プログラミング 18－15
(1998. 3. 23)

リングネットワーク上での排他制御問題に対する故障封じ込め自己安定アルゴリズム

千星 裕 桝田 秀夫 辻野 嘉宏 都倉 信樹

大阪大学大学院基礎工学研究科情報数理系専攻
〒 560-8531 大阪府豊中市待兼山町 1-3
電子メール:{senboshi, h-masuda, tsujino, tokura}@ics.es.osaka-u.ac.jp

あらまし
自己安定アルゴリズムとは，任意のネットワーク状況から有限時間内にシステムが問題に対する望ましい状況 (解状況) に到達し安定する分散アルゴリズムである．このため，任意個のプロセスが一時故障が生じても，その後十分長い間故障が生じなければ，解状況に到達するという特徴がある．しかし通常，分散システムで同時に多数の故障が生じることは稀であると考えられる．そこで，少数個の故障が生じた状況から効率よく解状況に到達する自己安定アルゴリズムが望まれる．本稿では，(1) 動作可能なプロセスは公平に動作する，(2) 一時故障はどのプロセスでも等確率で生じる，と仮定する．このとき，リングネットワーク上で排他制御問題を解き，さらに唯一つのプロセスで一時故障が生じた状況から平均して定数時間で解状況に到達する自己安定アルゴリズムを，2 つの通信モデルのもとでそれぞれ提案する．

キーワード 分散アルゴリズム, 一時故障, 自己安定, 故障封じ込め

Fault Containing Self-Stabilizing Algorithms for Mutual Exclusion Problem on Ring Networks

Yutaka Senboshi, Hideo Masuda,Yoshihiro Tsujino, Nobuki Tokura

Graduate School of Engineering Science, Osaka University
1-3 Machikane-yama, Toyonaka, Osaka 560-8531, Japan
E-mail:{senboshi, h-masuda, tsujino, tokura}@ics.es.osaka-u.ac.jp

Abstract
Self-stabilizing algorithms are designed to guarantee convergence into some desired stable configuration (legitimate configuration) from any initial configurations arising out of an arbitrarily large number of transient faults. However, in a well-designed system, the simultaneous occurrence of a large number of faults is rare. As one of approaches focusing on this aspect, it is desirable to design self-stabilizing algorithms that efficiently recover from small number of faults. In this paper, we present two algorithms for mutual exclusion problem on ring networks. These algorithms are not only self-stabilizing, but also converge in constant time on average from a configuration with a single transient fault into legitimate configuration, if scheduler is fair and a transient fault occurs on every process with equal probability.

key words
distributed algorithm, transient fault, self-stabilization, fault-containment

1. はじめに

通信リンクで接続されたネットワーク上で，複数のプロセスが自律的に動作し，メッセージを交換しながら協調して問題を解くアルゴリズムを分散アルゴリズム (distributed algorithm) という．通常の分散アルゴリズムでは，アルゴリズム実行の開始時において，全てのプロセスの内部状態と通信リンクの状態で決まるネットワーク状況が，ある状況に初期化されているものとしている．

これに対し，任意のネットワーク状況からアルゴリズムを開始しても，有限時間内にシステムが問題に対する望ましい状況 (解状況) に到達し安定する分散アルゴリズムを自己安定アルゴリズム (self-stabilizing algorithm) という．自己安定アルゴリズムの特徴として，ネットワーク上の各プロセスを初期化する必要がないということが挙げられる．このことからアルゴリズム実行中にプロセスの持つ内部変数などのデータ (プログラムカウンタの値を含む) の内容の変質といった一時故障 (transient failure) やネットワーク形状の変化が起きても，有限時間内に問題を解くことができる．

