Vol.2009-DPS-139 No.2
2009/6/18

P2P 基盤ソフトウェア musasabi の仮想ピアにおける通信方式

鹿野将典[†1] 上田達也[†1] 安倍広多[†1]
石橋勇人[†1] 松浦敏雄[†1]

著者らが研究・開発中の P2P 基盤ソフトウェア musasabi では，P2P ネットワーク上の複数のピア上で同一のアプリケーションプログラムを同時に実行することで，実行中のプログラムの耐故障性を向上させる仮想ピア機能を備えている．本稿では，仮想ピアにおける通信処理の実現法について述べる．提案手法では，仮想ピアが実際は複数のピアを用いて一貫性を保ちながら動作していること，仮想ピアを構成するピアの構成が時間とともに変化する可能性があることなどを考慮している．

Communication Methods for Virtual Peers on *musasabi* P2P Platform

Masanori Shikano,[†1] Tatsuya Ueda,[†1] Kota Abe,[†1]
Hayato Ishibashi[†1] and Toshio Matsuura [†1]

The authors have been developing *musasabi* P2P platform, which supports *virtual peers*. Virtual peers achieve fault-tolerance of running program by executing the same program on multiple peers simultaneously. In this paper, communication methods for virtual peers are described. The methods take into account the fact that a virtual peer consists of multiple peers which run in consistent with others, and that the membership of a virtual peer might be changed as time goes on.

†1 大阪市立大学大学院創造都市研究科
Graduate School for Creative Cities, Osaka City University

1. はじめに

Peer-to-Peer(P2P) 方式は，サーバクライアント方式と比較すると，単一故障点であるサーバが存在しないため耐故障性に優れているとされているが，一方ピアが予告なく離脱することに対する対策が必須であり，サーバクライアント方式よりも実装が難しい．

著者らは，P2P システムにおいて耐故障性を確保するために，P2P ネットワーク上の複数のピアを用いて**仮想ピア**を構築する手法を提案し，P2P 基盤ソフトウェア musasabi に実装している（**図 1**）[1)]．仮想ピアではその上でプログラムを実行できる．仮想ピアの各ピア（**メンバピア**と呼ぶ）が冗長系を構成することで，一部のピアが離脱（故障を含む）しても仮想ピア上で実行しているプログラムは停止しない．また，メンバピアが離脱した場合，P2P ネットワーク上の他のピアから補充することでメンバピアの数を回復する．このため，仮想ピアの障害の確率は十分低く，安定した（離脱しない）ピアとみなすことができる．

仮想ピアを使って P2P サービスを構築することで，従来の P2P アプリケーションの実装では不可欠だったピアの障害対策（リモートピアの離脱に備えてデータの複製を配置したり，バックアップポインタを維持するなど）を簡素化できる．また，仮想ピアを仮想的なサーバとして利用することでハイブリッド型 P2P における単一障害点（サーバ）を除去するなどの応用も考えられる．

仮想ピアを用いて P2P サービスを実現するためには，仮想ピアと通常のピア，および仮想ピア同士の通信が必要となるが，仮想ピアは複数のメンバピアから構成され，また各メンバピアの状態を一貫性を持って更新する必要があるため，通信処理は単純ではない．

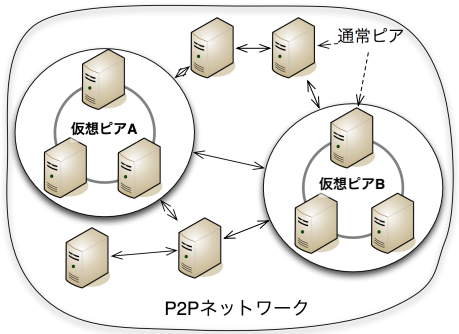

図 1 仮想ピアと通常ピア

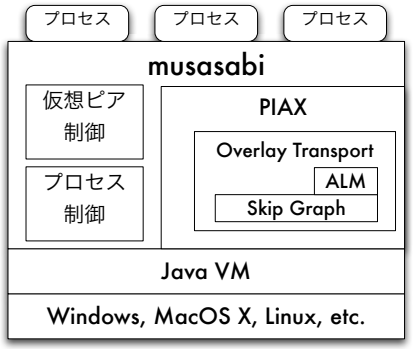

図 2 musasabi の構造

本稿では，仮想ピアにおける通信の方式について述べる．

2. P2P 基盤ソフトウェア musasabi

ここでは，P2P 基盤ソフトウェア musasabi について簡単に述べる（詳細は文献 1) を参照されたい）．

musasabi は，P2P アプリケーションを実行するための基盤ソフトウェアであり，全て Java 言語で実装している．musasabi を実行している計算機間で通信するために PIAX[2] を利用している．構成を**図 2** に示す．

musasabi 上で実行するユーザのアプリケーションプログラムも Java 言語で記述する．musasabi 上で実行中のアプリケーションプログラムを（通常の OS と同様）**プロセス**と呼ぶ．

