分散計算における全順序保証プロトコルの適応的選択

安武芳紘[†1] 小田謙太郎[†2] 吉田隆一[†2]

分散環境では複数のプロセスが通信をするため，メッセージの全順序保証を利用した通信は不可欠である．また，さまざまな実行環境が存在し，環境はたえず変化するため，最適な全順序保証プロトコルをあらかじめ選択することは困難である．そこで，環境変化へ動的に適応する全順序保証プロトコルが求められる．従来の適応的全順序保証プロトコルは悲観的全順序保証プロトコルを基盤にしている．そこで本論文では悲観的手法に加えて楽観的手法も対象とした適応方法を提案する．適応方法を楽観的手法にまで広げたことにより，異なる手法のプロトコルを動的に選択し環境変化に適応することが可能である．2 つの手法のコストを比較した場合，楽観的手法はメッセージの送信頻度に影響を受けやすく，悲観的手法は通信遅延の影響を受けやすい．適応的選択は，時々刻々変化する実行環境に対して，順序付けコストが最小の手法を選択することになり，これを実現するために，それぞれの手法に対して順序付けコストを評価し比較する方法を提案する．また，メッセージの全順序保証を維持するため，楽観的手法を考慮したプロトコルの切替え方法を提案する．例として Time Warp と ABCAST を対象としたコスト評価・比較，切替えについて具体的に述べ，本方式の有効性をシミュレーションにより検証する．

Adaptive Selection of Total Ordering Protocols in Distributed Computation

Yoshihiro Yasutake,[†1] Kentaro Oda[†2] and Takaichi Yoshida[†2]

In distributed systems, the communication among group members often needs ordered messages to guarantee that every member receives the messages in the same order. Oftentimes, changes in distributed computing environment occur and this undermines the assumption of any ordering algorithm. Therefore, it is difficult to presume a suitable ordering algorithm and apply a single algorithm throughout the lifetime of the system. In this paper, we present a protocol to dynamically select a suitable ordering protocol from the optimistic and pessimistic protocols. With optimistic protocol, it is possible to reduce the affect of the network latency which is not negligible in the pessimistic protocol, because the optimistic protocol delivers messages immediately after receiving them. The optimistic protocol though incurs high ordering cost during rollback, and so in this case, it is worth using the pessimistic algorithm instead. We propose the estimation, evaluation, and switching methods to be used in selecting the appropriate ordering protocol. To verify the validity of our proposed protocol, we simulate the protocol using Time Warp and ABCAST.

1. はじめに

分散計算においてメッセージのマルチキャスト機能は重要な技術である．たとえば，耐故障性を向上させるためにオブジェクトのレプリケーションが用いられる場合，メッセージのマルチキャスト機能を用いることですべてのレプリカに同一のメッセージを効率良く送信できる．また複数のオブジェクトに対するイベントを通知する場合，メッセージのマルチキャスト機能を用いることで容易に実現することができる．しかし，マルチキャストを用いて複数のメッセージが送受信され，各メンバが互いに異なる順序でメッセージを処理してしまうとメンバ間で一貫性が保たれなくなってしまう．

この問題を解決する手法としてメッセージの全順序保証プロトコルをあげることができる．全順序保証プロトコル[1),2)] はいくつか提案されているが，それぞれのプロトコルは実行環境やアプリケーションの特徴によって性能が異なる．分散環境ではシステムの利用状況などによって環境がたえず変化するため，その変化

†1 九州産業大学情報科学部
Faculty of Information Science, Kyushu Sangyo University
†2 九州工業大学大学院情報工学研究科
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology

をあらかじめ正確に把握し，単一のプロトコルを選択することは困難である．そこでプロトコル自体が環境に適応する適応的プロトコルが提案されている[3)–6)]．従来提案されてきた適応的プロトコルは処理順序が確定してからメッセージを処理する悲観的順序付け手法を基盤とし，局所的な性能低下が全体に及ぶことを防ぐことに主眼を置いている．

本論文では分散トランザクション処理に注目し，さまざまなアプリケーション，分散して存在するデータへのアクセスを想定する．アプリケーションによって処理の粒度が異なり，また分散されたデータへのアクセスは，ネットワーク遅延や局所的な環境変化をもたらす．そこで，条件によっては効率の良い楽観的順序付け手法にまで適応の範囲を広げ，最適なプロトコルを目指して，楽観的手法と悲観的手法を動的に選択し環境の変化に適応する方法を提案する．楽観的手法はメッセージを受信後すぐにそのメッセージを処理し，メッセージの処理順序が間違っていたときにやり直しを行う．順序付けのためにメンバ間で情報交換をする悲観的手法に比べ，楽観的手法ではその必要性がないためメンバ間で発生する遅延の差の影響を受けないという利点がある．しかし，楽観的手法はメッセージの送受信の順序が逆転する場合にロールバックを起こすため，送信順序と受信順序が大きく異なる場合には悲観的手法を利用することが望ましい．

異なる 2 つの手法を動的に選択するためには，まず初めにそれぞれの手法によるプロトコルの処理コストを観測可能な環境情報をパラメータにより見積もり，比較することが可能でなければならない．さらに，その見積り結果は比較可能な形式で得る必要がある．次に，手法の異なるプロトコルの切替えが全順序保証を損なうことなく行われることが必要である．悲観的手法におけるプロトコルの切替えは Hybrid Protocol [4)] によって示されているが，楽観的手法を含んだ切替えを行う場合はやり直しが起きる可能性があることを考慮しなければならない．これらの問題に対する方法を 3 章で提案し，例として全順序保証プロトコルのうち，楽観的手法として Time Warp [1)]，悲観的手法として ABCAST [2)] をあげ，それらを動的に選択し切り替えることによる環境への適応をシミュレーションにより検証する．

2 章で全順序保証プロトコルについて述べ，3 章で全順序保証プロトコルを動的に切り替えるプロトコルを提案する．4 章で Time Warp と ABCAST を対象とした具体例を示し，5 章でシミュレーションによる評価を行う．

2. 全順序保証プロトコル

メッセージの全順序保証を実現するには，すべてのメッセージがグループ内で一意の順序で配送される必要がある．その方法にはトークンやシーケンスサーバを利用する方式やメッセージの順序が確定するまで配送を遅らせるシンメトリック方式などがある[7)]．

