解　説

フォールトトレラント分散システム向けアルゴリズム

5. 分散チェックポイント・ロールバックアルゴリズム†

真鍋義文††　青柳滋己††

1. はじめに

分散システムであるプロセス（プロセッサ）がなんらかの方法で検出可能な故障で停止したとする．その際，予備のプロセスに故障プロセスの仕事を引き継がせる，あるいは故障プロセスを回復させて処理を継続することがある．故障プロセスの途中までの実行結果は失われるので再実行が必要であるが，全プロセスが最初から再実行を行うのは手間がかかり，外部入力など再現が困難なものが存在することもある．そこで実行中に随時，状態を復帰するための情報を安定記憶（**2.** 参照）に保存し（チェックポイントを取る），故障発生時にはその情報を用いて以前の状態に戻る（ロールバックする）手法が考えられている．

分散システムでは各プロセスが勝手にチェックポイントを取ってはいけない．**図-1** でプロセス p_2 が故障を起こしたとする．p_2 は最新のチェックポイント c_{22} にロールバックするが，その後受信したメッセージ m_{32} の情報が失われているため再実行できない．したがってプロセス p_3 は m_{32} の送信前から再実行する必要があり，その前のチェックポイント c_{31} にロールバックする．しかし同様に m_{21} の情報が失われているため，p_2 がさらにその送信前にロールバックする必要がある．この例では各プロセスは初期状態 c_{10}, c_{20}, c_{30} まで戻ることになる．このようなロールバックの連鎖はドミノ効果 (domino effect) と呼ばれている[31),32)]．

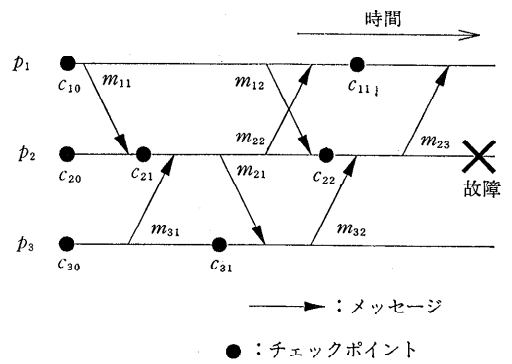


図-1　ドミノ効果

ドミノ効果を防止する分散チェックポイント・ロールバックの手法は協調型と自律型とに分類される．協調型チェックポイント・ロールバックでは，各プロセスがある意味で一斉にチェックポイントを取り，仮想的なある時点でのシステム全体の状態を得る．故障の際，各プロセスはそのチェックポイントの組のそれぞれにロールバックすることにより，ドミノ効果を防止する．

自律型チェックポイント・ロールバックでは，各プロセスが独立にチェックポイントを取る代わりに，プロセス間メッセージの内容および各プロセスの動作の因果関係情報を保持する．故障の際には因果関係情報からロールバック地点を求めることにより，ドミノ効果を防止する．

協調型の長所は，(1)プロセス動作時にチェックポイントのための動作を常に行う必要がない，(2)各プロセスの動作が非決定的（**2.** 参照）であってもよい点である．自律型の長所は，(1)各プロセスで都合の良いときにチェックポイントを取ることが可能である，(2)無故障プロセスがロールバックする可能性を低くできる点である．本稿ではこれらについて代表的なアルゴリズムを紹介する．

2. 諸定義

準備として用語の定義を行う．本稿では特に断

† Distributed Checkpoint and Rollback Algorithms by Yoshifumi MANABE and Shigemi AOYAGI (NTT Basic Research Laboratories).

†† NTT 基礎研究所

わらないかぎり以下の仮定をおく.

●各プロセスは揮発性記憶と安定記憶をもつ. 故障時に揮発性記憶の内容は失われるが, 安定記憶の内容は失われない. 安定記憶への書込みの遅れは大きく, プロセスの状態を常に安定記憶におくことはできない.

●プロセス間通信はメッセージ送受のみによる. プロセス対を結ぶチャネルは無故障で無限長のバッファをもつ FIFO キューである. プロセスはチャネルに対してメッセージ送信, 受信のコマンドを実行する. 通信は非同期すなわち, 通信の遅れは有限であるが不定である. 通信路のトポロジを, プロセスを頂点, チャネルを辺とした無向グラフ $G=(V,E)$ で表現する. プロセス数を n とすると, $|V|=n$ である.

●故障は停止故障 (crash failure) で検出可能である. ロールバック時には故障は起きない.

●単一プロセスに対して, チェックポイントを取るためのハードウェア的もしくはソフトウェア的な手法が存在する. 具体的手法については文献[5),27)]を参照のこと.

プロセス動作はある状態からイベントを実行し, 別の状態に遷移することの繰返しである. プロセス動作が決定的であるとは, 同一の状態で, 同一のメッセージを受信した場合には同一の状態に遷移することをいう. そうでない場合には非決定的であるという. プロセスのイベントに関して, 因果関係 (happened before) "$\rightarrow$" と呼ぶ半順序関係を以下のように定義する[20)].

定義 1 (因果関係)[20)]

(1) 任意のイベント a に対し, $a\rightarrow a$ である.