2.1 仮想ピア機能の概要

仮想ピアは，アプリケーションプログラムを複数のピア（メンバピア）を用いて実行することで，アプリケーションプログラムを長期間安定して動作させるための機構である．仮想ピア上で実行中のアプリケーションプログラムを，**仮想プロセス**と呼ぶ．仮想ピアと仮想プロセスは 1 対 1 に対応する．

仮想ピア上で仮想プロセスを実行する場合，仮想ピアを構成する複数のメンバピア上で同一のプログラムを同時に（冗長的に）実行させる．これにより，一部のメンバピアが離脱（故障などの障害を含む，以下同じ）しても，仮想プロセスの実行を継続できるようにしている．仮想ピアと仮想プロセス，メンバピアの関係を**図 3** に示す．

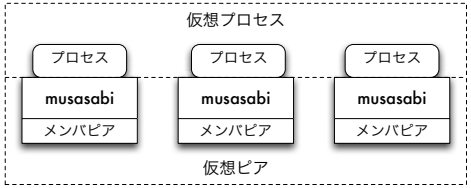

図 3 仮想プロセスと仮想ピア

メンバピア数を一定に保つため，メンバピアが離脱した場合は代わりに P2P ネットワークに参加している他のピアをメンバピアとして追加する．このため，仮想ピアを構成するメンバピアは固定されているわけではなく，時間の経過とともに変化する．

2.2 メンバピア間での一貫性確保

仮想プロセスは論理的には 1 つのプロセスであるため，仮想ピアのメンバピア上で動作する各プロセスは，外部からみて動作が一貫している必要がある．このために，Paxos 合意アルゴリズム[3),4)]（以後単に Paxos と表記する）を用いた State Machine Replication 方式[3)] を採用している．

この方式では，仮想ピア上で動作するアプリケーションはなんらかの入力を受け取って処理を行い，結果を返すものとする．また，アプリケーションは，同一の入力系列を与えると毎回同じように動作しなければならない（つまり決定的に動作する必要がある）．なお，アプリケーションは，ネットワーク経由で送信されるメッセージだけを入力とするものとする．

このような制約を設けた上で，メンバピア上で動作する複数のプロセスに対し同一のメッセージを入力として与えることで，各プロセスを同じように動作させる．メッセージの受信順序を統一するために Paxos を用いる．

2.2.1 Paxos 合意アルゴリズム

Paxos 合意アルゴリズムは，分散システムの複数の参加者（ここではメンバピア）間で値の合意を形成するアルゴリズムである．参加者の過半数が正常に動作していればいずれ合意が成立する．合意の対象は値の系列に拡張できる (Multi-Paxos)．Paxos では，唯一のリーダ（参加者の中から何らかのリーダ選出アルゴリズムを用いて選出する）だけが値を提案できる．提案する値にはシーケンシャル番号が付与される．参加者の過半数が提案された値を受理することで合意が成立する．

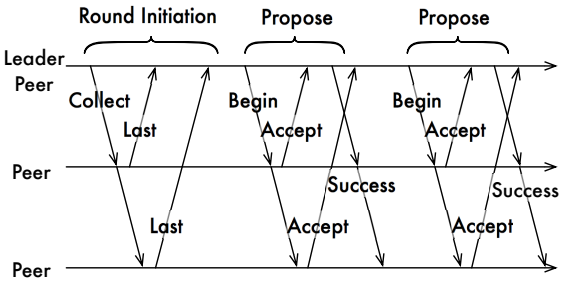

図 4 Paxos プロトコルシーケンス

Paxos で値を提案・合意するためのシーケンスの概略を以下に示す（**図 4** 参照）[⋆1].

(1) リーダは最初に Collect メッセージを全参加者（メンバピア）に送信する.
(2) 各参加者は Collect メッセージに対して Last メッセージで応答する.
(3) リーダが過半数の参加者から Last メッセージを受信すると，リーダは値を提案できるようになる.
(4) 値を提案する場合，リーダは Begin メッセージを全参加者に送信する．複数の提案値を区別するため，Begin メッセージにはシーケンス番号が埋め込まれている.
(5) 各参加者は Begin メッセージを受信して提案された値を受理すると，Accept メッセージをリーダに送信する.
(6) 過半数の参加者から Accept メッセージを受信したら，合意成立を Success メッセージで全参加者に通知する.

