分散システムにおけるフォールトトレランスのためのグループ通信*

柿垣 博章 †　　曽根岡 昭直 ‡
NTT ソフトウェア研究所　　NTT 総合企画本部

概要
分散システムの構成オブジェクトを複製プロセスからなるプロセスグループで実現することによって、システムの耐故障性を高めることができる。本稿では、サーバクライアントモデル分散システムを対象として提案された ordered multicast protocol の適用が困難であるパートナモデル分散システムを対象としたフォールトトレランス実現のためのグループ通信アルゴリズムを提案する。提案アルゴリズムは、プロセスグループ間通信を複製プロセス数の線形オーダのメッセージ数で実現し、プロセス故障情報をプロセスグループに隠蔽することによってオーバヘッドを削減している。

1　はじめに

分散システムでは、複数のプロセッサと記憶装置がネットワークで接続され、複数のオブジェクトがメッセージ交換によって通信して処理を行なう。分散システムは、この多重性によって集中システムに比べて潜在的により可用性、信頼性の高いものになる。オブジェクトを複製して異なるプロセッサに配置することは、分散システムにおけるフォールトトレランス実現のための有効な手法のひとつである。

一方、通信システムや計算機ネットワークといった大規模分散システム、分散制御システム、協調型問題解決(分散 AI)、協調作業支援 (CSCW) など、近年の重要な分散システムの多くは、多数のオブジェクトから構成され、それらの関係が複雑化している。そのため、オブジェクトの関係を主従関係に限定したサーバクライアントモデルや RPC モデルではなく、オブジェクトが対等な立場でメッセージを交換するパートナモデル分散システム (対等分散システム) として適切にモデル化できる [8]。本稿では、パートナモデル分散システムに適用可能な複製プロセスを用いたフォールトトレランス実現手法を提案する。

オブジェクトを複製プロセスで実現する主要なフォールトトレランス手法は、active replication と passive replication に分類される。passive replication では、各オブジェクトを実現する複製プロセスのひとつだけが動作し (primary process)、その他 (backup process) は処理を行なわずに待機する。primary process は、あらかじめ設定されたチェックポイントで backup process に状態情報を転送する。primary process の故障発生時には backup process のひとつが最新のチェックポイントから動作を開始する。このとき、状態情報がリストアされ、オブジェクトの動作がチェックポイントへロールバックする。したがって、primary process が故障前に実行したイベントが再実行されるため、故障に伴なう処理の中断時間が長い。さらに、チェックポイント設定による開発オーバヘッドも必要である。一方、active replication では同一オブジェクトを実現する全複製プロセスが緊密に同期して全イベントを同時に実行する。このため、チェックポイントにおける状態情報の転送や故障発生時のロールバックによるオーバヘッドが不要である。

しかし、分散システムではプロセス間通信にメッセージ伝達遅延が伴なうため、複製プロセスを緊密に同期させることはできない。そこで、複製プロセスが同一の状態遷移を行なうことのみを保証することによって active replication フォールトトレランスを実現する手法が考えられる。メッセージ伝達遅延の正確な予測は不可能であるため、複製プロセスに送られたメッセージの受信順序はプロセスごとに異なる。このため、各複製プロセスの状態遷移は異なり、プロセス故障発生時にオブジェクトの動作の一貫性を保持することができない。この問題を解決するために、オブジェクトをプロセスグループとして実現し、同一プロセスグループに属する全複製プロセスが同一順序でメッセージを受信することを保証する ordered multicast protocol を用いる方法が提案されている。これは point-to-group の通信プロトコルであるため、サーバクライアントモデル分散システムにおけるサーバオブジェクトのフォールトトレランスを実現する手法として有効である。Clouds[1] Circus[5] Delta-4[7] における active replication の実現や ISIS[3] Auragen[4] における passive replication の実現に用いられている。しかし、これらはクライアントオブジェクトのフォールトトレランスについては考慮していないため、パートナモデル分散システムに単純に適用することは困難である。パートナモデル分散システムに ordered

