通信リンクのブロックによる移動故障エージェントの封じ込め

半澤　陽 *†　　山内　由紀子 ‡§

概要

本研究では，分散システム内を移動する移動故障エージェントがプロセスを故障させる状況を想定し，ビザンチン故障を引き起こすビザンチン故障エージェント，停止故障を引き起こす停止故障エージェント，一時故障を引き起こす一故障エージェントを考える．各プロセスが通信リンクをブロック（受信拒否）または切断（受信，送信拒否）することによって，移動故障エージェントを高々 2 個のプロセスに封じ込める手法を提案し，さらに，移動故障エージェントを封じ込めた後に通信グラフの連結性を回復するための条件とその手法を示す．

1　はじめに

大規模な分散システムは，多数のプロセスや通信リンクから構成されるため，プロセスや通信リンクの故障を回避することができない．そのため分散システムの一部が故障を起こしても正常に動作することができる故障耐性について様々な研究が行われている．

プロセスの内部変数を任意に書き替える**一時故障**に対する故障耐性としては，自己安定性が提案されている [1]．**自己安定プロトコル**は，任意のシステム状況からプロトコルを開始しても目的のシステム状況に到達することを保証する分散アルゴリズムである．

Pease らは，故障プロセスが任意に振る舞う**ビザンチン故障**が存在する場合に，プロセス間で合意を形成する合意形成問題である**ビザンチン合意問題**を提案した [4]．Garay はプロセスにビザンチン故障を起こす**移動ビザンチン故障エージェント**がプロセス間を移動することで，ビザンチン故障のプロセス集合が変化する**移動ビザンチン故障**を提案し，その故障の下で合意形成を行う**移動ビザンチン合意問題**を提案した [2]．この論文では，プロセス自身が故障していたことを認識できる**故障感知性**をもち，常に正常なプロセスが少なくとも 1 つ存在するという仮定の下，$n > 6t$ の場合に移動ビザンチン合意問題を解く分散アルゴリズムが示された．

移動故障エージェントがプロセス間を移動し続けることで全プロセスが故障する可能性がある．そこで，移動故障エージェントの移動を分散システムの一部に制限することで，故障する可能性のあるプロセスを削減できる可能性がある．Inoue らは，移動故障エージェントを封じ込めるために，故障直後のプロセスが移動故障エージェントの移動先のプロセスを認識できる**探知可能モデル**，移動先のプロセスを認識できない**探知不可能モデル**を導入し，探知可能モデルの下でプロセスが隣接プロセスからのメッセージを受信しない通信リンクの**ブロック**を複数行うことで，単一の移動ビザンチン故障を分散システムの一部に封じ込めながら，分散システム上に全域木を構成する自己安定プロトコルを提案した [3]．しかし，Inoue らの手法では通信グラフの強連結性が保証されておらず，故障エージェントを封じ込めた後，プロセス間で双方向に通信を行うことができない．

本研究では，通信リンクのブロック，切断を用いて移動故障エージェントの封じ込めについて考える．ここで，**通信リンクの切断**とは，プロセスがある隣接プロセスにメッセージの送信，受信を行わない状態である．さらに，封じ込め後，通信グラフの強連結性を保証するために，ブロックした通信リンクを解除することで連

* 九州大学大学院システム情報科学府
† hanzawa@tcs.inf.kyushu-u.ac.jp
‡ 九州大学大学院システム情報科学研究院
§ yamauchi@inf.kyushu-u.ac.jp

結性を回復する．本研究では MBA に加え，移動故障エージェントとして新しく 2 つの移動故障エージェントである移動一時故障エージェント (MTA)，移動停止故障エージェント (MCA) を新しく導入する．MTA は滞在するプロセスに一時故障を起こさせ，MCA は滞在するプロセスに送信，受信，内部計算を行わない**停止故障** [5] を起こさせる．MBA については，通信リンクのブロックを用いることでは MBA を封じ込めることができないことを示し，通信リンクの切断を用いて，MBA を単一のプロセスに封じ込めることができることを示す．MTA，MCA については，通信リンクのブロックを用いて，高々 2 つのプロセスに封じ込めることができることを示し，封じ込め後，通信グラフの連結性を回復できることを示す．

2 準備

2.1 システムモデル

プロセス集合を $\Pi = \{p_1, p_2, ..., p_n\}$ ，プロセス間の双方向通信リンクの集合を E とし，分散システムを無向グラフ $G = (\Pi, E)$ と表す．プロセス p_i と p_j について $\{p_i, p_j\} \in E$ である時，p_i と p_j は**隣接**していると言い，隣接プロセスは相互にメッセージの送受信による通信を行うことができる．プロセス p_i の隣接プロセスの集合を $N_i = \{p_j \mid \{p_i, p_j\} \in E\}$ と表す．プロセス p_i の固有の識別子を i とし，プロセスは自身の識別子を送信メッセージに添付することで，そのメッセージを受信したプロセスは送信プロセスを識別することができる．

