

分散 Linda のフォールトトレラント性

野里 貴仁 杉本 明

三菱電機株式会社 中央研究所

Linda モデルを使った分散システムのフォールトトレラントを実現する方法について述べる。我々の方法は、チェックポイントとデータ複製と組み合わせたものである。Linda モデルでは、システムはプロセスとタプルスペースから構成される。故障が起きても、計算を続けるために、プロセスに対しては、チェックポイントを残し、タプルスペースに対しては、プロセッサ毎にレプリカを用意している。チェックポイントは、個々のプロセスが独立に残す。この時、他のプロセスの情報を必要としない。また、チェックポイントからのロールバックは、履歴を使い、他のプロセスのロールバックを引き起こさない。

Fault-Tolerance on Distributed Linda

Takahito Nozato and Akira Sugimoto

Central Research Laboratory, Mitsubishi Electric Corporation
8-1-1 Tsukaguchi-Honmachi, Amagasaki, Hyogo, 661, JAPAN

This paper describes a method for fault tolerance on a distributed Linda system. Our method is combination of checkpointing and data replication. In a Linda model, a system contains processes and Tuple Space. In order to continue computations in spite of failures, processes take checkpoints and Tuple Space are replicated in every processor. Processes can take checkpoints independently, and require no information of other processes. Rollbacking from the checkpoint is based on histories, and does not cause other processes rollback.

1 はじめに

近年ネットワークの発達により、複数計算機のネットワークからなる分散システムが増加している。このような分散システムでは、単一計算機を使ったシステムよりも、信頼性を高くすることができる [1]。計算機が一つしかないシステムでは、その計算機の故障がシステム全体の故障になるが、複数の計算機から構成されるシステムでは、一つの計算機の故障が即座にシステム全体の故障とはならないからである。分散システムでは、一つの計算機に故障が起これども、残りの計算機を使ってシステムの実行を続けることが可能である。

分散システムを構築するためのモデルには、メッセージ通信を使うモデルと、共有メモリを使うモデルがある [2]。メッセージ通信を使うモデルでは、共有メモリを持たず、計算機上の実行実体 (プロセスと呼ぶ) がローカルな情報を持っており、互いにメッセージ通信を行なう。共有メモリを使うモデルでは、分散環境上にすべてのプロセスに共有される仮想的な空間が存在し、プロセスはこの共有空間を介して通信する。我々は分散システムの構築に後者の1つである Linda モデル [3, 4] を用いる。Linda モデルでは、タブルスペースと呼ばれる、仮想共有空間がある。プロセス間通信は、タブルをタブルスペースに加え、他のプロセスがそのタブルを読み込むことで、行なう。そのため、プロセス間の通信は明示的に相手を指定することがなく、プロセス間の依存関係は緩いものになっている。

故障が発生しても、システムが止まらないようにするために、チェックポイントを残す方法が広く用いられている [5][6]。プロセスはチェックポイントとして、その状態を安定記憶 (stable storage) に残し、故障が起きれば、最も新しいチェックポイントまで戻り (ロールバック)、そこから実行をやり直す。しかし、チェックポイントを残す手法の多くは、メッセージ通信をモデルとして使っており、Linda モデルにそのまま適用するには、タブルスペースの扱いが問題になる。また、ロールバックが発生することは、既に実行したものをキャンセルすることになり、外部環境への出力がある場合に不適である。Linda モデルを使ったフォールトトレランスに関する研究には [7] がある。しかし、これは、タブルスペースのフォールトトレランスについてのみであり、タブルを操作するプロセスに対しては、故障が起きないものと仮定している。Linda のフォールトトレランスにはタブルスペースのフォールトトレランスだけでは、不完全である。

本論文では Linda モデルの特徴を生かし、フォールトトレラントなタブルスペースとチェックポイント方式を融合した、フォールトトレラントな Linda システムについて述べる。我々の方法は、他のプロセスのチェックポイントの状態を知る必要がなく、また、他のプロセスのロールバックを引き起こすこともない。2 節で、我々の想定している分散システムのモデルを述べた後、3 節では、我々の方法を示す。4 節で、我々のフォールトトレランスの方法に重要な意味を持つタブルスペースの実現について述べ、5 節で、チェックポイント方式との比較を行なう。