*Group-to-Group Communications for Fault-Tolerance in Distributed Systems
†Hiroaki HIGAKI, NTT Software Laboratories
‡Terunao SONEOKA, NTT General Planning Headquarters

multicast protocolをそのまま適用すると、全複製プロセスが同一メッセージを重複して送信する。N個の複製プロセスを含むプロセスグループでオブジェクトを実現した場合、オブジェクト間通信に対して$O(N^2)$のプロセス間通信メッセージ(または$O(N)$のブロードキャストメッセージ)が必要になり、多くの通信バンド幅とプロセッサ時間が消費される。この冗長なメッセージ伝達を避けるために、送信イベントではprimary processのみがメッセージを送信し、backup processはメッセージを送信せずにイベント実行を終了する手法が考えられる。これによって、$O(N)$のプロセス間通信メッセージ($O(1)$のブロードキャストメッセージ)でオブジェクト間通信を実現することができる。しかし、primary processが故障した場合、primary processが故障前にどのイベントまで実行したのかをbackup processが知ることはできない。そのため、新しいprimay processはメッセージ送信を開始するべき送信イベントを正確に決定できず、送信済みのメッセージが重複送信されたり、メッセージ未送信の送信イベントを生じたりすることがある。従来のordered multicast protocolでは、プロセス故障の検出と送信イベント実行の順序が制御されていないために、この問題を解決できない。

パートナモデル分散システムにおけるpassive replicationによるフォールトトレランス手法として3-way multicast protocolが提案されている[2]。各送信イベントではprimary processが宛先プロセスグループに属する複製プロセスにメッセージを送信するとともにbackup processに状態情報を転送する。しかし、[2]の手法は、ブロードキャストネットワークを用いたatomic multicast [6]が下位レイヤ機能として提供されることが必要である。また、backup processによる状態情報の保持とprimary process故障時のロールバックが必要である。

本稿では、パートナモデル分散システムにおけるactive replicationによるフォールトトレランスを実現するために、ordered multicast protocolと3-way multicast protocolを拡張したgroup-to-group通信アルゴリズムを提案する。本アルゴリズムは、primary processの送信イベント実行とプロセス故障との因果関係に関する情報をプロセスグループの全複製プロセスが共有することを保証する。また、backup processがprimary processに先行して送信イベントを実行しないように制御し、故障したprimary processをbackup processが引き継ぐタイミングを適当な送信イベントまで遅延させることによってオブジェクト間通信メッセージ伝達の重複や欠落を回避し、オブジェクト動作の一貫性を保つ。

2 システムモデル

2.1 プロセスと通信チャネル

分散システムを構成するプロセスと通信チャネルに対して以下を仮定する。なお、各プロセスは図1に示すように3つのレイヤから構成されるものとする。

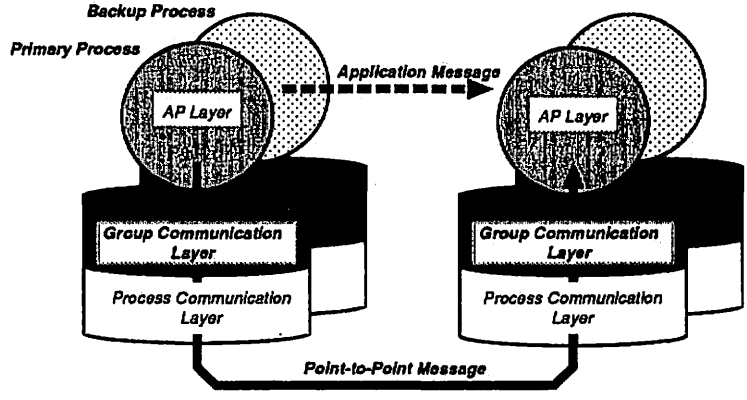

図 1: The Three Layers of Processes

決定的プロセス: プロセスはアプリケーションレイヤで分散アプリケーションプログラムを実行するが、この動作は決定的状態遷移機械でモデル化できると仮定する。アプリケーションの状態は、送信イベントとローカルイベントから成るイベント列を実行して遷移する。このイベント列は、受信イベント実行時点の状態とグループ通信レイヤから配付されるメッセージのみによって決まる。したがって、同一プロセスグループに属する複製プロセスが同一状態から実行を開始し、同一メッセージを同一順序でアプリケーションレイヤに配付するならば、同一メッセージを送信し、同一状態遷移を行なう。

クラッシュ故障: プロセスの故障形態はクラッシュ故障のみであると仮定する。故障プロセスは完全に停止し、他のプロセスに対して一切メッセージを送信しない。故障発生頻度については、あるプロセスグループと