2.2.2 musasabi での Paxos 実装

musasabi の実装では，メンバピアの中で最も古くから参加しているピアがリーダになる.

リーダピアでは，仮想プロセス宛に送信されたメッセージを受信すると（どのように受信するかについては 3 章で述べる），まず Paxos を用いて当該メッセージを提案する（musasabi ではシリアライズ可能な Java オブジェクトを値として Paxos で提案できる）.

各メンバピアでは値の合意が確認されたら（リーダの場合は過半数の参加者から Accept メッセージを受信した場合，非リーダの場合は Success メッセージを受信した場合），合意された値（メッセージ）をプロセスに渡す．プロセスは渡されたメッセージに基づいて処理を行う（メッセージドリブン）.

なお，Paxos は複数の値の提案を非同期に処理するため，メッセージが提案された順（シーケンス番号の順に）に合意されるとは限らない．また，Success メッセージがネットワーク上で喪失する可能性もある．このため，musasabi では合意されたメッセージをプロセスに渡す前に，シーケンス番号順に並べ替える．もし欠落しているシーケンス番号がある場合は埋まるまで待ち，なかなか埋まらない場合はリーダの場合再提案，非リーダの場合はリーダに値の問い合わせを行う．これにより，全てのプロセスが同一の順番に同一の入力を受け取ることができる.

2.3 メンバピア離脱時の処理

メンバピアの 1 つが離脱した場合，P2P ネットワーク中の通常ピアの中から適当なピアを選び（現在はランダムに選択），そのピアと離脱したピアを入れ替える．新しく参加したピア上でも仮想ピアのプロセスを動作させる必要があるが，そのプロセスは既存のメンバピア上で既に実行しているプロセスと同じ状態にする必要がある．このため，musasabi ではプロセス移送技術を用いてリーダピア上で実行中のプロセスを新しく参加したピアに複製する[1].

3. 仮想ピアにおける通信方式

仮想ピアでの通信方式について述べる．送信ピア（通常ピアあるいは仮想ピア）が，受信ピア（仮想ピア）に何らかの要求をメッセージとして送信し，受信ピアが応答メッセージを返すというモデルを想定する.

なお，送信ピアが複数のメッセージを同一の受信ピアに応答を待たずに送信したとき，メッセージ送信順序とメッセージ受信順序は一致する必要はないものとする．また，ピア間（物理的なノード間）で送受信するメッセージは，任意の時間遅延，もしくは喪失する可能性があるものとする.

以下，3.1 節で通常ピアと仮想ピア間での通信について述べ，次に 3.2 節で仮想ピア間での通信について述べる.

3.1 通常ピアと仮想ピアが通信する場合

ここでは，通常ピア p が仮想ピア v にメッセージ m を送信し，v が p に応答を返す場合の処理を検討する.

3.1.1 仮想ピアへのメッセージの配送

まず，仮想ピアは構成するメンバピアが固定されていないため，p からのメッセージを v

⋆1 メッセージ名は参考文献 4) に従っている.

のメンバピアに配送する方法が問題となる．

この問題はアプリケーションレベルマルチキャスト (ALM) を用いて解決した（ALM は PIAX の持つ機能で，構造化 P2P ネットワークの 1 つである Skip Graph[5] を利用して実装されている）．仮想ピア v の全メンバピアは，仮想ピアの ID($v.id$) をキーとする 1 つのマルチキャストグループに所属する．p から v へメッセージを送信する際には，宛先に $v.id$ を指定してマルチキャスト送信する．これにより p は v のメンバピア構成を知らなくても v の全メンバピアにメッセージを送信できる．

3.1.2 再送の必要性

2.2 節で述べたように，仮想ピアでは各メンバピア間でメッセージの受信順序を統一するため，受信したメッセージを一度 Paxos を用いて提案する必要がある．このため，リーダピアが ALM 経由でメッセージ m を受信すると，Paxos を用いて m を提案する（非リーダの場合は単に破棄する）．各メンバピア上で動作するプロセスは，Paxos のシーケンス番号の順番に合意されたメッセージを受信し，処理する．

しかし，p が v に送信したメッセージは必ずしも合意されるとは限らない．合意されない原因としては，例えば (1) リーダに届く前にネットワーク側でメッセージが喪失する，(2) リーダに届いた場合でも，リーダが提案する前にリーダが離脱してしまう，などが考えられる．

このため，p は v からの応答を受信するまで定期的にメッセージを再送する．また，p は送信したメッセージを v からの応答を受信するまで記憶しておく．

3.1.3 メッセージ ID の必要性

p が再送したメッセージと，v からの応答がすれ違う可能性があるため，不必要な再送が発生する可能性がある．

1 回しか送信されていないメッセージを仮想プロセスが複数回処理することを避けるため，仮想ピア宛に送信されるメッセージには ID（メッセージ ID）を付与する．これにより，仮想ピアは受信したメッセージが新規のものか再送されたものかを区別する．