2.1 悲観的手法と楽観的手法

トークンやシーケンスサーバ方式はあらかじめメッセージの順序を決定する．また，シンメトリック方式ではメッセージの正確な処理順序が決定するまで配送を待つ．よってこれらは悲観的手法に分類することができる．メッセージの処理のやり直しが起きない一方，各メッセージの正確な順序を求めるため次に述べる楽観的手法に比べメッセージ 1 つあたりの処理に多くの時間を要する．

楽観的手法は Time Warp に代表される順序付け手法である．メッセージは受信後ただちに配送される．その後，配送されたメッセージよりも小さいタイムスタンプのメッセージを受信した場合，正しい順序で処理されるようにやり直しが行われる．受信した順序と処理すべき順序が同じ場合は，やり直しがないため効率の良いメッセージ配送が可能になる．しかし，受信した順序に対し配送すべき順序にばらつきがある場合，やり直しを行う確率が高くなる．このやり直される処理が他のプロセスに対するメッセージを送信した場合，そのメッセージを処理したプロセスもやり直さなければならなくなる．

2.2 適応的全順序保証プロトコル

トークン，シーケンスサーバ方式では順序付けを 1 点で行うため，メンバ数の増加やメッセージ数の増加によって順序付けの効率が低下することから，拡張性に乏しいといえる．また，シーケンスサーバ方式では各メンバの情報から順序を保証するため，一部のメンバの性能低下が全体の性能低下を招くことになる．これらの問題を解決する方法として，適応的全順序保証プロトコルが提案されている．

ATOP（Adaptive Totally Ordered Multicast Protocol）[3)] は各メンバのメッセージの送信頻度の変化に応じて分布と呼ばれる受信スケジュールを動的に変更し，送信頻度の変化に適応する．また，ATOP の拡張[6)] ではドメイン内でシーケンスサーバを利用することにより，ネットワーク構成に応じて ATOP とシーケンス方式を統合した順序付け方式を利用する．その結果，シーケンスサーバへの負荷の軽減とメッセージ数の削減を実現している．Hybrid Protocol [4)] は各プロ

セスをメッセージの送信頻度とネットワーク遅延からメンバを active node と passive node に分類し，ノードごとにトークン方式とシンメトリック方式を選択して用いることができる．個々のノードに適したプロトコルを利用することにより，グループ全体の情報交換が円滑に行われる．ToTo[5] は過半数以上のメンバからの応答を得ることによって配送が可能であり，すべてのメンバからの情報を待つ必要がない．実行環境に適した応答数を設定することにより，通信遅延の大きいメンバとの情報交換の遅れがグループ全体に影響することを防ぐことができる．

これら従来の適応的全順序プロトコルは配送前に順序を決定する手法，つまり悲観的手法を利用し，その欠点である低い拡張性や局所的な性能低下がグループ全体の性能を低下させることを改善する手法であると考えることができる．

2.3 適応的選択

我々は対象とする全順序保証プロトコルを楽観的手法の利用にまで広げ，それぞれの性質の異なる順序付け手法の特徴を活かすことにより実行環境への適応範囲を広げることを提案する．悲観的手法はグループ内のメンバの情報を用いるため，情報を集約するサーバが存在するか各メンバが情報を集める必要がある．この手法の問題点は，情報を集約するサーバの負荷が高くなった場合やあるメンバの性能が低下した場合，局所的なネットワーク遅延が発生した場合にグループ全体の性能低下が起こることである．一方，楽観的手法はメッセージを受信後すぐにそのメッセージを処理するためメンバ間で通信する必要がなく，他のメンバやネットワーク遅延の影響を受け難いという利点がある．よって，楽観的手法を用いることにより局所的な性能低下がグループ全体に及ぶことを防ぐことができる．

ただし，楽観的手法は配送前に確認を行わないため，やり直しが起こることがある．メッセージがタイムスタンプの順序どおりに受信する場合が最も理想的な状況であるが，1 度でもタイムスタンプの小さなメッセージを遅れて受信するとやり直しが必要となる．メッセージ送信頻度が高いマルチキャストは，送受信期間が重なる可能性が高いため，受信者間で到着順序が異なるメッセージ数は多くなる．すなわち，メッセージの送信頻度が高いときに，メッセージの遅延の変動が大きいとメッセージが逆転する状況が発生する．このような場合に，楽観的手法は不利である．

適切なプロトコルを選択するには，環境変化に応じてそれぞれのプロトコル処理のコストがどのように変化しているかを比較可能な形式で知る必要がある．全順序保証プロトコルのコスト評価はさまざまな目的で行わるが，複数のプロトコルを比較する従来の評価方法は悲観的手法を用いたプロトコルを対象としており，楽観的手法は含まれていない．そこで我々は手法の違いからは独立したプロトコル処理に要するコストを求めるために，メッセージの受信から配送までの時間を評価に用いることにした．

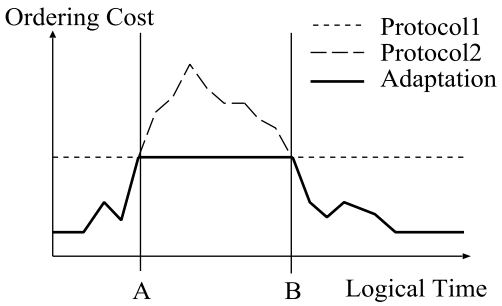

図 1 順序付けコスト
Fig. 1 Ordering cost.