各プロセスは**ラウンド**毎に動作し，各ラウンドでメッセージの**送信**，**受信**，**内部計算**を行う．送信されたメッセージは同じラウンド内で必ず受信される．このモデルを**同期モデル**と呼ぶ．各プロセスはいくつかの内部変数を管理しており，これらの内部変数の値がプロセスの状態を表す．各プロセスは共通の分散アルゴリズムを用いて内部計算を行い，自身の内部変数の値を更新し，次のラウンドで送信するメッセージを決定する．

2.2 移動故障エージェント

本研究では，故障プロセスが任意に振る舞うビザンチン故障，故障プロセスがメッセージの送信，受信，内部計算を行わない停止故障，故障プロセスの内部変数が任意に書き換えられる一時故障の 3 種類のプロセスの故障について考える．ビザンチン故障プロセスは各隣接プロセスに異なるメッセージを送信することができる**二地点間通信**を行うものとするが，識別子を偽ることはできないとする．各故障によってプロセスが持つ分散アルゴリズムは改ざんされないと仮定する．

故障プロセスの集合を F と表し，$\Pi \backslash F$ に含まれるプロセスを**正常プロセス**と呼ぶ．本研究では**移動故障エージェント**が送信フェーズの直後に各プロセスを移動することで故障プロセス集合 F が変化する場合を考える．プロセスは故障エージェントが滞在する間，そのプロセスが故障を起こすと考える．移動故障エージェントは送信フェーズの直後に故障プロセスから離れ，隣接プロセスへ到着する．移動故障エージェントが移動したプロセスの軌跡を移動故障エージェントの**移動経路**と呼ぶ．プロセスに停止故障，一時故障，ビザンチン故障を起こす移動故障エージェントをそれぞれ**停止故障エージェント**（MCA），**一時故障エージェント**（MTA），**ビザンチン故障エージェント**（MBA）と呼ぶ．MTA はプロセスに滞在している間にプロセスの内部変数を高々 1 回書き換えるとする．ラウンド r で故障エージェントがあるプロセスから離れると，そのラウンド r でそのプロセスを *cured* プロセスと呼ぶ．移動故障エージェントがラウンド r の受信フェーズの直前で p_i に到着し，ラウンド $r+1$ の送信フェーズの直後に p_i から離れ，$p_j \in N_i$ に到着する．プロセス p_i はラウンド $r+1$ で *cured* プロセスとなる（図 1）．本研究では，*cured* プロセスが直前のラウンドで自身が故障プロセスであったことを認識できる**故障感知性**を持つとする．

Inoue らは，*cured* プロセスが移動故障エージェントの移動先のプロセスを認識できる**探知可能モデル**と認識できない**探知不可能モデル**を提案した [3]．プロセス p_i は変数 $dest_i$ を持ち，$dest_i$ は $\{\perp, 1, ..., n\}$ の要素を値とする．移動故障エージェントがラウンド r で p_i から p_j へ移動した時，ラウンド r の送信フェーズの直後

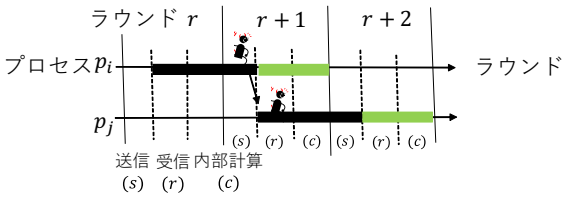

図 1　プロセス間を移動する移動故障エージェント．

に $dest_i$ は j となり，ラウンド $r+1$ で $\perp$ に戻る．探知不可能モデルでは，常に $dest_i = \perp$ である．

Inoue らは，プロセスがある隣接プロセスからのメッセージを受信しない通信リンクの**ブロック**を提案した．プロセス p_i が $p_j \in N_i$ の送信メッセージを受信しない時，p_i は $\{p_i, p_j\}$ を**ブロック**していると言う．プロセス p_i が再び p_j からのメッセージを受信する時，p_i が p_j との通信リンク $\{p_i, p_j\}$ を**解除する**と言う．本研究ではさらに，プロセス p_i が $p_j \in N_i$ に対して送信，受信を行わない通信リンクの**切断**を導入する．今後，p_i が p_j との通信リンク $\{p_i, p_j\}$ をブロックする時，p_i が p_j をブロックすると言う．通信リンクの切断についても同様である．

