2005－MPS－56（4）
2005／9／21

インターネットエージェントシステムのための実用的な耐ビザンチン故障方式

櫟　粛之[1], KK. Senthil Kumar[2]

[1]日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
[2]株式会社 ユニグレイト

あらまし　本論文では，エージェントシステムに対する実用的な耐ビザンチン故障方式を提案する．これまでにサーバ・クライアントシステムに対して，効率の良い耐ビザンチン故障方式が Castro と Liskov によって提案されている．ここではその方式の基本となる replica 間の３フェーズ合意プロトコルを利用して，エージェントシステムに対応した耐故障方式を与える．Castro-Liskov のサーバ・クライアントシステムの耐故障方式では，クライアントは故障しないことを前提とするが，エージェントシステムの場合，通信の両側の故障を許す．また，エージェントの自律的動作特性に対処するため，どの時点で，いままで受信したメッセージを処理するかについての replica 間の合意も導入する．

Practical Byzantine Fault Tolerance for Internet Agent Systems

Tadashi Araragi[1], KK. Senthil Kumar[2]

[1] NIPPON TELEGRAPH AND TELEPHONE CORPORATION, NTT Communication Science Laboratories
[2]UNIGRATE

Abstract: This paper presents a Byzantine fault tolerance method for Internet agent systems. We extend Castro and Liskov's well-known practical Byzantine fault tolerance method for the server-client model to a method for the agent system model. There are two main differences between the methods. First, in agent systems we have to create replicas on both sides of the communicating agents, while in the server-client model of Castro and Liskov's method, replicas are created only on the server side, and the client is assumed to be non-faulty or is treated differently from a replica model. This two-sided replica model complicates the synchronization protocol. Second, due to the autonomous behavior of agents, we have to synchronize the timing of the receiving of messages among replicas. Agents decide their actions based on their current state of knowledge and do not wait indefinitely for messages that may not reach them.

1. まえがき

インターネット上で, 人間の代わりになって様々なタスクを実行するソフトウェアをインターネットエージェントと呼ぶ[1]．現在，このようなエージェントを相互に接続して世界規模のサービスを展開する試みが進められている（FIPA, agentcities）．一方，インターネットのようなオープンな環境では, 悪意のある侵入者がホストを乗っ取り, そこで働くエージェントを自由にあやつる危険性が大きい．このように，ホストの障害・乗っ取りにより，その上のシステムが本来の意図とは別の動作を行う障害を, 一般的にビザンチン故障と呼ぶ．従来，対応が難しいとされていたビザンチン故障に対し[3]，近年，Castro と Liskov は, メッセージ遅延に関する現実的な仮定のもとに，primary replica と view change(primary change)の概念を導入してサーバ・クライアント型の非同期システムに対する実用的な耐ビザンチン故障方式（BFT）を導入した[2].

本論文では, インターネット上で動作するエージェントに対して耐ビザンチン故障方式を与える．そこでは，Castro-Liskov の耐ビザンチン故障方式で提案された合意プロトコルを利用する. サーバ・クライアントシステムを前提とした Castro-Liskov の方式では, 通信の片側（サーバ側）にしか故障を仮定しなかったのに対し, 我々のエージェント向けの耐ビザンチン故障方式では, 通信の両側のエージェントに故障を許す.

2. Castro-Liskov の耐ビザンチン故障方式の概要

Castro-Liskov の方式ではサーバの replica を作り，それぞれ異なったホストで動作させる．高々f 個のホストが乗っ取りなどでビザンチン故障することを前提に，１つのサーバに対して 3f+1 個の replica を作る．これらの replica は，各々が違った順序で受信した可能性のあるさまざまなリク

エストに対して，replica 間で合意をとり（正しい合意のために 3f+1 個が必要），共通の sequence number をこれらリクエストに割り当て，その順番でリクエストを処理することで同一の動作を維持する．これにより，一部のエージェントが乗っ取られても（高々f 個），f+1 個の一致する動作から，本来の正しい動作を判別することができる．Castro-Liskov の方式ではこの合意手続きをコントロールする primary と呼ばれる特別な replica が用意される．それ以外の replica は backup と呼ばれる．もし，primary が故障を起こしたかもしれないと判断されたら backup の合意のもと，別の backup が primary になる（view change）．