2 分散システムのモデル

2.1 システムに関する仮定

我々の想定しているハードウェアは LAN によって接続された計算機群である。分散システムは、これらの計算機上のプロセスから構成される。プロセスはローカルな状態を持っており、プロセスどうしは互いに情報のやりとりを行なう。計算モデルとして Linda モデルを用

いる。プロセスどうしの情報のやりとりは、タブルスペースを介した通信であり、メッセージを使わない。また、プロセスはある時点の状態とそれ以後のタブルスペースの交信記録(順番、タブルの内容)から状態を再構築できるものとする。

故障に関しては、プロセッサの故障のみを考える。以下、本論文では故障はプロセッサの故障の意味で用いる。プロセッサは故障を起こすと、そのまま、止まり(フェイルストップ)、その故障は有限時間内にすべてのプロセッサに伝えられるものとする。プロセッサ間の通信に関しては、十分信頼でき、通信内容が途中でなくなることや、重複した送信はないものとする。また、システムには安定記憶が存在し、その内容は、故障によって失われることはないものとする。

2.2 Linda

Linda モデルでは、タブルスペースと呼ばれる仮想的な共有空間上でタブルと呼ばれるデータを用いてプロセス間の通信を行なう。タブルは型のついたフィールドの並びである。フィールドには文字列などのデータや変数が入る。タブルの例として(‘foo’, 1, ‘Tom’)や(‘begin’)などがある。Linda モデルではタブルスペースに対してタブルを読むこと、加えること、取り除くことしかできない。タブルを直接変更する操作がないために、タブルの変更はタブルを取り除き、タブルを新たに加えることで行なう。このためにタブルの操作には排他制御の機構が組む込まれることになり、プログラマが明示的にロックをかける必要はない。また、タブルの指定はパターンマッチングを使って行なわれ、パターンに対応するタブルがない場合はマッチするタブルが現れるまで実行がブロックされる。このために同期を自然な形で記述することができる。

タブルに対する操作は次の4つである。

- out(t)
タブルスペースにタブルtを加える。
- in(s)
s とマッチするタブルがあれば、それをタブルスペースから取り除く。この時sに仮引数があればそれに対応する値が代入される。マッチするタブルがない場合はそのようなタブルが現れるまで実行を中断する。
- read(s)
タブルスペースからタブルを取り除かないことを除いてinと同じである。
- eval(t)
t を別プロセスで評価し、その結果をタブルに置く。

Linda モデルを利点を以下に挙げる。

- C などの既存の逐次型言語に4つのプリミティブを付加することで、アプリケーション記述の言語にすることができる。このため、既存の言語を使用していたプログラマにとって、新たに言語を習得する手間が少ない。
- ある種の共有メモリである、タブルスペースを用いるので、明示的に通信を意識することがない。これも、今までに、並列言語を用いたことがないプログラマには相性が良いと考えられる。

- 操作に排他制御、同期機構が組み込まれているので、明示的なロックなどが不要である。

3 Linda のフォールトトレランス

Linda はプロセスとタブルスペースから構成される。Linda のフォールトトレランスには、この2つについて、故障の対策を立てることが必要である。プロセスの故障対策としては、プロセスが通常の実行時にチェックポイントを安定記憶に残し、故障発生後、そのチェックポイントからロールバックする方法 [5][6] が広く研究されている。また、タブルスペースについては、レプリカを用いる方法 [7] が研究されている。本節では、チェックポイントとフォールトトレラントなタブルスペースについて述べた後に、我々の方法を示す。

3.1 チェックポイント

チェックポイントを残す時には、チェックポイント間で矛盾が生じないようにすることが重要である。チェックポイント間の矛盾した状態とは、送り手のチェックポイントでは、メッセージを送った情報を残していない(メッセージ送信以前の状態を残している)が、受け手のチェックポイントではメッセージを受け取った後の状態を記録している場合である。矛盾しないチェックポイントを残すために、何らかの形でシステムのグローバルな状態を残す必要がある。[5]では、メッセージにタイムスタンプを付加することで、プロセス間の依存関係を形成する。この依存関係を調べることで、矛盾したチェックポイントを見つける。[6]ではチェックポイントを残す時に、プロセス間の同期を取るメッセージを用いている。しかし、Linda ではタブルスペースをシステムのグローバルな状態であると考えることができる。なぜなら、タブルスペースはすべてのプロセスに共有されているからである。このような、グローバルな状態が存在するシステムに対して、新たに、プロセスにシステムの状態を残すことは、無用な通信を増やすことになってしまう。

