システムソフトウェアと
オペレーティング・システム 64－6
マルチメディア通信と分散処理 65－6
（１９９４．５．１９）

分散システムにおける動的改版のためのグループ通信（その2）

桧垣 博章[1]　平川 豊
NTT ソフトウェア研究所

概要

大規模分散システムにおける多様な変化の影響を受容し、サービスの安定提供を支援する分散実行環境「リセプティブプラットホーム」が提案されている。ここでは、プロセス改版技術が重要な要素技術である。これまでに、未定義受信による異常状態を回避し、プロセスの同期停止を要しない「動的改版手法」を提案し、実現のためのグループ通信アルゴリズムを構築した。本稿では、従来アルゴリズムでは未解決のデッドロック問題に対して、タイムアウト設定を用いたデッドロック検出と、未定義受信への対処のためのチェックポイントーロールバックアルゴリズムを利用したデッドロック解消とを可能とするグループ通信アルゴリズムについて述べる。本アルゴリズムにより、改版への制約を緩和し、適用領域を拡大することが可能である。

Group Communications Algorithm for Dynamically Updating in Distributed Systems

Hiroaki HIGAKI　Yutaka HIRAKAWA
NTT Software Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo 180, JAPAN

Abstract

'Receptive platform' has been proposed to support stable service provision in large-scale distributed systems. It provides functions to tolerate various effects of changes. In order to achieve 'receptive platform', updating technique plays an important role. The concept of dynamically updating and the group communications algorithm to implement this concept have also been proposed. However, the proposed algorithm cannot solve the problem of deadlock occurrence. This paper describes a novel group communications algorithm that can detect deadlock occurrence by timeout setting and can resolve it by applying the checkpoint-rollback method for avoiding unspecified reception occurrence. This algorithm can support more updating styles than the existing algorithm.

[1]E-mail:higaki@sdesun.slab.ntt.jp

1 はじめに

近年の通信技術、計算機技術の発達により、大規模ネットワークを活用して高度かつ多様なサービスを提供する大規模分散システムの構築が可能になっている。しかし、環境やユーザ要求の経時的変化のために、大規模分散システムが定常状態で安定的に運用され続けることは稀である。これらの変化にともなうシステムへの影響を受容し、サービスの安定提供を実現するための分散実行環境として「リセプティブプラットホーム」が提案されている [1]。

運用中の大規模分散システムへの機能追加や機能変更を実現するための改版技術は、重要な要素技術である。本稿では、「運用中の実行実体 (旧版プロセス) から機能追加や機能変更がなされた実行実体 (新版プロセス) への置換操作」を改版と定義する。この改版処理によってシステムが異常状態に陥ることを避けなければならない。分散システムにおけるプロトコル的変更に注目した場合には、未定義受信の発生が典型的な異常状態である。

これまでに、改版処理による未定義受信の発生を完全に回避する手法が提案されている [2, 3]。その基本方針は、改版に関係するプロセスを同期停止させた安定状態でプロセス置換を実行するというものであり、停止対象プロセス集合の縮小によって、改版処理コストの減少を目指してきた。適用対象システムは、プロセス間関係が主従関係のみで構成され、その静的な解析のみで停止対象プロセスを決定できる RPC 型やクライアントサーバ型に限定されていた。ところが、プロセスが自律的に動作、通信する対等型分散システムにおけるプロセス間関係には、以下の特徴がある。(1) 同時に多数のプロセスが関係する。(2) 関係は動的に決定される。(3) 関係は対等である。このため、プロセスが停止可能な安定状態の獲得は困難である。また、複雑で動的な関係のために、停止対象プロセス集合の決定が困難であり、決定可能であっても一般に集合が大きくなる。以上により、改版操作による影響の波及方向、波及範囲決定の容易さに依存した従来手法を対等型分散システムに適用することは、本質的に困難である [4]。そこで、未定義受信の発生を完全に回避するのではなく、未定義受信に対処可能な機構を備えることによって安全な改版を実現する動的改版手法が提案された [5]。この手法は、複数プロセスの同期停止を必要としないため、対等型分散システムにおいてもサービス停止コストの小さい改版を実現できる。