クライアントと replica の間の通信

(1) クライアントはサーバの現在の primary にリクエストを送信する．

(2) クライアントは，サーバの f+1 個の異なる replica から，同じ結果を受け取ったとき，それを正しい結果として受理する．もしそのような f+1 個の結果を決められた時間内に受け取れなかったら，クライアントはサーバの replica 全体に直接もとのリクエストをマルチキャストする．

合意プロトコル：３フェーズ

(1) primary はリクエストを受け取ると，そのリクエストに sequence number を割り当て，全ての replica にそのリクエストと sequence number の情報を含む PRE-PREPARE メッセージをマルチキャストする．

(2) replica が PRE-PREPARE メッセージを受信したとき，PRE-PREPARE と同じ内容を含んだ PREPARE メッセージを全ての replica にマルチキャストする．

(3) replica が valid な 2f+1 個の同じ PREPARE を異なる replica から受信したとき，replica はそのリクエストと sequence number に対して prepared 状態になる．そして，replica は PREPARE と同じ情報を含む COMMIT メッセージを全ての replica にマルチキャストする．

(4) replica が 2f+1 の COMMIT メッセージを異なる replica から受信したとき，replica はそのリクエストと sequence number に対して committed 状態になる．そして，replica はリクエストを実行し，その結果をクライアントに返す．

View change

各 replica は，クライアントから，間接あるいは直接にリクエストを受信したとき，そのリクエストに対する timer を起動する．そしてその timer が終了したとき，replica は現在の primary を変更しようとする(view change)．この時，replica は VIEW-CHANGE メッセージを全ての replica にマルチキャストする．このメッセージには次の view で現状を無矛盾に回復する情報が含まれている．replica が 2f+1 個の VIEW-CHANGE メッセージを受信したとき，集めた情報から失われた情報を再現し，次の view に移動，即ちローカルに次の primary を決める．

３フェーズの合意プロトコルにより，どの replica も，同じリクエストに対して異なった sequence number で prepared 状態になることはない．さらに，もしある replica が committed 状態に達したら，view が変わって，別の replica が，そのリクエストに対して committed 状態に達したとき，その sequence number は同じであることが保障されている．これらの性質は Byzantine quorum の概念を用いて証明されている．

3．エージェントシステムの BFT

3.1 メッセージの sequence number に対する合意プロトコル

メッセージに共通の sequence number を割り当てるという問題は Castro-Liskov の BFT と同じであるが，問題の前提が大きく異なる．

エージェント BFT の基本的な枠組み

エージェントシステムでは，各エージェントは同等の立場にあり，サーバやクライアントなどの区別はない．エージェントは，ある時はサーバとなり，またある時はクライアントとなる．さらにエージェントがやりとりするメッセージをリクエストやリプライに分類することはできないし，エージェントは全てのリクエストに応答する必要は無いし，通常のリクエスト，リプライを別の用途に利用することもある．このような状況で，我々のフレームワークでは，エージェントのメッセージ送信を Castro -Liskov の BFT でのリクエストと捉え，それに対する ack の送信をリプライと捉える．どのエージェントも faulty になりうるので，図１に示すような形で各エージェントの replica を作る．

下記のプロトコルでの本質的なアイデアは，メッセージ送信するエージェントの non-faulty の replica は，共同で受信エージェントの view change に責任を持つことである．

エージェント間の通信

エージェント A，B の通信，特にエージェント A

からエージェント B へのメッセージの送信は以下のようになされる．

(1) エージェント A の replica はエージェント B の現在の primary にメッセージ msg を送信する．そして，自分のタイマー(*sndr timer* と呼ぶ)をスタートさせる．

(2) A の replica が sndr timer が終了する前に f+1 個の *msg* に対する ack を B の異なる replica から受信したら，その ack を受理する．