メッセージ ID を一意にするためには，p のピア ID（ピアを識別するために PIAX によって付与される識別子）とメッセージのシーケンス番号を組み合わせればよい．

また，p が応答メッセージを受信した際は，その応答がどの送信メッセージに対応するものかを判断する必要がある．メッセージ ID はこのためにも使用する（v がメッセージ m に対して送信する応答メッセージには m のメッセージ ID を埋め込む）．

3.1.4 通常ピアへの応答の送信

v 上の仮想プロセスが m を処理し，p に応答を送信する場合，メンバピア上の各プロセスで応答を送信するための API が呼び出されるが，全てのメンバピアが応答を送信するとトラフィックの無駄であるため，応答を実際に送信するのはリーダピアだけに限定する．

3.1.5 応答の再送

v のリーダピアから送信された m に対する応答は，p に到達せずに喪失する可能性がある．この場合，p は m を再送するが（3.1.2 節参照），再送された m を受け取った仮想プロセスでは実際は m の処理は完了していることに注意する必要がある．

この場合，仮想プロセスが再度メッセージを処理するのではなく，応答のみを p に再送しなければならない．このため，v では送信した応答を一定時間記憶しておく必要がある．

3.1.6 提 案 方 式

以上を踏まえ，以下の方式を提案する．

送信側の動作

送信側の通常ピア p は，まずメッセージ m のメッセージ ID を生成する．次に仮想ピア v の ID($v.id$) をキーとするマルチキャストグループに対して m をマルチキャスト送信する．

一定時間待っても v からの応答がない場合，m を再送する．

受信側の動作

受信側の仮想ピアの処理は，実際は仮想ピアの各メンバピア上の処理になる．

受信側では，同一のメッセージを複数回処理しないように，受信したメッセージの状態（以下の 3 つ）を記憶するための表 *history* と，応答を記録しておくための表 *reply* を用いる．これらの表はメッセージ ID をキーとするハッシュ表として実装する．

INITIAL　まだメッセージを受信していない（初期値）．

PROCESSED　プロセスにメッセージの処理を依頼済みである．

REPLIED　応答を送信済みである．

以下に受信側アルゴリズムを示す．

(1) ALM 経由でメッセージ m を受信した場合，リーダピアは以下の処理を行う（リーダでない場合は単に無視する）．
  (a) *history* から m の状態 (*state*) を得る．
  (b) *state* が INITIAL ならば，初めて受信するメッセージなので，Paxos を用いて m を提案し，終了．
  (c) *state* が REPLIED ならば，*reply* に記録しておいた応答を m の送信ピアに送

信し，終了．

（d） *state* が PROCESSED ならば，無視して終了．

（2） Paxos で m が合意されたという通知があった場合，全てのピアで以下の処理を行う．

（a） *history* から m の状態 (*state*) を得る．

（b） *state* が INITIAL でないならば，何もせずに終了する（再送されたメッセージが提案され，合意された場合である）．

（c） *state* を PROCESSED に変更する．

（d） メンバピア上で動作しているプロセスに m の処理を依頼して，終了．

（3） プロセスから m に対して応答 (*res*) を送信する API では，以下の処理を行う（各メンバピアで実行される）．

（a） m の状態 (*state*) を REPLIED に変更し，*history* に記録する．

（b） *reply* に，*res* を記録する．

（c） リーダピアは，*res* を m の送信ピアに送信する（非リーダピアでは送信しない）．

上記のアルゴリズムでは，*history* と *reply* のエントリ数は減ることがない．この問題に対しては，(1) p が v に m を送信した場合，p が m に対する応答を受信したことが確認できれば v は m に関するエントリを削除できるため，p から v に引き続いて送信するメッセージ m' を用いて m の応答受信完了を通知する，(2) p がメッセージを再送できる最大時間 max を設定する．受信側ではメッセージの状態を REPLIED に変更してから max よりも十分長い時間経過したらエントリを削除する，などの方法が考えられる．

3.1.7 メッセージシーケンスの例

通常の場合（仮想ピアのリーダは離脱せず，メッセージも喪失しない場合）のメッセージシーケンスを**図 5** に，メッセージの喪失により通常ピアから仮想ピアへのメッセージ再送が発生する場合のメッセージシーケンスを**図 6** に示す．なお，図中の太破線はメンバピア上のプロセスがメッセージを処理している時間を表している．図 6 では一つのメッセージが 2 回提案されているが，各プロセスが処理する回数は 1 回だけとなっている．また，ピア 2 には *Success(m, 100)* よりも先に *Success(m, 101)* が到着しているが，2.2.2 節で述べたように，Paxos のシーケンス番号が欠落した場合には埋まるまで待機するため，*Success(m, 100)* が到着してから m が処理される．