(2) あるプロセスにおいてイベント a を実行したのち b を実行するとき, $a\rightarrow b$ である.

(3) メッセージ m に対して送信イベントを $s(m)$, 受信イベントを $r(m)$ とすると, $s(m)\rightarrow r(m)$ である.

(4) $a\rightarrow b$ かつ $b\rightarrow c$ のとき, $a\rightarrow c$ である. ■

たとえば図-1 において, (2)より $s(m_{11})\rightarrow s(m_{12})$, (3)より $s(m_{12})\rightarrow r(m_{12})$ が成立し, これらと(4)より, $s(m_{11})\rightarrow r(m_{12})$ が成立する.

因果関係はイベント間の影響の存在を表す. $a\rightarrow b$ であれば, a の結果は b に影響を与える可能性がある. また, 因果関係はイベントの生起順序も表す. $a\rightarrow b$ かつ $a\neq b$ であれば, a のほうが b より必ず先に起きている. $a\nrightarrow b$ かつ $b\nrightarrow a$ であれば, プロセスの動作速度および通信の遅れにより, 先に起きるのが a,b どちらの場合もある. 図-1 において $s(m_{12})\nrightarrow s(m_{32})$ かつ $s(m_{32})\nrightarrow s(m_{12})$ であり, 先に起きるのがどちらの場合もある. また, これらのイベントは互いにその結果が他に影響を与えない.

3. 協調型チェックポイント・ロールバック

Chandy ら[6)]は一貫性のある全域状態を定義し, その状態で各プロセスがチェックポイントを取るアルゴリズムを示した.

定義 2 (一貫性のある全域状態)[6)]

分散システムの全域状態 (global state) とは, 各プロセス p_i のイベント e_i からなる n 個組 $(e_1, e_2, \cdots, e_n)$ である☆.

全域状態 $(e_1, e_2, \cdots, e_n)$ が一貫性がある (consistent) とは, 任意のメッセージ m の送信プロセスを p_i, 受信プロセスを p_j とすると, $r(m)\rightarrow e_j$ ならば $s(m)\rightarrow e_i$ が成立することである. ■

全域状態 $(e_1, e_2, \cdots, e_n)$ は, プロセス p_i がイベント e_i の実行を終了し, その次のイベントを実行する前の状態を表す. 一貫性があるとは, 未送信かつ受信済のメッセージが存在しないことである. **図-2** の C_1 で表される全域状態 $(s(m_{11}), s(m_{22}), s(m_{32}))$ は一貫性があるが, C_2 で表される全域状態 $(r(m_{22}), s(m_{21}), s(m_{31}))$ は m_{22} が定義 2 の条件を満足しないので一貫性がない.

一貫性のある全域状態は, 全プロセスを同時に監視できる仮想的な機構が存在した場合に起こり得る, ある時刻における全プロセスの状態の組である. すなわち, プロセスの実行速度および通信の遅れによっては, p_i がそれぞれ e_i を実行した直後の状態になる場合がある.

一貫性のある全域状態だけではメッセージの内容が失われているためロールバックには不十分である. そこで, 一貫性のある全域状態とその時点で送信済で未受信メッセージの内容 (図-2 の C_1 においては m_{22} および m_{32}) を保存情報とする. ロールバックの際には各プロセスは状態を戻すとともに, 各プロセス間のチャネルをクリアし, 保

☆ 全域状態を各プロセスの初期状態からのイベントの系列の n 個組と定義することもある.

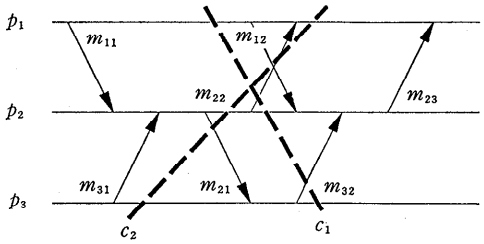


図-2　一貫性のある全域状態（C_1）と一貫性のない全域状態（C_2）

存メッセージをチャネルに戻す．この状態からは実行の継続が可能である．これが協調型チェックポイントの基本アイデアである．

3.1 Chandy らの協調型アルゴリズム

Chandy らのアルゴリズムでは，プロセス間の通常のメッセージとは異なる，マーカと呼ぶメッセージを送り合うことで一貫性のある全域状態を求め，かつ未受信メッセージの検出を行う．アルゴリズムの概要を以下に示す．

アルゴリズム 1[6]

一つ以上のプロセスが開始プロセスとなる．

[開始プロセス p_i の手続き]

begin

　p_i のチェックポイントを取る．

　p_i に接続されているすべてのチャネルにマーカを送信する．

end.　{通常のプロセス動作を続ける．}

[プロセス p_j（開始プロセスも含む）がチャネル c からマーカを受信したときの手続き]

begin

　if p_j はチェックポイントを取っていない **then**

　　begin

　　p_j のチェックポイントを取る．

　　空系列をチャネル c の状態として安定記憶に保存する．{未受信メッセージなし．}

　　p_j に接続されているすべてのチャネルにマーカを送信する．

　　end

　else

　　p_j がチェックポイントを取ってから今までに c から受信したメッセージの系列をチャネル c の状態として安定記憶に保存する．

end.　{通常のプロセス動作を続ける．}　■

マーカメッセージは $O(|E|)$ 送信される．**図-3** に動作例を示す．通信路のトポロジ G は完全グラフ，開始プロセスを p_1 とする．マーカメッセージの送受により，図-2 の C_1 の状態の組とメッセージ m_{22}，m_{32} の内容を保存情報としている．