また、[5][6]ともに、故障後のチェックポイントからの実行が他のプロセスのロールバックを引き起こす。このために、外部環境への出力などのやり直しのできないものに対しては、ロールバックが起こらないと保証されるまで実行しない、などの制約が課せられる。

3.2 タブルスペースのフォールトトレランス

Linda のフォールトトレランスの研究には [7] がある。[7]では、タブルスペースのレプリカが常に同一の内容のタブルスペースを保持している。プロセスは自分が通信していたレプリカが動かなくなれば、別のレプリカと通信し、計算を続ける。レプリカ間で同一の内容を保つためのアルゴリズムと、通信するレプリカを変更するアルゴリズムについて述べている。しかし、[7]では、プロセスの故障は発生しないと仮定しており、Linda のフォールトトレランスには不完全である。

3.3 チェックポイントとレプリカの融合

我々はフォールトトレラントな Linda システムの構築のため、レプリカ方式と、チェックポイントを組み合わせる方式を用いる。すなわち、タブルスペースにレプリカ方式を使い、プロセッサの故障が発生すると、残りのサーバを使って、タブルスペースの内容を保つ。そ

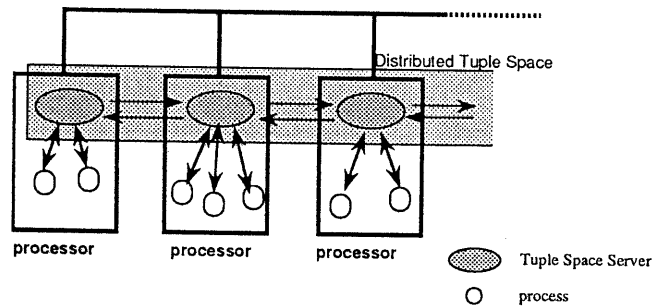


図 1: タブルスペースサーバ

して、プロセスはチェックポイントを残し、故障が発生すると、故障したノードで動いていたプロセスを、チェックポイントから、再び動かす。

すべてのプロセッサには同一内容のタブルスペースを保持するサーバが存在している。このサーバをタブルスペースサーバと呼ぶ。タブルスペースサーバどうしは、タブルスペースの内容を保つために、互いに通信している。プロセスは同じプロセッサ内にあるサーバと通信することで、タブルスペースにアクセスする (図 1)。

プロセスは故障対策のために、チェックポイントを安定記憶に残し、タブルの操作の記録を、履歴としてタブルスペースに残す。履歴についての詳細は 4 節で述べる。チェックポイントは適当な時に残し、タブルの履歴はタブルの操作と同時に残す。故障後の回復のために、プロセスをチェックポイントからロールバックするが、この時のタブルの操作は、履歴を参照することで行なう。このために、回復時に新たにタブルの付加や削除は行なわない。すなわち、回復作業によってタブルスペースの内容を更新することはない。そのため、あるプロセスのロールバックが他のプロセスのロールバックを引き起こすことはない。また、チェックポイントからの実行は履歴を参照して行なうので、再実行の計算は故障前と全く同一に行なわれる。そのために、一度外部環境へ出力したデータと全く同じデータを再構築することになり、再実行時の外部環境への出力については、出力を無視して、計算を続ければ良い。

回復は故障直前の状態に、たどり着いた時に終わる。その後、プロセスは通常の実行となり、タブルの操作も、タブルスペースにアクセスするものとなる。

回復に使われる通信は、プロセスが履歴を参照するための通信だけである。これは、ロールバックが他のプロセスの状態に全く依存しないで、行なわれることを意味している。すなわち、個々のプロセスのロールバックは、独立して行なわれる。そのために、チェックポイントも個々のプロセスが独立して残すことができる。同時に、ロールバックを独立して行なうことは、新しくチェックポイントを残した時に、以前のチェックポイントまで戻る必要がないことを意味している。よって、チェックポイントと履歴は新しくチェックポイントを残した時に、すべて捨て去ることができる。