3.2 仮想ピア間の通信方式

次に，仮想ピア間で通信する際の処理を検討する．仮想ピア s が仮想ピア r にメッセージ m を送信し，r が s に応答を返すものとする．

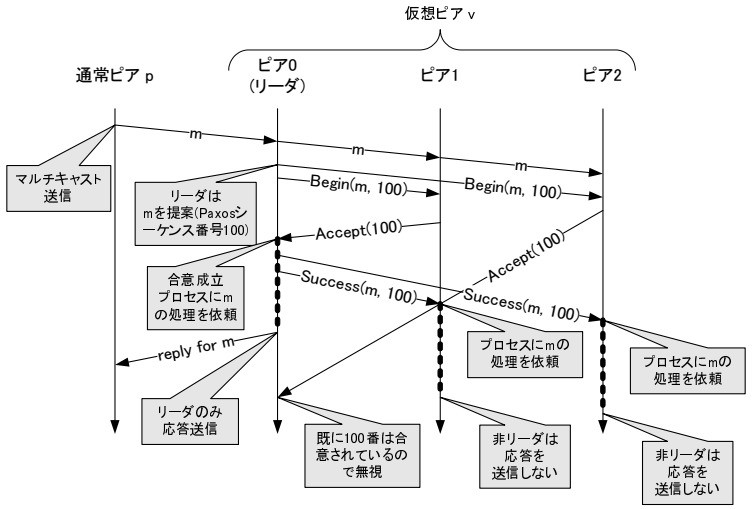

図 5 通常ピアと仮想ピアとのメッセージシーケンス例（再送が発生しない場合）

仮想ピア（仮想プロセス）は受信メッセージの系列を合意しながら処理することで動作する．このため，仮想ピア s が r にメッセージを送信する場合，メッセージ送信を引き起こした受信メッセージ（これを M とする）が存在するはずである．M の送信ピアは通常ピア，仮想ピアの別を問わない．また，M は s が送信したメッセージに対する応答メッセージでもよい．s の各メンバピアで動作するプロセスは，M を受信したことにより r に m を送信するための API を呼び出す．

仮想ピア間での通信処理は，3.1 節で述べた通常ピア・仮想ピア間での通信方式がベースとなる（s のリーダピアが，送信側の通常ピア p に相当する）が，さらに以下の点を考慮する必要がある．

3.2.1 応答の送信方法

s のリーダピア s_l が r に ALM で m を送信してから，応答を受信する前に s のメンバ構成が変化する可能性がある．このため r からの応答も ALM で送信する．

s_l が r にメッセージを送信してから r からの応答を受信する前に s_l が離脱する可能性はそれほど高くないと思われるため，応答は時間のかかる ALM で送信するのと同時に，s_l に直接送っておくと応答時間を短縮できる．この場合 s_l は応答を 2 回受信する可能性がある