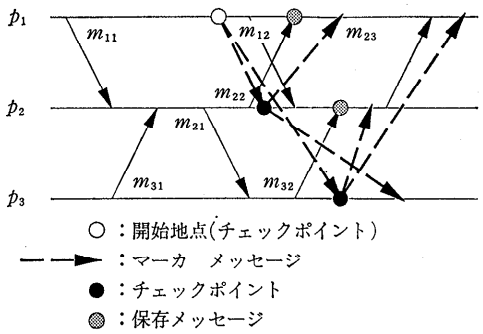


図-3　Chandy らのアルゴリズムの動作例

3.2 その他の協調型アルゴリズム

Lai ら[19]および Venkatesh ら[39]は，マーカを通常のプロセス間メッセージに付加するアルゴリズムを示している．Venkatesh ら[39]のアルゴリズムでは，各プロセスが開始したチェックポイント動作は独立に処理される．マーカとプロセス間メッセージへの付加情報を併用して通信路が FIFO でない場合にも適用可能なアルゴリズムも考えられている[33),36)]．

Koo ら[18]は，失われたメッセージの再送は下位レイヤが行うという前提のもとで，一貫性のある全域状態のみを保存するアルゴリズムを示した．Leu ら[21]は Koo らのアルゴリズムを改良し，(1)通信路が FIFO でない場合にも適用可能，(2)必要なプロセスのみロールバックを行い，ロールバック中の故障にも対処可能，という特徴をもつアルゴリズムを示している．

これらのアルゴリズムの比較を**表-1** に示す．

Venkatesan[38] は前回のチェックポイントとの差分を求めることにより，チェックポイントを取るためのメッセージ数を減少させるアルゴリズムを示している．また，各プロセスの時計がある程度合っている場合には各プロセスが自分の時計で一定時刻になるとチェックポイントを取る手法が考えられている[7),37)]．

Leu ら[22]らは，チェックポイント，ロールバック，メッセージ送受信をデータベースにおけるトランザクションのオペレーションと同じようにモ

表-1 主な協調型チェックポイント・ロールバックアルゴリズムの比較
通信計算量：チェックポイントアルゴリズム実行時のメッセージの総数
付加情報量：プロセス間メッセージに付加するカウンタ値などの数

研究	通信計算量	付加情報量	特記事項
Chandy ら[6]	$O(\|E\|)$	0	
Lai ら[19]	0	$O(1)$	
Venkatesh ら[39]	0	$O(n)$	開始プロセスごと別々のチェックポイント
Silva ら[33]	$O(n)$	$O(1)$	非 FIFO
Leu ら[21]	$O(\|E\|)$	0	非 FIFO，必要なプロセスだけロールバック

デル化して，互いに他をブロックする必要があるか否かを示している．

4. 自律型チェックポイント・ロールバック

以下では，各プロセスの動作は決定的であると仮定する．自律型アルゴリズムのためのメッセージ内容保存法として最初に考えられたのは，悲観的ログ保存法 (pessimistic logging)[4),30)] である．これは，メッセージ受信時にまずその内容を安定記憶に保存し，保存が終了するまで受信動作をブロックする手法である．これはオーバヘッドが大きいという欠点をもつ．その後考えられた楽観的ログ保存法 (optimistic logging)[35)] は，メッセージ内容の安定記憶への保存を待たずにプロセス動作を行う方法である．以下では楽観的ログ保存法をベースとした手法について述べる．

自律型アルゴリズムでは，故障によって失われた実行から影響を受ける実行部分のみをロールバックする．たとえば図-4 において，故障プロセス p_2 のチェックポイント c_{20} および受信メッセージ m_{31} の内容が安定記憶に保存されているとする．p_2 はこの保存情報を用いて $s(m_{22})$ まで再実行可能である．プロセス動作が決定的という仮定より，再実行時の m_{21}，m_{22} の内容は故障前の実行時と同一である．したがって，m_{12} の内容が p_1 に保存されていたとすると，p_1 は再実行を行わなくても p_2 からの要求に応じて m_{12} を再送信できるので，p_1 は $r(m_{22})$ 以前にロールバックする必要はない．

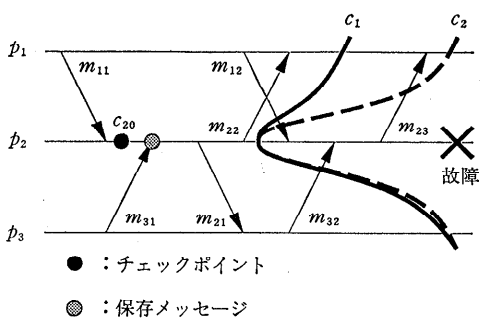


図-4 自律型アルゴリズムによるロールバック地点

しかし m_{12} と m_{32} の受信順序の情報は失われている．したがって，再実行時の $s(m_{22})$ より後の p_2 のメッセージ送信（送り先および内容）は故障前と異なる可能性がある．よって，p_1 は $r(m_{23})$ の直前にロールバックする必要がある．

