部分スナップショットアルゴリズムの効率的な並行実行の実現

渡部連太郎[1] 金鎔煥[2] 大下福仁[3] 角川裕次[1] 増澤利光[1]
1 大阪大学大学院 情報科学研究科
2 名古屋工業大学 情報工学専攻
3 奈良先端科学技術大学院大学 情報科学研究科

概要 計算機（以降ノードとする）同士が協調的に動作する分散システムにおいて，一部のノードの故障や離脱がシステム全体へ影響を与える可能性がある．そのため，いかに分散システムに耐故障性を持たせるかが重要である．システムに耐故障性を持たせる手法の 1 つにチェックポイントロールバック手法がある．これは，システムに故障が発生した際，事前に記録済みの安定状態 (チェックポイント) へシステムを復元 (ロールバック) する手法である．分散システムにこの手法を用いるため，スナップショットアルゴリズムを利用してシステム全体の状態を保存する必要がある．スナップショットとは，分散システム全体の状態を局所状態の集合として記録したものであり，スナップショットアルゴリズムはスナップショットを矛盾なく作成する手法を指す．本稿では CSS アルゴリズム [Kim et al, IEICE E97, No.1, 2014] の概要を紹介し，CSS アルゴリズムのメッセージ複雑度を改善するアルゴリズムを提案する．

1 はじめに

ネットワークを介して複数の計算機が相互に通信しながら協調して動作するシステムを分散システムと言う．分散システムでは一部のノードの故障がシステム全体に悪影響を及ぼすことがある．近年の分散システムでは大規模化が進んでおり，それに伴いシステムのどこかで故障が発生する確率も高くなる．このような背景のもとで，システムの耐故障性が重要になってきている．

システムに耐故障性を持たせる手法の 1 つにチェックポイントロールバックがある．これはシステムが正常なうちにチェックポイントとして安定状態を記録しておき，故障発生時には最新のチェックポイントへシステムを復元（ロールバック）するものである．分散システムに対してこの手法を適応する際，チェックポイントとして使用する分散システム全体の状態を保存する必要がある．矛盾のない分散システム全体の状態を記録するアルゴリズムをスナップショットアルゴリズムと言う [1]．

Chandy と Lamport によるスナップショットアルゴリズム (CL アルゴリズム)[1] は，手続きの簡潔さからその代表的なものである．しかし CL アルゴリズムでは，メッセージを送信可能なノード全てに送信して伝播させるため，適応するシステムの規模が大きくなるとメッセージ複雑度が増大し，実用が難しいという問題があった．

CL アルゴリズムを基に考案された Sub-Snapshot(SSS) アルゴリズム [2] はこの問題を解決している．これはスナップショットアルゴリズムを最初に開始するノード（initiator）と一部のノード集合 (スナップショットグループ) のみでスナップショットをとるアルゴリズムである．これにより同時にスナップショットをとるノード数を削減し，ノードの参加や離脱への対応にも成功した．しかし SSS アルゴリズムは並行的に実行されることを仮定していない．この仮定は，多くのノードがスナップショットを取得する必要のある大規模なシステムにおいては現実的ではない．仮に異なるノードを initiator としてアルゴリズムを並行実行すると，スナップショットグループが重複 (衝突) した場合に生成されるスナップショットに矛盾が発生する可能性がある．

SSS アルゴリズムを基に処理の並行実行を可能にした Concurrent Sub-Snapshot(CSS) アルゴリズム [3] は，スナップショットグループの衝突が発生した場合，スナップショットグループ同士を統合し，1 つのスナップショットグループとして機能するよう改良された．しかしスナップショットグループを統合するためには多くのメッセージ通信を行う必要がある．本稿では，各アルゴリズムの概要を述べた後に CSS アルゴリズムのメッセージ複雑度を改善するアルゴリズムを提案する．

2 既存研究

2.1 CL アルゴリズム

CL アルゴリズム [1] はマーカと呼ばれる特別なメッセージを用いてシンプルな動作を行うだけで，矛盾のないスナップショットを効率よく作成するアルゴリズムであり，以降で紹介するアルゴリズムの基盤となっている．initiator は最初に局所状態を記録し，各隣接ノードへマーカを送信する．initiator 以外のノードは，このマーカを受信することでアルゴリズムが開始する．初めてマーカを受信したノードは局所状態を記録し，各隣接ノードへマーカを送信する．initiator を含めた全てのノードは，全ての隣接ノードからマーカを受信するとアルゴリズムを終了する．