アルゴリズム A の実行は**大域状態** $C_0, C_1, \ldots,$ の連続であり，各 $C_i (i = 0, 1, 2, \ldots,)$ はラウンド i での内部計算後の，すべてのプロセスと通信リンクの状態から構成されている．アルゴリズム A の初期大域状態 C_0 から始まる実行には，移動故障エージェントの移動経路や振る舞い方によって複数の実行が存在する．

2.3　封じ込めと連結性の回復

プロセスが通信リンクのブロックや切断を行うことで，移動故障エージェントの移動を制限する．移動故障エージェントを封じ込めるために行った通信リンクのブロック，または切断を行った状態では，プロセス間でメッセージ通信を行えない．封じ込め後，正常プロセス間でのメッセージ通信を行うために，封じ込め後，$\Pi \setminus X$ のプロセス間の連結性を考える．プロセス集合 $X \subsetneq \Pi$ について，$p_i \in X$ と $p_j \in \Pi \setminus X$ のような辺 $\{p_i, p_j\}$ の集合を $E(X)$ と表す．

定義 2.1. 移動故障エージェントの封じ込め
アルゴリズム A が実行 $C_0, C_1, C_2, \ldots$ のあるラウンド r で，移動故障エージェントが $X \subsetneq \Pi$ に含まれるプロセスに滞在し，$\Pi \setminus X$ の各プロセスが X のプロセスとの通信リンクをブロック（または切断）しており，ラウンド r 以降，$\Pi \setminus X$ のプロセスが $E(X)$ のブロック（もしくは切断）を解除しない場合，アルゴリズム A は移動故障エージェントを X に封じ込めると言う．

アルゴリズム A がグラフ G の任意の初期大域状態から始まる任意の実行について，移動故障エージェントをプロセスの真部分集合で封じ込める場合，アルゴリズム A は移動故障エージェントの**封じ込める**と言う．

定義 2.2. 封じ込め後の連結性
アルゴリズム A が実行 $C_0, C_1, \ldots$ で移動故障エージェントをあるプロセス集合 $X \subsetneq \Pi$ に封じ込めたとする．ブロックまたは切断されている辺を $\bar{E} \subseteq E$ とする．つまり，$E(X) \subseteq \bar{E}$ とする．その後のラウンド r' でグラフ $G' = (\Pi \setminus X, E \setminus \bar{E})$ が連結である場合，アルゴリズム A は封じ込め後，連結性を回復すると言う．

アルゴリズム A がグラフ G の任意の初期大域状態から始まる任意の実行について，移動故障エージェントの封じ込め，封じ込め後，連結性を回復する場合，アルゴリズム A は封じ込め後，**連結性を回復する**と言う．

3　MBA の封じ込め

分散システム中に 1 個の MBA が存在する場合を考える．初めに，探知可能モデルにおいて，MBA の移動先のプロセスのみをブロックする時，MBA を封じ込めることができないことを示す．次に，探知可能モデルにおいて，MBA の移動先のプロセスのみを切断する時，MBA を単一のプロセスに封じ込めることができることを示す．

補題 3.1. 各プロセスが *MBA* の移動先のプロセスのみをブロックする場合，プロセス p_i が p_j をブロックしている時，p_j は p_i をブロックできない．

引き続き，各プロセスは MBA の移動先のプロセスのみをブロックする場合を考える．補題 3.1 より，通信リンクを双方向にブロックできないため，以下の定理が言える．

定理 3.1. 各プロセスが *MBA* の移動先のプロセスのみをブロックすることでは，単一の *MBA* を封じ込めることはできない．

次に，MBA をプロセスの部分集合に封じ込めるために通信リンクの切断を用いる．MBA があるプロセスに滞在している時に，MBA は切断している通信リンクを解除する可能性があるが，まず初めに，あるプロセスに滞在している MBA が切断している通信リンクを解除できない場合について考え，MBA を単一のプロセスに封じ込めることができることを示す．次に，あるプロセスに滞在している MBA が切断している通信リンクを解除できる場合，MBA を封じ込めることができるのは，グラフが木の場合，またその場合に限ることを示す．

MBA が移動しない時，プロセスはビザンチン故障プロセスを認識できず，MBA をプロセスの部分集合に封じ込めることはできない．

定理 3.2. 単一の *MBA* を考える．この *MBA* はいずれかのプロセスに永久に滞在し続けることはなく，通信リンクの切断を解除しないとする．この時，各プロセスが *MBA* の移動先のプロセスを切断することで *MBA* を単一のプロセスに封じ込めることができる．