2 動的改版手法

動的改版手法の概要は以下の通りである。

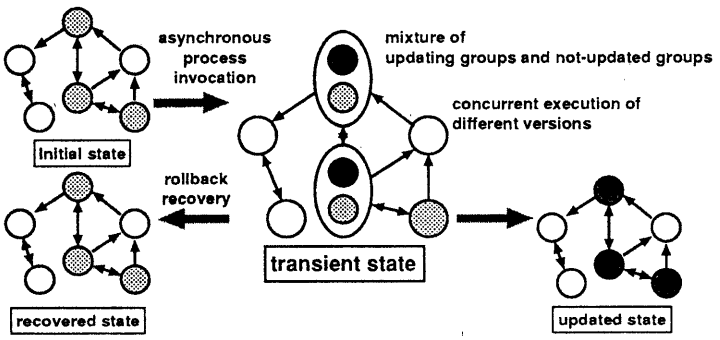

図 1: Overview of dynamically updating.

(1) 旧版プロセスは各々ひとつのプロセスグループに属する。改版対象プロセスは、他と同期することなく新版プロセスを同一プロセスグループ内に起動する。システムには、改版過程グループと未改版グループが混在する過渡状態が形成される。

(2) 改版過程グループでは、旧版プロセスが新版プロセスと同一状態遷移可能な範囲で並行動作を継続する。ロールバック回復処理後の未定義受信発生を回避するために、旧版プロセスは新版プロセスの通信メッセージを観測し、適切な状態にチェックポイントを設定して一時停止する。

(3) 過渡状態で未定義受信を発生することなく、全改版対象プロセスで新版プロセスを起動した場合には、旧版プロセスを停止して改版処理を終了する。

(4) 過渡状態で未定義受信が発生した場合には、新版プロセスの動作継続を断念し、設定済みのチェックポイントから旧版プロセスに処理を継続させることによってロールバック回復を実現する。

2.1 グループ通信アルゴリズム

過渡状態におけるプロセス間通信を支援し、ロールバックコスト最小のチェックポイント設定により、動的改版を実現するグループ通信アルゴリズムの概要を示す[1]。

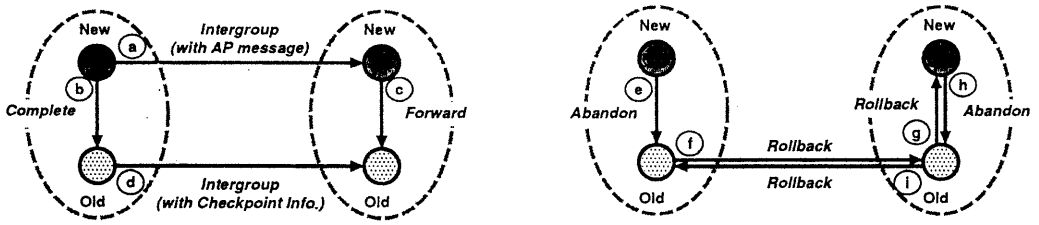

図 2: Group communication and rollback recovery.

