論理時計を用いた通信効率の良い Checkpoint-Rollback アルゴリズムの考察

斎田誠宏 [1] 金鎔煥 [1] 中村 純哉 [2] 片山喜章 [1]
Masahiro Saida Yonghwan Kim Junya Nakamura Yoshiaki Katayama

名古屋工業大学大学院工学研究科情報工学専攻 [1]
Nagoya Institute of Technology, Graduate School of Computer Science and Engineering
豊橋技術科学大学 情報メディア基盤センター [2]
Toyohashi University of Technology, Information and Media Center

概 要

本論文では, 複数の計算機（以下、ノード）が通信リンクで繋がっている分散システムにおいて、故障耐性を持たせる代表的な手法の一つである Checkpoint-Rollback アルゴリズムについて考える。Checkpoint-Rollback とは、分散システムの正常な状況を定期的に安定した記憶装置に保存し（Checkpoint）、システムの一部に故障が発生した場合、保存した過去の正常な状況を用いて復元（Rollback）させる手法である。分散システムにおいて、システムの全体状況を矛盾なく正しく把握することは、分散システム分野において重要な課題の一つであり、効率良く全体状況を把握するための様々な研究が紹介されている。

本研究では、効率の良い Checkpoint-Rollback 手法の実現のため、論理時計を用いて通信を必要としない Checkpoint 手法のアイデアを紹介する。さらに、システムに不具合が発生した場合、各ノードが保存した状態を矛盾なく復元する手法も紹介する。本研究の戦略は、分散システムの運用において、Checkpoint の頻度が Rollback の頻度より高いという観点から、Checkpoint の負荷を減らすことで故障耐性を実現するための総合的な負荷の削減を目標としている。

1 研究目的

分散システムはノードと呼ばれる計算機で構成され, リンク (通信路) で相互に接続されている. 各ノードは, これらのリンクを介してメッセージを交換することにより, 他のノードと通信する. 少数のノードでの障害によりシステム全体が故障するため, 膨大な数の計算機で構成される分散システムは障害を起こしやすい. したがって, 分散システムの故障耐性を考えることは重要な問題である. チェックポイント・ロールバック [2] は, 故障耐性を持たせるための一般的で代表的な手法である. システム全体の状態を定期的に不揮発性ストレージに記録 (チェックポイントの取得) し, システムに障害が発生すると, システムは記録された状態を使用して自身を復元 (ロールバック) する. クライアントサーバ型分散システムの場合, 任意のタイミングでサーバがチェックポイントを取得すれば, システムの故障時にクライアントに対してサーバから状態に一貫性があるチェックポイントにロールバックにすることが出来る. しかし, サーバが故障した際にシステム全体が障害となる単一障害点 (SPOF) になってしまう. また, サーバーで集中管理を行うので, クライアントからの通信が集中してしまうと動作不良を起こしてしまい, パフォーマンスが低下する. そこで近年, システムの大規模化に伴い増大するサーバの負荷軽減と処理の高速化のため, 従来のクライアントノードがサーバを介さない相互通信により処理を進める P2P 型の分散システムが構成されることも多い. だが, ノード間で相互に通信を行う分散システムでは, 各ノードが独立にチェックポイントの取得をするだけではドミノ効果 [1] が起こり, 状態の一貫性を維持したままチェックポイント状態へロールバックさせることができない. そこで, 分散システムの構成を記録するためにスナップショットアルゴリズムと呼ばれる特定のアルゴリズムが必要になる. ただし, 多くのスナップショットアルゴリズムでは, システム内のすべてのノード間の調整が必要になる. したがって, システムが大きい場合, スナップショットアルゴリズムを頻繁に実行するには, システム全体のパフォーマンスが低下するほどの通信コストが必要になる. 分散システムにおいて物理時間を利用して全ノードを同期させてチェックポイントを取得することは難しい. そこで, 物理時計を利用せずに, プロセス速度やメッセージ遅延にかかわらずプロセス間の動作の因果関係を捉える手法が考えられている. これによりノード間でメッセージ送受信の一貫性が維持されている無矛盾な状況を構成することができる. しかし, 動作の遅いノードが故障した際に動作の速いノードが古いチェックポイントに戻らなければならなくなる可能性がある. 本論文では, サーバを持たずノード間で相互に通信を行う分散システム論理時計を用いたチェックポイントの取得とロールバックを効率的に行う手法について考察する.