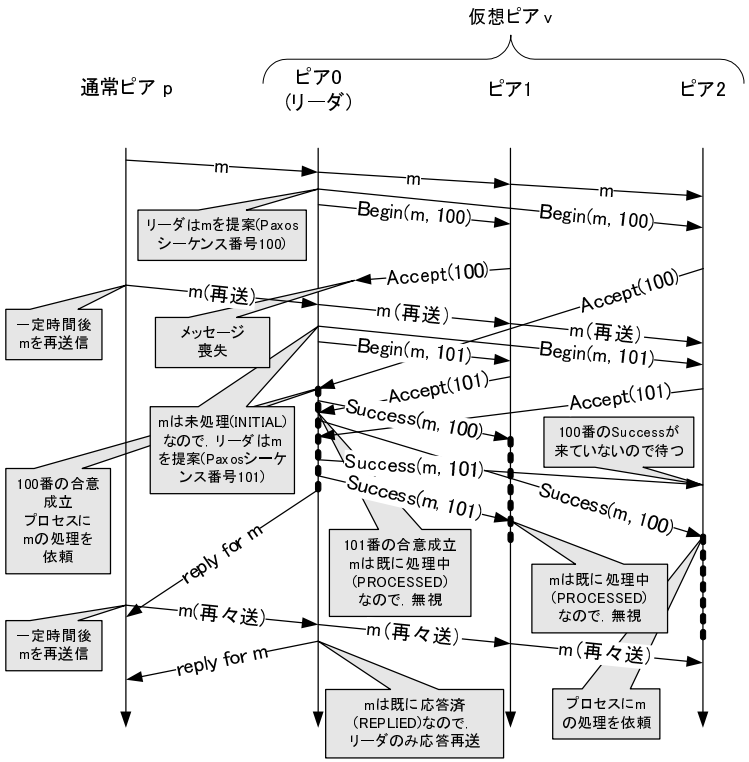

図 6 通常ピアと仮想ピアとのメッセージシーケンス例（再送が発生する場合）

が，2 つめの応答は無視する．

3.2.2 応答の合意の必要性

一般に，s 上で動作する仮想プロセスは r から受信した応答メッセージを処理すると内部状態が変化する．このため，s の各メンバピアでは，応答メッセージも他のメッセージと同様に扱い，受信順序を統一しなければならない．つまり，s のリーダは応答メッセージを受信したら，一度 Paxos を用いて提案する必要がある．応答メッセージは提案が合意されてから処理する．

3.2.3 メンバ構成の変化

s で M を合意した後，r から応答を受信する前に s のリーダ s_l が離脱する場合を考える．以下のシナリオが考えられる．

- r に m が送信される前に s_l が離脱する場合: さらに以下の 2 通りに分類できる．
  （1） s の次のリーダ（s'_l とする）がリーダになる前に M を処理している（M の Success メッセージを受信している）場合: s'_l が M を処理したときはリーダではなかったため，r に m を送信する API を実行したとき，実際はメッセージを送信していない．
  しかし，s'_l はリーダになったため，r からの応答待ちのタイムアウトにより今度は m を実際に送信する必要がある．このため，送信 API を実行したときは自分がリーダかどうかに関わらず送信メッセージを記憶しておく必要がある．
  （2） s'_l がリーダになってから M を処理する場合: この場合，リーダになってから M を処理するため，r に m を送信する．
- r に m が送信された後に s_l が離脱する場合: さらに以下の 2 通りに分類できる．
  （1） r からの応答を s'_l が受信する場合: この場合，問題なく s'_l は応答を Paxos で提案する．
  （2） r から応答が送信された時点で s_l は離脱しているが，次のリーダがまだ存在しない場合: この場合，s で応答は提案されず，喪失することになる．次のリーダ (s'_l) が決まると，s'_l は応答を得るために m を再送する必要がある．

なお，メンバピアでは受信したメッセージの処理は Paxos のシーケンス番号順に行うため，M の処理を行う前に m の応答を処理することはない（m の応答も合意されてから処理されるため，m の応答のシーケンス番号は M のシーケンス番号より必ず大きくなる）．

3.2.4 提 案 方 式

以上を踏まえ，以下の方式を提案する．

送信側の動作

送信側の仮想ピア s の処理は，実際は仮想ピアの各メンバピア上の処理になる．

- プロセスから仮想ピア r に m を送信する API では，以下の処理を行う（各メンバピアで実行される）．
  （1） メッセージ m のメッセージ ID を生成する．全てのメンバピアで m のメッセージ ID を統一する必要がある．このため，メッセージ ID 生成に用いるピア ID には，各メンバピアのピア ID ではなく，仮想ピアのピア ID を用いる．
  （2） リーダピアは，仮想ピア r の ID($r.id$) をキーとするマルチキャストグループに対して m をマルチキャスト送信する（非リーダピアでは送信しない）．

(3) 再送のために m を記憶し，タイマをスタートする．
(4) タイマが満了しても r からの応答がない場合，リーダピアは m を再送する（非リーダピアでは再送しない）．

一連の処理を実行中に s のリーダが交替する場合，新しいリーダは上の (4) で再送することになる．

- 仮想ピア s のリーダピアが r から応答 (res) を受信した場合，3.1.6 節で述べた受信側アルゴリズムの (1)，(2) とほぼ同様の方法で res を処理する（ただし (2) の (c) の後，タイマの停止処理を追加する）．

受信側の動作

受信側の仮想ピア r におけるメッセージ受信処理は，3.1.6 節で述べた受信側アルゴリズムとほぼ同様であるが，応答を送信する際には仮想ピア s の ID($s.id$) をキーとするマルチキャストグループに対して応答をマルチキャスト送信する点が異なる．

3.2.5 メッセージシーケンスの例

通常の場合（s のリーダが離脱しない場合）のメッセージシーケンス例を**図 7** に，r に m が送信された後に s のリーダが離脱する場合のメッセージシーケンス例を**図 8** に示す．図 8 では新リーダになった $s1$ は M をシーケンス番号 100 で提案しているが，これは，Paxos アルゴリズムにおいて，Last メッセージで提案されたが合意されていないメッセージが通知された場合，リーダは同じシーケンス番号でそのメッセージを再提案しなければならない[3),4)] と規定されているためである．