プロセスの一時故障を考えるときに，システム中のプロセスが同時に多数故障するということはあまり考えられない．このことから，少数個のプロセスで一時故障が生じた状況から解状況に到達する過程においてシステムをどのように振る舞わせるかを議論することは重要であると考えられる．しかし，従来のリングネットワーク上で排他制御問題を解く自己安定アルゴリズムは，任意の状況から有限時間内に解状況に到達することを保証しているに留まり，このような少数個のプロセスで一時故障が生じた状況から解状況までの到達過程の議論はあまり行われてこなかった．しかし，最近では，少数個のプロセスでの故障から解状況までの到達過程の自己安定アルゴリズムの振る舞いに関して故障封じ込め (fault-containment) と強安定 (super-stabilization) [1, 2] という 2 つのアプローチがなされている．

自己安定アルゴリズムに対する故障封じ込めの概念は文献 [3] で提案されている．故障封じ込めアルゴリズムは，解状況から少数個 (通常 1 個) のプロセスで一時故障が生じた状況から，効率良く解状況に到達する分散アルゴリズムである．効率とは，回復までの時間，または解状況に到達するまでのプロセスの状態の変化する回数をいう．リーダ選択問題や生成木構成問題を解く故障封じ込め自己安定アルゴリズムが提案されている [4, 5]．また文献 [3] では，解状況において動作可能なプロセスが存在しない問題 (non-reactive な問題) に対して，それを解く自己安定アルゴリズムを，故障封じ込め自己安定アルゴリズムに変換する手法を提案している．

本稿では排他制御問題を解く故障封じ込め自己安定アルゴリズムについて議論する．排他制御問題は解状況においても動作可能なプロセスが存在する問題 (reactive な問題) である．この性質から，故障封じ込めアルゴリズムを施しても，故障プロセス，及び，その故障プロセスから定数距離にあるプロセスが動作せずに，故障前から特権を持つプロセスが動作し続けるような遷移をシステムがする場合があるので，定数時間で解状況に到達するとは言えなくなる．そのため，(1) 動作可能なプロセスは公平に動作する (2) どのプロセスでも等確率で一時故障が生じる，という仮定のもとで解状況までの平均的な到達時間を考える．ここで時間とはネットワーク状況の変化の回数である．本稿では，この仮定のもとで排他制御問題に対する故障封じ込めの定義をする．

次に，本稿の故障封じ込めの定義のもとで，既存の自己安定アルゴリズムが故障封じ込めであるかどうかを確かめるために，プロセス数 n の方向感覚付きリングネットワーク上で排他制御問題を解く Dijkstra の状態数 n の決定性自己安定アルゴリズム [6] を取り上げる．このアルゴリズムの実行により，一度解状況に到達し安定した状況でただ一つのプロセスで一時故障が生じた状況から，解状況に到達するまでの時間については議論されていない．本稿では，その平均的な到達時間，および最悪の場合の到達時間について，シミュレーションプログラムによる解析をする．その結果を見る限りにおいては，Dijkstra の自己安定アルゴリズムは，故障封じ込め自己安定アルゴリズムではないと判断できる．

そこで，この Dijkstra のアルゴリズムを，故障封じ込めを考慮したアルゴリズムに拡張する．そのために，まず故障封じ込めを考慮しやすい通信モデルとして，リングネットワーク上で各プロセスは 2 つ後方のプロセスまでの状態を参照することができると仮定した通信モデル (2 単方向状態通信モデル) のもとで，排他制御問題を解く決定性故障封じ込め自己安定アルゴリズム $FCSSA\_2uni$ を提案し，その正当性を示す．この通信モデルは一般的ではないが，本稿で述べる故障封じ込めのためのアイディアをストレートに議論することができる．

さらに，リングネットワーク上で各プロセスは双方向のプロセスの状態を参照することができると仮定したモデル (双方向状態通信モデル) のもとで，排他制御問題を解く決定性故障封じ込め自己安定アルゴリズム $FCSSA\_bi$ を提案し，その正当性を示す．このアルゴリズムのアイディアは $FCSSA\_2uni$ と同じであるが，2 つ後方のプロセスの状態を参照するかわりに，各プロセスは状態変数に加え補助変数を利用する．なお，本論文中の補題，定理の証明の詳細は省略する ([7] を参照)．