CL アルゴリズムを実行するためには，各ノードは隣接ノードを予め知っていなければならないという制約がある．実際の分散システムは動的な変化 (ノードの参加/離脱などによるネットワーク構造の変化) が頻繁に起こるが，制約により CL アルゴリズムはこれに対応していない．また，各ノードは送信できる全てのノードに対してマーカを送信するため，ネットワークが大規模化するとメッセージ数が膨大になる．

[1] 分散システムにおいて，あるノードが送信していないメッセージを他のノードが受信しているとき，このメッセージを orphan message と呼ぶ．分散システム全体で orphan message が存在しない記録の状態を“矛盾がない”という．

2.2 SSS アルゴリズム

SSS アルゴリズム [2] は，CL アルゴリズムを基に動的なノードの参加/離脱への対応とマーカ送信回数の削減に成功している．これはアルゴリズム内で initiator と動的な因果関係を持つノード集合 (スナップショットグループ) を決定し，その要素ノードのみとスナップショットをとることによる．

このアルゴリズムでは依存関係の概念を導入する．各ノードは，送信/受信，生成/被生成イベントが発生した際，関与したノード ID を動的に依存関係集合 (DS) へ追加していく．アルゴリズムを開始すると，initiator はまず CL アルゴリズムと同様に局所状態を保存し，ID を添付したマーカを，保持している DS に含まれる全てのノードに送信する．初めてマーカを受信したノードは局所状態を保存し，DS の各要素にマーカを送信，さらに initiator に DS を送信する．initiator はスナップショットグループを決定するため，受信した DS の和集合とその送信ノード ID の集合を管理する．この 2 つの集合が一致したとき，それが決定したスナップショットグループである．その後 initiator は，各ノードがマーカを受け取らなければならない受信元ノード集合を計算し，それぞれのノードへ送信してアルゴリズムを終了する．initiator からその集合を受信したノードは，集合に属する全てのノードからマーカを受信するとアルゴリズムを終了する．

注意点として，このアルゴリズムでは並行的に 2 つ以上の initiator がアルゴリズムを開始させることはないと仮定している．仮に異なる複数のノードを initiator として複数のアルゴリズムが同時実行された場合，スナップショットグループの衝突が発生すると正しい動作を保証できなくなる．

2.3 CSS アルゴリズム

SSS アルゴリズムを並行実行できるよう改良したものが CSS アルゴリズム [3] である．CSS アルゴリズムでは，並行実行した二つのアルゴリズム間でスナップショットグループの衝突が発生した際，矛盾がないよう衝突したスナップショットグループを統合する．衝突したスナップショットグループを管理していた initiator のうち一方が統合後のグループを管理する．この initiator を main initiator，他方を sub initiator と呼ぶ．

以下で統合の流れを説明する．あるグループからのマーカを既に受信しているノードが新たなグループからのマーカを受信したとき，ノードは衝突を検知し，属するグループの initiator へ衝突を報告する．報告を受け取った initiator は，もう一方のグループの initiator へ統合要請メッセージを送る．そしてメッセージを受け取った相手 initiator が，どちらが main initiator になるか決定し返信する．最後に sub initiator となる initiator は，main initiator へ保持していた依存関係集合やノード ID などの情報を送信する．

このアルゴリズムの注意点として，衝突回数が増加するとメッセージの送信回数が膨大になる点が挙げられる．衝突が増加し多くのスナップショットグループが統合されていると，衝突報告/統合要請メッセージを目的ノードへ届けるために必要な中継ノードが増加する．よってこのとき新たな衝突が発生するとメッセージの複雑度が大きくなると考えられる．

3 提案アルゴリズム

本章では，CSS アルゴリズムに比べ，アルゴリズム実行中の衝突回数が大きくなった場合のメッセージ送信回数を削減するアルゴリズムを提案する．CSS アルゴリズムでは衝突発生時，スナップショットグループを矛盾なく統合することにより並行実行を可能にしているが，衝突や統合の前後において，関連ノードは最初にマーカを受信した際に保存した局所状態の削除や，新たな局所状態の保存といった変更を行わない．つまり“統合の有無に関わらず，最終的に取得するスナップショットの要素になるのは最初にマーカを受信した際に保存した局所状態である”と言える．そこで本アルゴリズムでは衝突発生時にスナップショットグループの統合処理は行わず，衝突相手のグループの記憶のみを行う．以降で，アルゴリズムを構成する 2 つのステップを説明し，その後ステップ 2 のメッセージ複雑度を改善する手法を紹介する．