故障プロセス p_i が再実行可能な最後の状態の次の受信イベントを e_i' とすると，e_i' 以降の実行の影響を受ける，すなわち $e_i' \rightarrow e_j$ を満足するプロセス p_j の実行 e_j はロールバックする必要がある．したがって p_j のロールバック地点は，$e_i' \not\rightarrow e_j'$ が成立する p_j の最後のイベントである．図-4 の例ではロールバック地点の組は C_1 となる．無故障プロセスのロールバックは，ロールバック地点以前の最新のチェックポイントから行うことができる．

さらに，もし m_{12} と m_{32} の受信順序を p_2 以外のプロセスに保存してあれば，それをもとに p_2 の再実行時の動作が故障前と同一になるように制御でき，無故障プロセスのロールバックは不要となる．ロールバック地点の組は図-4 の C_2 となる．

以下ではまず，無故障プロセスもロールバックする Strom ら[35)]のアルゴリズムの概要を述べ，その後の研究については主な違いのみを述べる．

4.1 Strom らの自律型アルゴリズム

準備として，状態区間 (state interval) $I_k(m)$ と呼ぶ状態の列を定義する．$I_k(m)$ はプロセス p_k に

おける m 番目のメッセージ受信イベントの終了から $m+1$ 番目のメッセージ受信イベントの直前までの区間と定義する．$I_k(m)$ の状態の状態区間番号 (state interval index) STI を m とする．STI は各プロセスごと独立につけられる値である．

各プロセスの動作の因果関係を依存ベクトル (dependency vector) DV で表す*．プロセス p_i の依存ベクトル DV_i の第 k 成分 $DV_i(k)$ は p_i の現在のイベント e_i に対して $e_k \rightarrow e_i$ を満足する p_k のもっとも後のイベント e_k の状態区間番号とする．また，$DV_i(i)$ は p_i の現在の STI の値とする．DV は以下のようにして求めることができる．初期状態において，p_i の依存ベクトル DV_i を

$$DV_i(i)=0, \quad DV_i(k)=-1 (k \neq i).$$

とする．各送信メッセージにはその時点での DV_i を付加する．メッセージ受信時には，付加された依存ベクトルを RDV とすると，

$$DV_i(k)=\begin{cases} DV_i(k)+1 & \text{if } k=i \\ max(DV_i(k), RDV(k)) & \text{otherwise} \end{cases}$$

によって計算する．たとえば**図-5** において p_1 で $r(m_{23})$ 実行時に，$DV_1=(1,2,0)$ と $RDV=(0,4,1)$ から $DV_1=(2,4,1)$ を得る．$r(m_{23})$ は p_2 の $I_2(4)$ の実行および p_3 の $I_3(1)$ の実行の影響を受ける．

Strom ら[35]のアルゴリズムは一つの故障が $O(2^n)$ 回のロールバックを起こすことがある．その点を改良した Sistla ら[34]のアルゴリズムを示す．

アルゴリズム 2[34]

[プロセス p_i の手続き]

● 実行中，随時チェックポイントを取る．これは他プロセスと同期を取らずに行う．

● 送信メッセージにはその時点の DV_i の値を付加する．送信メッセージのログ保存終了通知が受信側から到着するまでメッセージ内容を揮発性記憶に保存する．ただし通知を待たずにプロセス動作は継続する．

● メッセージ受信時には，上記の式で DV_i を計算するとともに，メッセージ内容を安定記憶に保存する．保存の終了を待たずにプロセス動作は継続する．保存終了時に送信プロセスにログ保存終了を通知する*．

● 故障発生時には，安定記憶に保存した情報から再実行を行い，再実行終了時の $DV_i(i)$ の値をロールバックメッセージとして全プロセスに送る．

● 無故障プロセスはロールバックメッセージ受信時には，現在の $DV_i(i)$ の値をロールバックメッセージとして全プロセスに送る．

● 全プロセスからのロールバックメッセージ受信終了時には（p_j から受信した内容を RDV_j とする），$DV_i(j) \leq RDV_j$ $(j=1,\cdots,n)$ が成立する最後の状態までロールバックする． ■

図-6 に上記アルゴリズムの動作例を示す．故障時に p_2 はチェックポイント c_{20} の状態に復帰し，さらに安定記憶の保存メッセージを用いて $s(m_{22})$ まで実行する．そしてその時点での $DV_2(2)$ の値 2 を他プロセスに送る．このメッセージを受信した p_1，p_3 はそれぞれその時点での $DV_1(1)$ の値 2 および $DV_3(3)$ の値 1 をロールバックメッセージとして送る．したがって各プロセスは DV が $(2,2,1)$ 以下になる状態までロールバックする．p_3 は現在の $DV_3=(0,2,1)$ なのでロールバックの必要がない．p_1 は $DV_1=(2,4,1)$ なので $(2,2,1)$ 以前となる $r(m_{22})$ 実行後までロールバックする．