2. システムモデル

2.1 分散システム

分散システム D は，2 項組 $D = (R, \mathcal{A})$ から成る．R はリングネットワークで，$\mathcal{A}$ はある目的を達成するためのアルゴリズムである．

リングネットワーク R は，2 項組 $R = (P, L)$ で定義される．P はプロセスの集合であり，L は通信リンクの集合である．本稿では，プロセス数が n であるとし，

- $P = \{P_0, P_1, \cdots, P_{n-1}\}$
- $L = \{(P_i, P_{(i+1)\mathbf{mod}_n}) \mid 0 \leq i \leq n-1\}$

とする．つまり，プロセス $P_0, P_1, \cdots, P_{n-1}$ はこの順で単方向に並んでいる．以下簡単のため，特に言及する必要がないときは，全ての加減法は n の法のもとで計算されるとし，$P_{(i+1)\mathbf{mod}_n}$ は単に P_{i+1} と表す．さらに，P_i と P_{i+1} とは，通信リンクで結ばれているので，互いに隣接プロセスであるという．また，P_i にとって P_{i+1} を前方プロセス (successor) といい，P_i から P_{i+1} への方向を前方向という．同様に，P_i にとって P_{i-1} を後方プロセス (predecessor) といい，P_i から P_{i-1} への方向を後方向という．

アルゴリズム $\mathcal{A}$ は各プロセスの動作を定義するものである．つまり，各プロセスは状態機械 (state machine) であるとみなし，アルゴリズム $\mathcal{A}$ によって，各プロセス P_i に状態集合 S_i と状態遷移関数 δ_i が与えられる．ここで，各プロセスは隣接プロセスの状態を参照することができるという通信モデルを仮定する．この通信モデルを**状態通信モデル**という．本稿では次の 3 つの状態通信モデルのもとで議論を進める．

- 単方向状態通信モデル：各プロセス P_i は，隣接プロセスのうち後方プロセス P_{i-1} の状態を参照可能とするモデル
- 2-単方向状態通信モデル：状態通信モデルの仮定を拡張して，各プロセスは 2 つ後方のプロセスまでの状態を参照可能とするモデル
- 双方向状態通信モデル：各プロセス P_i は，2 つの隣接プロセス P_{i-1}, P_{i+1} の両方の状態を参照可能とするモデル

また，本稿では準均一 (semi-uniform) ネットワークを扱う．これは，定数個 (通常 1 個) の特別なプロセスがあり，それら以外は同一の状態機械であるネットワークをいう．

2.2 アルゴリズムの実行とスケジューラ

リングネットワークのシステム状況は各プロセスの状態の n 項組で表される．つまり，$C = S_0 \times S_1 \times \cdots \times S_{n-1}$ とすると，C がリングネットワークシステムの全状況集合を表し，$c \in C$ がネットワーク状況 (以後，単に状況) を表す．

$Q(\subseteq P)$ をプロセス集合の空でない部分集合とする．ある状況 c から Q に属するすべてのプロセスの動作によって状況 c' に遷移したとする．つまり，$c = (s_0, s_1, \cdots, s_{n-1})$，$c' = (s'_0, s'_1, \cdots, s'_{n-1})$ とし，この 2 つの状況で次が成立する．

$$s'_i = \begin{cases} \delta_i(s_i, \cdot) & P_i \in Q \\ s_i & P_i \notin Q \end{cases}$$

これを，$c' = \Delta(c, Q)$ と表す．ここで，$\delta_i(s_i, \cdot)$ は，単方向通信のもとでは $\delta_i(s_i, s_{i-1})$，2-単方向通信のもとでは $\delta_i(s_i, s_{i-1}, s_{i-2})$，双方向通信のもとでは $\delta_i(s_i, s_{i-1}, s_{i+1})$ を意味する．

