Vol.2011-AL-134 No.20
2011/3/7

DAG を構成する故障封じ込め自己安定プロトコルについて

三浦哲平[†1]　片山喜章[†1]
和田幸一[†1]　高橋直久[†1]

自己安定プロトコルとは，任意のネットワーク状況から実行を開始しても，やがて解を求めて安定する分散プロトコルである．故障封じ込めは，解状況から少数のプロセスが故障した場合に，再び解状況に到達するまでに状態遷移するプロセス数，及び時間を制限し，素早く再安定することを目的としている．本稿では，1 故障状況から変動プロセス数が Δ^2+1，再安定時間が $O(1)$ である DAG を構成する故障封じ込め自己安定プロトコルを提案する．

A Fault-Containing Self-Stabilizing Protocol for Constructing a Directed Acyclic Graph

Teppei Miura ,[†1] Yoshiaki Katayama ,[†1]
Koichi Wada [†1] and Naohisa Takahashi[†1]

Self-stabilizing protocols guarantee convergence to some predefined legitimate configuration starting from an arbitrary network configuration. A fault-containing is not only self-stabilizing, but in addition, can reconverge from a small number of faults, with only a bounded number of processes changing state and time during reconvergence. In this paper, we present a fault-containing self-stabilizing protocol for constructing a DAG. From 1-faulty configuration, proposed protocol can stabilize $O(1)$ time, and its contamination number is Δ^2+1.

†1 名古屋工業大学大学院 工学研究科 情報工学専攻
Graduate School of Computer Science, Engineering Science, Nagoya Institute of Technology

1. はじめに

通信リンクによってプロセスが相互に接続されたネットワーク（分散システム）上で，プロセスが互いに協調して問題を解くためのプロトコルが分散プロトコルである．通常の分散プロトコルでは，プロトコル開始時のネットワーク状況があらかじめ決められた初期状況であると仮定される．一方，自己安定プロトコルとは，任意の初期ネットワーク状況から実行を始めても問題を解くことができる分散プロトコルである[1)]．この性質から，自己安定プロトコルはプロセスの故障に対して耐性のある分散プロトコルであり，長期に渡ってネットワーク状況を安定に保ち，一時故障に柔軟に対応することを求められる分散システムを動作させる場合に適している．

分散システム上で分散プロトコルを長期に渡って動作させた場合，同時に多くのプロセスが故障することは稀であり，一般には少数のプロセスが一時故障を起こし，解状況からかけはなれた状況ではなく，わずかに変動（小変動）したネットワーク状況になることがほとんどである．一般的に自己安定プロトコルでは，小変動状況から再び安定するまでの間の動作については一切考慮されない．このため，ただ一つのプロセスが一時故障したにも関わらず，再び安定するまでにネットワーク上のすべてのプロセスが動作することもある．従って，小変動状況から再安定するまでの動作について考慮した自己安定プロトコルは重要である．

DAG とは閉路のない有向グラフであり，これまでに DAG に関連する様々な分散プロトコルが提案されている[2)–5)]．DAG は分散プロトコルを動作させる初期グラフとしてもよく利用される[2)–4)] ため，分散システム上に安定した DAG を構成するプロトコルは有用である．

本稿では，指定したプロセス集合がシンクとなる DAG を構成する自己安定プロトコルを提案し，さらに 1 故障状況からの実行において，状態遷移するプロセス数 Δ^2+1（Δ はプロセス次数の最大値），再安定時間 $O(1)$ で安定する故障封じ込め自己安定プロトコルに拡張する．これによって，分散システム上での一時故障に耐性を持ち，かつ安定後の 1 プロセスの故障を局所的に抑える DAG を構成することができる．

2. システムモデルと問題の定義

本章では，本稿で扱うモデルと DAG 構成問題について述べる．

2.1 ネットワークとプロセス

本稿では，n 個のプロセスが通信リンクで互いに接続された任意の形状のネットワーク N を扱う．一般にネットワークは，プロセスを頂点，通信リンクを辺としたグラフで表現でき

るので，グラフに対する用語や記法をネットワークに対しても用いる．

ネットワーク N は 2 項組 $N = (\mathcal{P}, \mathcal{L})$ で表され，$\mathcal{P} = \{P_0, P_1, ..., P_{n-1}\}$ をプロセスの集合，$\mathcal{L}$ を通信リンクの集合とする．各プロセスはすべて相異なる識別子を持ち，簡単のためにプロセス P_i の識別子をそのまま P_i と表す．$(P_i, P_j) \in \mathcal{L}$ のとき，プロセス P_i, P_j 間に全二重リンクが存在し，P_i, P_j は互いに隣接しているという．各プロセスは自分の識別子，接続するリンク，隣接プロセスの識別子の集合 N_i を知っているとする．プロセス間の通信は，隣接するプロセスが互いの内部状態を直接知ることができる状態通信モデルを仮定する．

2.2 有向グラフと DAG 構成問題の定義

有向グラフ $\overrightarrow{G}$ は 2 項組 $\overrightarrow{G} = (V, \overrightarrow{E})$ で表され，$V = \{v_0, v_1, ..., v_{n-1}\}$ を頂点集合，$\overrightarrow{E}$ を有向辺の集合とする．v_i から v_j への有向辺が存在するとき，v_i は v_j への外向辺（outgoing edge）を持ち，v_j は v_i からの内向辺（incoming edge）を持つといい，$\overrightarrow{(v_i, v_j)} \in \overrightarrow{E}$ と表す．

閉路のない有向グラフを DAG といい，内向辺しか持たないノードをシンク，外向辺しか持たないノードをソースと呼ぶ．本稿で扱う DAG 構成問題を以下のように定義する．

定義 1.（DAG 構成問題の定義）任意の連結ネットワーク N，独立頂点集合 $T \subset \mathcal{P}$[⋆1] に対して，以下の条件を満たす有向グラフ $\overrightarrow{G}$ を構成する問題を，DAG 構成問題という．