2 関連研究

東と守屋 [3] は付加情報量が定数サイズの場合は他プロセスの時計値を知ることが困難なため, 全プロセスが同時にロールバックするために利用できるチェックポイントが取得されるまで時間がかかる点に問題があるとして, ハイブリッド P2P システムのように一般のプロセスとサーバから成るシステムを利用し, サーバはチェック

ポイントの取得とロールバックの管理を行うだけで, プロセスでメッセージのやり取りや内部状態の変化を実行する. プロセスがチェックポイントを取得する前にサーバを介してプロセス間のランポートタイムスタンプ [4] の値を調整することや, 物理時計ではないタイマーにより一定時間 σ を検知しプロセスが σ 経過した後に動作する前にチェックポイントを取得するようにして, 各プロセスがロールバックに利用するのに有効なチェックポイントを取得し，また，実際に故障が発生したときにロールバックを行う手法を提案をした. この手法により, チェックポイントの取得時間を短縮することには成功しているが, サーバが単一故障点になっていることやサーバとの通信遅延はサーバと各プロセスが近いことから無視できるほど小さいとすること, サーバがチェックポイントの管理を行うことによるサーバへの負荷がプロセスに比例していることが問題になる.

3 提案手法

本論文では, サーバを持たず各ノードで論理時計を用いることでノード間の順序関係を保障する. 動作の遅いノードが故障した時, 動作の速いノードが古いチェックポイントに戻らなければならなくなる問題をチェックポイント取得間隔 Tδ とタイマーによって取得できる一定時間 σ を使用することによって動作のないノードであったとしても必ず長くても σ・Tδ 間隔でチェックポイントを取得を行う. ロールバックを行う際に, 故障ノードはチェックポイントから故障するまでにメッセージの送信を行ったノードに対してロールバックを行うメッセージを送信し, その後自身をロールバックする. メッセージを受信したノードは今の状態からロールバックするチェックポイントまでに送信した事のあるノードに対してロールバックを行うメッセージを送信し, その後自身をロールバックする. これを行うことにより故障ノードに関係性のないノードはロールバックの必要がなくなる. ロールバックを行うメッセージに関しては通信遅延がないものとすると各ノードが保存しておけばいいチェックポイントは　σ・Tδ　以内に取得したチェックポイントを保存しておけばいい.

3.1 チェックポイントの取り方

タイムスタンプの間隔 Tδ を定めておき，タイムスタンプが初めて k · Tδ 以上 (k = 0, 1, 2,...) になるイベントの直前にチェックポイントを取得する. 無矛盾な状況に含めるメッセージについては，タイムスタンプが k · Tδ 以上のイベントで受信するメッセージのうち，送信時のイベントのタイムスタンプが受信側が k · Tδ より小さいメッセージを含めるようにする. タイムスタンプは各プロセスが動作しない限り増加しない各ノードがいつチェックポイントを取得するかわからないのでタイマーにより一定時間　σ を 検知できることを利用して動作の遅いノードを最低でも σ 間隔でタイムスタンプを強制的に進めることで長くても σ・Tδ 間隔でチェックポイントを取得する.

3.2 ロールバック

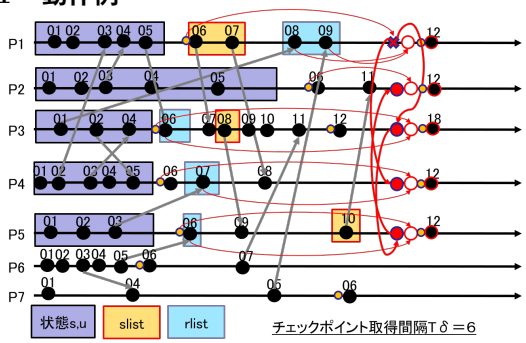
故障ノードを Pi としその他のノードを Pj,Pv とする (i ≠ j ≠ v)
故障ノード Pi

1. Pi が故障
2. Pi の状態を保存
   - ロールバックするチェックポイントの状態 s
   - ロールバックするチェックポイント取得後に受信したメッセージのうち添付のタイムスタンプがロールバックするチェックポイント直後のイベントのタイムスタンプより小さいメッセージのリスト rlist
   - チェックポイント取得後に送信したメッセージのリスト slist
3. Pi は slist にあるメッセージを送信した Pj に対して Pi からの受信破棄してほしいメッセージの通知
4. Pi はロールバックするチェックポイントの状態 s を復元
5. Pi は状態 s から rlist に含まれる全てのメッセージの受信処理を順に反映
6. Pi は次のイベントでチェックポイントを取得出来る様にタイムスタンプの値を増加させて動作を再開