スケジュール T とは，プロセスの空でない部分集合の無限系列をいう．$\mathcal{A}$ をアルゴリズム，c_0 を ($\mathcal{A}$ によって決まる) 状況，$T = Q_0, Q_1, \cdots$ を任意のスケジュールとする．このとき，ネットワーク状況の無限系列 $E = c_0, c_1, c_2, \cdots$ が，任意の i $(i \geq 0)$ について，$c_{i+1} = \Delta(c_i, Q_i)$ を満たすとき，E を，「初期状況 c_0，スケジュール T に対するアルゴリズム $\mathcal{A}$ の実行」という．

分散システムでは，システム全体の動作は非決定的である．つまり，ネットワーク中のどのプロセスがどういった順で動作するかは決定的ではない．そこで，プロセスの動作はデーモン (daemon) とよばれるスケジューラによって支配されているとする．

デーモンは仮定の違いによって幾つか分類できるが，本稿では以下の C デーモンを仮定する．

- C デーモン (Central daemon)
  各 $i(i \geq 0)$ に対して $|Q_i| = 1$ が成り立つスケジュール $T = Q_0, Q_1, \cdots$ のみを考えるモデル．つまり，動作可能なプロセスが複数個存在しても，同時には 1 つのプロセスしか動作しないスケジューラモデルである．

C デーモンによる状況の 1 回の遷移を 1 ステップとよぶ．さらに，ある状況において動作可能なプロセスが複数個存在するとき，C デーモンがそれらの中から実際に動作するプロセスを等確率で選ぶものとする．このとき，C デーモンは公平なスケジューラであるという．

3. 排他制御問題を解く故障封じ込め自己安定アルゴリズムの定義

リングネットワーク上での排他制御問題を考える．次に排他制御の解状況 (legitimate configuration) 及び，排他制御に対する要求を満たす実行を定義する．ここで，プロセス P_j が特権 (privilege) を持つとは，P_j があらかじめ定められた条件を満足しているときをいう．

定義 3.1 システム $D = (R, \mathcal{A})$ において，排他制御のための解状況 c とは，特権を持つプロセスがネットワーク上に高々一つ存在する状況をいう．さらに解状況の集合 c の集合 L を排他制御の解状況集合という． □

定義 3.2 システム $D = (R, \mathcal{A})$ において，$E = c_0, c_1, \cdots$ を $\mathcal{A}$ の実行系列とする．もし，E が次の条件を満たしているとき，E を排他制御の実行であるという．

1. 任意の状況 c_i $(i \geq 0)$ において，特権を持つプロセスがネットワーク上に高々 *1* つ存在する．
2. 任意のプロセス P_j $(0 \leq j \leq n-1)$ が実行系列 E において無限にしばしば特権を持つ． □

次に，アルゴリズム $\mathcal{A}$ が排他制御問題を解く自己安定アルゴリズムであることの定義をする．

定義 3.3 分散システム $D = (R, \mathcal{A})$ において，アルゴリズム $\mathcal{A}$ の実行が次を満たすとき，$\mathcal{A}$ は排他制御のための自己安定アルゴリズムであるという．

1. 任意の状況からアルゴリズム $\mathcal{A}$ を開始しても，ついには，特権を持つプロセスが *1* つになる (収束性)
2. 特権を持つプロセスが *1* つになると，それ以降は特権を持つプロセスが常に高々 *1* つのままである (閉包性)
3. 全てのプロセスは無限にしばしば特権を持つ (公平性)

□

自己安定アルゴリズムは解状況集合のある部分集合で安定する．自己安定アルゴリズム $\mathcal{A}$ による安定状況 (stable configuration) を次に定義する．

定義 3.4 分散システム $D = (R, \mathcal{A})$ において，状況 c から開始されるアルゴリズム $\mathcal{A}$ の実行を $E(c)$ とする．次を満たす状況 c を (自己安定アルゴリズム $\mathcal{A}$ による) 安定状況，その集合 L_s を安定状況集合という．