順序付けプロトコル処理のコストは，実行環境やアプリケーションの性質によって変化する．たとえば，メッセージの遅延はトラフィックの変動により変化し，トランザクションの数が昼夜で変化するなど，実行環境は時刻とともに変化することがある．

例として，時間帯によりシステムにアクセスするユーザ数や処理の種類が変化するシステムを考える．ネットワーク帯域には余裕があり，通信遅延はほとんど変化しないと想定したとき，ユーザ数や処理の種類の変化により，各プロセスのメッセージの送信頻度は時間帯により変化することになる．**図 1** は時間の経過に従ってメッセージの送信頻度が変化した場合の悲観的手法を用いた全順序保証プロトコルと楽観的手法を用いた全順序保証プロトコル，そしてプロトコルの適応的選択のコストである．初期の環境（A までの期間）はメッセージが少なく，メッセージの到着間隔は長いとする．この期間では多くのメッセージが送信された順に受信されるため，やり直しが発生する確率が低いため楽観的手法（図 1 の Protocol2）を利用することが有効である．その後，利用者の増加や処理の変化によってメッセージ数が増加し，やり直しが発生する確率が高くなった場合（A から B の期間）は，悲観的手法（Protocol1）が有効となる．その後は再び初期の環境と同じ状態になり，楽観的手法（Protocol2）のコストが低下している．このように楽観的手法における順序制御のコストと悲観的手法のコストを比較し，コストの低い手法を選択する（Adaptation）ことにより，システムのライフタイムを通じて一定の順序付け

コストを維持することができる．また，ネットワーク内のメッセージ数が変化し遅延時間が変化する場合には，悲観的手法のコストが変化するので，状況はより複雑になり適応的順序付けプロトコルの必要性はますます高まる．

3. 適応的選択プロトコル

環境の変化に応じて複数の全順序保証プロトコルから最適なプロトコルを選択し利用する．そのためにはそれぞれの全順序保証プロトコルの評価を行い，比較することによって最適なプロトコルを選択する手段が必要である．そして次に，全順序保証プロトコルの切替えはメッセージの順序を損なうことなく，かつ切替えの処理や待ち時間が本来の処理に与える影響をできるだけ抑えた方法である必要がある．本章ではこれらに求められる要件をあげ，我々が提案する手法について述べる．

3.1 概　　要

提案する適応的選択プロトコルはグループメンバがネットワーク上に分散して存在し，それぞれが異なる環境で実行されることを想定している．そこでそれぞれのメンバが定期的に全順序保証プロトコルのコストを見積もり，それらを比較し，各自にとって最適なプロトコルを選択する．その後，グループ内で合意をとりグループで 1 つのプロトコルを選択し，現在のプロトコルと異なる場合はそのプロトコルに切り替える．今回は非同期メッセージ交換を利用し，グループメンバの変更はないものとする．適応的選択のプロトコル概要は以下のような手順となる．

(1) コストの見積り：対象とする全順序保証プロトコルのコストをそれぞれ見積もる．すべてのプロトコルを実際に適用することはできないため，メッセージの送信頻度やネットワーク遅延などの観測結果から順序付けコストを見積もる．
(2) コストの比較：プロトコルのコストを比較し，最もコストが小さいものを選択する．頻繁にプロトコルが切り替わることを防ぐため，小さなコスト差ではプロトコルの切替えが起こらないようにする．
(3) メンバ間の合意：各メンバが次のプロトコルを投票しグループで一意のプロトコルを決定する．プロトコル変更が決定した場合，プロトコル切替えの準備，そして切替えを行う．
(4) プロトコル切替えの準備：プロトコル変更後においても一貫してメッセージの全順序は保証されなければならない．そこで全順序を損なうことなくある論理時刻において切替え可能な状態にする．
(5) プロトコルの変更：ある論理時刻から次のプロトコルを利用した順序付けを開始する．

以下に各手順について説明を行う．

3.2 コストの見積り

複数の順序付けプロトコルから実行環境に適した手法を選択するには，楽観的手法と悲観的手法のそれぞれの全順序保証プロトコルのコストを見積もり，比較する必要がある．

全順序保証プロトコルの性能評価に関する研究では，CPU 時間と通信遅延時間，メッセージの数からコストを見積もる Contention-Aware Mertics[8] などがある．これらの方式は悲観的手法を対象としているため，メッセージの順序決定処理が配送時までに行われていなければならない．悲観的手法はメッセージの順序が配送前に確定するため順序決定の処理からプロトコルを評価することができるが，楽観的手法は順序決定プロセスが存在せずやり直しを行うことができるため同様の評価方法を適用することができない．このように楽観的手法では処理のやり直しのために配送時に順序が決定しているとはいえず，この評価方法は不向きである．

我々は分散トランザクションシステムを想定しているため，分散環境においてさまざまなネットワーク遅延が発生し，それぞれのアプリケーションによりトランザクション処理の粒度が異なる．これらの要素を考慮したプロトコルの評価方法が必要である．また，本研究は複数プロトコルを比較することが必要不可欠である．性質の似たプロトコルであれば複数の指標，たとえば上で紹介した見積り方法のように CPU 時間とメモリ量からシステムに与える影響を比較することができるが，楽観的手法と悲観的手法のように性質がまったく異なるプロトコルの場合はそれらの指標で比較することは困難である．

そこでやり直しが発生することを考慮した全順序保証プロトコルのコストとして，受信者がメッセージを受信してから配送するまでの実時間を採用する．対象とするプロトコルとして順序決定の処理がメッセージの受信者側で個々に行われるプロトコルに注目する．これらは Defago による全順序アルゴリズムの分類[7] では Communication History と Destination Agreement に相当するプロトコルである．メッセージにタイムスタンプが付加される時期はプロトコルによって送信時や受信時などであるが，これらのプロトコルに共通する特徴は付加されたタイムスタンプを使いメッ

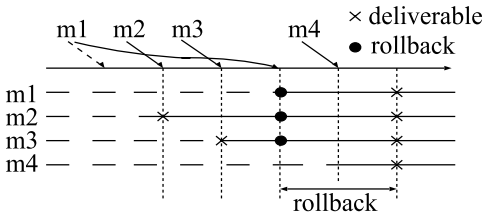

図 2 楽観的プロトコルのコスト
Fig. 2 Optimistic protocol's cost.

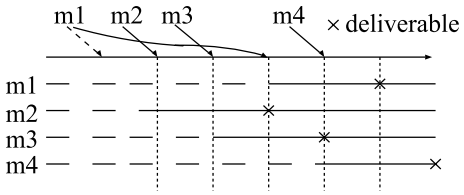

図 3 悲観的プロトコルのコスト
Fig. 3 Pessimistic protocol's cost.