(1) 送信イベント実行時には、新版プロセスは AP メッセージを含むグループ通信信号を受信側新版プロセスへ送信し (図 2:a)、送信側旧版プロセスへ同一 AP メッセージを含む送信完了通知信号を送信する (図 2:b)。旧版プロセスは、この送信完了信号を S-queue(FIFO キュー) に登録し、送信イベント実行時にこの信号に含まれる AP メッセージと送信すべきメッセージとを比較する。同一ならば、チェックポイント未設定情報を含むグループ通信信号を受信側旧版プロセスに送信する (図 2:d)。異なるならば、チェックポイント設定済み情報を含むグループ通信信号を受信側旧版プロセスに送信し (図 2:d)、チェックポイントを設定する。
(2) 新版プロセスは受信したグループ通信信号を R-queue(FIFO キュー) に登録し、この信号に含まれる AP メッセージを転送信号を用いて旧版プロセスに通知する (図 2:c)。旧版プロセスはこの転送信号を R-queue に登録し、受信イベント実行時にこの信号に含まれる AP メッセージをアプリケーションに配付する。このとき、AP メッセージの受理が不可能または対応するグループ通信信号がチェックポイント設定情報を含む場合には、チェックポイントを設定する。
(3) チェックポイント設定済み旧版プロセスは、チェックポイント設定済み情報を含むグループ通信信号を送受したグループを記録する。新版プロセスで未定義受信が発生した場合は、旧版プロセスが記録したグループに回復信号を送信する (図 2:f)。最初の回復信号を受信時には、新版プロセスを停止し、旧版プロセスが同様に回復信号を送信する (図 2:i)。記録された全てのグループから回復信号を受信した時点でチェックポイントからの動作を再開する。

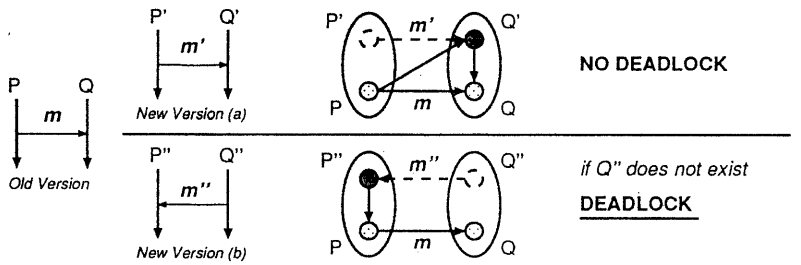

図 3: Deadlock in dynamically updating.

2.2 問題点

グループ通信アルゴリズムを適用するには、分散アプリケーションが以下の条件を満たさなければならない。
[条件] 新旧版プロセスの混在動作によってデッドロックが発生しない。

このため、旧版プロセス仕様と新版プロセス仕様の分岐点となるイベントは、ともに送信イベントあるいはともに受信イベントのいずれかに限定される。図 3 において、新版仕様 (a) への改版では、仕様の分岐点は P では送信同士、Q では受信同士であり、デッドロックは発生しない。しかし、新版仕様 (b) への改版では、仕様の分岐点は P、Q ともに送受信イベントが混在し、Q'' が存在しない場合にはデッドロックが発生する。機能追加の多くが受信分岐

[1] アルゴリズムの詳細と正当性は [5] 参照。

の追加あるいは条件分岐にともなう送信分岐の追加によって実現されるので、従来アルゴリズムで有効な対処が可能である。しかし、機能変更や同期メッセージの追加 / 削除、プロセス構成変更などにおいては、この条件が新版仕様に対して強い制約になる場合がある。

3 提案アルゴリズム

3.1 提案アルゴリズムの概要

前章で述べた問題の解決法について述べる。

送受信イベント混在分岐への対処 旧版プロセスは、送信完了通知信号、転送信号を個別の信号キュー (S-queue/R-queue) に登録し、イベント実行時に対応するキューから信号を取り出すことで、新版プロセスのメッセージ送受信を観測していた。このため、送受信イベント混在分岐への対処が不可能であった。そこで、イベント実行時には以下の処理を行なうこととする。
(1) 新版プロセスは送受信イベントいずれにおいても、実行したイベントの内容を通知するイベント通知信号を旧版プロセスに送信し、旧版プロセスはこれを E-queue(FIFO キュー) に登録する。
(2) 旧版プロセスはイベント実行時に E-queue から信号を取り出し、新版プロセスが実行したイベントと現在実行すべきイベントとが同一であるならば従来通りの処理を行なう。異なる場合には、以下の処理を行なう。
(2-1) 新版プロセスが送信イベントを実行した場合には、受信側旧版プロセスにチェックポイント設定済み情報を含むグループ通信信号を送信し、チェックポイントを設定する。
(2-2) 新版プロセスが受信イベントを実行している場合には、ただちにチェックポイントを設定する。
また、従来は転送信号と送信完了通知信号を個別に直列化すれば十分であったため、新版プロセスは転送信号をグループ通信信号受信時に送信していたが、イベント通知信号は送受信イベント実行履歴を通知するものであるため、受信イベントに関しても実行時に旧版プロセスに通知することとする (‡)。