ロールバックメッセージは $O(n^2)$ 送信される**．このアルゴリズムで各プロセスには故障回復以降の実行のメッセージと回復前の実行のメッセージの両方が届くことがある．これらの区別のため，各メッセージにはさらに世代番号 (incarnation number) IN およびセッション内番号 (session sequence number) SSN を付加する．IN の初期値は 0 であり，故障時の再実行終了時に 1 増加させる．SSN はチャネルごと，送信メッセージに対して 1 から順に付加する番号である．これらを用

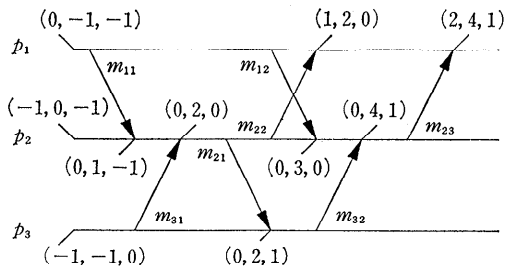


図-5　依存ベクトル DV の値

* Strom らは DV を状態区間番号と後述の世代番号の組のベクトルで定義している．

* 終了通知は他のメッセージに付加してもよい．

** 文献 13) のアルゴリズムを適用すればメッセージ数は $O(n)$ となる．

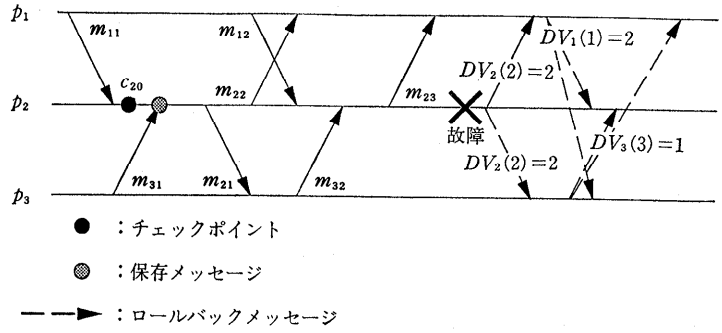


図-6 Sistla らのアルゴリズムの動作例

表-2 主な自律型チェックポイント・ロールバックアルゴリズムの比較

通信計算量：ロールバックアルゴリズム実行時のメッセージの総数（ただし失われたメッセージの再送分を含まない）
応答メッセージ数：プロセス間メッセージ一つに対して送られる受信確認などのメッセージ数
同時故障数：ロールバック可能な最大同時故障プロセス数
保存プロセス：メッセージ内容の安定記憶保存プロセス
無故障再実行：無故障プロセスのロールバック有無
D：グラフGの直径，k：故障プロセス数，$\Sigma\delta_i$：各故障プロセスの隣接プロセス数の総和

研究	通信計算量	付加情報量	応答メッセージ数	同時故障数	保存プロセス	無故障再実行	特記事項
Sistla ら[34]	$O(n^2)$	$O(n)$	1	1	受信側	あり	
Johnson ら[11]	$O(n^2)$	$O(1)$	2	1	送信側	あり	
Juang ら[13]	$O(n)$	$O(1)$	0	1	受信側	あり	Gはリング
Juang ら[14]	$O(D\cdot\|E\|)$	$O(1)$	0	1	受信側	あり	
Juang ら[13]	$O(kn)$	$O(n)$	1	任意	送信側	あり	
Juang ら[15]	$O(n\cdot\|E\|)$	0	0	任意	送信側	あり	
Juang ら[15]	$O(\Sigma\delta_i)$	$O(1)$	1	任意*	送信側	なし	* ただし $(p,q),(p,r)\in E$ を満たす故障プロセス p,q,r なし
Elnozahy ら[8]	$O(kn)$	$O(n^2)$	0	任意	送信側	なし	非 FIFO

いて故障回復前のメッセージの破棄および再実行時のメッセージの再送要求を行う☆．また，安定記憶の情報が増加するのを防ぐため，ロールバックに使用されなくなった情報を破棄する手続きも述べられている．詳細は文献 34)，35) を参照のこと．

4.2 その他の自律型アルゴリズム

本節ではその他の自律型アルゴリズムを示す．各手法の比較を**表-2** に示す．

Johnson ら[12] はチェックポイントおよびメッセージ内容の安定記憶への保存完了の情報を一カ所に集め，故障発生時のロールバック地点を求める集中型のアルゴリズムを示している．複数プロセスの同時故障にも対処できるが集中型のため，計算量などの比較は不可能である．

Juang ら[13),14)] は，ネットワークトポロジが限定された場合の効率のよいアルゴリズムを示している．Juang ら[15)]は，通常メッセージに情報を付加しないアルゴリズムを示している．Johnson ら[11)]はメッセージの内容を送信側で安定記憶に保存するアルゴリズムを示している．