セージの順序を決定する処理が受信側のそれぞれのノードに存在し，受信から配送までが 1 カ所で行われることである．この特徴を利用することにより，悲観的手法を用いたプロトコルではメッセージが受信されてから配送可能になるまでの時間を全順序保証処理に要した時間と考えることができる．楽観的手法を用いたプロトコルでは，メッセージは受信後すぐに配送可能とするため受信決定処理はないが，それぞれのノードで行われるやり直し処理を受信から配送可能になるまでの時間に含めて考えることができる．

図 2 はメッセージ $m1$ が遅れて到着し，それ以前に受信されたメッセージ $m2$ と $m3$ により行われた処理がロールバックされた例である．メッセージ $m2$ は受信後ただちに配送可能となるが，ロールバックが行われ再度配送可能となる．このときのメッセージ $m2$ のコストは受信されてから最終的に配送可能となった時間（ロールバックを含む）までと考えることができる．よって，全順序保証プロトコルの処理コストを受信から配送可能となるまでの時間から見積もることにより，楽観的手法と悲観的手法の両方を対象とした全順序保証プロトコルの性能評価を行うことができる．悲観的手法の例である**図 3** は，図 2 と同様にメッセージが受信される例であるが受信者側で全順序保証のみを行うため，メッセージが受信されてから配送可能になるまでの時間をコストと考えることができる．

やり直しの処理をどのように全順序保証プロトコルの性能評価に反映するかは 1 つの問題である．代表的な楽観的手法を用いた全順序保証プロトコルである Time Warp の性能やコストの評価についてはさまざまな研究[9)–11)] がなされている．これらの研究はプロトコルの性能が実行環境の変化によってどのように変化するかに着目しており，やり直し処理が性能に大きく影響し，やり直しは各メンバの論理時刻の進み具合にばらつきがあるときに発生することや，イベントの粒度が大きい場合は環境の変化を受けにくいという特徴があることが分かっている．これらの研究結果をふまえて，全順序保証処理に要する時間をメッセージの到着頻度とメッセージ順序のばらつき（受信順序と配送されるべき順序），各イベントの処理時間から見積もる方法を用いることにする．コストの見積り方法の詳細は 4.1 節で述べる．

3.3 コストの比較

個々のメンバは見積もったコストを比較し，それぞれにとってコストが最も低いプロトコルを選択する．コストの比較は現在の順序付けのコスト $Cost_d$ に対し他のプロトコルでの順序付けコスト $Cost_d(NEXT)$ と切替えコスト $Cost_s$ の合計を比較することによって行う．プロトコルの切替えのコストを考慮することにより，頻繁にプロトコルが変更されるのを防ぐ．すなわち，プロトコル切替えの判断は式 (1) によって行う．

$$Cost_d > Cost_d(NEXT) + Cost_s \qquad (1)$$

現在の順序付けコストよりも切替えを行い他の順序付けを行う場合のコストが小さい場合は，このメンバはプロトコルの切替えを選択することになる．

3.4 メンバ間の合意

各メンバで選択したプロトコルに切り替えるかどうかをグループで一意に決定する．この処理は順序付けとは独立しており，別のスレッドとして動作することが可能である．

メンバは任意の間隔で各自が独立にローカルの情報を用いてコストの見積り・比較を行い最適なプロトコルを決定する．最適なプロトコルが現在のプロトコルと異なる場合，グループ内でプロトコル切替え判断の処理をコーディネータに依頼することで開始する．すでにグループ内に 1 つのコーディネータが存在するときは，そのコーディネータを中心に多数決処理を行う．存在しないときはまず文献 12)–14) などに提案されている選出アルゴリズムによりコーディネータを決定し，その後コーディネータを中心に処理を行う．メンバ数が N のとき $O(N)$ のメッセージ数でコーディネータを選出することができる．

その後，コーディネータがメンバに投票要求を送り，投票要求を受けた各メンバは，提案されたプロトコルが各メンバにおいて最適か否かを決定して投票を行う．投票を受けたコーディネータはそれらを集計し，半数

を超える賛成で合意となり，結果をメンバに通知する．合意に至らなかったときは，現在のプロトコルを引き続き利用する．運用にあたってはあらかじめ合意アルゴリズムを定める必要がある．なぜなら単純な多数決では最適なプロトコルが選択されない場合が考えられるからである．たとえば，楽観的プロトコルが 1 つ，悲観的プロトコルが 2 つの計 3 つのプロトコルが候補である場合，悲観的プロトコルで票が割れることにより，票数では楽観的プロトコルが上回る可能性がある．

3.5 準備と切替え

プロトコルの切替えを安全に行うには，切替え前後でメッセージの順序がメンバ間で同一になるように保障する必要がある．Hybrid Protocol [4] ではプロトコルが動的に切り替わる方法が用いられているが，これらは楽観的手法に分類されるプロトコルを対象としていない．そこで楽観的プロトコルも切替え対象に含むための要件をあげる．プロトコルの変更はグループメンバが共通の論理時刻を境にプロトコルを変更することにより実現する．ある時刻 T にプロトコル p からプロトコル q へ切り替えると仮定した場合，T より小さなタイムスタンプ $t_m \leq T$ を持つメッセージ m はプロトコル p が適用され，T より大きなタイムスタンプ $t_n > T$ を持つメッセージ n はプロトコル q が適用されなければならない．よって，

$$t_m \leq T < t_n \tag{2}$$

となるような時刻 T をグループ内で一意に定める必要がある．切替え時刻 T の決定は 3.4 節と同じくメンバ間の同意によって決めることができる．このプロセスは全順序保証とは並行して行うことができるため，全順序保証プロトコルのコストに影響は与えない．

切替え時刻 T はグループ内で一意のグローバルな時刻であり，どのメンバの時刻よりも未来の時刻である．この時刻 T を決めるにはすべてのメンバから切替え希望時刻を集め，その時刻の中で最大の値を持つ時刻（最も未来の時刻）を選択する．それぞれの希望時刻はメンバがローカル時刻の進む割合を考慮して決定する．メンバはローカルの時刻が T に到達したときはその時刻で他のメンバを待つことになる．しかし，プロトコルを切り替えるときは十分に遠い将来の時刻を設定したとしても，メンバは切替え時刻まで自分の時刻を進めることができる．時刻を進めることは順序付け処理の効率に影響はないため，切替え希望時刻は十分に未来の時刻とすることができる．切替え時刻は，3.4 節で述べた最適プロトコルを決める合意処理と同時に（同一のメッセージで）決定することが可能であるので，切替え時刻決定自身のオーバヘッドは無視できる．