- 実行 $E(c)$ が排他制御の実行である．
- 実行 $E(c)$ で出現する全ての状況は無限にしばしば出現する． □

定義からも，$L_s \subseteq L$ であることがわかる．一旦，安定状況に到達すると，それ以後のアルゴリズムの実行により安定状況集合 L_s 中の状況を遷移し続け，L_s から逸脱しない．しかも，安定状況到達後の実行は排他制御の実行である．自己安定アルゴリズムにより，任意の状況から解状況に到達し，さらに安定状況に到達すると安定状況集合内を遷移し続ける．

さらに，排他制御の安定状況 $c \in L_s$ からただ 1 つのプロセスで一時故障が生じた場合，どれだけのステップ数で解状況集合 L に到達するかを考える．そのために，まず以下に 1-故障状況を定義する．

定義 3.5 分散システム $D=(R,\mathcal{A})$ において，L を排他制御の解状況集合，L_s をアルゴリズム A によって決まる安定状況集合とする．安定状況 $c \in L_s$ からただ 1 つのプロセス p で一時故障が生じ，状況 c' $(c' \notin L)$ に変化したとする．このとき，状況 c' を 1-故障状況といい，またプロセス p を 状況 c の故障プロセスという．

さらに，次の集合 F を 1-故障状況集合という．

$$F = \{c' \in C | c \in L_s \ : \ diff(c,c') = 1 \wedge c' \notin L\}$$

ここで，$diff(c,c')$ とは状況 c と c' とで状態が異なるプロセスの数である．

□

本稿では次のような仮定のもとで，排他制御問題を解く故障封じ込めアルゴリズムの定義をする．

(仮定 1) C デーモンは公平なスケジューラである．

(仮定 2) どのプロセスでも等確率で一時故障が生じる．

つまり，プロセス数が n のネットワークにおいて，任意の解状況 $c \in L$ から一時故障が生じ 1-故障状況 $c' \in F$ になったとき，その故障プロセスが P_i $(i \in \{0..n-1\})$ である確率は $\frac{1}{n}$ である．

さらに，プロセス数が n のネットワークにおいて，状況 $c' \notin L$ から 1 ステップで遷移する状況の集合を $\xi(c')$ とする[1]．また，状況 $c' \notin L$ から解状況に到達するまでの実行系列の集合を $E(c')$ とし，実行 $e \in E(c')$ に対する系列の長さ (つまり，状況 c' から実行 e によって解状況に到達するまでのステップ数) を $l(e)$ とする[2]．ここで，状況 $c' \in C$ からの平均到達ステップ数 $avg(n,c')$ を次のように再帰的に定義する．

$$avg(n,c') = \begin{cases} \frac{1}{|\xi(c')|} \cdot \sum_{c \in \xi(c')} avg(n,c) \ + \ 1 \ , & c' \notin L \\ 0 \ , & c' \in L \end{cases} \quad (1)$$

また，状況 $c' \notin L$ からの最悪到達ステップ数 $wst(n,c')$ を

$$wst(n,c') = \max\{l(e) \mid e \in E(c')\} \quad (2)$$

とする．さらに，1-故障状況集合 F からの平均到達ステップ数 $average(n)$ を，

$$average(n) = \frac{1}{|F|} \cdot \sum_{c' \in F} avg(n,c') \quad (3)$$

とし，また，1-故障状況集合 F からの最悪到達ステップ数 $worst(n)$ を，

$$worst(n) = \max\{wst(n,c') \mid c' \in F\} \quad (4)$$

とする．

このときに，排他制御問題を解く故障封じ込め自己安定アルゴリズム $\mathcal{A}$ を次に定義する．

定義 3.6 リングネットワーク R において次の条件を満たすとき，$\mathcal{A}$ は排他制御問題を解く故障封じ込め自己安定アルゴリズムであるという．