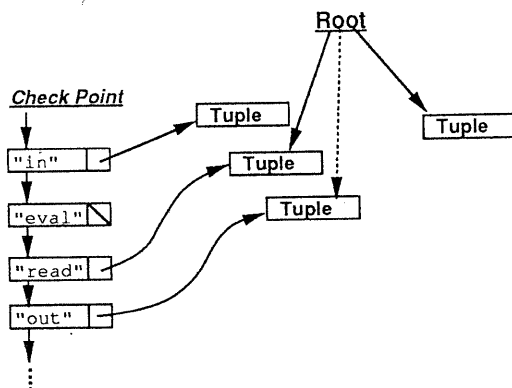


図 2: タブルの実現

4 タブルスペースの実現

本節では、フォールトトレランスのためのタブルスペースの実現について述べる。まず、タブルスペースサーバ内での履歴の実現手法を示し、次に、履歴の使われ方を示す。最後にノード間でタブルと履歴を共有するための操作を述べる¹。

我々のタブルスペース実現の特徴は、回復のために必要な情報をタブルスペースサーバが含んでいることである。すなわち、プロセスの実行履歴はすべてタブルスペースサーバ内で管理され、ノード間で一貫性を持って共有される。しかも、履歴の共有はタブルスペースの一貫性管理における通信と同時に行われるので、通信のコストは大きくならない。

4.1 タブルスペースサーバ内での履歴の実現

プロセスの通常実行時には、タブルスペースサーバは履歴として、4つの操作(out、in、read、eval)について、行なった操作とその結果を、実行した順番に記録している。

履歴も含めてタブルスペースサーバ上ではタブルを図2のように実現する。タブルスペースサーバ内ではタブルに対するリンクが存在する(図2でのRootからの矢印)。通常のプロセスの実行時にはこのリンクを使ってたどれるタブルだけをin/readする。このリンクをグローバルリンクと呼ぶ。通常のinではこのリンクを切ることで、タブルを見かけ上タブルスペースから削除している。履歴はチェックポイントからのびるリストとして実現する。履歴のエントリは、タブルに対する操作と操作したタブルへのリンクからなる²。

通常のプロセスの実行時には各操作に対して、タブルスペースサーバは履歴を次のように生成する。

out タブルを生成しタブルへのグローバルリンクを張る。履歴にoutを追加する。履歴からタブルへリンクを張る。

¹簡略のため、説明は1つのプロセスについて行なう。

²タブルに対する操作名はコンテキストから判別できるので、実際には必要ないが、説明を簡易にするために残している。

eval 履歴に eval を追加する。この時、タプルへのリンクは NULL ポインタである。
引数の評価が終わり、タプルが生成されれば、out と同じ操作を行なう。

in タプルへのグローバルリンクを切り、in を履歴に追加し、in されたタプルへリンクを張る。

read read を履歴に追加し、読み込まれたタプルへリンクを張る。

履歴の削除は、チェックポイントを取った時、および、プロセスが正常に終了した時に行なう。履歴を削除する時には、タプルを指していたリンクもなくなる。タプルの削除は、タプルへのリンクの数 (履歴からのリンク数とグローバルリンクの和) が 0 になった時に、行なう。タプルスペースサーバはノード間にわたって一貫性を保つので、削除されたタプルはすべてのノードのタプルスペースサーバ内から削除される。