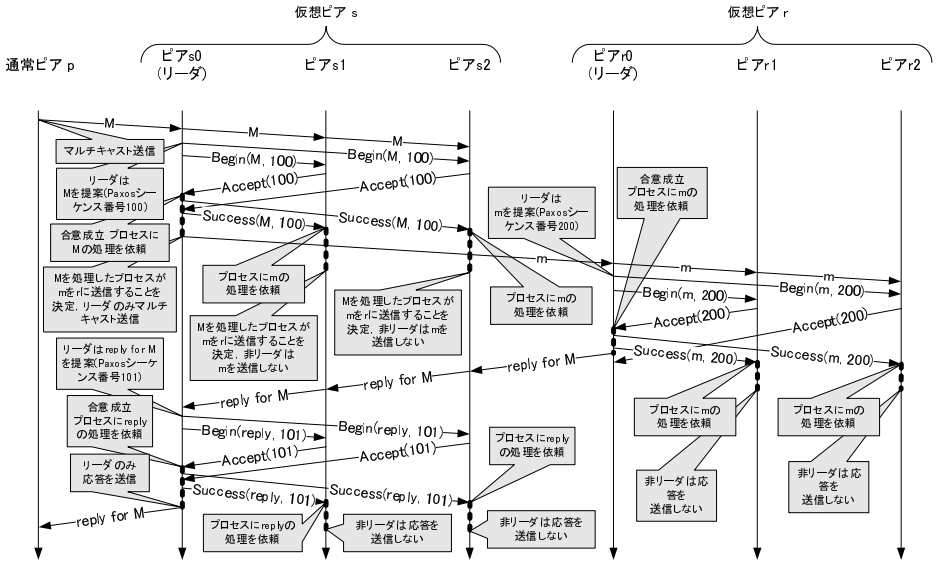

図 7 仮想ピア間のメッセージシーケンス例（s のリーダが離脱しない場合）

4. 考　　察

インターネット上のノード間で直接メッセージを送受信した場合の片方向遅延時間を t とする（ラウンドトリップタイム (RTT) は $2t$）．このとき，提案方式での RTT（通常ピア，あるいは仮想ピアが仮想ピアにメッセージを送信し，応答を受信するまでの時間）を考察する．なお，通信中にリーダが離脱したり，メッセージが喪失することはないものとする．

4.1 通常ピアと仮想ピア間の RTT

通常ピア p が仮想ピア v と通信する場合の RTT を求める．

p は ALM を用いて v のメンバピアにメッセージを送信するが，PIAX の ALM は Skip Graph を用いてルーティングするため，メッセージがメンバピアに到達するには平均 $\log_2(n)$ ホップ必要である（n は Skip Graph に参加しているピアの数，ホップ数はオーバレイネットワーク上でのホップ数）．メッセージがリーダピアに到達すると，リーダピアは Paxos で提案するために自分を除く全てのメンバピアに Begin を送信し，過半数のメンバピアからの Accept を待つ．メンバピア間のネットワークに十分な帯域があり，過半数のメンバピアの応答速度が十分速ければこの処理はほぼ $2t$ 時間で実行できる．リーダピアから送信した応答は t で p に到達する．これらを合計すると，通常ピアと仮想ピア間での RTT は，$(\log_2(n)+3)t$ となる．

ALM のオーバヘッドを削減するには次の方法が考えられる．v のリーダピアから p に送信する応答の中に v のメンバ構成の情報を含める．p はこの情報を記憶することで，以後の通信では ALM を使用せず，直接 v の全てのメンバピアにメッセージを送信する．このときの RTT は $4t$ となる．メンバピアの構成は変化する可能性があるが，p が送信した相手にリーダピアが含まれていれば問題ない．相手にリーダピアが含まれない場合でも，送信相手の中にメンバピアが含まれていれば，新しいリーダを p に教えることは可能である．送信相手にメンバピアが全く含まれない場合，通信エラーやタイムアウトにより ALM を利用するようにフォールバックする．