デッドロックへの対処 グループ通信アルゴリズムでは、改版過程にある旧版プロセスは、送受信イベントいずれにおいても新版プロセスからの制御信号の受信動作を行なう。
[性質 1] あるサブシステムの単一版プロセスと新版プロセスが全て受信イベント実行中であり、互いに受信待ちである場合にデッドロックが発生する。
同一版からなるシステムではデッドロックは発生しない、という仮定から、次の性質が成り立つ。
[性質 2] デッドロックしているプロセスグループ群には必ず未改版プロセスグループが含まれており、その単一版プロセスは受信イベント実行中である。
そして、この受信イベントに対応する送信イベントを含むプロセスグループ内では、新版プロセスが受信イベント実行中で、旧版プロセスがチェックポイント設定済みであるため、対応する送信イベントは実行されない。
[性質 3] デッドロックしている単一版プロセスが実行中の受信イベントに対応する送信イベントを含むプロセスグループのロールバックによって、デッドロックが解消できる。
デッドロック検出 / 解消アルゴリズムを以下に示す。
(1) 単一版プロセスは、受信イベント実行時に (十分長い) タイムアウトを設定する。
(2) グループ通信信号受信待ちでタイムアウトした場合は、対応する送信イベントを含む旧版プロセスにデッドロック検出信号を送信する。
(3) デッドロック検出信号を受信した旧版プロセスは、ロールバック回復処理を開始する。
未改版プロセスグループがデッドロック検出信号を受信した場合には、これに含まれる単一版プロセスが必ずタイムアウトすることから、受信信号を無視すればよい。
提案アルゴリズムのためのタスク群を付録に示す。

3.2 提案アルゴリズムの性質

前節の提案アルゴリズムは次の利点、欠点を持つ。
[利点 1] 適用可能な改版の範囲を拡大する。
新版仕様に対して、旧版仕様との分岐点が全プロセスで送信イベント同士または受信イベント同士という制約が不要である。
[利点 2] デッドロック解消のためコストが最小のロールバックを実現する。
デッドロック解消のためには、デッドロック検出信号送信先でのロールバックが必修である。このロールバックを提

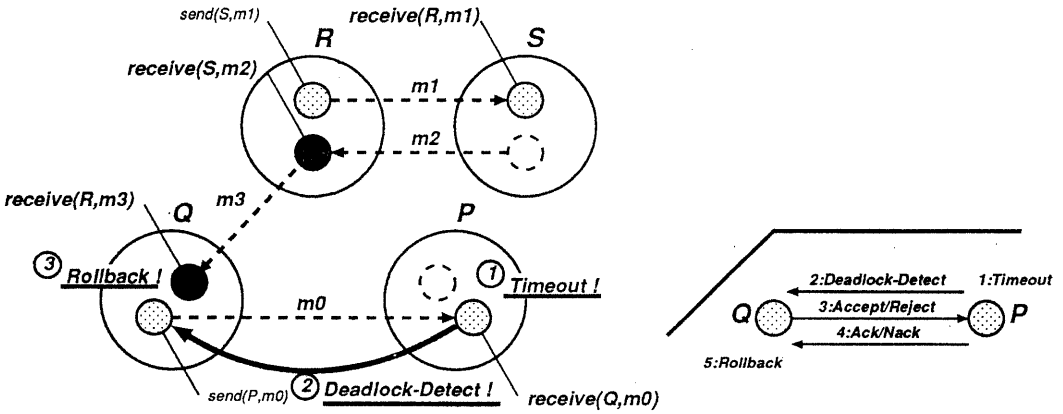

図 4: Deadlock detection and resolution.