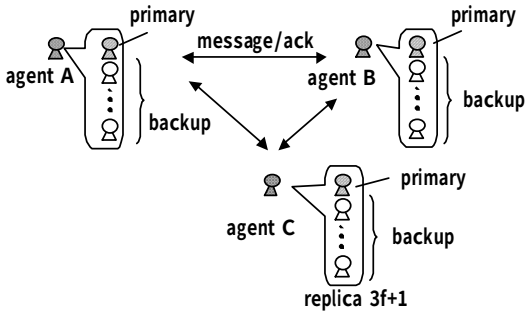

図 1 : replica の枠組み

(1) B の primary が f+1 の同じメッセージ *msg* を A の replica から受け取ったとき，その primary は *msg* に sequence number を割り当て，対応する PRE-PREPARE メッセージを B の全ての replica にマルチキャストする．ここから Castro-Liskov の 3 フェーズ合意プロトコルを始める．B の replica は，*msg* を受信したとき，その *msg* に対するタイマー(*rcvr timer*)をスタートさせる．

(2) 3 フェーズ合意プロトコルで，committed 状態になった replica は，エージェント A の全ての replica に *msg* の ack をマルチキャストする．

View change

(1) A の replica が，*sndr timer* が終了する前に，B の replica たちから f+1 個の ack を受信できなかった場合，その replica は，B の全ての replica に元のメッセージをマルチキャストする．

(2) B の replica が，その primary から PRE-PREPARE メッセージを受信する前に，f+1 個の A の異なる replica からその PRE-PREPARE メッセージに対応する同じメッセージを直接受け取ったら，replica は *rcvr timer* をスタートさせ，受信したメッセージを primary に転送する．

(3) B の replica の *rcvr timer* が終了したとき，その replica は Castro-Liskov の BFT の場合と同様の view change プロトコルを開始する．

エージェント間の通信リンクにおける FIFO 性を保つため，replica は，メッセージの送り先のエージェントが ack を返すまで (即ち，そのエージェントの f+1 個の replica が ack を返すまで)，そのエージェントに次のメッセージを送信しない．さもなければ，faulty replica は，メッセージの受信順序を変えることができてしまう．

3.2 メッセージ受信のタイミングに関する合意

最初に「メッセージ受信のタイミングに関する合意」が対象とする問題を説明する．下記の(エージェント agt1 の)単純なエージェントプログラムを考える．

procedure{

L1: if m1 ∈ MB then send(agt2: m3), rm(m1);

L2: if m2 ∈ MB then send(agt3: m4), stop;

L3: goto L1;}

このプログラムの実行は，最初の行である L1 からスタートする．まず *agt1* は，自分がメッセージ *m1* を受信しているかどうかをチェックする．もし受信していたら，*agt1* は，エージェント *agt2* にメッセージ *m3* を送信し，自分のメッセージベース MB から *m1* を取り除く．そして，行 L2 へ実行を移す．そこでは，まず，自分がメッセージ *m2* を受信していないかをチェックし，受信していたら *agt3* にメッセージ *m4* を送信し，実行を終える．L2 の時点で *m2* を受信していなければ，L3 に移動し，そこから L1 に再び戻って同じ処理を繰り返す．このプログラムでは，*agt1* はメッセージ *m1* を将来必ず受け取る確信が無く，そのためにこのプログラムでは *agt1* は *m1* を永久に待ち続けることはしない記述になっている．一方，メッセージ *m2* は必ず受け取る確信があり，それを受信するまで待つプログラムになっている．このプログラムの実行は *agt1* が *m2* を受信するまで L1, L2, L3, L1, L2, L3, L1, . . .と進んでいく．このプログラムは，エージェントの自律的な振る舞いの一端を示している．即ち，*agt1* は，他のエージェントがメッセージ *m1* をいつかは送信してくれるという前提に依存することなく振舞っている．

次に，上記エージェント *agt1* の 2 つの replica *rp1* と *rp2* を考える．各々メッセージ *m1*，*m2* を受信するとする．*rp1*，*rp2* はこのメッセージの受信順序を *m1* が最初，*m2* がその次と合意したとする．*rp1* が L1 から実行を開始し，L1 と L2 の間で *m1* と *m2* を受信したとする．すると *rp1* は L2 で *m4* を送信して実行を終了する．一方，*rp2* は L1，L2 と進んだあと *m1* を受信し，L3，L1 と実行を進め，その後 *m2* を受信したとする．すると *agt1* は 2 回目の L1 で *m3* を送信し，さらに L4