次に，MBA が滞在しているプロセスにおける通信リンクの切断を解除する場合の MBA の封じ込めについて示す．

定理 3.3. 単一の *MBA* を考える．この *MBA* はいずれかのプロセスに永久に滞在し続けることはなく，通信リンクの切断を解除することができるとする．この時，*MBA* を単一のプロセスに封じ込めることができるのは，グラフ G が木構造である場合，またその場合に限る．

4　MCA の封じ込めと連結性の回復

分散システム中に 1 個の MCA が存在する場合を考える．故障プロセスは隣接プロセスへメッセージを送信しない．各プロセスが毎ラウンド何らかのメッセージを全ての隣接プロセスへ送信するアルゴリズムを想定する．故障プロセスの隣接プロセスはメッセージを受信しないことから，故障プロセスを認識し，ブロックすることができる．

補題 4.1. 1 個の *MCA* があるプロセスに連続して 2 ラウンド以上滞在する場合，そのプロセスに *MCA* を封じ込めることができる．

観察 4.1. グラフ G が 1 頂点連結グラフの場合，連結性を回復することができない．

以降，探知可能モデル，探知不可能モデルそれぞれについて，まず MCA が毎ラウンドでプロセス間を移動する場合を考え，次に同じプロセスに 2 ラウンド以上滞在する場合について考える．探知可能モデルでは MCA を単一のプロセスに封じ込めるアルゴリズムを示し，探知不可能モデルでは MCA を高々 2 つのプロセスで封じ込めるアルゴリズムを示す．そして，各モデルで連結性を回復するアルゴリズムを示す．

4.1 探知可能モデル

1 個の MCA が毎ラウンド，プロセス間を移動する場合を想定する．観察 4.1 より，グラフ G が 2 頂点連結グラフである場合を考える．各プロセスは隣接プロセスと毎ラウンドメッセージを交換し，メッセージ送信を行わなかった隣接プロセスをブロックするアルゴリズムを想定する．各プロセス p_i は *bool* 値を保持できる配列 $block_i[1..n]$ を管理し，p_i の $block_i[j] = true$ である時，p_i は p_j からのメッセージを受信しない．*cured* プロセスは MCA の移動先のプロセスをブロックする．各プロセスは一度ブロックした通信リンクをラウンド $2n$ まで解除しない．

補題 4.2. 各プロセスが *Algorithm 1* を実行すると，*MCA* が 1 度訪問したプロセスを再び訪問した以降は，*MCA* はプロセスの部分集合のみしか移動できず，さらにその移動経路は閉路である．

MCA を単一のプロセスに封じ込めるアルゴリズム Algorithm 1，2，3 を示す．ラウンド r で正常プロセスは Algorithm 2 を実行し，*cured* プロセスは Algorithm 3 を実行する．各プロセス p_i は各隣接プロセスが故障していたラウンドを格納する配列 $r_i^{f_1}[1..n]$，2 度目に故障した時のラウンドを格納する変数 $r_i^{f_2}$ を持つ．ラウンド $r+1$ の受信フェーズの直前に MCA がプロセス p_i から離れ p_j を訪問したとする．プロセス p_i の隣接プロセス $p_k(k \neq j)$ は $r+1$ で p_i からメッセージを受信しないので p_i をブロックし，p_i は p_j をブロックする．さらに各プロセス p_k は p_i が故障していたラウンド r を $r_i^{f_1}[i]$ に格納する．このラウンド r を p_k の p_i に対する**故障ラウンド**と呼ぶ．プロセス p_i は故障していたことを認識できるので，p_i 自身が故障していたラウンド r を $r_i^{f_1}[i]$ に格納する．MCA が移動を続け，その後のラウンド r' の受信フェーズの直前で MCA がプロセス p_i を再度訪問すると，p_i は 2 度目の故障ラウンド r' を $r_i^{f_2}$ に格納する．

補題 4.2 より，MCA はプロセスの部分集合のみしか移動できず，その移動経路はいずれ閉路 $p_{j_0}(= p_i), p_{j_1}, \ldots, p_{j_k}(= p_l)$ となる．この閉路上のプロセスで MCA が初めに滞在したプロセスを p_i，最後に滞在したプロセスを p_l とする．プロセス p_i は $r_i^{f_2}$ と $r_i^{f_1}[i]$ の差から，MCA が p_l から p_i へ 3 度目に訪れるまでのラウンド数である**再訪周期**を計算することができる．MCA は毎ラウンド移動するので，ラウンド $r_i^{f_2}+2$ で MCA が p_{j_1} から離れる．MCA が p_{j_1} から p_i へ移動することはないので，プロセス p_i は p_l のみをブロックする．MCA はラウンド $r_i^{f_2}$ から高々 n ラウンドで p_l を訪問する．したがって，MCA を p_l に封じ込めることができる．