1. $\mathcal{A}$ が排他制御を解く自己安定アルゴリズムであり，かつ
2. 1-故障状況集合 F からの平均到達ステップ数 $average(n)$ が $O(1)$ である．

□

[1]排他制御問題では，状況 $c' \notin L$ において，動作可能なプロセスが 2 つ以上存在するので，$|\xi(c')| \geq 2$

[2]自己安定アルゴリズムでは，任意の状況から解状況に有限ステップ内に到達するので，$|E(c')|, l(e)$ は有限

4. Dijkstra の状態数 n の自己安定アルゴリズム

Dijkstra は，方向感覚付きのリングネットワークで排他制御問題を解く準均一な決定性自己安定アルゴリズムを提案した [6]．ここではプロセスの状態数 n のアルゴリズムを説明する[3]．このアルゴリズムでは状態通信モデルを用いており，C デーモンの下で動作をするものとする．また，各プロセスは一時故障を生じても，方向感覚は失わないものとする．

アルゴリズムは，幾つかの文 (statement) からなり，各プロセスに対して与えられる．さらに，文は条件部 (guard) とそれに対する動作 (action) からなる．各プロセスのいずれかの文の条件部が成立するとき，そのプロセスは動作可能であるという．条件部の成立する文がデーモンによってスケジュールされたときに，プロセスがその条件部に対する動作をアトミックに実行する．

Dijkstra の排他制御問題を解く自己安定アルゴリズム $DSSA$ を図 1 に示す．アルゴリズム中の s_i は状態を表す変数であり，状態変数 s_0 に関する加減法は n の法のもとで計算され，各 s_i は集合 $\{0..n-1\}$ のいずれかの値をとる．

[Dijkstra's SSA (以下，$DSSA$)]

Algorithm for process P_0 :

$s_{n-1} = s_0 \quad \Rightarrow \quad s_0 := (s_0 + 1) \mathbf{mod} n$

Algorithm for process P_i $(1 \leq i < n)$:

$s_{i-1} \neq s_i \quad \Rightarrow \quad s_i := s_{i-1}$

図 1: Dijkstra の状態数 n の自己安定アルゴリズム

ここで以後，$DSSA$ の条件部を次のようにあらわす．

$$\text{条件部}: \quad t(i) \equiv \begin{cases} s_{i-1} = s_i, & i = 0 \\ s_{i-1} \neq s_i, & 1 \leq i < n \end{cases}$$

P_i がトークンを持つとは，プロセス P_i の条件部 $t(i)$ が成立する状態をいう．P_i がトークンを放すとは，プロセス P_i で $t(i)$ が成立し，かつデーモンにスケジュールされ動作することをいう．$DSSA$ において P_i が特権を持つとは，P_i がトークンを持つときをいう．$DSSA$ において P_i が動作可能であることと，トークンを持つこと，及び特権を持つことは同義である[4]．

$DSSA$ に関して，(仮定 1), (仮定 2) のもとで，1-故障状況からの平均的な到達ステップ数を解析する．つまり，1-故障状況集合 F からの平均到達ステップ数 $average(n)$ を 式 (3) に従って求める．しかし，この式を理論的に求める事は困難である．そこで本稿では，プロセス数 n の値を与えることで $average(n)$ を計算するシミュレーションプログラムを作成した．この計算は一台の計算機上で以下のような要領で行った．

- プロセス数 n を入力として，P_i $(0 \leq i < n)$ のみが特権を持つ安定状況を生成する．

[3]本来は状態数 K のアルゴリズム (K は $K \geq n$ なる任意の定数) であるが，K の値が n とかけ離れているほど解状況に到達するのに時間がかかる．ここでは議論の簡単のため，リングサイズは n で固定し，どのプロセスもリングサイズ n を知っているとする