ロールバックメッセージを受信する Pj

1. Pj は他のノードからメッセージの受信破棄とロールバックにすることを受信
2. Pj の状態を保存
   - 受信破棄するメッセージの中で Pj 側で一番タイムスタンプの小さい値の直前に取得したチェックポイントの状態 u
   - ロールバックするチェックポイント取得後に受信したメッセージのうち添付のタイムスタンプがロールバックするチェックポイント直後のイベントのタイムスタンプより小さいメッセージのリスト rlist
   - チェックポイント取得後に送信したメッセージのリスト slist
3. Pj は slist にあるメッセージを送信した Pv に対して Pj からの受信破棄してほしいメッセージの通知
4. Pj はロールバックするチェックポイントの状態 u を復元
5. Pj は状態 u から rlist に含まれる全てのメッセージの受信処理を順に反映
6. Pj は次のイベントでチェックポイントを取得出来る様にタイムスタンプの値を増加させて動作を再開

3.2.1 動作例



ノード P1

1. P1 が故障
2. P1 が状態 s・slist・rlist の 3 つを保存
3. P1 はロールバックが必要なことを slist{(06,P3),(07,P4)} の状態から P3,P4 に対して通知
4. P1 は状態 s を復元
5. P1 は状態 s から rlist{(08,P1),(09,P7)} に含まれる全てのメッセージの受信処理を順に反映
6. P1 は次のイベントでチェックポイントを取得出来るようにして動作を再開

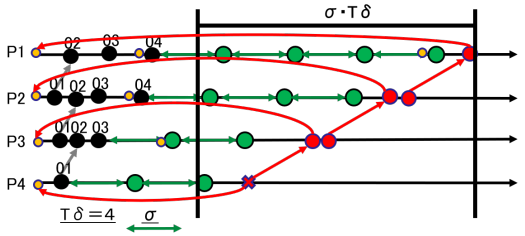
ノード P3

1. P3 が P1 メッセージの受信破棄とロールバックにすることを受信
2. P3 が状態 u・slist・rlist の 3 つを保存
3. P3 はロールバックが必要なことを slist{(08,P5)} の状態から P5 に対して通知
4. P3 は状態 u を復元
5. P3 は状態 u から rlist{(05,P1)} に含まれる全てのメッセージの受信処理を順に反映
6. P3 は次のイベントでチェックポイントを取得出来るようにして動作を再開

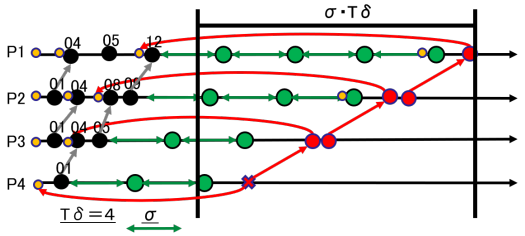
P5 のようにロールバックが必要になるノードで繰り返すことによりシステム全体で必要な所だけロールバックしていく.

4 考察と今後の課題

本論文では，論理時計を用いたチェックポイントの取得とロールバックを効率的に行う手法について考察をおこなった. 今後の課題としてはロールバックによる通信遅延を 0 としているためにロールバックの通信通信遅延を考慮する場合, 下図の通りノードのメッセージ送信関係から順番にロールバックの通知を行わなければならないパターンの際にロールバックを行うチェックポイントが σ・Tδ よりも前を使用しなければならない. その時のロールバックによる通信時間を D, ノードの台数 n とすると最大で σ・Tδ+(n-1)・D 時間のチェックポイントを保持する必要がある.



その問題を解決するためにノードが受信する前にチェックポイントを取得することを考える. よって物理時間において受信イベントよりも前の送信イベントの間には必ずチェックポイントが取得されていることになる. しかし, メッセージ通信よりロールバックの通信が必ず短くなることがいえなければメッセージの通信遅延を M とすると σ・Tδ+(n-1)・(D-M) のチェックポイントを保持していなければならない.



参考文献

[1] B. Randell, “ System structure for software fault tolerance, ”Proc. International Conference on Reliable Software, pp.437–449, 1975.

[2] Koo R, Toueg S. Checkpointing and rollback‐recovery for distributed systems. IEEE Trans Softw Eng. 1987; SE‐13(1): 23‐31. https://doi.org/10.1109/TSE.1987.232562

[3] 東 瞭斗, 守屋 宣. 分散アプリケーションのためのハイブリッド P2P 型分散チェックポイント・ロールバックアルゴリズムとその評価. 電子情報通信学会論文誌 D. 2020. J103-D. 1. 1-12

[4] L. Lamport, “ Time, clocks, and the ordering of events in a distributed system, ” Commun. ACM, vol.21, no.7, pp.558–565, 1978.