次に MCA を封じ込めた後, グラフの連結性を回復するためのアルゴリズム Algorithm 4 を示す．Algorithm 1 によって単一のプロセス p_l に MCA が封じ込められる時，p_l の隣接プロセス p_k が保持する故障ラウンドの値が全プロセスの中で最大である．ラウンド $2n$ 後，各プロセス p_i は $r_i^{f_1}[1..n]$ の中から最大の故障ラウンド以外のプロセスとの通信リンクのブロックを解除する．各プロセスは自身の最大の故障ラウンドを隣接プロセスと比較し，大きい故障ラウンドに更新していく．プロセス p_l 以外のプロセスとの通信リンクを解除することで連結性を回復することができる．

定理 4.1. グラフ G を 2 頂点連結グラフとする．探知可能モデルにおいて，各プロセスが故障感知性を持つ時，1 個の *MCA* がグラフ上を毎ラウンド移動する場合，*Algorithm 1* は $2n$ ラウンドで *MCA* を単一のプロセスに封じ込め，その後，連結性を回復することができる．

Algorithm 1 $Alg_{MCA,MTA}$

プロセス $\mathbf{p_i}$の内部変数

$r_i^{f_1}[1..n]$：p_i が隣接プロセスをブロックした時の故障ラウンド．隣接していれば初期値 0，隣接していなければ $\perp$.

$r_i^{f_2}$：故障エージェントが p_i に 2 度目に滞在した時の故障ラウンド，初期値 $\perp$.

per_i：再訪周期 p，初期値 $\perp$.

$period_i$：p_i が再訪周期を計算できた時 true，それ以外は false.

max_{r_i}：p_i がもつ故障ラウンドの中で最大の故障ラウンド，初期値 $\perp$.

max_{p_i}：p_i の最大の故障ラウンドでブロックしたプロセスの ID，初期値 $\perp$.

$block_i[1..n]$：p_i が隣接プロセス p_j をブロックしている時 true，それ以外は false.

$dest_i$：故障エージェントの移動先のプロセスの ID，$\{\perp, 1, ..., n\}$.

$cured_i$：p_i が直前のラウンドで cured だった場合 true，それ以外は false.

$destination_i$：故障エージェントが移動したプロセスの ID．$\{\perp, 1, ..., n\}$.

アルゴリズム

for *all* $p_j \in N_i$ **do**

　$(r_i^{f_1}[1..n], per_i, max_{p_i}, max_{r_i}, cured_i, destination_i)$ を送信

1: **while** *round* $r \leq 2n$ **do**
2:　**if** p_iが *cured* プロセス **then**
3:　　**if** 故障エージェントが MCA **then**
4:　　　*Algorithm* 3
5:　　**if** 故障エージェントが MTA **then**
6:　　　*Algorithm* 6
7:　**else**　//p_i が正常プロセス
8:　　**for** *all* $p_j \in N_i$ **do**
9:　　　**if** 故障エージェントが MCA **then**
10:　　　$x = 2$
11:　　　*Algorithm* 2
12:　　　**if** 故障エージェントが MTA **then**
13:　　　$x = 3$
14:　　　*Algorithm* 5
15:　　　**if** $(r = r_i^{f_2} + x) \wedge (period_i = true)$ **then**
16:　　　　**if** $p_j = max_{p_i}$ **then**
17:　　　　$block_i[j] = true$
18:　　　　**else**
19:　　　　$block_i[j] = false$
20: *Algorithm* 4を実行　//連結性を回復するためのアルゴリズム

Algorithm 2 $MCA_{correct}$

1: **if** p_jからメッセージを受信しなかった **then**
2:　$block_i[j] = true$
3:　$r_i^{f_1}[j] = r - 1$
4:　$max_{p_i} = j$
5:　$max_{r_i} = r_i^{f_1}[j]$