プロトコルの切替え手法は楽観的から悲観的へ移行とその逆とで異なる．

楽観的プロトコルから悲観的プロトコルへの移行

楽観的プロトコルから悲観的プロトコルに移行するためには，式 (2) のプロトコル変更時刻を設定しグループ内のメンバがその時刻において変更可能状態であることが必要である．以下の 2 つの条件を満たすことにより次のプロトコルへ変更可能となる．

- すべてのメンバの時刻が変更時刻まで進んでいる．
- 切替え時刻より小さなタイムスタンプを持つメッセージはすべて配送済みである．

各メンバが持つローカルの論理時刻と送信履歴，メッセージキューに関する情報を交換することによって変更可能であるかを確認することができる．確認はメンバ間の遅延を d とすると $O(d)$ の時間が必要となり，さらにメンバの論理時刻のばらつきによる影響も受ける．よって，確認頻度を高めることはプロトコル変更を早めることができるが，同時に計算資源を費やすことにもなる．また，ローカルの論理時刻が変更時刻より進んだメンバは確認を待たず次のプロトコルを開始することができるが，上の 2 つの条件を満たすまではロールバックの処理が発生する可能性がある．これらはトレードオフであり，計算環境やアプリケーションの特性を考慮して決める必要がある．

悲観的プロトコルから楽観的プロトコルへの移行

式 (2) によって定めた時刻 T まで悲観的プロトコルを適用し，以降は各自同意をとらずに楽観的プロトコルでの処理を開始することができる．

4. 実際の適用例と評価

環境に適応した全順序保証プロトコル選択の具体的適用を示す．ここでは楽観的手法として Time Warp [1]，悲観的手法として ABCAST [2] を仮定する．

Time Warp では，送信者がメッセージにそのメッセージが配送されるべき論理時刻をタイムスタンプとして添付し送信する．メッセージの受信者はメッセージを受信すると，自身が持つ論理時計とメッセージのタイムスタンプを比較する．タイムスタンプに記された時刻の方が未来を指し示していれば論理時計をタイムスタンプの時刻を値に更新しメッセージの配送を行う．もしタイムスタンプが受信者の時計より過去の場合はその時刻まで処理を戻し，再び処理を開始する．

今回利用する ABCAST は Skeen のアルゴリズムと呼ばれ，初期の Isis システムで用いられているものである．このプロトコルでは，メッセージの受信者が

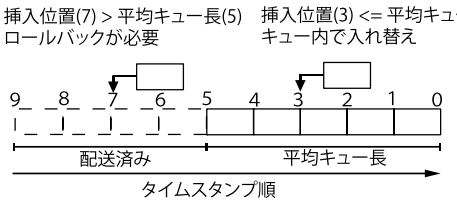

図 4 コストの見積り（Time Warp）
Fig. 4 Cost estimation (Time Warp).

受信メッセージに論理時計が示す時刻のタイムスタンプを添付し送信者に返す．送信者はすべての受信者からタイムスタンプ付きメッセージを受信するとそのタイムスタンプの中で最も未来の時刻を示すタイムスタンプをすべての受信者に再送する．このタイムスタンプの時刻によって受信者はメッセージの配送順序を判断する．

4.1 コストの見積り

3.2 節で述べたとおり，プロトコルのコストを受信者がメッセージを受信してから配送するまでに要する実時間とする．順序付け処理が 1 点で行われるトークン方式やシーケンスサーバ方式は性能が単一のノードによって左右されるが，Time Warp と ABCAST の 2 つのプロトコルは各メンバで順序付け処理がそれぞれ行われるためメンバ全体によって性能が決定することになる．

Time Warp

楽観的手法では，ローカルの論理時間より古いタイムスタンプのメッセージを受け取ると処理のやり直しをするためにロールバックが必要となる．ロールバックのコストは一般に順序付けに比べ非常に大きいため，順序付けプロトコルのコストはロールバックの発生する確率に大きく左右される．例として，処理の粒度が大きくメッセージ交換の頻度が低い場合は古いタイムスタンプを持つメッセージが到着する可能性が低くコストは低い．逆に処理の粒度が小さくメッセージ数が多い場合はメッセージが到着する順序が入れ替わる可能性が高いためコストも高くなる．そこで，ここでは Time Warp のコストをキュー内にあるメッセージの統計をとることにより求める手法を提案する．

配送キュー内にはメッセージがタイムスタンプ順に格納されているとする．あるメッセージを受信したとき，そのメッセージがキュー内でどの位置に挿入されるかはタイムスタンプによって決まる．**図 4** のようにすでに処理済みメッセージを含む仮想のメッセージキューを考える．タイムスタンプの順序と受信順序が一致しているとき，すなわちメッセージが理想的な順序で受信されているときのメッセージはつねにキューの末尾に格納されるが，そうでない受信メッセージはタイムスタンプに従い適切な位置に挿入される．ロールバックの必要なメッセージはすでに処理されたメッセージの中に挿入されるため，そのメッセージより後ろにあるメッセージの数は配送可能キューの平均値よりも長くなる．この性質を利用し以下の方法でコストを見積もる．

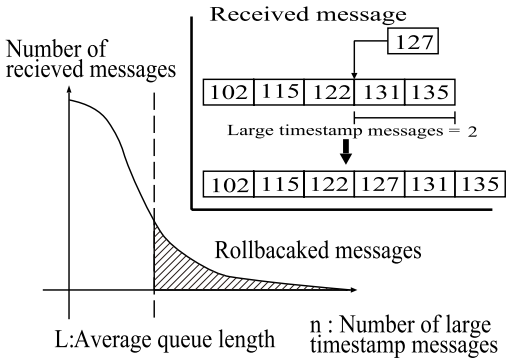

図 5 受信メッセージ分布
Fig. 5 Message distribution.