(1) T に含まれるすべての頂点はシンクとなる．
(2) $\mathcal{L}$ 中のすべてのリンクは方向付けられている．
(3) $\overrightarrow{G}$ は有向閉路を持たない． □

2.3 スケジュールと実行

各プロセス P_i の状態を q_i として，ネットワーク状況（以下，単に状況と呼ぶ）を $c = (q_0, q_1, ..., q_{n-1})$ で表す．状況 c_i からプロセスの部分集合によるプロトコルの 1 原子動作の実行によって c_{i+1} になったとする．このとき，状況の極大系列（場合によっては無限系列）$E = c_0, c_1, ...$ を実行と呼ぶ．本稿では，実行についてすべてのプロセスが無限回プロトコルを実行可能な公平なスケジュールを仮定する．また，同時に 1 つ以上のプロセスが原子動作を実行可能，つまり c_i から c_{i+1} に状況が遷移する際に複数のプロセスが状態遷移可能なスケジューラである D デーモンを仮定する．D デーモン下では，非同期実行が可能である．非同期実行においては実行時間を評価する尺度として（非同期）ラウンドを用いる．$E = c_0, c_1, ...$ を任意の実行とする．このとき，第 1 ラウンド $E_1 = c_1, c_2, ..., c_k$ とは，c_1 から c_k に状況が遷移する間にすべてのプロセス P_i が少なくとも 1 回プロトコルの 1 原子動作を実行する極小系列である．第 2 ラウンド以降は直前のラウンド直後の状況を初期状況として再帰的に定義される．

2.4 自己安定

任意の状況の集合を $\mathcal{LE}$ とする．プロトコル $\mathcal{A}$ は，次の 2 つの条件を満たす時かつその時のみ，$\mathcal{LE}$ に関して自己安定であるという．

(1) 到達可能性：任意の初期状況から実行を開始しても，有限時間内に $\mathcal{LE}$ に含まれる状況に到達する．
(2) 閉包性：一度 $\mathcal{LE}$ に含まれる状況に到達すれば，以降の状況はすべて $\mathcal{LE}$ に含まれる．

プロトコル $\mathcal{A}$ が $\mathcal{LE}$ に関して自己安定であるとき，$\mathcal{LE}$ を正当な状況と呼ぶ．

2.5 故障封じ込め

プロトコル $\mathcal{A}$ を正当な状況 $\mathcal{LE}$ に関して自己安定であるとする．ある状況 $c \in \mathcal{LE}$ において，1 プロセスの状態を任意に変えることによって得られる状況 c' を 1 故障状況と呼び，状態を変えたプロセスを故障プロセスと呼ぶ．

実システムでは 1 故障状況から再安定までに動作するプロセス数やそれにかかる時間は重要であり，故障の影響を局所的なものに抑え，かつできる限り迅速に再安定することが望まれる．これらを制限した自己安定プロトコルを，故障封じ込め自己安定プロトコルという．故障封じ込めの性能は，次に定義する変動プロセス数と再安定時間で評価する．

定義 2.（変動プロセス数と再安定時間）プロトコル $\mathcal{A}$ を $\mathcal{LE}$ に関して自己安定であるとする．任意の 1 故障状況 c から始まる任意の実行 E が，再び $\mathcal{LE}$ に到達するまでに状態遷移する最大プロセス数を「変動プロセス数」，それにかかる同期スケジュール（すべてのプロセスが同時にプロトコル $\mathcal{A}$ の原子動作を実行）の下での時間を「再安定時間」という．□

3. DAG を構成する自己安定プロトコル $\mathcal{DAG}_{mk}$

本章では，DAG 構成問題を解く自己安定プロトコル $\mathcal{DAG}_{mk}$ について述べる．

3.1 プロトコル $\mathcal{DAG}_{mk}$

$\mathcal{DAG}_{mk}$ はネットワーク $N = (\mathcal{P}, \mathcal{L})$ とシンクとして指定するプロセス集合 $T \subset \mathcal{P}$ を入力として，定義 1 の DAG 構成問題を解く自己安定プロトコルである．各プロセスは指定するシンクプロセス集合 T が自分を含むとき，外部からそのことを知らせてもらう述語 $Sink_i$ を持っており，一時故障によって $Sink_i$ が変化してもすぐに正しく再設定されるとする．

DAG 構成問題を解く自己安定プロトコルの戦略は以下の通りである．

⋆1 ここでは各 $P_i \in T$ は互いに独立であると仮定する．もし隣接する場合は隣接プロセス間の辺の縮約により独立な集合を考えることが可能なため，一般性を失わない．

(1) ネットワーク上の各プロセスは自分から最も近いシンクプロセスへの距離（以降では，レベルと呼ぶ）を求める．
(2) 各プロセスは隣接プロセスとレベルを比較し，レベルの大きいプロセスから小さいプロセスへ向かう有向辺をつくる．
(3) レベルが等しいプロセス間では識別子を比較し，識別子の大きいプロセスから小さいプロセスへ向かう有向辺をつくる．

以上の戦略によって方向付けられた辺を表現するために，各プロセスは自分に接続する外向辺によって隣接する隣接プロセス（以降，親プロセスと呼ぶ）の集合を管理するが，親プロセス集合は DAG を構成するための本質的な情報ではなく，単に DAG を利用するプロトコルに対して辺の向きを与えるためにのみ利用されるものであり，この集合（変数）が一時故障によって改変されても周りのプロセスに影響を及ぼすことはないことに注意する．また，たとえ親プロセス集合が故障によって改変されても，隣接プロセスと一度通信して自分と隣接プロセスの正しい情報を知れば，正しい親プロセスの集合を決定できる．つまり，ネットワーク上の各プロセスがシンクプロセスからの正しいレベルを知ることができれば，DAG 構成問題を解くことができる．よって，以降ではネットワーク上の各プロセスがシンクからのレベルを求める戦略（プロトコル）のみに限定して議論していく．