Algorithm 3 MCA_{cured}

1: **for** *all* $p_j \in N_i$ **do**
2: **if** $r_i^{f_1} \neq \perp$ **then**
3: $r_i^{f_2} = r - 1$
4: **if** 受信した全ての per_jについて，$per_j = \perp$ **then** $//p_i$ の故障が 2 回目
5: $per_i = r_i^{f_2} - r_i^{f_1}[i]$ //再訪周期を計算
6: $period_i = true$
7: $max_{r_i} = r_i^{f_2} - 1$
8: **if** 受信した $r_*^{f_1}[*]$ の中で $r_j^{f_1}[j]$ が最大 **then**
9: $max_{p_i} = j$
10: **else**
11: $per_i = per_j$
12: $max_{r_i} = max_{r_j}$
13: $max_{p_i} = max_{p_j}$
14: **else** $//p_i$ の故障が 1 回目
15: $r_i^{f_1}[i] = r - 1$
16: **if** $r_i^{f_1}[dest_i] = \perp$ **then**
17: $r_i^{f_1}[dest_i] = r$
18: $max_{r_i} = r_i^{f_1}[dest_i]$
19: $max_{p_i} = dest_i$
20: **if** $dest_i \neq \perp$ **then**
21: $block_i[dest_i] = true$ //移動故障エージェントの移動先のプロセスをブロック

Algorithm 4 $Alg_{connect}$

1: max_{p_i}以外のプロセスとの通信リンクのブロックを解除
2: **for** *all* $j \in N_i$ **do**
3: **if** $max_{r_i} < max_{r_j}$ **then**
4: $block_i[max_{p_i}] = false$
5: $max_{r_i} = max_{r_j}$
6: $max_{p_i} = max_{p_j}$

4.2 探知不可能モデル

探知不可能モデルにおいて MCA が毎ラウンドでプロセス間を移動する場合を想定する．Algorithm 1 を用いて，各正常プロセスは隣接プロセスをブロックする．*cured* プロセスは MCA が移動したラウンドで MCA の移動先のプロセスを分からず，MCA の移動先のプロセスをブロックすることができない．

補題 4.3. 探知不可能モデルにおいて，各プロセスが故障感知性を持ち，*Algorithm 1* を実行する時，*MCA* が直前のラウンドで滞在していたプロセスへ移動できるのは，*MCA* の移動経路が閉路でない場合，またその場合に限る．

補題 4.4. 探知不可能モデルにおいて，各プロセスが故障感知性を持つ時，*MCA* があるプロセス p_j から直前のラウンドで滞在していたプロセス p_i へ移動する場合，*Algorithm 1* は *MCA* をプロセス集合 $\{p_i, p_j\}$ に封じ

込めることができる．

MCA を単一のプロセスで封じ込める場合と 2 つのプロセス間で封じ込める場合が考えられる．各プロセスが MCA を単一のプロセスへ封じ込めたのか，2 つの隣接プロセスで封じ込めたのか判断できることを示す．MCA を 2 つのプロセス間で封じ込めた場合，補題 4.3 より，再訪周期を計算するプロセスが存在しない．したがって，各プロセスが MCA を単一のプロセスへ封じ込めたのか，2 つの隣接プロセスで封じ込めたかを再訪周期が $\perp$ かどうかによって判断することができる．

MCA を単一のプロセスに封じ込めた場合，Algorithm 4 を実行することで連結性を回復することができる．MCA を 2 つのプロセス間で封じ込めた場合，グラフの連結性を回復するアルゴリズムを提案する．各プロセスは隣接プロセスと最大の故障ラウンドを比較し，隣接プロセスの故障ラウンドのほうが大きければ，そのプロセスはブロックを解除する．各プロセスは全プロセスの中で最大の故障ラウンドとその差が 1 の故障ラウンド以外のブロックを解除することで連結性を回復することができる．

定理 4.2. グラフ G を 3 頂点連結グラフとする．探知不可能モデルにおいて，各プロセスが故障感知性を持つ時，1 個の *MCA* がグラフ上を毎ラウンド移動する場合，*Algorithm 1* は $2n$ ラウンドで *MCA* を高々 2 つのプロセスに封じ込め，その後，連結性を回復することができる．

4.3 MCA が 2 ラウンド以上滞在する場合

グラフ G を単一のサイクルを除く 2 頂点連結グラフとし，各プロセスが Algorithm 1 を実行する．MCA が 2 ラウンド以上同じプロセス p_i に滞在する場合を想定する．補題 4.1 より，p_i で MCA を封じ込めることができる．その後，連結性を回復するために各プロセスがブロックの解除を行うと，p_i の隣接プロセスが p_i との通信リンクを解除してしまう場合がある．あるプロセス p_i で 2 ラウンド以上滞在する MCA について，Algorithm 1 が MCA を p_i に封じ込めることができる場合は，MCA の移動経路が閉路となる前に MCA が p_i で 2 ラウンド以上滞在する場合，またその場合に限る．