案アルゴリズムが最小コストで実現することは [5] で証明済みである。

[欠点 1] 新旧版プロセスイベント実行の同期が強い。

従来は送信イベントのみでの同期であったが、(♯) により、提案アルゴリズムでは送受信イベントでの同期が必要である。このため、旧版プロセスが受信待ちになる機会が増加する。ただし、アプリケーション実行は新版プロセスによって制御されているため、その影響は小さい。

[欠点 2] タイムアウトの長さが不十分である場合、不要なロールバックが発生することがある。

ローカルなデッドロック検出法を用いているため、完全に回避することはできない。ただし、不要なロールバックによるシステム誤動作は回避しなければならない。そのため、次の 2 つの場合に対処する必要がある。

・ デッドロック検出プロセスの受信待ちグループ通信信号が、デッドロック検出信号受信以前に送信済みの場合。

・ デッドロック検出信号受信プロセスがチェックポイント未設定の場合。

これらは以下の処理で対処できる。

(1) 検出信号受信プロセスは、チェックポイント設定済みならば Accept、未設定ならば Reject を返信する。

(2) 検出プロセスは、Reject 受信時には再びタイムアウトを設定する。Accept 受信時には、受信待ちグループ通信信号が未受信ならば Ack を、受信済みならば Nack を送信する。

(3)Ack を受信した場合にのみロールバックを開始する。

4 まとめ

本稿では、動的改版実現のためのグループ通信アルゴリズムにおけるデッドロック発生に対処するため、タイムアウト設定による検出とチェックポイントーロールバック手法による解消とを実現する新しいグループ通信アルゴリズムを提案した。デッドロック解消のためのロールバックは、従来の未定義受信発生時のロールバックと同様に最小コストで実現することができる。本アルゴリズムによって、従来の新旧版仕様間の制約は不要になり、より広範囲の改版へ動的改版手法を適用することができる。

参考文献

[1] 桧垣, 森保, 奥山, 平川, 市川, “システム進化支援環境 リセプティブプラットホーム,” 情報処理学会第 47 回全国大会, 6E–1, pp.201–202 (1993).

[2] M.E. Segal and O. Frieder, “Dynamically Updating Distributed Software: Supporting Change in Uncertain and Mistrustful Environments,” Proc. of IEEE Conf. on Software Maintenance, pp. 254–261 (1989).

[3] J. Kramer and J. Magee, “The Evolving Philosophers Problem: Dynamic Change Management,” IEEE Trans. Softw. Eng. , vol.16, No.11, pp. 1293–1306 (1990).

[4] M.R. Barbacci, D.L. Doubleday, and C.B. Weinstock, “Application Level Programming,” Proc. of 9th ICDCS, pp. 458–465 (1990).

[5] 桧垣, “分散システムにおける動的改版のためのグループ通信,” 情報処理学会マルチメディア通信と分散処理ワークショップ, pp.81–90 (1993).

付録 提案アルゴリズム

Send(GID,APmes)

if Vop_p =Single then
 if $CPL_p = \emptyset$ then send(All(GID),Intergroup(MID,False,GID_p,APmes));
 else send(All(GID),Intergroup(MID,True,GID_p,APmes)); append(CPL_p,GID);
else if Vop_p =New then
 send(New(GID),Intergroup(MID,Empty,GID_p,APmes)); send(Old(GID_p),EventInform(MID,Send,GID,APmes));
else
 SIG:=dequeue($E\text{-queue}_p$);
 if $EVENT_{SIG}$ =Send then
  if $APmes_{SIG}$ =APmes and GID_{SIG} =GID then send(Old(GID),Intergroup(MID_{SIG},False,GID_p,APmes));
  else
   append(CPL_p,{GID_p,GID_{SIG}}); send(Old(GID_{SIG}),Intergroup(MID_{SIG},True,GID_p,$APmes_{SIG}$));
   p takes a checkpoint and stops executing AP events;
 else *p takes a checkpoint and stops executing AP events*;