各プロセスがシンクプロセスからのレベルを求める戦略は以下の通りである．

(1) 各プロセスがシンクプロセスからのレベルを求めるプロトコル $\mathcal{DAG}_{mk}$.
  (a) シンクプロセスは常にレベルを 0 とする．
  (b) シンク以外のプロセスは隣接プロセスの中で一番小さいレベルに 1 を加えた値を自分のレベルとする．
  (c) (a), (b) をくり返す．

プロトコル中で各プロセスが扱う定数，変数，述語とプロトコル $\mathcal{DAG}_{mk}$ を示す（図 1）．

- N_i: プロセス P_i の隣接プロセスの集合を表す定数．$N_i = \{P_j | (P_i, P_j) \in \mathcal{L}\}$ である．
- $d.i$: プロセス P_i のレベルを表す変数．$d.i$ の値は非負整数である．
- $Sink_i$：自分がシンクプロセスかどうかを表す述語．$P_i \in T$ のとき $Sink_i = true$，$P_i \notin T$ のとき $Sink_i = false$ である．

3.2 正当性の証明

正当な状況を定義するために，各プロセスの無矛盾状態を表す述語 $cons(i)$ を定義する．各プロセスは自分がシンクプロセスでかつレベルが 0 のとき，及びシンク以外のプロセスでかつ自分のレベルが隣接プロセスで最小のレベル +1 のとき，無矛盾状態である．

入力変数： $d.j(P_j \in N_i)$
出力変数： $d.i$
　シンクプロセス（$Sink_i = true$）：
　　R1: $d.i \neq 0 \longrightarrow d.i := 0$
　シンク以外のプロセス（$Sink_i = false$）：
　　R2: $d.i \neq min_{P_j \in N_i}[d.j + 1] \longrightarrow d.i := min_{P_j \in N_i}[d.j + 1]$

図 1 自己安定プロトコル $\mathcal{DAG}_{mk}$

定義 3.（無矛盾状態）以下に定義される述語 $cons(i)$ を満たすプロセス P_i を「無矛盾状態」という．また満たさないものを「矛盾状態」という．

$P_i \in T$ のとき，$cons(i) = [d.i = 0]$
$P_i \notin T$ のとき，$cons(i) = [d.i = min_{P_j \in N_i}[d.j + 1]]$ □

次に，正当な状況 $\mathcal{LE}_{DAG}$ を定義する．

定義 4.（正当な状況）ネットワーク中のすべてのプロセス P_i が $cons(i)$ のとき，正当な状況といい，正当な状況の集合を $\mathcal{LE}_{DAG}$ と表す． □

$\mathcal{DAG}_{mk}$ が正当な状況 $\mathcal{LE}_{DAG}$ に関して自己安定であることを示す．ここで，プロセス P_i から最も近いシンクまでの距離を D_i とする．ただし，$P_i \in T$ のとき $D_i = 0$ とする．

補題 1. 第 1 ラウンド後では，シンクプロセス P_i（$D_i = 0$）のレベルは $d.i = 0$ となり $cons(i)$ が成立する．以降のラウンドでは故障が発生しない限り $d.i$ の値は変化せず，常に $cons(i)$ が成立する．また，シンク以外のプロセス P_j（$D_j \geq 1$）では $d.j \geq 1$ となる．

証明. シンクプロセス P_i が $\mathcal{DAG}_{mk}$ を実行すると，$d.i = 0$ となり，$cons(i)$ が成立し，以降のラウンドも $d.i = 0$ から変化しない．すべてのプロセスのレベルは非負整数なのでシンク以外のプロセス P_j が $\mathcal{DAG}_{mk}$ を実行すると，$d.j \geq 1$ となる． □

補題 2. シンクプロセスの離心数の最大値を $\mathcal{D}$ とする．第 $k(2 \leq k \leq \mathcal{D})$ ラウンド後において，$D_i = q(1 \leq q \leq k-1)$ であるプロセス P_i では $d.i = q$ となり，$cons(i)$ が成立する．

証明. ラウンド数による帰納法で証明する．

第 2 ラウンド後を考える．$D_i = 1$ であるプロセス P_i はシンクプロセスに隣接しており，補題 1 より $d.i = 1$ となり，$cons(i)$ が成立する．$D_j \geq 2$ であるプロセス P_j は，$d.k \geq 1$ となった $D_k \geq 1$ であるプロセス P_k に隣接しており，$\mathcal{DAG}_{mk}$ を実行すると $d.j \geq 2$ となる．

第 $k(3 \leq k \leq \mathcal{D})$ ラウンド後において，$D_i = q(1 \leq q \leq k-1)$ であるプロセス P_i で

$d.i = q$ であり，$D_j \geq k$ であるプロセス P_j で $d.j \geq k$ が成立すると仮定する．

第 k ラウンドに続く第 $k+1$ ラウンドの実行を考える．$D_j = k$ であるプロセス P_j は，$D_i = k-1$ であるプロセス P_i に隣接している．また P_i 以外の隣接プロセス P_l は，$d.l \geq k$ なので，P_j が $\mathcal{DAG}_{mk}$ を実行すると $d.j = k$ となる．また，$d.j = k$ と $d.l \geq k$ から，P_l が $\mathcal{DAG}_{mk}$ を実行すると $d.l \geq k+1$ となり，P_j において $cons(j)$ が成立する． □

補題 3． プロトコル $\mathcal{DAG}_{mk}$ は正当な状況 $\mathcal{LE}_{DAG}$ に対して自己安定性を満たす．

証明． シンクプロセスの離心数の最大値を $\mathcal{D}$ とする．補題 1，補題 2 より，第 $\mathcal{D}+1$ ラウンド後ではすべてのプロセス P_i において $cons(i)$ が成立する．よって，到達可能性を満たす．