4.2 履歴によるプロセスの再実行

プロセスの再実行には、タプルの操作を次のように行なう。

out 即座に次の実行に移る。タプルスペースに対しては何の操作もしない。

eval タプルへのリンクが NULL なら eval を再度実行する。これは引数の評価が終わる前に、故障が発生した場合である。そうでなければ、out と同様に即座に次の実行に移る。

in 履歴を使ってタプルへのリンクをたどり、もとのアクティビティが in したタプルと同じタプルを読み込んでくる。

read in と同様に履歴を使ってタプルを読み込んでくる。

再実行には、プロセスは履歴を使ってタプルを扱うので、タプルスペースに対する変更は全く行なわない。このために再実行によるノード間の余分な通信は起こらない。

4.3 ノード間での一貫性の実現

ノード間で一貫性を保つために、[7] の実現を用いる。ここで述べる実現方法は [7] の実現に対して履歴の一貫性を追加し、通信の信頼性が低かったための制約を取り除いたものである。out、in、read を次のように実現する。eval は out と同じ操作になるので、ここでは省略する。

out タプルをすべてのサーバに放送する。受け取ったサーバはタプルをタプルスペースに加え、履歴のエントリを新たに付け加える。これらの作業を終え、そのことを送り手のサーバに知らせる。すべてのサーバからこの返事が返ってくればプロセスに終了を知らせる。

in ローカルなタプルスペース内でマッチするタプルを探す。そのようなタプルがあれば、そのタプルをすべてのサーバに放送する。受け取ったサーバは履歴を作り、履歴からタプルへのリンクを張り、グローバルリンクを切断し、作業が完了したことを送り手のサーバに知らせる。送り手のサーバはすべてのサーバから返事が返ってくればプロセスにタプルの値を返す。マッチするタプルがない場合は、一致するタプルが現れるまで待つ。

read タプルへのグローバルリンクを切断する以外は、in と同じである。

5 考察

チェックポイント方式として [5] を取り上げ、我々の方法との比較を行なう。

[5] では、プロセスが、チェックポイント以外にも、メッセージログ、他のプロセスのチェックポイントの状態を記録し、これらを用いて、矛盾した状態を見つける。我々の方法では、プロセスはチェックポイントを残すのみである。それ以外の情報は、すべてタブルスペースに残している。そして、個々のプロセスが勝手にチェックポイントを残しても、チェックポイント間の矛盾した状態は起こり得ない。

我々の用いた履歴は [5] でのメッセージログに対応する。[5] では、すべてのメッセージに関するログが残るとは限らないが、我々はタブルスペース上で、タブルの操作に含めて残すために、履歴が完全に残る。

また、[5] では、チェックポイントを捨て去るために、他のプロセスのチェックポイントの進行具合を用いている。我々は、新たにチェックポイントを取った時点で、古いチェックポイントと履歴を捨て去ることができる。

6 まとめ

Linda モデルを使った分散システムのフォールトトレラントについて述べた。故障対策として、プロセスはその状態をチェックポイントに残し、タブルスペースはノード毎にレプリカを持っている。チェックポイントは、個々のプロセスが独立に残し、チェックポイント間の状態を矛盾のないようにするために、何ら付加的なことを行なわない。チェックポイントからの再実行は、履歴を使い、故障前と同一の実行をする。このために、故障に関係ないプロセスには影響を与えず、他のプロセスのロールバックを引き起こさない。また、履歴の管理はタブルの一貫性管理に組み込まれるので、履歴を残すことによる通信量の増加は少ない。

参考文献

- [1] Bal, H.E, Steiner, J.G., and Tanenbaum, A.S. : Programming Languages for Distributed Computing Systems *ACM Computing Survey*, Vol.21, No.3(1989),pp.261-322.
- [2] Stumm, M. and Zhou, S: Algorithms Implementing Distributed Shared Memory, *IEEE Computer*, Vol. 23, No. 5(1990), pp.54-64.
- [3] Ahuja, S., Carriero, N. and Gelernter, D: Linda and Friends, *IEEE Computer*, Vol.19, No.8(1986), pp.26-pp34.
- [4] Carriero, N. and Gelernter, D: *How to Write Parallel Programs*, MIT press, 1990
- [5] Strom, R. E.,and Yemini, S.: Optimistic recovery in distributed systems, *ACM Transaction on Computer Systems*, Vol.3, No.3(1985),pp.204-226.
- [6] Koo, R. and Toueg, S.: Checkpointing and Rollback-Recovery for Distributed Systems, *IEEE Transacion on Software Engineering*, Vol. SE-13, No.1(1987), pp.23-31.
- [7] Xu, A.S.: A Fault-tolerant Network Kernel for Linda, MIT/LCS/Tr-424, 1988