5 MTA の封じ込めと連結性の回復

分散システム中に 1 個の MTA が存在することを想定する．MTA はプロセスに滞在する間に，そのプロセスの内部変数を書き換える．MTA の書き換えは，送信，受信，内部計算，各フェーズで行われる可能性がある．MTA が送信フェーズの直前で内部変数を書き換える場合，そのプロセスは書き換えられたメッセージを送信する可能性がある．MTA が内部計算フェーズの直前で内部変数を書き換える場合，プロセスが受信したメッセージが書き換えられる可能性がある．書き換えられたメッセージをもとに計算を行い，次のラウンドで送信するメッセージが本来のメッセージと異なる可能性がある．MTA が受信フェーズの直前で内部変数を書き換える場合，そのプロセスの内部変数を書き換える可能性はあるが，隣接プロセスから受信するメッセージは書き換えることはできない．本研究では，MTA の書き替えは受信フェーズの直前に滞在しているプロセスの内部変数を書き替えるとし，滞在中に高々 1 回書き換えることができると仮定する．

5.1 探知可能モデル

探知可能モデルにおいて，単一の MTA を単一のプロセスに封じ込めるアルゴリズム Algorithm 5，6 を提案する．MTA が同じプロセスに複数ラウンド滞在する場合があるが，初めに MTA が毎ラウンド，プロセス間を移動し続ける場合での MTA の封じ込めと連結性の回復について考え，次に MTA が複数ラウンド滞在する場合の MTA の封じ込めと連結性の回復について考える．

各プロセス p_i は自身が直前のラウンドで *cured* だったかどうかを隣接プロセスへ伝えるための変数 $cured_i$

を保持し，p_i が $cured$ だった場合 $cured_i = true$ とする．$cured_i = true$ を受信した各プロセスは p_i が故障していたことを認識できるので p_i をブロックする (Algorithm 5，2,3 行目)．各プロセスはラウンド $2n$ まで，このブロックを解除しない．ラウンド r で正常プロセスは Algorithm 5 を実行し，$cured$ プロセスは Algorithm 6 を実行する．

定理 5.1. グラフ G を 2 頂点連結グラフとする．探知可能モデルにおいて，各プロセスが故障感知性を持つ時，1 個の MTA がグラフ上を毎ラウンド移動する場合，*Algorithm 1* は $2n$ ラウンドで MTA を単一のプロセスに封じ込め，その後，連結性を回復することができる．

Algorithm 5 $MTA_{correct}$

1: $cured_i = false$
2: **if** $cured_j = true$ **then**
3: $\quad block_i[j] = true$ //故障していたプロセス p_j をブロック
4: $\quad r_i^{f_1}[j] = r - 2$
5: $\quad max_{r_i} = r_i^{f_1}[j]$
6: $\quad max_{p_i} = j$
7: $\quad$ **if** $destination_j = i$ **then**
8: $\quad\quad destination_i = destination_j$
9: **if** $destination_j \in N_i$ **then**
10: $\quad block_i[destination_j] = true$
11: $\quad r_{b_i}[destination_j] = r - 2$
12: $\quad max_{r_i} = r_i^{f_1}[destination_j]$
13: $\quad max_{p_i} = destination_j$

5.2 MTA が複数ラウンド滞在する場合

グラフ G を単一のサイクルを除く 2 頂点連結グラフとし，各プロセスが Algorithm 1 を実行する．MTA が 2 ラウンド以内に各プロセスを移動する場合，高々 $2n$ ラウンドで 1 度訪問したプロセスを再び訪問した以降は，MTA はプロセスの部分集合のみしか移動できず，その移動経路は閉路である．高々 $4n$ ラウンドで単一のプロセスに MTA を封じ込めることができ，ラウンド $4n$ 後，Algorithm 4 を実行することで連結性を回復することができる．

最後に MTA が同じプロセスに 3 ラウンド以上滞在する場合の封じ込めと連結性の回復について考える．MTA が同じプロセスに 3 ラウンド以上滞在する場合，その滞在しているプロセスで MTA を封じ込める手法を提案する (Algorithm 5，7 から 13 行目)．各プロセス p_i は MTA の移動先の ID を格納する変数 $destination_i$ を保持する．ラウンド $r+1$ の受信フェーズの直前で p_i から $p_j \in N_i$ へ移動する．ラウンド $r+1$ で p_i は $destination_i = j$ とし，$r+2$ で送信する．ラウンド $r+2$ で $destination_i = j$ を受信したプロセスは p_j をブロックする．ラウンド $r+3$ で p_j は $destination_j = j$ とし，$r+3$ で送信する．プロセス p_j の隣接プロセスが p_j をブロックする (図 2)．あるプロセス p_i で 3 ラウンド以上滞在する MTA について，Algorithm 1 が MTA を p_i に封じ込めることができる場合は，MTA の移動経路が閉路となる前に MTA が p_i で 3 ラウンド以上滞在する場合，またその場合に限る．