$\mathcal{LE}_{DAG}$ の定義より，正当な状況での $\mathcal{DAG}_{mk}$ の実行によってレベルを変更するプロセスは存在しない．よって，閉包性を満たす． □

補題 4． すべてのプロセス P_i において，レベル $d.i$ と識別子 P_i の二項組 $(d.i, P_i)$ を考える．これらを辞書式順序で比較することで，すべてのプロセスに全順序が定義できる．

証明． プロセスの識別子が全順序であることより明らか． □

補題 5． DAG を構成する戦略 (1)-(3) に従って構成する有向グラフ $\overrightarrow{G}$ は DAG である．

証明． $\overrightarrow{G}$ に有向閉路が存在すると仮定し矛盾を導く．補題 4 と戦略 (1)-(3) より，P_i から P_j へ向かう有向辺が存在する場合，$P_i > P_j$ であるとする．有向閉路を構成するプロセス系列を $< P_0, P_1, ..., P_m, P_0 >$ とすると，$P_0 > P_m$ かつ $P_m > P_0$ が成立するので矛盾． □

補題 6． 正当な状況 $\mathcal{LE}_{DAG}$ では，ネットワーク上に定義 1 で示した DAG を構成できる．

証明． 補題 5 より，DAG を構成する戦略 (1)-(3) に従って構成する有向グラフは DAG である．$\mathcal{LE}_{DAG}$ において，シンクプロセス P_i のレベルは 0 であり，レベル 0 より小さいプロセスは存在しないため P_i はシンクとなる．シンク以外のプロセス P_j の隣接にはレベルが「P_j のレベル -1」のプロセスが存在し，外向辺を持つので P_j はシンクにならない． □

補題 3, 補題 6 より以下の定理が成り立つ．

定理 1． プロトコル $\mathcal{DAG}_{mk}$ は DAG 構成問題を解く自己安定プロトコルである． □

3.3 時間複雑度の解析

定理 2． プロトコル $\mathcal{DAG}_{mk}$ は，$O(\mathcal{D})$ ラウンドで DAG 構成問題を解く自己安定プロトコルである．($\mathcal{D}$ はシンクプロセスの離心数の最大値)

証明． 補題 3 より，$\mathcal{DAG}_{mk}$ は高々$\mathcal{D}+1$ ラウンド動作すると，正当な状況に到達する． □

4. 故障封じ込め自己安定プロトコル $\mathcal{DAG}$

プロトコル $\mathcal{DAG}_{mk}$ は，1 故障状況からの再安定までに，多くのプロセスが変数 $d.i$ を変更する場合がある．たとえば，あるプロセスが $d.i$ を実際のレベルより非常に小さい値にするような故障をした場合，故障プロセスが動作する前にその隣接プロセスが動作すると，誤ったレベル情報をもとにレベルを変更し，それが次々に伝播することで，多くのプロセスが $d.i$ を変更する可能性がある．これを防ぐには，1 故障状況からの実行において，各プロセスが 1 故障状況と思われる状況では動作せず，かつ故障プロセスが動作すれば正当な状況になることを保証すればよい．本章では，$\mathcal{DAG}_{mk}$ に対して故障封じ込めの性質を持たせたプロトコル $\mathcal{DAG}$ について述べる．

4.1 プロトコル $\mathcal{DAG}$

正当な状況から単一プロセス P_f が故障した 1 故障状況を考える．$\mathcal{DAG}_{mk}$ は，正当な状況に到達するとそれ以降各プロセスは状態を変化させないので，1 故障状況においてレベルの値を変更可能なプロセスは，P_f および P_f に隣接するプロセスのみである．つまり，1 故障状況において，プロセス P_i が矛盾状態になるのは以下のいずれかの場合である．

(1)　P_i が故障プロセス

(2)　P_i の隣接プロセス P_j が故障プロセス

(1) の場合は，P_i が $\mathcal{DAG}_{mk}$ を実行すれば，ただちに正当な状況に再安定する．一方，(2) の場合は，P_i は $\mathcal{DAG}_{mk}$ を実行せずに，P_j が $\mathcal{DAG}_{mk}$ を実行し正当な状態になるのを待たなければならない．つまり，故障封じ込めを実現するには，$\mathcal{DAG}_{mk}$ を実行してよい場合と，いけない場合を各プロセスが判断する必要があり，この判断のための条件を $\mathcal{DAG}_{mk}$ に追加することで $\mathcal{DAG}$ を得る．

あるプロセス P_i が矛盾状態で，かつその隣接プロセスの中にも矛盾状態のプロセス P_j が存在する場合を考える．このとき，P_j のレベル $d.j$ が $d'.j$ に変化することで，P_i，P_j が無矛盾状態になるような $d'.j$ を安定代入値と呼び，「P_i から見て P_j に安定代入が存在する」という．これを表す述語 $can\_stab(i)$ を定義する．

定義 5． ($can\_stab(i)$ の定義) 互いに隣接するプロセス P_i, P_j に関して，P_i と隣接プロセスからなる部分ネットワーク $P_i \cup N_i$ の状況を $C[i]$ とし，同様に $P_j \cup N_j$ の状況を $C[j]$ とする．また，その時の $cons(i)$ と $cons(j)$ をそれぞれ $cons(C[i])$，$cons(C[j])$ と表記する．

ここで，ある状況 $C[i]$，$C[j]$ (P_j のレベルを $d.j$ とする) と，$C[i]$，$C[j]$ に対して P_j のレベルの値のみを $d'.j$ に変化させた状況 $C'[i]$，$C'[j]$ を考える．このとき，述語 $can\_stab(i)$ を以下のように定義する．

$$can\_stab(i) = \neg cons(C[i]) \wedge \exists P_j \in N_i[\neg cons(C[j]) \wedge cons(C'[i]) \wedge cons(C'[j])]$$