図 5 のグラフの横軸はあるメッセージが受信されたときにそのメッセージのタイムスタンプより大きなタイムスタンプを持ったメッセージの数を表す（図 5 の例では 2 つ）．縦軸は上記のようなメッセージの単位時間あたりの累積個数である．すなわち横軸 n のときの縦軸の値は末尾から $n+1$ 番目に挿入されたメッセージの単位時間あたりの累積個数である．たとえば，図 5 のように自らのタイムスタンプよりも大きなタイムスタンプを持ったメッセージが 2 つあった場合にそのようなメッセージを単位時間にいくつ受信したかを表す．

理想的な順序で受信されたメッセージの数は横軸 0 のときの値となり，図 5 の例のようにキューの末尾から 3 つ目に挿入されるメッセージの数は横軸 2 のときの値となる．図 5 の例では，理想的な順序で受信されたメッセージが最も多く，タイムスタンプと受信順序のずれが大きくなるに従いそのようなメッセージは少なくなっている．

ロールバックに要する平均実時間 T をとすると，図 5 の斜線部の面積がロールバックを要するメッセージ数を表すので，単位時間あたりにロールバックに要したコスト $Cost(TW)$ を次式で見積もることができる．

$$Cost(TW) = \int_{L}^{\infty} f(n)dn \times T \qquad (3)$$

実装では分布関数 f を求める代わりに，メッセー

ジの受信履歴を用いてロールバックを必要とするメッセージの割合を求めることができる．

単位時間の受信メッセージ数は到着率 λ のポアソン分布に従い，メッセージを処理する時間はサービス率 μ の指数分布に従うと仮定すると，トラフィック密度 ρ は式 (4) で示される．

$$\rho = \frac{\lambda}{\mu} \tag{4}$$

これにより，キュー内の平均メッセージ数 L を式 (5) により求めることができる．

$$L = \frac{\rho^2}{1-\rho} \tag{5}$$

$Cost(TW)$ はメッセージの到着率（受信の頻度）とサービス率（処理の粒度）に依存することになる．ここで，λ と μ は観測可能である．

ABCAST

悲観的手法では順序を決定するためにメッセージを交換しメンバ間で同意をとる．よって，ネットワーク遅延が小さい場合に効率が良く，大きい場合には効率が悪い．ABCAST ではメッセージ受信後に送信者と受信者間で acknowledge メッセージと commit メッセージの 2 個のメッセージをやりとりする．送信者はすべての受信者からの acknowledge メッセージを待つ．したがって，コストの見積りにはグループ内のメンバ間の最大遅延を d とした場合，$2d$ で見積もることができる．よって，単位時間あたりのコストはメッセージの到着率 λ から求めることができる．

$$Cost(ABCAST) = \lambda \times 2d \tag{6}$$

4.2 コストの比較

それぞれのプロトコルにおける順序付けコストを式 (1) に従って比較することにより，最適なプロトコルを選択する．切替えのコストは，Time Warp からの切替えと ABCAST からの切替えで異なる．切替え方法については次節で述べることとする．

プロトコル切替えのコストはプロトコル切替えの実装に依存するので，以降のコスト計算において s とおく．たとえば，具体的な実装方法として我々は，再構成可能オブジェクトモデル[15] を用いて，順序付けを行うエンティティを入れ替える方法を提案している[16]．

Time Warp による順序付け

Time Warp による順序付けが行われている場合，プロトコルを切り替える時刻を設定する．この時刻より前のメッセージは Time Warp を適用し，後のメッセージは ABCAST による順序付けを適用する．切替え時刻を未来に設定し，メッセージの送信者は自身が持つローカルのタイムスタンプを同意の得られた時刻まで進めることにより，以後のメッセージは切替え時刻より大きなタイムスタンプを持ち，次のプロトコルで順序が決定される．切替え時刻 T（式 (2)）の決定と GVT の更新は順序付けプロトコルと並行に行われる．GVT が切替え時刻 T に達するまでの時間を α とすると，Time Warp による順序付けが行われている場合のコスト比較は式 (7) のようになる．

$$Cost(TW) > Cost(ABCAST) + s + \alpha \tag{7}$$

ABCAST による順序付け

ABCAST においても切替え時刻 T（式 (2)）の決定は順序付けプロトコルと並行に行われる．メンバは個々に次のプロトコルを開始できることから，コストの比較は式 (8) のようになる．

$$Cost(ABCAST) > Cost(TW) + s \tag{8}$$

4.3 準備と切替え

Time Warp から ABCAST への切替え

プロトコルを切り替える上記の 2 つの条件は，切替え時刻後にロールバックが発生しないことを保証することである．これは Time Warp アルゴリズム[1] における Global Virtual Time（GVT）を用いることができる．**図 6** のようにプロトコルを切り替える時刻 T（式 (2)）を定めたとき，すべてのメンバが時刻 T まで処理が進んでいること，すなわち GVT が T まで達していることが楽観的プロトコルによる処理が完了していることを保証し，以後のメッセージはすべて悲観的プロトコルを適用することができる．メンバはそれぞれの論理時刻（Local Virtual Time，LVT）は切替え時刻 T に到達するが，LVT より小さなタイムスタンプを持つメッセージが到着する可能性があるため，GVT が切替え時刻 T に達するまで待った後に悲観的プロトコルを開始する．この待ち時間が式 (7) の α である．待ち時間 α を小さくするには GVT の更新頻度を高くすることが考えられる．

次に具体的な切替え処理の例を述べる．

- 切替え時刻 T より小さなタイムスタンプを持つメッセージを処理する．切替え時刻は十分に未来の時刻であるが，もしローカル時刻がその時刻に達した場合はそれ以後の処理は行わない．
- 処理するべきメッセージがなくなったら，送信済みメッセージのうちタイムスタンプが最大のものと GVT を比較し，自ら送信したメッセージが処理されたことを確認する（切替え時刻 T 以前の処理がすべて終了したことを確認する）．
- 確認した後，GVT マネージャに処理が終了していることを報告する．
- GVT マネージャはすべてのメンバの処理が終了

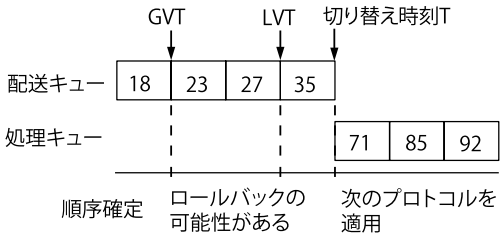

図 6 楽観的プロトコルの切替え
Fig. 6 Switch optimistic protorol.

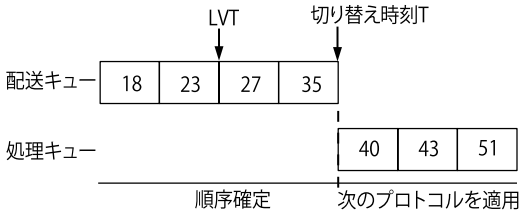

図 7 悲観的プロトコルの切替え
Fig. 7 Switch pessimistic protocol.