[4]この 3 つは $DSSA$ では同義だが，後述のアルゴリズムでは異なるので注意．各アルゴリズムでプロセスが特権を持つときの条件が異なる．

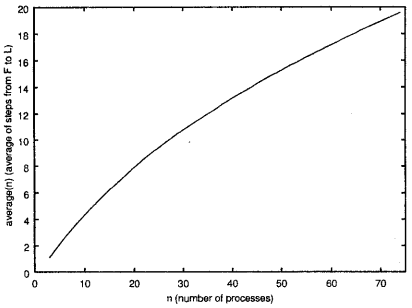

図 2: プロセス数 n に対する平均到達ステップ数

n	$c \in F$ i	j	$\|E(c)\|$	$avg(n,c)$	$wst(n,c)$
3	0	1	4	1.333333	2
4	3	1	30	2.003150	12
5	3	1	4191	2.807178	17
6	3	1	942598	3.479215	22
7	3	1	199136267	4.061629	27
8	3	1	39334757287	4.579724	32
9	3	1	7436098598008	5.049443	37
10	3	1	1368619002914280	5.481411	42

図 3: 計算結果の抜粋

- 生成した安定状況 $c \in L$ それぞれに対し, P_j $(0 \leq j < n)$ の状態変数 s_i の値を変えて 1-故障状況を生成する.
- 生成した 1-故障状況 c' のそれぞれに対し, c' から $DSSA$ を実行を開始し, 解状況に到達する全実行系列を求め, 式 (1), 式 (2) に従い $avg(n,c'), wst(n,c')$ を求める.
- さらに式 (3), 式 (4) から, $average(n), worst(n)$ を求める.

この解析では, 1-故障状況 $c' \in F$ から解状況に到達するまでの全ての実行系列を捜査している. $DSSA$ の実行によりプロセス数 n のリングネットワーク上で 1-故障状況から解状況までの平均到達ステップ数 $average(n)$ を解析した結果を図 2 に示す. 横軸がプロセス数 n で 3 から 75 まで, 縦軸がそれに対する $average(n)$ の値である. この結果を見る限りにおいては, $average(n)$ の値は n に関して増加しており, $DSSA$ は故障封じ込め自己安定アルゴリズムでないと判断できる.

図 2 はシミュレーションプログラムの出力データの抜粋である. プロセス数 n が 3 から 10 のリングネットワークのそれぞれに対して, プロセス P_i のみが特権を持つ状況でプロセス P_j で一時故障が生じた状況 $c \in F$ から解状況に到達するまでの全実行系列の集合の大きさ $|E(c)|$, 平均到達ステップ数 $avg(n,c)$, 及び, 最悪到達ステップ数 $wst(n,c)$ を示している. ここで示したデータは n に対して, $worst(n) = wst(n,c)$ となる状況 c に対するデータを抜粋している. 計算結果からは $DSSA$ では, 任意の 1-故障状況から 高々 $5n$ ステップで収束することがわかる.

5. 2-単方向状態通信モデルでの FC-SSA

5.1 故障封じ込めのためのアイディア

$DSSA$ をベースにした故障封じ込め自己安定アルゴリズムを提案する. そのためのアイディアを示す.

$DSSA$ による安定状況において, プロセス P_i で一時故障が生じ状態変数 s_i の値が変質し, $\neg t(i)$ が成立する状況から[5] $t(i), t(i+1)$ が成立する状況になったときを考える. このとき $DSSA$ の実行により,

- P_i が P_{i+1} より先にスケジュールされれば, s_i の値が故障前の値に戻り, 安定状況に戻る.
- P_{i+1} が P_i より先にスケジュールされれば, P_{i+1} の特権が前方へ移動し $t(i+2)$ が成立する. この結果, リング上のトークン数は減らない.