つまり，矛盾状態であるプロセス P_i の矛盾状態である隣接プロセス P_j にプロセス P_i と

入力変数： $d.j,\ d'.j(P_j \in N_i)$
出力変数： $d.i$
SR1： $can\_stab(i) \longrightarrow no\ move$
SR2： $\neg can\_stab(i) \longrightarrow execute\ protocol\ \mathcal{DAG}_{mk}$

図 2 故障封じ込め自己安定プロトコル $\mathcal{DAG}$

P_j を無矛盾状態にするような現在のレベル $d.j$ と異なる値 $d'.j$ が存在するとき,「P_i から見て P_j に安定代入が存在する」といい，$can\_stab(i)$ を満たす. □

プロセス P_i が $can\_stab(i)$ を評価するには，隣接プロセス P_j が無矛盾状態となるレベルを知り，かつ P_j がそのレベルになることで自分が無矛盾状態になるかどうかを判定すればよい．P_j が無矛盾状態になるレベルを知るために，P_j の隣接プロセスの情報，つまり P_i から見て「隣接の隣接プロセス」の情報が必要なので，本章では各プロセスが「隣接の隣接プロセス」の状態を直接知ることができると仮定する．通常の状態通信モデル（隣接プロセスの情報のみ）で $can\_stab(i)$ を評価するプロトコルについては第 5 章で述べる．

定義 3 より，シンクプロセス P_i は隣接プロセスのレベルによって矛盾・無矛盾状態を変更しないので，P_i のすべての隣接プロセスに安定代入は存在せず，常に $\neg can\_stab(i)$ である．

故障プロセス P_f による 1 故障状況において，P_f で $\neg cons(f)$ かつ $\neg can\_stab(f)$ が成立する（4. 2 節）．また，P_f の隣接プロセス P_i で $cons(i)$ または $can\_stab(i)$ が成立する（4. 2 節）．よって，各プロセスで can_stab が成立するとき $\mathcal{DAG}_{mk}$ を実行しないことで，1 故障状況における故障封じ込め自己安定プロトコル $\mathcal{DAG}$ を実現する（図 2 参照）.

4.2 1 故障状況における故障プロセスとその隣接プロセスの状態

1 故障状況における故障プロセスとその隣接プロセスのとり得る状態について，より詳細に述べる．1 故障状況において，プロセス P_i が矛盾状態になり得るのは P_i が故障プロセスの場合と P_i の隣接プロセス P_j が故障プロセスの場合のみである．ここで以下の事実が言える.

- プロセス P_i が故障プロセスの場合，P_i は明らかに矛盾状態である.
- プロセス P_i の隣接プロセス P_j が故障プロセスの場合
  - P_i が矛盾状態の場合，隣接する故障プロセス P_j の故障後のレベルが $d.j$ とすると，P_j を無矛盾状態にするレベル $d'.j$ は，P_j の隣接プロセスのレベルの最小値 $+1$ なので，一意に定まる．P_i は $d'.j$（安定代入値）を計算可能であり，かつそれによって自分も無矛盾状態になる[⋆1]ことが分かる．これは，P_i 以外の P_j の任意の隣接プロセスでも言える．従って，P_j のすべての隣接プロセスから見て P_j に安定代入が存在（can_stab が成立）し，かつその安定代入値は一意に決定される.
  - P_i が無矛盾状態の場合，$\mathcal{DAG}_{mk}$ を実行してもレベルを変更しない.

以上から，1 故障状況において故障プロセスの隣接プロセスがレベルを変更しないことがわかる．よって，1 故障状況で故障プロセス P_f で常に $\neg can\_stab(f)$ となることを示せばよい．シンクプロセスは常に $\neg can\_stab$ なので，シンク以外のプロセスが故障する場合を考える．正当な状態での故障プロセス P_f のレベルを $d.f^-$，故障後のレベルを $d.f$ として，$d.f \leq d.f^- - 1$ と $d.f \geq d.f^- + 1$ の二つの場合に分けて考える.

- $d.f \leq d.f^- - 1$ の場合
  故障後のプロセス P_f のレベルが $d.f = d.f^- - 1$ のとき，P_f が無矛盾状態になるには，隣接プロセス P_i にレベルが $d.i = d.f^- - 2$ で無矛盾状態になるプロセスが必要である．しかし，プロセス P_i の隣接プロセス P_j のレベルの範囲は $d.f^- - 2 \leq d.j \leq d.f^- + 2$ であるため，そのような P_j は存在せず，$can\_stab(f)$ は成立しない．故障後のプロセス P_f のレベルが $d.f \leq d.f^- - 2$ の場合も同様の理由で，$can\_stab(f)$ は成立しない.
- $d.f \geq d.f^- + 1$ の場合
  故障後のプロセス P_f のレベルが $d.f = d.f^- + 1$ のとき，P_f が無矛盾状態になるには，隣接プロセスのレベルの最小値が $d.f^-$ でなければいけない．一方，P_f の隣接にはレベル $d.i = d.f^- - 1$ のプロセス P_i が存在する．P_i は無矛盾状態であり，レベルを変更しないので $can\_stab(f)$ は成立しない．また，故障後のプロセス P_f のレベルが $d.f \geq d.f^- + 2$ の場合も同様の理由で，$can\_stab(f)$ は成立しない.

以上より，故障プロセス P_f がシンクでない場合も P_f で常に $\neg can\_stab(f)$ が成立する.

本節の議論より，プロセス P_f の故障による 1 故障状況において，P_f は $\neg cons(f)$ かつ $\neg can\_stab(f)$ であり，P_f の隣接プロセス P_i は $cons(i)$ もしくは $can\_stab(i)$ である.

4.3 正当性の証明

プロトコル $\mathcal{DAG}$ の正当性を証明する前に，can_stab の分類について述べる.

分類：1 故障状況において「P_i から見て P_j に安定代入が存在する」場合，安定代入値 $d'.j$ の値によって $d.i \leq d'.j \leq d.i + 1$，$d'.j = d.i - 1$ の二つの場合に分類できる[⋆2].

