは，コンセンサス問題を解く代表的なPaxosアルゴリズムを意識して設計されている．Paxosアルゴリズムは，理論が複雑で，難解であり，実装も困難である．この問題を解決するため，Raftアルゴリズムは，Paxosアルゴリズムと異なる構造を持つ．具体的には，Raftアルゴリズムは二つの機能から成り立つ．Leader ElectionとLog Replicationの二つである．Leader Electionでは，プロセスの中からLeaderを選出する．Log Replicationではログの複製を，Leaderを中心にして行い，全てのプロセスで同じデータが記憶されていることを保証する．この動作を行うことによってコンセンサスを得ることが出来る．これらの機能はtermという変数を用いて，制御されている．

図1では，Raftの動作の大まかな流れを，termを用いて示す．時間軸をtermによって分割する．termの最初は必ずLeader Electionから始まる．このとき，Leaderが選挙によって選ばれたときのみ，Log Replicationに移行する．選ばれなかったときはtermを一つ増やし，新たなtermでもう一度Leader Electionが行われる．

Raftではビザンチン故障[5]が起こらないという条件で，プロセスのクラッシュと回復，ネットワークの遅延，分断，もしくは，パケットのロスや重複などが起きても，絶対に間違った結果を返却しないという安全性(Safety)が保証されている．また，過半数以上のプロセスが正常に動作していれば，故障しているプロセスは動作の進行を妨げないという性質も保証されている．本稿では，

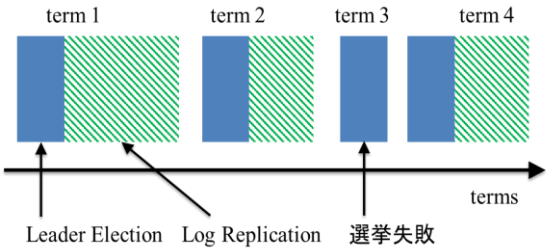

図 1. termによる時間軸の分割

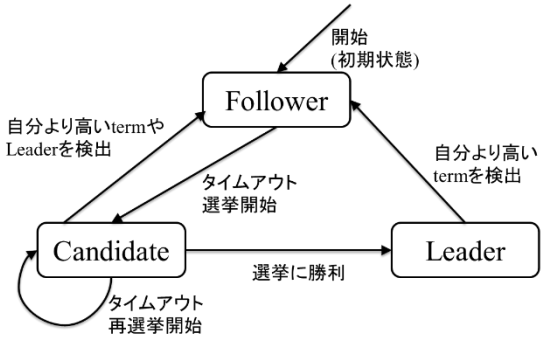

図 2. Leader Election 時における，各サーバの状態遷移図

RaftアルゴリズムのLeader Electionの部分についての検証を行う．

3. Leader Election

Raft アルゴリズムにおける Leader Election の役割は，複数のプロセスからリーダーを選出するというものである．これにより選出されたリーダーを中心に，次の Log Replication の動作が行われる．まず，Leader Election において，それぞれのプロセスは，以下に示すいずれかの状態になる．

・Follower：立候補者に対して，投票を行う
・Candidate：立候補者
・Leader：選挙によって選ばれたリーダー

各プロセスの状態遷移図を図 2 に示す．

全てのプロセスは初期状態であるFollowerからスタートする．次に，プロセスがタイムアウトを起こし，Candidateに遷移する．Candidateは全てのサーバに投票を促す．Candidate自身は自分に投票

Verification of a Consensus Algorithm Using Model Checking

Shinya NAKANO, Hideharu KOJIMA and Tatsuhiro TSUCHIYA

し，Followerは最初にメッセージを受信したCandidateに対して，Leaderになれるかどうかをtermによって判断し，投票する．投票は1termにつき1回までとなっている．過半数以上の投票によって選ばれたCandidateはLeaderとなり，全てのサーバにメッセージを送信する．このとき過半数以上の投票が得られなかった場合は，termを一つ進めて再投票を行う．

4. 検証項目

検査方法は，満たすべき性質を線形時相論理式(LTL)で表現し，この性質の否定を表すneverオートマトンとモデルとの非同期積をとることによって合成を行い，モデル検査を行う．まず，保証されるべき安全性の性質である「一つのtermにおいて，常に，最大で一人のLeaderしか存在しない」[3]という性質を，term i でLeaderが最大一人という命題をp_iとして，LTLで表現すると，(1)式になる．

$$[\,](p_0 \&\& p_1 \ldots \&\& p_k) \qquad -(1)$$

ここでkは検証対象とするtermの最大値とする．さらに，「いつか必ずLeaderが選出される．」という活性(Liveness)の性質を，Leaderが選出されているという命題をqとして，LTLで表現すると，(2)式になる．

$$<> q \qquad -(2)$$

これらの否定を表すオートマトンを，システムの動作を表すオートマトンと合成することで，モデル検査を行う．

5. 実験結果

検証対象とするRaftのモデルは全ノード数3とし，termは無限に増加するため，有限値として扱い検証を行った．まず，(1)式の安全性について検証した結果を表1に示す．このときのterm値は最大で3とした．全探索では使用できるメモリの限界量を超えてしまったため，検査することが出来なかった．よって，Super Traceモードで実行したところ，探索が終了し，性質が満たされた．すなわち，Leader Electionにおける安全性は保証されることが示された．次に，(2)式の活性について検証した結果を表2に示す．このときのterm値は最大で10とした．状態を200個程度探索したときに受理サイクルを発見し，性質を満たさない判例が見つかったことがわかる．よって，活性は満たされないことが示された．

表 1. 安全性の検証

Mode	Super Trace	Exhaustive
States	42840837	>769000000
Transitions	66967815	>1130000000
Memory[MB]	417.978	>127937.202
Time[s]	7340	>2500
LTL	TRUE	UNKNOWN

表 2. 活性の検証

Mode	Super Trace	Exhaustive
States	166	141
Transitions	237	190
Memory[MB]	397.665	407.518
Time[s]	0.036	0.003
LTL	FALSE	FALSE

6. まとめ

termを有限値として扱ったが，有限の範囲において安全性は保証され，活性は満たされないことが示された．また，今後の課題として，アルゴリズム全体の検証[6,7]や，状態爆発を引き起こすもとになるtermの値を境界値として，境界値分析が出来るようにモデル化，およびそのモデルの妥当性についての検討などが考えられる．

参考文献

[1] LAMPORT, Leslie. The part-time parliament. ACM Transactions on Computer Systems (TOCS), 1998, 16.2: 133-169.

[2] LAMPORT, Leslie. Paxos made simple. ACM Sigact News, 2001, 32.4: 18-25.

[3] ONGARO, Diego; OUSTERHOUT, John. In search of an understandable consensus algorithm. In: Proc. USENIX Annual Technical Conference, 2014, p. 305-320.

[4] HOLZMANN, Gerard J. The SPIN model checker: Primer and reference manual. Reading: Addison-Wesley, 2004.

[5] LAMPORT, Leslie; SHOSTAK, Robert; PEASE, Marshall. The Byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS), 1982, 4.3: 382-401.

[6] NOGUCHI, Tatsuya; TSUCHIYA, Tatsuhiro; KIKUNO, Tohru. Safety Verification of Asynchronous Consensus Algorithms with Model Checking. In: Proc. Pacific Rim International Symposium on Dependable Computing, 2012, p. 80-88.

[7] TSUCHIYA, Tatsuhiro; SCHIPER, André. Verification of consensus algorithms using satisfiability solving. Distributed Computing, 2011, 23(5-6): 341-358.