Algorithm 6 MTA_{cured}

1: $per_i = \perp,\ r_i^{f_2} = \perp,\ period_i = false$
2: **for** *all* $p_j \in N_i$ **do**
3: $\quad block_i[j] = false$
4: $\quad$ **if** 受信した全ての $r_j^{f_1}[i]$ について，$r_j^{f_1}[i] \neq \perp$ **then** $\quad //p_i$ の故障が 2 回目
5: $\quad\quad r_i^{f_1}[i] = r_j^{f_1}[i]$
6: $\quad\quad r_i^{f_2} = r - 1$
7: $\quad\quad$ **if** 受信した全ての per_jについて，$per_j = \perp$ **then**
8: $\quad\quad\quad per_i = r_i^{f_2} - r_i^{f_1}[i]$ $\quad$ //再訪周期を計算
9: $\quad\quad\quad period_i = true$
10: $\quad\quad\quad max_{r_i} = r_i^{f_2} - 1$
11: $\quad\quad\quad$ **if** プロセス p_jの $r_j^{f_1}[j]$ が受信した $r_j^{f_1}[j]$ の中で最大 **then**
12: $\quad\quad\quad\quad max_{p_i} = j$
13: $\quad\quad$ **if** $per_j \neq \perp \wedge cured_j = true$ **then**
14: $\quad\quad\quad per_i = per_j$
15: $\quad\quad\quad max_{r_i} = max_{r_j}$
16: $\quad\quad\quad max_{p_i} = max_{p_j}$
17: $\quad$ **else** $\quad //p_i$ の故障が 1 回目
18: $\quad\quad r_i^{f_1}[i] = r - 1$
19: $\quad\quad$ **if** $r_i^{f_1}[dest_i] = \perp$ **then**
20: $\quad\quad\quad r_i^{f_1}[dest_i] = r$
21: $\quad\quad max_{r_i} = r_i^{f_1}[dest_i]$
22: $\quad\quad max_{p_i} = dest_i$
23: **if** $dest_i \neq \perp$ **then**
24: $\quad block_i[dest_i] = true$ $\quad$ //移動故障エージェントの移動先のプロセスをブロック
25: $destination_i = dest_i$
26: $cured_i = true$

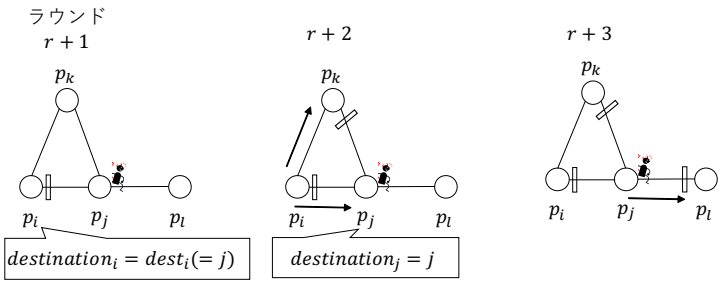

図 2　MTA があるプロセス p_j に 3 ラウンド以上滞在する場合，MTA を p_j に封じ込めることができる例.

6　まとめと今後の課題

本研究では，移動故障エージェントを封じ込め，その後，通信グラフの連結性を回復するアルゴリズムを提案した．移動故障エージェントが MBA の場合，通信リンクのブロックを用いては MBA を単一のプロセスに封じ込めることができないことを示し，通信リンクの切断を用いることで MBA を単一のプロセスで封じ込めることができることを示した．MCA，MTA の場合，通信リンクのブロックを用いて高々 2 つのプロセスに封じ込めるアルゴリズムを提案し，封じ込め後，通信グラフの連結性を回復するアルゴリズムを提案した．

今後の課題は，MBA については，通信リンクの切断を用いて単一のプロセスに MBA を封じ込めた後，連結性を回復できるのかについて検証することである．MTA については，探知不可能モデルにおいて MTA を封じ込め，その後，連結性を回復するアルゴリズムの考案である．そして MBA，MCA，MTA の各移動故障エージェントで合意問題を解くアルゴリズムの考案である．

参考文献

[1] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[2] Juan A. Garay. Reaching (and maintaining) agreement in the presence of mobile faults (extended abstract). In *Proceedings of the 8th International Workshop on Distributed Algorithms*, pages 253–264, 1994.

[3] Koki Inoue, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. A strongly-stabilizing protocol for spanning tree construction against a mobile Byzantine fault. In *Proceedings of the 26th International Colloquium, SIROCCO*, pages 353–356, 2019.

[4] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[5] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems - principles and paradigms, 2nd Edition.* Pearson Education, 2007.