3.1 ステップ 1

ステップ 1 では，各 initiator が CSS アルゴリズムと同様にマーカを伝播させることでスナップショットグループを決定する．このとき，衝突が発生した場合は CSS アルゴリズムのように統合を行わず，衝突相手の initiator の ID の記憶のみ行う．また，本来衝突が起きたノード間の依存関係によりマーカを送るべきであったノードは全て衝突相手のスナップショットグループで局所状態が保存されるので，以降そのアルゴリズム中ではスナップショットグループに含めない．

3.2 ステップ 2

ステップ 2 では，ステップ 1 で記憶しておいた衝突関係のあるグループの initiator 同士でメッセージを交換することで，互いにステップ 1 が終了していることを確認した後，アルゴリズムを終了する．ステップ 2 を開始した initiator はまず，ステップ 1 で記憶していた，自身が直接衝突したグループの各 initiator へ，自身の ID を載せた終了確認メッセージを送信する．initiator はこのメッセージの broadcast，convergecast により全連結 initiator の終了を確認する．すなわちこの時送信した

メッセージが各 initiator から返ってきた時，衝突関係により構成される initiator 間のネットワークに含まれる全ての initiator は正常にスナップショットグループを決定している．自身のものではない ID が載ったメッセージを受信した場合，その送信元以外の隣接 initiator へメッセージを伝播し，それら全てから同メッセージが返ってくると，最初の送信元へメッセージを返す．ネットワーク内のどれか 1 つでも initiator が全 initiator の終了を確認すると，その initiator は最後の終了許可メッセージを broadcast する．これを受信した initiator は，メッセージを伝播するとアルゴリズムを終了する．

3.3 ステップ 2 の改良

前述のステップ 2 では，全ての initiator が broadcast と convergecast を試みるが，実際はどれか 1 つの initiator だけが行えばよいため，無駄なメッセージが多く発生していた．この点を改良し，メッセージ複雑度を小さくしたアルゴリズムを以降で紹介する．

新たなステップ 2 では，initiator 群の中から代表 initiator（ID が最も小さい initiator）を 1 つ選出する．この代表選出プロセスのために，各 initiator は自身の知る最小 ID を保存する変数 *MinID* を持つ（初期値は自身の ID）．ステップ 2 を開始した initiator はまず，ステップ 1 で記録した衝突先 initiator へ，自身の ID id_i を載せた終了確認メッセージを送信する．このメッセージを受け取った initiator は，メッセージに添付された ID id_i と自身の *MinID* を比較し，$id_i > MinID$ ならば，メッセージの送信元へ，最小の ID を知らせるメッセージに *MinID* を載せて送信する．$id_i < MinID$ ならば，*MinID* の ID を保持する initiator へ，最小の ID が更新されたことを知らせるメッセージに id_i を載せて送信する．更にこのメッセージを受信した initiator も，メッセージに添付された ID と自身の *MinID* を比較し，代表となる initiator を決定，最小の ID が更新されれば，それをメッセージで伝える．各 initator はこのような処理を繰り返す．このプロセスにより，代表 initiator は自身が代表であることを知っている状態が保持される．このようにして代表となる initiator が決定すると，衝突関係にある全ての initiator へアルゴリズム終了を許可するメッセージを伝播させ，受信した initiator の管理するスナップショットグループはアルゴリズムを終了する．

このアルゴリズムは衝突毎の統合を行わないため，CSS アルゴリズムに比べて，衝突回数の増加によるメッセージ送信回数の増加が抑えられることが期待できる．

4 おわりに

本稿では CSS アルゴリズムをメッセージ送信回数の観点から見たその問題点を述べ，それを改善するアルゴリズムの概要を示した．アルゴリズムのステップ 2 は従来のものから改良がなされた．アルゴリズムの正当性は証明済みであるため，今後はシミュレーション実験を行い，それにより実際にメッセージ送信回数が削減されていることを示そうと考える．

参考文献

[1] K.Chandy and L.Lamport, “ Distributed snap-shots : Determining global states of distributed systems,”ACM Trans. Computer Systems, Vol.3, No.1, pp.63-75, 1985.

[2] S.Moriya and T.Araragi,“Dynamic Snapshot Algorithm and Partial Rollback Algorithm for Internet Agents, ” Proceeding of the 15th International Symposium on Distributed Computing, Brief Announcements, pp.23-28, 2001.

[3] Y.Kim, T.Araragi, J.Nakamura and T.Masuzawa, “ A Concurrent Partial Snapshot Algorithm for Large-scale and Dynamic Distributed Systems, ”IEICE Transactions on Information and Systems, Vol.E97-D, No.1, Jan.2014.