⋆1 $d'.j$ として少なくとも P_j が故障する前のレベルの値が存在するので，必ず安定代入が存在する.
⋆2 これら以外の場合，P_j のレベルが $d'.j$ になっても P_i もしくは P_j が矛盾状態のままであることが容易に分かる.

- *can_stab* パターン 1：$d.i \leq d'.j \leq d.i+1$ の場合，以下のすべてが成立する状況である．
  （1） P_i の隣接（P_j を除く）にレベル $d.i-1$ のプロセスが存在する．
  （2） P_i の隣接（P_j を除く）プロセスのレベルは $d.i-1$ 以上である．
  （3） P_j のレベル $d.j$ は $d.i-1$ 未満である[⋆1]．
- *can_stab* パターン 2：$d'.j = d.i-1$ の場合，以下のすべてが成立する状況である．
  （1） P_i の隣接（P_j を除く）プロセスのレベルは $d.i-1$ 以上である．
  （2） P_j がシンクプロセスでなければ，P_j の隣接（P_i を除く）にレベルの値が $d.i-2$ のプロセスが存在する．

補題 7. 隣接する二つのプロセス P_i, P_j において，「P_i から見て P_j に安定代入が存在する」かつ「P_j から見て P_i に安定代入が存在する」は成立しない．

証明. 成立すると仮定して，矛盾を導く．プロセス P_i, P_j の両方で *can_stab* パターン 1 が成立すると仮定すると，*can_stab* パターン 1 の状況 (3) より，矛盾．

次に，片方で *can_stab* パターン 2，もう一方で *can_stab* パターン 1 または 2 が成立する場合を考える．

（1） P_i：*can_stab* パターン 2，P_j：*can_stab* パターン 1 が成立すると仮定する．
can_stab パターン 1 の状況 (3) より，$d.i < d.j-1$ である．また，*can_stab* パターン 2 の状況 (2) より，P_j の隣接にレベル $d.i-2(< d.j-1)$ のプロセスが存在することになり，P_j において *can_stab* パターン 1 の状況 (2) に矛盾．

（2） P_i：*can_stab* パターン 2，P_j：*can_stab* パターン 2 が成立すると仮定する．
この場合，一般性を失うことなく $d.i \leq d.j$ を仮定できる．*can_stab* パターン 2 の状況 (2) より，P_j の隣接にレベル $d.i-2(< d.j-1)$ のプロセスが存在することになり，P_j において *can_stab* パターン 2 の状況 (1) に矛盾．

以上より，仮定はすべての *can_stab* パターンで矛盾である． □

補題 8. プロセスの系列 $< P_0, P_1, ..., P_m > (m \geq 2)$ において，「$P_x(0 \leq x < m)$ から見て P_{x+1} に安定代入が存在する」かつ「P_m から見て P_0 に安定代入が存在する」は成立しない．

証明. 成立すると仮定して，矛盾を導く．プロセスの系列から，連続する三つのプロセス P_i, P_{i+1}, P_{i+2} を抜き出して考える．

（1） P_i で *can_stab* パターン 1 が成立する場合，*can_stab* パターン 1 の状況 (3) より，以下が成立する．
P_{i+1} のレベル $d.i_{+1}$ は $d.i-1$ 未満（$d.i-1 > d.i_{+1}$）⋯ (1.1)

（2） P_i で *can_stab* パターン 2 が成立する場合，*can_stab* パターン 2 の状況 (2) より，P_{i+1} の隣接にレベルの値 $d.i-2$ のプロセスが存在する．
- P_{i+2} のレベルが $d.i_{+2} \neq d.i-2$ の場合，P_{i+1} の隣接 (P_{i+2} を除く) にレベル $d.i-2$ のプロセスが存在することになり，P_{i+1} における *can_stab* パターン 1 の状況 (2)，又はパターン 2 の状況 (1) より，以下が成立する．
$d.i-2 \geq d.i_{+1}-1$ ⋯ (2.1)
- P_{i+2} のレベルが $d.i_{+2} = d.i-2$ の場合，P_{i+1} における *can_stab* パターン 1 の状況 (2)，又はパターン 2 の状況 (1) より，以下が成立する．
$d.i \geq d.i_{+1}-1$ ⋯ (2.2)

以上より，(1.1), (2.1) の場合はプロセス P_i, P_{i+1} のレベルは $d.i > d.i_{+1}$ である．つまり，レベルが小さくなっている．一方，それ以外 (2.2) の場合は $d.i \geq d.i_{+1}-1$ となり，P_{i+1} のレベルは P_i と等しいか，1 大きい．

プロセスの系列 $< P_0, P_1, ...P_m > (m \geq 2)$ において，「$P_x(0 \leq x < m)$ から見て P_{x+1} に安定代入が存在する」かつ「P_m から見て P_0 に安定代入が存在する」が成立し，かつすべてのプロセスで (1.1), (2.1) の *can_stab* が成立すると仮定すると，$d.0 > d.1 > ... > d.m$ かつ $d.m > d.0$ が成立することになり，矛盾．

次に，系列中のあるプロセスにおいて (2.2) の *can_stab* が成立する場合を考える．プロセス系列の m の値によって場合分けをして考える．

- $m = 2$ の場合
  P_i で (2.2) の *can_stab* が成立すると仮定すると，$d.i_{+2} = d.i-2$ が成立する．P_i と P_{i+2} は隣接しているので，P_i において *can_stab* パターン 2 の条件 (1) に矛盾する．
- $m = 3$ の場合
  P_i で (2.2) の *can_stab* が成立すると仮定すると，$d.i_{+2} = d.i-2$ が成立する．P_{i+2} で (2.2) の *can_stab* が成立する場合とそうでない場合に分けて考える．
  – P_{i+2} で (2.2) の *can_stab* が成立すると，$d.i = d.i_{+2}-2$ が成立することになり，矛盾．
  – P_{i+2} で (1.1), (2.1) の *can_stab* が成立すると，$d.i_{+2} > d.i_{+3}$ が成立することから，$d.i_{+2} = d.i-2 > d.i_{+3}$ が成立する．P_i と P_{i+3} は隣接しているので，P_i において *can_stab* パターン 2 の条件 (1) に矛盾する．
- $m \geq 4$ の場合