Receive()

if Vop_p =Single then
 if *timeout expires without enqueuing any signals to* $E\text{-queue}_p$ then send(Old(Sender),DeadlockDetect());
 else
  SIG:=dequeue($E\text{-queue}_p$);
  if CP_{SIG} =True then
   if $CPL_p = \emptyset$ then append(CPL_p, GID_p); *p takes a checkpoint*;
   append(CPL_p, GID_{SIG});
   if acceptable(GID_{SIG},$APmes_{SIG}$) then deliver(GID_{SIG},$APmes_{SIG}$);
   else send(Old(∀CPL_p),Rollback(GID_p)); append(RBL_p,GID_p);
  else deliver(GID_{SIG}, $APmes_{SIG}$); if $CPL_p \neq \emptyset$ then enqueue($RB\text{-queue}_p$, SIG);
else if Vop_p =New then
 SIG:=dequeue($E\text{-queue}_p$);
 if acceptable(GID_{SIG},$APmes_{SIG}$) then
  deliver(GID_{SIG},$APmes_{SIG}$); send(Old(GID_p),EventInform(MID_{SIG},Receive,Empty,GID_{SIG},$APmes_{SIG}$));
 else send(Old(GID_p),Abandon()); *p stops execution*;
else
 SIG:=dequeue($E\text{-queue}_p$);
 if $EVENT_{SIG}$ =Receive then
  if CP_{SIG} =Empty then
   p suspends until receiving WSIG, MID_{WSIG} = MID_{SIG}; remove($P\text{-buffer}_p$,WSIG); CP_{SIG} :=CP_{WSIG};
  if CP_{SIG} =True then append(CPL_p,{GID_p,GID_{SIG}}); *p takes a checkpoint and stops executing AP events*;
  else deliver(GID_{SIG},$APmes_{SIG}$);
 else
  append(CPL_p,{GID_p,GID_{SIG}}); send(Old(GID_{SIG}),Intergroup(MID_{SIG},True,GID_p,$APmes_{SIG}$));
  p takes a checkpoint and stops executing AP events;

SignalReception(SIG){Intergroup}

if Vop_p =Single then if CP_{SIG} ≠Empty then enqueue($E\text{-queue}_p$,SIG);
else if Vop_p =New then enqueue($R\text{-queue}_p$,SIG);
else
 if CP_{SIG} =Empty then send(New(GID_p), Intergroup(MID_{SIG},CP_{SIG},GID_{SIG},$APmes_{SIG}$));
 else if (∃QSIG∈$R\text{-queue}_p$) MID_{QSIG} =MID_{SIG} then CP_{QSIG} :=CP_{SIG};
 else if $CPL_p = \emptyset$ then append($P\text{-buffer}_p$,SIG);
 else if CP_{SIG} =True then append(CPL_p,GID_{SIG}); else enqueue($RB\text{-queue}_p$,SIG);

SignalReception(SIG){EventInform}

if $CPL_p = \emptyset$ then
 enqueue($E\text{-queue}_p$,SIG); if (∃BSIG∈$P\text{-buffer}_p$) MID_{BSIG} =MID_{SIG} then CP_{SIG} :=CP_{BSIG}; remove($P\text{-buffer}_p$,BSIG);
else if $EVENT_{SIG}$ =Send then append(CPL_p,GID_{SIG}); send(Old(GID_{SIG}),Intergroup(MID_{SIG},True,GID_p,$APmes_{SIG}$));

SignalReception(SIG){DeadlockDetect}

if Vop_p =Old then send(Old(∀CPL_p),Rollback(GID_p)); append(RBL_p,GID_p);