Juang ら[13)]は，メッセージ内容を送信側で安定記憶に保存するアルゴリズムを用いて複数プロセスの同時故障にも対処可能にしている．**図-7** は m_{23} の内容を受信側プロセス p_1 が安定記憶に保存する直前に故障し，送信側の p_2 も同時に故障した場合である．最新のチェックポイント c_{11} と c_{21} は一貫性があるが，この状態にそれぞれがロールバックすると m_{23} の内容は失われる☆．

☆ 文献34) では述べられていないが，故障プロセスはロールバック終了時に SSN の値を他プロセスに送ることにより，再送要求を行う必要がある．

☆ 文献 13),14) の一部のアルゴリズムはこの問題に言及せず，複数プロセスの同時故障に対処可能と述べている．

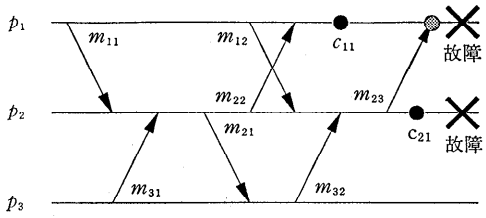


図-7　複数プロセスの同時故障

m_{23} の内容保存を送信側 p_2 で行えば，c_{21} の保存が終了した時点では m_{23} の内容保存が終了しているので，上記のメッセージ消失は起こらない．

無故障プロセスのロールバックがないアルゴリズムは，Juang ら[15]によって最初に考えられた．プロセス p_i は前述の状態区間番号 STI_i をもち，メッセージ受信時に STI_i の値を受信確認として送信プロセスに送る．STI_i は p_i での受信順序を表す値である．p_i の故障時には各プロセスは p_i へ送信したメッセージを，受信確認として受け取った STI の値とともに再送する．p_i は STI の値の順に再受信することにより実行の再現を行う．

Elnozahy ら[8]は，先行グラフ(antecedence graph) AG と呼ぶ受信順序を表す有向グラフを構成し，それを送信メッセージに付加することにより，無故障プロセスのロールバックのないアルゴリズムを示している．AG の頂点 σ_j^i は状態区間 $I_i(j)$ を表す．有向辺 (σ_j^i, σ_l^k) が存在するのは，(1) $i=k$ かつ $l=j+1$，または (2) $I_i(j)$ で送信したメッセージの受信により p_k で状態区間 $I_k(l)$ が開始する場合である．AG の例を**図-8** に示す．たとえば図-4 で p_2 が故障した場合も，m_{12} と m_{32} の受信順序の情報は m_{23} に付加されて p_1 に送られているため再現可能である．しかしメッセージへの付加情報量は多く，平均 $O(n^2)$ となる場合がある☆．

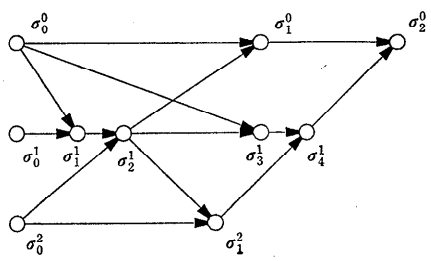


図-8　図4の $r(m_{23})$ における先行グラフ AG

Wang ら[40]はプロセス動作が非決定的である場合にロールバック地点を求める集中型のアルゴリズムを示している．また，Lowry ら[25]は大規模システムにおいて階層構造を導入して依存関係の保存情報量を減らす手法を考察している．

5. その他の分散チェックポイント・ロールバックアルゴリズム

前章までは分散プログラムの停止故障に関するチェックポイント・ロールバックアルゴリズムについて述べた．本章では，それ以外の応用に関する分散チェックポイント・ロールバックアルゴリズムなどについて述べる．

Bhargava ら[3]は，協調型チェックポイントと自律型チェックポイントについて，CPU 時間のオーバヘッドを実測している．また，Li ら[23]は，単一プロセスのプログラムに対して，コンパイル時にチェックポイントを取る時点を決めるアルゴリズムを与えている．

停止故障以外の故障に対処するために，リカバリブロックという手法が考えられている[17),31]．各プロセスのプログラムを複数のブロックに分解し，ブロックの最後にはテスト条件をおく．ブロックの先頭ではチェックポイントを取る．ブロックの実行の終了時にはテストを行って実行の正しさを検証する．テストに失敗すればそのブロックの実行は誤りとみて，先頭で保存したチェックポイントから再実行する．その場合にはそのブロックの動作から影響を受けた他のプロセスの実行もロールバックする手法である．

分散データベースのチェックポイントに対しては，チェックポイントのトランザクション一貫性 (transaction consistent) という概念が考えられている[9]．これは，トランザクションの実行順序に相当する系列 $\sigma_1\sigma_2\cdots\sigma_n$ に対して，$\sigma_1\cdots\sigma_i$ が終了して $\sigma_{i+1}\cdots\sigma_n$ を開始する前と等価な全域状態であることを表す．トランザクション一貫性をもつチェックポイントを取るためのアルゴリズムが考えられている[29]．

このほかに，安定全域状態 (stable global state)