⋆1 これ以外の場合 P_i で $cons(i)$ が成立するので，$can\_stab(i)$ が成立しない．

$m \geq 4$ の場合も，$m = 3$ と同様の議論をくり返すことで矛盾を導くことができる．□

補題 9． プロトコル $\mathcal{DAG}$ は，正当な状況 $\mathcal{LE}_{DAG}$ に関する自己安定プロトコルである．

証明． $\mathcal{DAG}$ は $\mathcal{DAG}_{mk}$ に実行を抑制する条件を付加したものなので，$\mathcal{LE}_{DAG}$ に関する $\mathcal{DAG}$ の自己安定性を示すには，正当でない状況 c において，$\mathcal{DAG}_{mk}$ を実行し，レベルを変更するプロセスが存在することを示せばよい．c において，矛盾状態のプロセス P_i が存在する．P_i で $\neg cons(i) \wedge \neg can\_stab(i)$ が成立する場合，証明終了．P_i で $\neg cons(i) \wedge can\_stab(i)$ が成立する場合，P_i の隣接に矛盾状態のプロセス P_j が存在する．

(1) $P_j \in T$ の場合，シンクプロセスでは常に $\neg cons(j) \wedge \neg can\_stab(j)$ が成立する．

(2) $P_j \notin T$ の場合

(a) P_j が $\neg can\_stab(j)$ の場合，$\neg cons(j) \wedge \neg can\_stab(j)$ が成立する．

(b) P_j が $can\_stab(j)$ の場合，補題 7 より，P_j の隣接に P_i 以外で矛盾状態のプロセス P_k が存在し，P_k に対しても (1)，(2) の議論が適用できる．以降の議論で (2)-(b) であるプロセスがくり返し現れる場合があるが，補題 8 より，いずれ $\neg cons \wedge \neg can\_stab$ が成立するプロセスが現れる．□

4.2 節の議論より，以下の補題が成立する．

補題 10． 1 故障状況から始まるプロトコル $\mathcal{DAG}$ の実行で，$\mathcal{DAG}_{mk}$ を動作するのは故障プロセスのみである．□

定理 1，補題 9，補題 10 より，以下の定理が成り立つ．

定理 3． プロトコル $\mathcal{DAG}$ は DAG 構成問題を解く故障封じ込め自己安定プロトコルである．□

5. 状態通信モデルへ変更した故障封じ込めプロトコル $\mathcal{DAG}_{syn}$

本章では，状態通信モデル（隣接プロセスの情報のみ）で $can\_stab(i)$ を評価できるように，文献[6] と同様に同期機構を用いて必要な情報を隣接プロセス $P_j(P_j \in N_i)$ から得られるように変更したプロトコル $\mathcal{DAG}_{syn}$ について述べる．

5.1 状態通信モデルへ変更したプロトコル $\mathcal{DAG}_{syn}$

すべてのプロセス P_i は，各隣接プロセス P_j に対して以下の変数 q_{ij}, a_{ij} を持つ．

- q_{ij}：$\perp, ask$ のいずれかを持つ．P_i から P_j への質問が発行中かどうかを表す変数．
- a_{ij}：P_j が P_i に発行した質問に対する返答を表す変数．返答は q_{ji} 及び P_i と P_i の全隣接プロセスの状態（$all(i)$ と表す）から決定される．つまり，a_{ij} は 2 引数関数 f_i を用いて，$a_{ij} = f_i(q_{ji}, all(i))$ となる．質問が発行されていない場合，$\perp$ となる．

入力変数： $d.i,\ d.j,\ a_{ji}(P_j \in N_i)$
出力変数： $a_{ij}(P_j \in N_i)$

SR10： $\neg cons(i) \wedge q_{ij} = \perp \wedge a_{ji} = \perp \longrightarrow q_{ij} := ask$

SR11： $a_{ij} \neq f_i(q_{ji}, all(i)), \exists P_j \in N_i \longrightarrow a_{ij} := f_i(q_{ji}, all(i))$

SR12： $cons(i) \wedge q_{ij} \neq \perp \longrightarrow q_{ij} := \perp$

図 3 同期機構プロトコル

同期機構の動作は，次のとおりである．P_i が質問 q_{ij} に対する正しい返答 a_{ji} を得るには，a_{ji} が $\perp$ になるのを待って q_{ij} を ask にする．次に a_{ji} が $\perp$ 以外の値を持った場合，それは q_{ij} を ask にした（質問を発行した）時点以降の $all(j)$ から得られる値である．P_i が q_{ij} を ask に変えるのは，$can\_stab(i)$ の評価が必要な場合，つまり自分が矛盾状態の場合である．P_i は a_{ji} が $\perp$ 以外の値を持った場合，その値を用いてプロトコル $\mathcal{DAG}$ に従って動作する．その結果，$cons(i)$ が成立すると q_{ij} を $\perp$ にする．この同期機構プロトコルを図 3 に示す．

次に f_i（2 引数関数）を定義する．プロセス P_j で $can\_stab(j)$ を評価するには，矛盾状態の隣接プロセス P_i から P_i を無矛盾状態にする現在のレベルと異なる値 $d'.i(\neq d.i)$ を知らせてもらえばよい．つまり，f_i は $P_i \in T$ の場合は $a_{ij} = 0$，$P_i \notin T$ の場合は $a_{ij} = min_{P_k \in N_i}[d.k + 1]$ を返せばよいので，関数 f_i は以下のようになる．

$$f_i = \begin{cases} 0 & if \quad q_{ji} = ask \wedge P_i \in T \\ min_{P_k \in N_i}[d.k + 1] & if \quad q_{ji} = ask \wedge P_i \notin T \\ \perp & else. \end{cases}$$