他の同一プロセスグループとの連続するメッセージ交換の間に発生するプロセス故障数が複製プロセス数よりも小さいと仮定する。

タイムアウト: 分散システムにおけるプロセス間通信メッセージの伝達遅延は一定ではなく、確率的に分布しているが、ここではその上限値の存在を仮定する。通信チャネル内でメッセージが紛失することはあるが、継続的に故障することはなく、ある一定回数以上連続して紛失することはないと仮定する。したがって、ある時間間隔で一定回数以上メッセージ通信を試みることによって、タイムアウトを用いたプロセス故障の検出が可能である。

FIFO 通信チャネル: プロセス通信レイヤでは、送受信プロセス間のメッセージ通信機能が提供される。ここでは、ブロードキャストネットワークではなく、point-to-point の FIFO(first-in first-out) 通信チャネルから構成されるネットワークを仮定する。通信チャネル内でメッセージが紛失することはあるが、メッセージ複製やメッセージ内容の書き換えは起こらないと仮定する。

2.2 プロセス通信レイヤ機能

グループ通信レイヤの下位に位置するプロセス通信レイヤでは、point-to-point のプロセス間メッセージ通信機能とプロセス故障の検出機能を実現する。

プロセス故障は、'I'm alive' メッセージを互いに定期的に交換することによって検出可能である。各プロセスは、あるプロセスからこのメッセージが一定回数以上連続して受信できない場合、このプロセスは故障したものと判断する。タイムアウトの仮定から、この故障検出手法は正当であるといえる。しかし、実現に必要なメッセージ数が多い欠点がある。N 個のプロセスが互いの故障検出を可能にするためには $O(N^2)$ のメッセージが必要である。システムの全プロセスで 'I'm alive' メッセージを交換するならば、あるプロセス故障を一定時間経過後には全プロセスに正しく伝えることができる。しかし、N 個の複製プロセスを含む N_g 個のプロセスグループによって構成されるシステムでは $O({N_g}^2 \cdot N^2)$ のメッセージが必要である。提案手法では、'I'm alive' メッセージを同一プロセスグループの複製プロセス間のみで交換する。この手法では、他のプロセスグループで発生した故障に関する情報をただちには得ることができないために、プロセスグループの構成に関する情報を常に正確に保つことはできない。提案アルゴリズムは、この不正確な情報を基にしてプロセスグループ間メッセージが確実に宛先プロセスグループに属する全ての正常な複製プロセスに受信されることを保証する。このとき、プロセス故障検出のために必要な 'I'm alive' メッセージの数は $O(N_g \cdot N^2)$ であり、N_g が大きい大規模分散システムへの適用に対して有利である。

提案アルゴリズムでは、プロセスグループ間通信のためにプロセスグループ内で伝達される全制御メッセージを primary process が送信する。このメッセージ通信には acknowledgement protocol を用いる。primary process は、acknowledgement を backup process から受信するまでメッセージを繰り返し送信する。これによって primary process は、acknowledgement によって backup process がメッセージを受信したことを確認するか、タイムアウトによって backup process の故障を検出するかのいずれかに必ず至る。結果的に、全ての正常な backup process がメッセージを受信する。一方、プロセスグループ間通信に対しては、送信プロセスが宛先プロセスグループの構成を正確に知ることができないため、いわゆる acknowledgement protocol を適用することができない。プロセスグループ間通信メッセージについては、プロセス通信レイヤでは特別な処理を行なわずにそのまま送信する。

プロセス通信レイヤはプロセス故障を検出すると、故障プロセスの ID を含む故障通知メッセージをグループ通信レイヤに配付する。プロセスグループ内通信処理の途中で primary process が故障すると、メッセージが一部の backup process のみに受信されている状態になる。複製プロセスが同一の状態遷移を行なうこと保証するためには、メッセージが全ての正常な backup process に受信されているか、あるいは全く受信されていないかのいずれかを実現しなければならない (atomic multicast[6])。そこで、primary process の故障を検出した backup process は、故障通知メッセージをグループ通信レイヤに配付する前に、故障した primary process から最後に受信したメッセージを新しい primary process へ送信する。primary process は最新の受信メッセージを送り返す。backup process は、送信したものより新しいメッセージを受信した場合にのみ、これをグループ通信レイヤへ配付する。このメッセージ交換についても backup process が primary process からメッセージを受信するまで繰り返し送信することでメッセージ紛失やプロセス故障に対処できる。