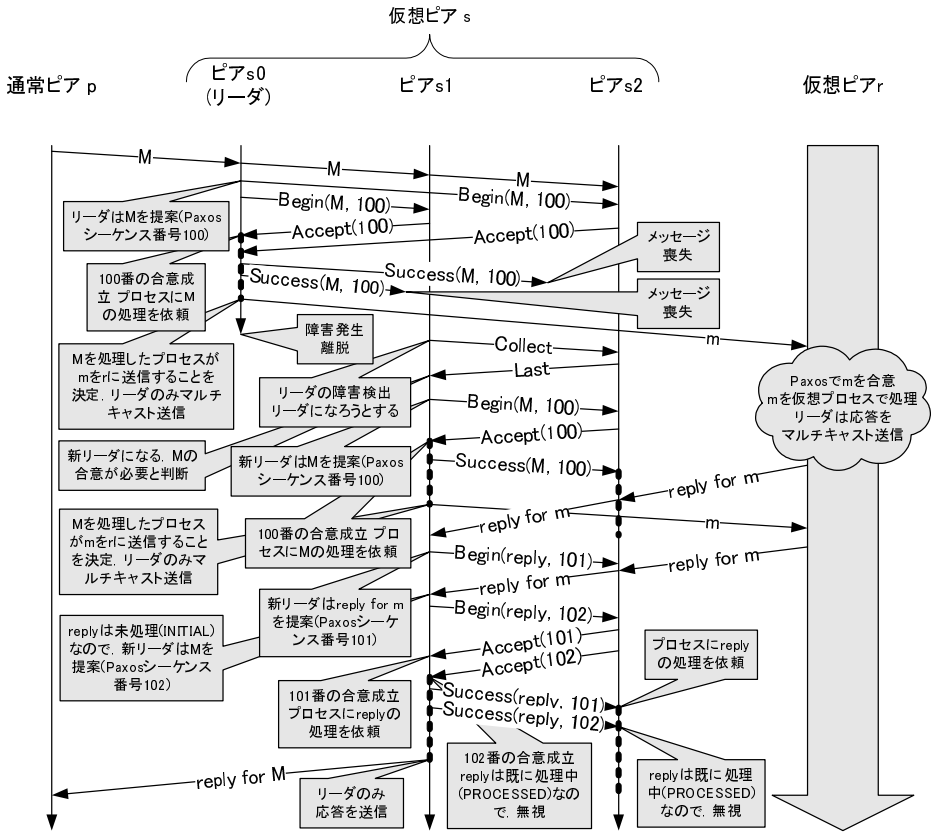

図 8 仮想ピア間のメッセージシーケンス例（r に m が送信された後に s のリーダが離脱する場合）

4.2 仮想ピア間の RTT

M が合意された後で，仮想ピア s のリーダピアが仮想ピア r にメッセージを送信し，応答を受信するまでの RTT を上と同様に求めると，$(2\log_2(n)+2)t$ となる．

ALM のオーバヘッドを削減するために，3.2.1 節で述べた方法を用いると，RTT は $(\log_2(n)+3)t$ となる．さらに，4.1 節で述べた方法を適用すると，s が r と 2 回目以降に通信する場合の RTT も $4t$ に削減できる．

5. おわりに

本稿では，P2P 基盤ソフトウェア musasabi の仮想ピアにおける通信方式について述べた．仮想ピアは複数のノードから構成され，構成も動的に変化するが，提案方式を用いることで仮想ピアと通常ピア，あるいは仮想ピア間でメッセージを交換することができる．仮想ピアと通常ピア間，および仮想ピア間での RTT は，2 回目以降の通信で ALM の使用を抑制することで，通常のノード同士での RTT の倍程度に抑えられることを示した．

現在，仮想ピアと通常ピア間の通信処理の実装がほぼ完了している（*history* や *reply* などのエントリの削除，2 回目以降の通信を効率化する方式は未実装）．

今後の課題としては，仮想ピア間の通信処理と 2 回目以降の通信の効率化，実環境（インターネット）上での性能測定などが挙げられる．

謝辞 本研究の一部は科研費 (18700069) 及び独立行政法人情報通信機構「高度通信・放送研究開発委託研究」の助成を受けている．

参考文献

1) 安倍広多. P2P システム上での安定したサービス提供基盤 musasabi. 情報処理学会研究報告, Vol. 2009-IOT-4, pp. 131–136, 2009.
2) 吉田幹ほか. マルチオーバレイと分散エージェントの機構を統合した P2P プラットフォーム PIAX. 情報処理学会論文誌, Vol.49, No.1, pp. 402–413, 2008.
3) Leslie Lamport, etal. The part-time parliament. *ACM Trans. on Computer Systems*, Vol.16, pp. 133–169, 1998.
4) Roberto De Prisco, etal. Revisiting the Paxos algorithm. In *Proc. of 11th Int. Workshop on Distributed Algorithms (WDAG 97)*, pp. 111–125. Springer-Verlag, 1997.
5) James Aspnes, etal. Skip graphs. *ACM Trans. on Algorithms*, Vol.3, No.4, pp. 1–25, 2007.