☆ 論文中では付加情報量は評価されていない．

と呼ぶ，送信済かつ未受信メッセージも存在しない全域状態を求める協調型のアルゴリズムも研究されている[16),28)]. Lin ら[24)]は，オブジェクト指向システムに対するチェックポイントアルゴリズムを与えている．また，Ahamad ら[1)]は複数の独立なプロセスがサーバを共有した場合の，互いに影響のないチェックポイント手法を与えている．

また，分散プログラムのデバッグのためのロールバックアルゴリズムも考えられている．ユーザがプロセス p_i のある状態 e_i をロールバック先として指定する，すなわち e_i の時点で誤りが存在すると考えているとき，プロセス p_j のロールバック先はその誤りの原因が存在し得る箇所，すなわち $e_j \rightarrow e_i$ を満足するもっとも後の状態 e_j とするべきである．このような全域状態にロールバックするアルゴリズムが考えられている[2),10),26)].

6. おわりに

分散チェックポイント・ロールバックアルゴリズムの代表的な研究について解説した．プロセスの内部状態すべてを保存する現在のアルゴリズムはコストが大きいので，今後は対象システムの性質および利用法をさらに考慮に入れることにより効率の向上が図られていくと思われる．

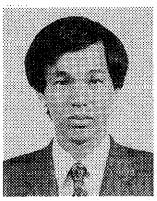
謝辞 NTT 基礎研究所の尾内理紀夫グループリーダに感謝いたします．

参考文献

1) Ahamad, M. and Lin, L.: Using Checkpoints to Localize the Effects of Faults in Distributed Systems, Proc. 8th Reliable Distributed Systems, pp. 2-11 (Oct. 1989).
2) 青柳，真鍋：分散デバッガのためのチェックポイント・ロールバックアルゴリズム，ソフトウェア科学会ソフトウェア研究会，SW-92-10-4 (Apr. 1992).
3) Bhargava, B., Lian, S-R. and Leu, P-J.: Experimental Evaluation of Concurrent Checkpointing and Rollback-Recovery Algorithms, Proc. 6th Data Engineering, pp. 182-189 (Feb. 1990).
4) Borg, A., Baumbach, J. and Glazer, S.: A Message System Supporting Fault Tolerance, Proc. 9th Operating System Principles, pp. 90-99 (Oct. 1983).
5) Bowen, N. S. and Pradhan, D. K.: Processor- and Memory-Based Checkpoint and Rollback Recovery, IEEE Computer, Vol. 26, No. 3, pp. 22-31 (Mar. 1993).
6) Chandy, K. M. and Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, ACM Trans. Computer Syst., Vol. 3, No. 1, pp. 63-75 (Feb. 1985).
7) Cristian, F. and Jahanian, F.: A Timestamp-Based Checkpointing Protocol for Long-Lived Distributed Computations, Proc. 10th Reliable Distributed Systems, pp. 12-20 (Oct. 1991).
8) Elnozahy, E. N. and Zwaenepoel, W.: Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback, and Fast Output Commit, IEEE Trans. Comput., Vol. 41, No. 5, pp. 526-531 (May 1992).
9) Fischer, M. J., Griffeth, N. D. and Lynch, N. A.: Global States of a Distributed System, IEEE Trans. Softw. Eng., Vol. SE-8, No. 3, pp. 198-202 (May 1982).
10) Fowler, J. and Zwaenepoel, W.: Causal Distributed Breakpoints, Proc. 10th Distributed Computing Systems, pp. 134-141 (May 1990).
11) Johnson, D. B. and Zwaenepoel, W.: Sender-Based Message Logging, Proc. 17th Fault-Tolerant Computing, pp. 14-19 (July 1987).
12) Johnson, D. B. and Zwaenepoel, W.: Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing, J. Algorithms, Vol. 11, No. 3, pp. 462-491 (Sep. 1990).
13) Juang, T. T-Y. and Venkatesan, S.: Efficient Algorithms for Crash Recovery in Distributed Systems, Proc. Foundations of Software Technology and Theoretical Computer Science, pp. 349-361 (Dec. 1990).
14) Juang, T. T-Y. and Venkatesan, S.: Efficient Crash Recovery in Sparse Low Diameter Distributed Systems, Proc. 29th Allerton Conf., on Computation, Control, and Computing, pp. 652-661 (Oct. 1991).
15) Juang, T. T-Y. and Venkatesan, S.: Crash Recovery with Little Overhead, Proc. 11th Distributed Computing Systems, pp. 454-461 (May 1991).
16) 角田，若原：効率良いプロトコルエラー回復のためのメッセージ送受信遷移系列の分散収集法，信学論 D-I, Vol. J 73-D-I, No. 2, pp. 134-140 (Feb. 1990).
17) Kim, K.H.: The PTC Scheme for Designing Loosely Coupled Recoverable Processes: Issues in Realizing Bounded Recovery Time, Proc. Future Trends of Distributed Computing Systems, pp. 287-296 (Apr. 1992).
18) Koo, R. and Toueg, S.: Checkpointing and Rollback-Recovery for Distributed Systems, IEEE Trans. on Softw. Eng., Vol. SE-13, No. 1, pp. 23-31 (Jan. 1987).
19) Lai, T. H. and Yang, T. H.: On Distributed Snapshots, Inf. Process. Lett., Vol. 25, No. 3, pp. 153-158 (May 1987).
20) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, Comm. ACM,