本節で述べたプロセス通信レイヤの機能を用いることによって、グループ通信レイヤでは以下を仮定することが可能である。

- あるプロセスグループ内のプロセス故障は、それに属する全ての正常な複製プロセスに通知される。
- プロセスグループ内通信メッセージは、primary process から全 backup process へ atomic に伝達される。

3 グループ通信アルゴリズム

3.1 要求条件

パートナモデル分散システムにおいて active replication によるフォールトトレランスを実現するためには、同一プロセスグループに属する全複製プロセスが同一状態遷移を行ない、個々の複製プロセスの故障発生はプロセスグループ内に隠蔽され、プロセスグループ間のメッセージ通信に一貫性が保たれていなければならない。グループ通信レイヤで機能する提案アルゴリズムは、以下の条件を満たすことが要求される。

- 同一プロセスグループに属する全ての正常な複製プロセスは、同一メッセージを同一順序でアプリケーションレイヤに配付する。
- アプリケーションレイヤの送信イベントに対応するプロセスグループ間通信メッセージの送信処理が欠落することがない。このメッセージは宛先プロセスグループの全ての正常な複製プロセスに受信される。
- アプリケーションレイヤの送信イベントに対応するプロセスグループ間通信メッセージの送信処理が重複して実行されることがない。宛先プロセスグループに属する複製プロセスでは、同一メッセージが重複してアプリケーションレイヤに配付されることがない。

3.2 準備

提案アルゴリズムでは、以下に挙げる 6 タイプの制御メッセージを用いる。

'Multicast', 'Forward': 送信プロセスグループの ID とメッセージシーケンスナンバ (MN) から構成されるメッセージ ID(MID) とアプリケーションレイヤで伝達されるメッセージ内容を含む。

'Ack', 'Complete': MID を含む。

'Fail', 'Recover': 故障あるいは回復したプロセスの ID を含む。

プロセス p は、グループ通信レイヤで以下の情報を保持する。

S-queue$_p$: primary process から受信した送信イベント実行済み通知 ('Complete') とプロセス回復通知 ('Recover')、プロセス通信レイヤの故障検出機能によって配付されるプロセス故障通知 ('Fail') を順序付けるための FIFO メッセージキュー。初期値は $\emptyset$。

R-queue$_p$: プロセスグループ間通信メッセージを同一順序でアプリケーションレイヤに配付するために primary process から受信した転送メッセージ ('Forward') を順序付けるための FIFO メッセージキュー。初期値は $\emptyset$。アプリケーションレイヤの受信イベントでは、このキューからメッセージを取り出す。

P-buffer$_p$: primary process の故障による不必要な再送信を回避するために、受信したグループ通信メッセージ ('Multicast') を保持するためのメッセージバッファ。初期値は $\emptyset$。

View$_p(G_i)$ $\forall G_i$: プロセスグループ G_i に属する複製プロセスの ID のリストを G_i のグループビューと呼び、プロセス p の持つ G_i のグループビューを View$_p(G_i)$ で表す。また、G_i の現在の正確なグループビューを Trueview(G_i) で表し、View$_p(G_i)$ の初期値とする。$p \in G_i$ が G_i にグループ通信メッセージを送信する場合、View$_p(G_i)$ に ID が含まれる全プロセスに送信する。また、p はプロセス ID が View$_p(G_i)$ の最上位にある場合は primary process として、それ以外の場合は backup process として動作する。

MI$_p(G_i)$ $\forall G_i$: $p \in G_i$ ならば G_i から送信されたグループ通信メッセージ数を、$p \notin G_i$ ならば G_i から受信したメッセージの最大シーケンスナンバの値を、それぞれ表すメッセージインジケータ。初期値は 0。

プロセス $p \in G$ は、'Multicast' メッセージと 'Ack' メッセージに View$_p(G)$ を付与して送信する。このメッセージを受信したプロセス $q \notin G$ は、保持していた View$_q(G)$ をメッセージに付与された View$_p(G)$ に更新することによって、送信プロセスグループ G の構成変更に関する情報を得る。