で *m4* を送信し，実行を終わる．このように，２つの replica は，メッセージの受信順序を同一にしても異なった動作をすることがあり，replica モデルの BFT の要求を満たさない．しかし，もしここで，２つの replica が *m1* の受信処理を２回目の L1 まで延期し，*m2* の受信を２回目の L2 まで延期したら２つの replica の動作は一致する．我々のプロトコルはこの同期を実現する．

このメッセージ受信のタイミングに関する同期処理を実現するために我々は receiving point の考え方を導入する．各 replica は replica 間で共通に決められた周期ごとの実行ステップでプログラムの実行を止める．この停止ポイントが receiving point である．各 receiving point で，non-faulty な replica はそれまでに受信したメッセージの中で, どのメッセージを本当に受信したことにするかの合意を取る．そして，合意の取れたメッセージだけ, 各々のメッセージベースに納め, 次の receiving point まで利用する．例えば，図 2 において，

replica は *msg1* の受信に関して合意を取るかもしれない．また故障システム数の上限 f = 1 の場合，replica は *msg1* と *msg2* の受信で合意を取ることもある．このとき，*replica4* はまだ *msg2* を受信していないが, 他の３つの replica は *msg2* を受信したと伝えられるから合意ができ，*replica4* は，次の実行を始める前に *msg2* の到着を待つ．

しかしながら，replica は *msg1*，*msg2*，*msg3* の３つを受信したと合意を取ることはできない. なぜなら，今の時点で *msg*3 を受信しているのは *replica1* だけであり，他の replica は *replica1* が faulty で, *msg3* を受信したというのは嘘かもしれないと考えるからである．

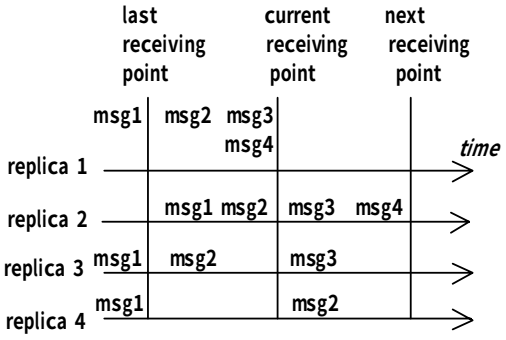

図２: メッセージ受信タイミングの合意プロトコル

基本的なアイデアは以下の通りである. 各 replica は受信したメッセージをその順序に関して合意を取って，即ち sequence number を割り当てて，仮のメモリに蓄える．各 receiving point で，どのメッセージを MB に移すかを replica 間で合意をとる．この時，primary はどの sequence number までのメッセージを受け取っているか，各 replica から聞き，2f+1 個の任意の返信から，多数決でメッセージベースを update するかを判断し，update する場合には, 受信が保障できる sequence number を定め（たとえば，*r* 個の replica が新しいメッセージの受信を報告したとき, 報告された sequence number の中で，*r-f* 番目に大きいもの），その根拠となる replica メッセージとともに，結果を replica にマルチキャストする．

また，view change に関しては，primary が最初のメッセージベース update の処理を怠ることに対応して, あらたなタイマーとコミットメントのフェーズが必要になるが，詳細は省略する．

3.3 正当性

上記のプロトコルにより下記の性質を保障することができる．

Safety: faulty replica がどのように振舞おうと non-faulty replica は同じ受信メッセージに対して，同じ sequence number を割り当てる．また，各 receiving point で，同じ valid sequence number に同意する．従って，エージェントシステムのモデルに従うと, non-faulty replica は, もし動作をブロックされていなければ, 互いにまったく同じ動作をする．

Liveness: 少なくとも f+1 個の non-faulty replica は，ブロックされることはない．

参考文献

[1] T. Araragi, P.C. Attie, I. Keidar, K. Kogure, V. Luchangco, N.A. Lynch, and K. Mano, “On Formal Modeling of Agent Computations,” Formal Approaches to Agent-Based Systems: First International Workshop, FAABS, LNCS 1871, pp.48-62, 2000.

[2] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” *In Proc. of the 3rd Symp. on Operating Systems Design and Implementation*, pp. 173-186, 1999.

[3] M. Fisher, N. Lynch, and M. Paterson, “Impossibility of distributed consensus with one faulty process,” Journal of the ACM,32(2), pp. 374-382, 1985.