Vol. 21, No. 7, pp. 558-565 (July 1978).
21) Leu, P-J. and Bhargava, B.: Concurrent Robust Checkpointing and Recovery in Distributed Systems, Proc. 4th Data Engineering, pp. 154-163 (Feb. 1988).
22) Leu, P-J. and Bhargava, B.: A Model for Concurrent Checkpointing and Recovery Using Transactions, Proc. 9th Distributed Computing Systems, pp. 423-430 (June 1989).
23) Li, C-C. J. and Fuchs, W. K.: CATCH—Compiler-Assisted Techniques for Checkpointing, Proc. 20th Fault-Tolerant Computing, pp. 74-81 (June 1990).
24) Lin, L. and Ahamad, M.: Checkpointing and Rollback-Recovery in Distributed Object Based Systems, Proc. 20th Fault-Tolerant Computing, pp. 97-104 (June 1990).
25) Lowry, A., Russell, J. R. and Goldberg, A. P.: Optimistic Failure Recovery for Very Large Networks, Proc. 10th Reliable Distributed Systems, pp. 66-75 (Oct. 1991).
26) Masuzawa, T. and Tokura, N.: A Causal Distributed Breakpoint Algorithm for Distributed Debugger, 信学会秋季大会, SD-1-8 (Sep. 1992).
27) 森山, 多田: 利用者レベルで実現したプロセス移送ライブラリ, 情報処理学会研究報告, 91-OS-51-6 (July 1991).
28) 森保, 曽根岡, 真鍋: 分散システムにおける全域状態監視アルゴリズム, 信学論 D-I, Vol. J-74-D-I, No. 4, pp. 273-282 (Apr. 1991).
29) Pilarski, S. and Kameda, T.: Checkpointing for Distributed Databases: Starting from the Basics, IEEE Trans. Parallel and Distributed Systems, Vol. 3, No. 5, pp. 602-610 (Sep. 1992).
30) Powell, M. L. and Presotto, D. L.: Publishing: A Reliable Broadcast Communication Mechanism, Proc. 9th Operating System Principles, pp. 100-109 (Oct. 1983).
31) Randell, B.: System Structure for Software Fault Tolerance, IEEE Trans. Softw. Eng., Vol. SE-1, No. 2, pp. 220-232 (June 1975).
32) Russell, D. L.: State Restoration in Systems of Communicating Processes, IEEE Trans. Softw. Eng., Vol. SE-6, No. 2, pp. 183-194 (Mar. 1980).
33) Silva, L. M. and Silva, J. G.: Global Check pointing for Distributed Programs, Proc. 11th Reliable Distributed Systems, pp. 155-162 (Oct. 1992).
34) Sistla, A. P. and Welch, J. L.: Efficient Distributed Recovery Using Message Logging, Proc. 8th Principles of Distributed Computing, pp. 223-238 (Aug. 1989).
35) Strom, R. E. and Yemini, S.: Optimistic Recovery in Distributed Systems, ACM Trans. Computer Syst., Vol. 3, No. 3, pp. 204-226 (Aug. 1985).
36) Tenma, T., Kono, S. and Tokoro, M.: Giving Persistency to Group of Communicating Concurrent Objects, Proc. Workshop Object Oriented Computing, pp. 1-17 (1991).
37) Tong, Z., Kain, R. Y. and Tsai, W. T.: Rollback Recovery in Distributed Systems Using Loosely Synchronized Clocks, IEEE Trans. Parallel and Distributed Systems, Vol. 3, No. 2, pp. 246-251 (Mar. 1992).
38) Venkatesan, S.: Message-Optimal Incremental Snapshots, Proc. 9th Distributed Computing Systems, pp. 53-60 (June 1989).
39) Venkatesh, K., Radhakrishnan, T. and Li, H. F.: Optimal Checkpointing and Local Recording for Domino-Free Rollback Recovery, Inf. Process. Lett., Vol. 25, No. 5, pp. 295-303 (July 1987).
40) Wang, Y-M. and Fuchs, W. K.: Optimistic Message Logging for Independent Checkpointing in Message-Passing Systems, Proc. 11th Reliable Distributed Systems, pp. 147-154 (Oct. 1992).

(平成5年6月7日受付)



真鍋 義文 (正会員)

1960年生. 1983年大阪大学基礎工学部情報工学科卒業. 1985年同大学院修士課程修了. 同年日本電信電話(株)入社. 現在, 同社基礎研究所情報科学研究部主任研究員. 分散アルゴリズム, グラフ理論の研究に従事. 工学博士. 電子情報通信学会, ACM 各会員.

青柳 滋己 (正会員)

1965年生. 1988年東京工業大学理学部情報科学科卒業. 1990年3月同大学院理工学研究科情報科専攻修士課程修了. 同年日本電信電話(株)入社. 現在, 同社基礎研究所情報科学研究部勤務. 耐故障分散アルゴリズムの研究に従事. 日本ソフトウェア科学会会員.