$can\_stab(i)$ を評価する時点で P_i と $P_j(\forall P_j \in N_i)$ の間の同期動作が終了していることを保証するために，以下の述語を導入する．

定義 6．（同期動作終了）同期機構が動作終了していることを表す述語 $synchro(i)$．

$$synchro(i) = \neg cons(i) \wedge (\forall P_j \in N_i, q_{ij} \neq \perp) \wedge (\forall P_j \in N_i, a_{ji} \neq \perp)$$ □

$synchro(i)$ が真の場合，同期機構は動作を終了している．従って，各プロセスが can_stab の評価に従って $\mathcal{DAG}_{mk}$ を実行するのは $synchro(i)$ のときのみであり，これらの拡張を行ったプロトコル $\mathcal{DAG}_{syn}$ を図 4 に示す．

5.2 正当性の証明

補題 11． 各プロセスが同期機構プロトコルの SR10〜SR12 を少なくとも一度ずつ評価す

入力変数： $d.j,\ a_{ji}(P_j \in N_i)$
出力変数： $d.i$
SR1： $\neg synchro(i) \vee can\_stab(i) \longrightarrow no\ move$
SR2： $synchro(i) \wedge \neg can\_stab(i) \longrightarrow execute\ protocol\ \mathcal{DAG}_{mk}$

図 4 故障封じ込め自己安定プロトコル $\mathcal{DAG}_{syn}$

ると，それ以降 a_{ji} の値は，直前に P_i で q_{ij} が ask になった時点以降に P_j で計算された値である．また，$\neg cons(i)$ の場合，いつか必ず $synchro(i)$ となり，$cons(i)$ が成立するまで $synchro(i)$ のままである．

証明． 同期機構プロトコルと $synchro$ の定義より明らかである． □

補題 12． 1 故障状況から始まる実行の任意の状況において，任意の隣接プロセス P_i, P_j に対して，$q_{ij} = ask \wedge a_{ji} \neq \perp$ であれば，a_{ji} は直前に P_i が q_{ij} を ask にした時点以降に計算された値である．

証明． 1 故障状況で，故障によって変数 a, q が変化しなければ補題が成立することは明らか，したがって P_i, P_j のいずれかで故障が起きたと仮定する．

(1) P_j が故障し，$a_{ji} \neq \perp, q_{ij} = \perp$ となった場合
同期機構プロトコルの SR10 で q_{ij} が ask になる前に，a_{ji} は SR11 で $\perp$ になる．よって，a_{ji} は直前に q_{ij} が ask になった後に計算された値である．

(2) P_i が故障し，$a_{ji} = \perp, q_{ij} = ask$ となった場合
明らかに a_{ji} は，直前に q_{ij} が ask になった後に計算された値である． □

定理 3，補題 11，補題 12 より，以下の定理が成り立つ．

定理 4． プロトコル $\mathcal{DAG}_{syn}$ は DAG 構成問題を解く故障封じ込め自己安定プロトコルである． □

5.3 1 故障状況からの変動プロセス数と再安定時間

定理 5． プロトコル $\mathcal{DAG}_{syn}$ は，変動プロセス数 $\Delta^2 + 1$，再安定時間 $O(1)$ で DAG 構成問題を解く故障封じ込め自己安定プロトコルである．

証明． 1 故障状況において，同期動作によって状態が変化するのは矛盾状態のプロセスとその隣接プロセスのみである．従って，同期動作を行うのは故障プロセスから距離 2 以内のプロセスで，レベルを変更するのは故障プロセスのみである．よって，変動プロセス数は $\Delta^2 + 1$ である．

1 故障状況において，故障プロセスが一度 $\mathcal{DAG}_{mk}$ を実行すると正当な状況になり，正当な状況で同期動作が行われないことは明らかである．よって，故障プロセスが動作し正当な状況になるまでに高々 $\Delta^2 + 1$ 個のプロセスが同期動作をおこない，かつ故障プロセスで $\mathcal{DAG}_{mk}$ を実行すると，それ以降同期動作は行われない．また同期動作は明らかに定数回の動作で停止する．従って，同期スケジュールによる実行において，各プロセスが定数回動作すれば再安定するので，再安定時間は $O(1)$ である． □

6. おわりに

本稿では，指定したプロセス集合をシンクとする DAG を構成する故障封じ込め自己安定プロトコル $\mathcal{DAG}_{syn}$ を提案した．このプロトコルは 1 故障状況からの実行において，変動プロセス数 $\Delta^2 + 1$，再安定時間 $O(1)$ である．今後の課題としては，シンク・ソース両方のプロセスを指定した DAG 構成プロトコルへの拡張などが挙げられる．

参 考 文 献

1) S. Dolev, A. Israeli, S. Moran, “Self stabilization of dynamic systems assuming only read/write atomicity”, Proc. 9th PODC, pp. 103-117, 1990.
2) S. Ghosh, M. H. Karaata, “A self-stabilizing algorithm for coloring planar graphs”, Distributed Computing(1993), Volume 7, Number 1, 55-59.
3) A. K. Datta, M. Gradinariu, M. Raynal, G. Simon, “Anonymous publish/subscribe in P2P networks”, In Mobile Computing and Networking, pages 263-270, 1999.
4) M. L. Neilsen, M. Mizuno, “A dag-based algorithm for Distributed Mutual Exclusion”, Distributed Computing System, 20. May. 1991-24. May. 1991.
5) V. D. Park, M. S. Corson, “A highly adaptive distributed routing algorithm for mobile wireless networks”, In Proc of IEEE INFOCOM, Kobe, Japan, Apr. 1997.
6) 片山 喜章, 増澤 利光, “重み最小生成木を構成する故障封じ込め自己安定プロトコル”, 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理 J84-D-I(9), 1307-1317, 2001-09-01.