3.3 グループ通信アルゴリズム

3.1で述べた要求条件を満たすグループ通信を実現するためには、複製プロセスが送信イベントに関して以下のように動作すればよい。

送信イベント実行の非先行制御: backup process は primary process に先行して送信イベントを実行しない。

primary process の引き継ぎ遅延制御: 故障した primary process が最後に実行した送信イベントまでは、全複製プロセスが backup process として動作する。

非先行制御によって、primary process がある送信イベントの実行前に故障し、故障通知が backup process のグループ通信レイヤに配付された時点では、全 backup process がその送信イベントを実行していないことが保証できる。したがって backup process をロールバックさせることなく (チェックポイント設定不要) この送信

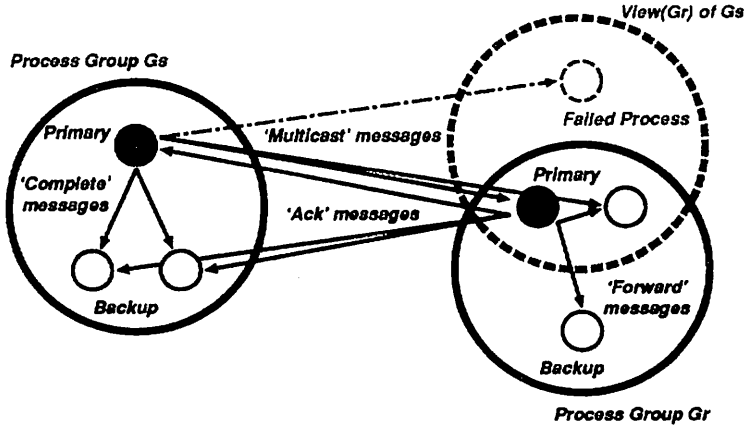

図 2: Overview of the Algorithm

イベントのメッセージ送信処理の欠落が避けられる。アプリケーションレイヤからの送信イベント実行要求を処理するアルゴリズムでは、送信プロセスグループの primary process がプロセスグループ間通信メッセージ ('Multicast') を送信し、受信プロセスグループから acknowledgement を受信した時点で、プロセスグループ間通信メッセージが送信済みであることを通知するメッセージ ('Complete') を backup process に送信する。これを受信するまで backup process がこの送信イベントで待機し、以降のイベントを実行しないことによって非先行制御が実現できる。さらに backup process がこの通知メッセージによって対応するイベントのメッセージ送信が不要であると判断し、実際に送信せずにこの送信イベントを終了することで、重複送信による通信バンド幅の消費を避けることができる。

引き継ぎ遅延制御によって、primary process が故障した場合でも送信済みのプロセスグループ間通信メッセージを重複送信することによるオーバヘッドを避けることができる。backup process は primary process から受信したプロセスグループ間通信メッセージ送信済み通知 ('Complete') とプロセス故障検出通知 ('Fail') を同一メッセージキュー (S-queue) を用いて順序付ける。これによって primary process の引き継ぎタイミングは適当な送信イベントまで遅延され、各プロセスは送信イベントの実行と複製プロセスの故障との因果関係を正しく管理することができる。

プロセスグループ間通信メッセージの紛失は、プロセスグループ間の acknowledgement protocol によって対処する。送信プロセスグループの primary process は、受信プロセスグループから acknowledgement('Ack') を受信するまでメッセージ ('Multicast') を繰り返し送信する。これらのメッセージをプロセス ID が宛先プロセスグループのグループビューに含まれる全複製プロセスに送信することによって、不正確なビューに基づく通信が可能になる。また、繰り返し送信されるメッセージに同一メッセージ ID を付与することでアプリケーションレイヤへの重複配付を避けることができる。

受信プロセスグループに属する全ての正常な複製プロセスがプロセスグループ間通信メッセージを同一順序でアプリケーションレイヤに配付するために、受信順序は primary process が決定して backup process に通知する。primary process は受信したプロセスグループ間通信メッセージ ('Multicast') をただちにメッセージキュー (R-queue) に登録してアプリケーションレイヤへ配付可能とするとともに、このメッセージを backup process に転送する ('Forward')。backup process は受信したプロセスグループ間通信メッセージ ('Multicast') をメッセージバッファ (P-buffer) に一時的に保持し、対応する転送メッセージを primary process から受信した時点でこれを取り出し、アプリケーションレイヤへ配付可能にする。プロセスグループ内通信メッセージはアトミックに伝達されることが仮定されているので、primary process が故障した場合も backup process が同一の転送メッセージを異なるプロセスから受信することはなく、受信順序の一意性が保証される。また、プロセスグループ間通信メッセージを backup process がメッセージバッファに保持する機構によって、メッセージ転送前に primary process が故障した場合でも、このメッセージを再送信する必要がない。