したことを確認した後に GVT を切替え時刻に更新し，メンバは次のプロトコルを開始する．

ABCAST から Time Warp への切替え

切替えは式 (2) によって定めた時刻 T より大きなタイムスタンプを持つメッセージから，各自が Time Warp の処理を始める．**図 7** のようにメンバのローカルの論理時刻（LVT）にかかわらず，順序が確定したメッセージのタイムスタンプをもとに切替えを行うことができるため，待ち時間が発生することはない．

切替え処理の流れは次のようになる．

- 切替え時刻 T より小さなタイムスタンプを持つメッセージを処理する．
- タイムスタンプが切替え時刻を越えたメッセージから次のプロトコルの適用を開始する．

5. 評　　価

環境の変化を再現するシミュレータを実装し，本論文で提案した手法の有用性を確認した．設計者が自由に環境の変化をつくりだすことにより，全順序保証プロトコルの切替えが起きたことを確認するとともに，切り替えたことによって順序付けコストが軽減されることを確認した．

5.1 シミュレーション

初めに，送信頻度によって Time Warp と ABCAST がどのような影響を受けるかを検証した．**図 8** はメッセージの送信頻度を変化させたときの Time Warp と ABCAST プロトコルのコストである．プロセスがメッセージを処理する平均時間は 5 ms とし，メッセージの送信頻度を 0.02 個/msec から 0.2 個/msec まで変化させている．また，メッセージ数の増加によって通信速度が低下することはないとする．Time Warp はメッセージの送信頻度の増加とともにロールバックの起こる確率が高くなり，コストが増大する．また，メッセージの頻度が 0.12 個/msec を超えてからはロールバックの確率が下がりコストが減少している．これはメッセージキュー内にメッセージが存在する場合に，キュー内で配送待ちのメッセージの並べ替えができるからである．ABCAST のコストはメッセージ送信頻度の変化による影響はなく，メッセージの転送遅延に応じたコストとなっている．

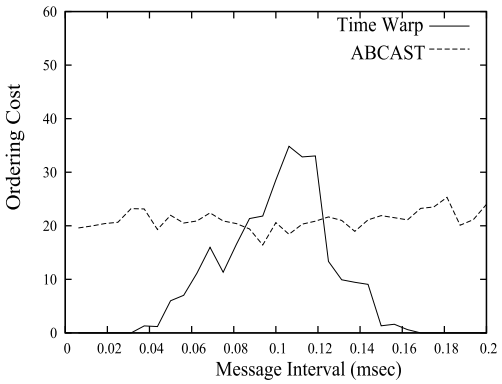

図 8 送信頻度の変化
Fig. 8 Message sending rate.

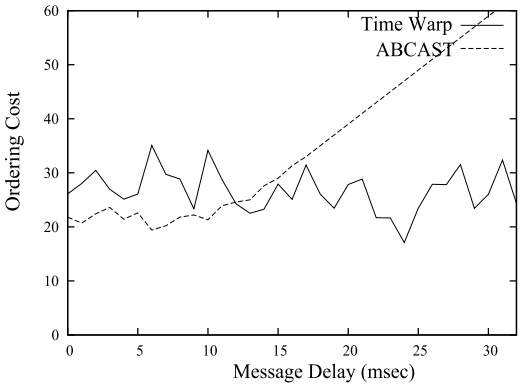

図 9 通信遅延の変化
Fig. 9 Communication delay.

次に，通信遅延による 2 つのプロトコルへの影響を検証した．**図 9** は 5 つのメンバ中 1 つのメンバとの通信遅延が変化したときのコストである．プロセスがメッセージを処理する平均時間は 5 ms，メッセージの送信頻度は 1 ms に 0.08 個とし，通信遅延を 10 ms とした．このとき 1 つのメンバとの通信遅延を次第に増加させ，32 ms まで変化させている．Time Warp で

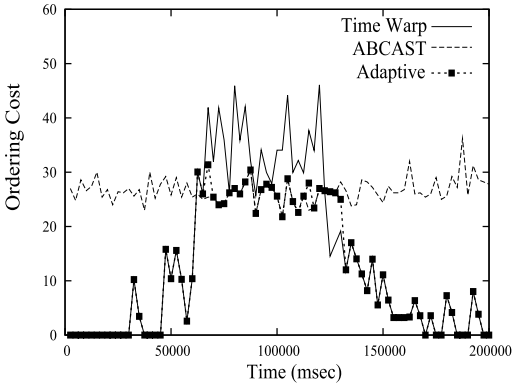

図 10 適応的選択
Fig. 10 Adaptive selection.

は，局所的な通信遅延の変化による著しい変化はみることができない．ABCAST のコストはあるメンバの通信遅延が平均の通信遅延を上回ってからは，そのメンバとの通信遅延の影響が順序付けコストに直接影響している．これは悲観的手法のプロトコルではすべてのメンバからの情報が順序付けに必要であるという特徴のためである．

以上の送信頻度（図 8）と通信遅延（図 9）の 2 つの検証により，Time Warp はメッセージ送信頻度の変化の影響を受けやすく，ABCAST は通信遅延による影響が大きいことが分かった．そこで，これらを動的に選択することによる環境変化への適応可能性を検証する．Time Warp と ABCAST の 2 つのプロトコルと本論文で提案する適応的選択を利用した実験結果を示す．**図 10** に時間の経過とともにメッセージの送信頻度を変化させたときの Time Warp と ABCAST プロトコル，適応的選択のコストを示す．通信遅延はばらつきがあるが，メッセージの数によって通信遅延が変化せずに一定とする．送信頻度は始め低く，50 sec から 150 sec までは送信頻度が高い期間とし，その後は再び頻度の低い期間としている．Time Warp は送信頻度の低いときに非常にコストが低いが，送信頻度の上昇とともに順序付けコストが著しく上昇している．一方，ABCAST は通信遅延が一定であるため，安定したコストを維持している．適応的選択ではコストの低いほうのプロトコルを選択するため，メッセージの送信頻度が低いときは Time Warp を選択し，送信頻度が上昇すると ABCAST を選択している．プロトコルの切替えが起こるときには切替えの処理が必要となるため，最適コストに切替えのコストが一時的に加わっている．