送信プロセスグループ G_s から受信プロセスグループ G_r へアプリケーションメッセージ M を伝達する場合、上述の機能を実現するグループ通信アルゴリズムは次の 4 つのフェーズで構成される (図 2)。

Multicast フェーズ: primary process $p_s \in G_s$ が M を含む 'Multicast' メッセージを G_r に送信する。G_r から 'Ack' メッセージを受信せずにタイムアウトした場合は再送信する。

Forward フェーズ: primary process $p_r \in G_r$ は受信した 'Multicast' メッセージを R-queue に登録し、M を含む 'Forward' メッセージを backup process に送信する。backup process は受信した 'Forward' メッセージを R-queue に登録する。

Acknowledge フェーズ: p_r が 'Ack' メッセージを G_s に送信する。

Complete フェーズ: p_s は 'Ack' メッセージを受信すると backup process に 'Complete' メッセージを送信する。backup process は受信した 'Complete' メッセージを S-queue に登録する。対応する送信イベントでは 'Complete' メッセージを S-queue から取り出すことによって 'Multicast' メッセージが送信済みであることを確認し、'Multicast' メッセージを送信をせずにこのイベントの処理を終了する。

アプリケーションレイヤから送受信イベントの処理を要求された場合にグループ通信レイヤで実行されるタスクを図3に示す。また、プロセスグループ間通信を実現するために伝達される制御メッセージがプロセス通信レイヤから配付された場合に実行されるタスクを図4に示す。

Send Event
  if p_s is the primary process then
    {*Multicast Phase:*}
      increment $MI_{p_s}(G_s)$.
      repeat sending 'Multicast' messages, MN(Multicast):=$MI_{p_s}(G_s)$, with application message M to the processes in $View_{p_s}(G_r)$ at intervals of timeout threshold till receiving 'Ack' message, MN(Ack)=$MI_{p_s}(G_s)$.
    {*Complete Phase:*}
      send 'Complete' messages, MID(Complete):=MID(Ack), to the backup processes in G_s.
  else
    if S-$queue_{p_s} = \emptyset$ then suspend the task till a message is enqueued.
    dequeue a message from S-$queue_{p_s}$.
    if the message is a 'Fail' message then
      execute the task '*Renewing View by Failure*' and '*Send Event*' from the beginning.
    else if the message is a 'Recover' message then
      execute the task '*Renewing View by Recovery*' and '*Send Event*' from the beginning.
    fi
  fi
Receive Event
  if R-$queue_{p_r} = \emptyset$ then suspend the task till a message is enqueued.
  dequeue a message from R-$queue_{p_r}$ and deliver it to its application.

図 3: The Algorithms for Event Execution.

Reception (Multicast)
  if p_r is the primary process then
    {*Forward Phase:*}
      if MN(Multicast)>$MI_{p_r}(G_s)$ then
        enqueue the message to R-$queue_{p_r}$ and set $MI_{p_r}(G_s)$:=MN(Multicast).
        send 'Forward' messages, MID(Forward):=MID(Multicast), with the application message in the 'Multicast' message to the backup processes in G_r.
      fi
    {*Acknowledge Phase:*}
      send 'Ack' messages, MID(Ack):=MID(Multicast), to the processes in $View_{p_r}(G_s)$.
  else
    if MN(Multicast)>$MI_{p_r}(G_s)$ then append the message to P-$buffer_{p_r}$ and set $MI_{p_r}(G_s)$:=MN(Multicast).
  fi
Reception (Forward)
  enqueue the message to R-$queue_{p_r}$ and set $MI_{p_r}(G_s)$:=max($MI_{p_r}(G_s)$,MN(Forward)).
  delete the 'Multicast' message received from G_s and MN(Multicast)≤MN(Forward) from P-$buffer_{p_r}$.
Reception (Ack)
  if MN(Ack)>$MI_{p_s}(G_s)$ then append the message to P-$buffer_{p_s}$ and set $MI_{p_s}(G_s)$:=MN(Ack).
Reception (Complete)
  enqueue the message to S-$queue_{p_s}$ and set $MI_{p_s}(G_s)$:=max($MI_{p_s}(G_s)$,MN(Complete)).
  remove the 'Ack' messages, MN(Ack)<MN(Complete), from P-$buffer_{p_s}$.

図 4: The Algorithms for Reception of Messages

3.4 グループ構成管理アルゴリズム

プロセスグループの構成に関する情報はグループビューとしてプロセスごとに保持される。

各プロセスは'I'm alive'メッセージのタイムアウトによって同一プロセスグループ内のプロセス故障を検出することができ、'Fail'メッセージによってプロセス通信レイヤからグループ通信レイヤに通知する。プロセス故障と送信イベント実行の因果関係を正しく保つためには、この通知によって得られる故障情報をただちにグループビューに反映させるのではなく、primary processから受信したプロセスグループ間通信メッセージの送信済み通知('Complete')と同一のメッセージキュー(S-queue)を用いて順序付ければよい。これによって、プロセスグループ間通信メッセージ送信の重複と欠落を回避することができる。プロセス通信レイヤから'Fail'メッセージが配付されたプロセス$p \in G$のグループ通信レイヤは以下の処理を行なう。

- pがprimary processならば、故障プロセスIDを$\mathrm{View}_p(G)$からただちに削除する。
- pがbackup processならば、'Fail'メッセージを$\mathrm{S\text{-}queue}_p$に登録する。送信イベントでこの'Fail'メッセージを取り出した時点で、故障プロセスIDを$\mathrm{View}_p(G)$から削除する。

新たに複製プロセスを生成してプロセスグループに加える場合には、primary processがbackup processに構成変更を通知し、backup processはプロセス故障の場合と同様のグループビュー更新処理を行なう。

以上の処理を行なうグループ通信レイヤのタスクを図5に示す。

Reception (Fail)
  if p is the primary process in G then
    eliminate the process ID of the failed process from $\mathrm{View}_p(G)$.
  else
    enqueue the message to $\mathrm{S\text{-}queue}_p$.
  fi
Renewing View by Failure
{*On dequeuing a 'Fail' message from $\mathrm{S\text{-}queue}_p$ in the task 'Send Event', $p \in G$ executes this task.*}
  eliminate the process ID of the failed process from $\mathrm{View}_p(G)$.
  if p becomes the primary process by this elimination then
    while $\mathrm{P\text{-}buffer}_p \neq \emptyset$ do
      {*Forward Phase:*}
        remove a 'Multicast' message from $\mathrm{P\text{-}buffer}_p$ and enqueue it to $\mathrm{R\text{-}queue}_p$.
        send 'Forward' messages, MID(Forward):=MID(Multicast), with the application message in the 'Multicast' message to the backup processes in G.
      {*Acknowledge Phase:*}
        send 'Ack' messages, MID(Ack):=MID(Multicast), to the processes in $\mathrm{View}_p(G_s)$ where G_s is the sender process group of the 'Multicast' message.
    od
  fi
Recovery
{*On recovering a backup process, the primary process $p \in G$ executes this task.*}
  append the process ID of the recovered process, PID_{p_r}, at the end of $\mathrm{View}_p(G)$.
  send 'Recover' message with the state information to p_r and 'Recover' messages with PID_{p_r} to the other backup processes in G.
Reception (Recover)
  if p is the recovered process then
    take out the state information from the message.
  else
    enqueues the message to $\mathrm{S\text{-}queue}_p$.
  fi
Renewing View by Recovery
{*On dequeuing a 'Recover' message from $\mathrm{S\text{-}queue}_p$ in the task 'Send Event', $p \in G$ executes this task.*}
  append the process ID of the recovered process at the end of $\mathrm{View}_p(G)$.

図 5: The Algorithms for Group Management.