6. ま と め

本論文では，メッセージの全順序保証に関して，楽観的順序付け手法と悲観的順序付け手法を動的に選択し切り替えることによって環境変化に適応する方法を提案した．まず，最適な手法の選択法として，メッセージの到着率とメッセージの処理時間から順序付けのコストを比較可能な形で求め，評価する手法を提案した．次に，全順序保証を損なうことなくプロトコルを切り替えるアルゴリズムを提案した．例として Time Warp と ABCAST をあげ，それぞれのプロトコルがメッセージの送信頻度と転送遅延の変化から受ける影響を検証した．検証の結果，楽観的手法はメッセージの送信頻度の影響を受けやすく，悲観的手法はメッセージの転送遅延の影響を受けやすいことが分かった．そこで，これらの環境変化に応じてコストを見積もることにより，最適なプロトコルを選択可能であることが分かった．さらに 2 つのプロトコルの動的選択を検証し，サービスを停止することなく最適な全順序保証プロトコルを利用できることをシミュレーションにより示した．

実行環境に適した全順序保証プロトコルを適応的に選択することにより，昼夜でトランザクション数が異なる実行環境や一時的にトラフィックが集中しネットワークの遅延が局所的に大きくなるような場合においても，全順序保証に関わる性能低下を防ぐことができる．

参 考 文 献

1) Jefferson, D.R.: Virtual time, *ACM Trans. Prog. Lang. Syst.*, Vol.7, No.3, pp.404–425 (1985).
2) Birman, K.P. and Joseph, T.A.: Reliable communication in the presence of the failures, *ACM Trans. Computer Systems*, Vol.5, No.1, pp.47–76 (1987).
3) Chockler, G.V., Huleihel, N. and Dolev, D.: An Adaptive Totally Ordered Multicast Protocol that Tolerates Partitions, *Proc. 17th annual ACM Symposium on Principles of Distributed Computing*, pp.237–246 (1998).
4) Rodrigues, L., Fonseca, H. and Verissimo, P.: Totally ordered multicast in large-scale systems, *Proc. 16th International Conference on Distributed Computing Systems*, pp.503–510 (1996).
5) Dolev, D., Kramer, S. and Malki, D.: Early delivery totally ordered multicast in asynchronous environments, *23rd Annual Interna-*

tional Symposium on Fault-Tolerant Computing, pp.544–553 (1993).
6) 細谷 篤, 佐藤文明, 水野忠則：適応的全順序マルチキャストの拡張, 情報処理学会論文誌, Vol.42, No.2, pp.138–146 (2001).
7) Defago, X., Schiper, A. and Urban, P.: Total order broadcast and multicast algorithms: Taxonomy and survey, *ACM Computing Surveys*, Vol.36, No.4, pp.372–421 (2004).
8) Urban, P., Defago, X. and Schiper, A.: Contention-aware metrics for distributed algorithms: Comparison of atomic broadcast algorithms, *Proc. 9th IEEE International Conference on Computer Communications and Networks*, pp.582–589 (2000).
9) Carothers, C.D., Fujimoto, R.M. and England, P.: Effect of Communication Overheads on Time Warp Performance: An Experimental Study, *Proc. 8th Workshop on Parallel and Distributed Simulation*, pp.118–125 (1994).
10) Carothers, C.D. and Fujimoto, R.M.: Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms, *IEEE Trans. Parallel and Distributed Systems*, Vol.11, No.3, pp.299–317 (2000).
11) Ferscha, A. and Luethi, J.: Estimating Rollback Overhead for Optimism Control in Time Warp, *28th Annual Simulation Symposium*, pp.2–12 (1995).
12) Garcia-Molina, H.: Elections in a Distributed Computing System, *IEEE Trans. Comput.*, Vol.31, No.1, pp.48–59 (1982).
13) Greg, N., Frederickson and Lynch, N.A.: Electing a leader in a synchronous ring, *J. ACM*, Vol.34, No.1, pp.98–115 (1987).
14) Singh, S. and Kurose, J.F.: Electing "good" leaders, *Journal of Parallel Distributed Computing*, Vol.21, No.2, pp.184–201 (1994).
15) Oda, K., Izumi, S., Yasutake, Y. and Yoshida, T.: A Simple Reconfigurable Object Model for a Ubiquitous Computing Environment, *International Journal of Computer Science and Information Security*, Vol.7, No.5, pp.8–16 (2007).
16) Yasutake, Y., Izumi, S., Oda, K. and Yoshida, T.: Implementation of an Adaptive Total Ordering Protocol, *International Journal of Computer Science and Information Security*, Vol.7, No.7, pp.153–159 (2007).

(平成 19 年 5 月 17 日受付)
(平成 19 年 11 月 6 日採録)

安武 芳紘（正会員）

2000 年九州工業大学情報工学部知能情報工学科卒業．2002 年同大学大学院情報工学研究科博士前期課程修了．2005 年同大学院情報工学研究科博士後期課程単位取得退学．現在，九州産業大学情報科学部知能情報学科助教．適応型分散ミドルウエア，分散システム等の研究に従事．

小田謙太郎（正会員）

1999 年九州工業大学情報工学部知能情報工学科卒業．2001 年同大学大学院情報工学研究科博士前期課程修了．2004 年同大学院情報工学研究科博士後期課程単位取得退学．現在，同大学院情報工学研究科情報創成工学専攻助教．適応型分散ミドルウエア，マルチエージェントシステム（RoboCup），分散システム等の研究に従事．日本ソフトウェア科学会，IEEE CS，ACM 各会員．

吉田 隆一（正会員）

1982 年慶應義塾大学工学部電気工学科卒業．1987 年同大学大学院工学研究科博士後期課程電気工学専攻修了．工学博士．同年九州工業大学情報工学部知能情報工学科助手．1990 年同助教授．1993 年から翌年にかけて，米国オレゴン科学技術大学院大学において客員研究員．2002 年九州工業大学大学院情報工学研究科情報創成工学専攻教授．現在に至る．オブジェクト指向計算，分散計算，地理情報システムに興味を持つ．日本ソフトウェア科学会，IEEE 各会員．