故障検出機構をプロセスグループ単位で適用するために、各プロセスは他のプロセスグループにおける構成変更の情報をただちにはグループビューに反映させることはできない。このため、プロセス$p \notin G$のグループビュー$\mathrm{View}_p(G)$はTrueview(G)と異なる場合がある。ここでは、グループビューに含まれるプロセスIDに順位を付け、最上位IDを持つプロセスがprimary processとして動作し、回復プロセスIDは最下位に登録する。この方法では$\mathrm{View}_p(G) \cap \mathrm{Trueview}(G)$にprimary processが必ず含まれる。さらに、primary processが受信したプロセスグループ間通信メッセージを転送する機構によって、送信プロセスが持つグループビューが不正確である場合でも、受信プロセスグループの全ての正常プロセスがメッセージを受信することが可能である。

4 評価

提案アルゴリズムのオーバヘッドを、N 個の複製プロセスを含む N_g 個のオブジェクトで構成された分散システムに適用した場合に必要な制御メッセージ数によって評価する。表1は、サーバクライアントモデル分散システムでフォールトトレランスを実現する代表的なシステムである ISIS が用いる ordered multicast protocol との比較を示したものである。アプリケーションメッセージの伝達に必要なグループ通信レイヤの制御メッセージ数は、サーバクライアントモデルでは提案アルゴリズムと従来手法は同等である。一方、$O(N^2)$ のメッセージが必要であるため ordered broadcast protocol の単純な適用は困難であるパートナモデルでは、提案手法によってグループ間通信を $O(N)$ のメッセージで実現することができる。また、プロセス故障の検出通知機構をプロセスグループ単位で適用することによって、'I'm alive' メッセージの数を $O({N_g}^2)$ から $O(N_g)$ に、故障発生時の処理に必要なメッセージ数を $O(N_g \cdot N)$ から $O(N)$ に減少させることができる。

表 1: Overhead (Number of Messages)

	Communication between Objects		Notification of Failure
	Group-to-Group (Partner)	Point-to-Group (Server-Client)	
proposed algorithm	$6N-4$	$3N-1$	$2N$
ISIS system	$3N^2$	$3N$	$3N_gN$

さらに、提案アルゴリズムは以下の特徴を持つ。

- チェックポイント設定が不要であるため、アプリケーションプログラムに対してフォールトトレランスが透過である。
- プロセスが故障した場合にロールバックリカバリが不要であるために処理中断時間が短い。
- プロセスグループ内では送信イベントにおける primary process の先行性を保てば十分であり、複製プロセスの緊密な同期は不要である。

5 おわりに

本稿では、サーバクライアントモデル分散システムを対象とした従来の point-to-group 通信では困難であるピアートナモデル分散システムにおける active replication によるフォールトトレランスの実現のために、group-to-group 通信アルゴリズムを提案した。提案方式は、複製プロセス数の線形オーダの制御メッセージの伝達によってプロセスグループ間通信を実現する。また、複製プロセスの故障情報をプロセスグループに隠蔽することによってオーバヘッドを削減しているが、複製プロセスが故障した場合でもプロセスグループ間メッセージ通信の一貫性を維持することができる。

参考文献

[1] M.Ahamad, P.Dasgupta, R.LeBlanc, and C.T.Wilkes, "Fault tolerant computing in object based distributed operating systems," Proc. of 6th Symp. on Reliable Distributed Systems, pp.115–125 (1987).

[2] O.Babaogle, "Fault-Tolerant Computing Based on Mach," ACM Operating System Review, 24(1), pp.502–508 (1985).

[3] K.Birman and T.Joseph, "Reliable communications in presence of failures," ACM Trans. on Comp. Syst. 5(1), pp.47–76 (1987).

[4] A.Borg, J.Baumbach, and S.Glazer, "A message system supporting fault tolerance," Proc. of 9th ACM Symp. on OS Principles, pp.90–99 (1983).

[5] E.C.Cooper, "Replicated distributed program," Proc. of 10th ACM Symp. on OS Principles, pp.63–78 (1985).

[6] F.Cristian, H.Aghili, and R.Strong, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," Proc. of 15th FTCS, pp.200–206 (1985).

[7] D.Powell, M.Chereque, and D.Drackley, "Fault-Tolerance in Delta-4," ACM Operating System Review, 25(2), pp.122–125 (1991).

[8] 吉田紀彦, "次世代並列分散システム開発のために," コンピュータ科学, Vol.2, No.4, pp.300–305(1992).