$DSSA$ においてトークンを持つプロセスが 2 個以上存在する場合, トークン数が減少する一条件は, トークンを持つプロセスが隣接し, かつそのうちの後方プロセスが先にスケジュールされることである. ことから, 次のようなことが実現できれば, 一時故障が生じた状況からすばやく解状況に到達できると考えられる : プロセス P_i がトークンを持つとき, プロセス P_{i-1} がトークンを持つかどうかを調べ,

- P_{i-1} がトークンを持たないなら, P_i は動作する
- P_{i-1} がトークンを持つなら, P_i は動作しない

しかし, このようなことを実現すれば, 全てのプロセスがトークンを持つ状況では, どのプロセスの条件部も成立せず, デッドロック状態に陥る. このような状況を避けるために, P_0 に限っては, P_0 がトークンを持つとき, P_{n-1} がトークンを持っているかどうかに関係なく, 動作することにする.

5.2 アルゴリズム $FCSSA\_2uni$

このアイディアをまず, 2-単方向状態通信モデルのもとで議論する. この 2-単方向状態通信モデルのもとでは, 2 つ後方のプロセスまでの状態が参照できる. プロセス P_i において, P_i がトークンを持つかどうかは s_i と s_{i-1} を, P_{i-1} がトークンを持つかどうかは s_{i-1} と s_{i-2} を参照することができるので, 既に述べた故障封じ込めのアイディアをそのままアルゴリズムに反映することができる. 次の図 4 にアルゴリズムを示す.

[FCSSA_2uni]
Algorithm for process P_0:
$t(0) \Rightarrow s_i := (s_{i-1} + 1) \mathbf{mod} n;$
Algorithm for process P_i $(1 \leq i \leq n-1)$:
$t(i) \wedge \neg t(i-1) \Rightarrow s_i := s_{i-1};$

図 4: FCSSA on 2-unidirectional ring

$FCSSA\_2uni$ において, P_i が特権を持つとは, P_i に与えられた文の条件部が成立することをいう.

次に, 任意の 1-故障状況 $c \in F$ からアルゴリズムの実行を開始した時に, 解状況に到達するまでの平均ステップ数を求めることで, $FCSSA\_2uni$ が故障封じ込め自己安定アルゴリズムであることを示す.

[5] $t(i+1)$ であっても $\neg t(i+1)$ であってもいい.

定理 5.1 アルゴリズム *FCSSA_2uni* は排他制御問題を解く自己安定アルゴリズムである. □

補題 5.2 状況 $c \in L_s$ において P_i $(i \in \{0..n-1\})$ が特権を持つプロセスとする. P_i で一時故障が生じ, 状況 c' になったとき, $c' \in L$ である. □

補題 5.3 P_i $(i \in \{1..n-1\})$ が特権を持つ安定状況 $c \in L_s$ において, P_0 で一時故障が生じ, 状況 c' になったとする. このとき, 状況 c' から解状況へは高々 $2n-i$ ステップで到達する. □

補題 5.4 P_i $(i \in \{1..n-1\})$ が特権を持つ安定状況 $c \in L_s$ において, P_j (ただし, $j \in \{1..n-2\}, i \neq j$) で一時故障が生じ, 状況 c' になったとする. このとき, 状況 c' から解状況への平均到達ステップ数は高々 2 ステップで, 最悪到達ステップ数は高々 $(j-i-1)\mathbf{mod}\ n$ ステップである. □

補題 5.5 P_i $(i \in \{1..n-2\})$ が特権を持つ安定状況 $c \in L_s$ において, P_{n-1} で一時故障が生じ, 状況 c' になったとき, 状況 c' から解状況へは高々 $2n$ ステップで到達する. □

定理 5.6 アルゴリズム *FCSSA_2uni* は排他制御問題を解く故障封じ込め自己安定アルゴリズムである.

(略証) 補題 5.3, 5.4, 5.5 より, プロセス数が n のリングネットワークにおいて,

$$average(n) < \frac{1}{n}\left(2n + \sum_{i=1}^{n-2} 2 + 2n\right) < 6$$

よって 1-故障状況から平均で高々 6 ステップという定数ステップで到達する. □

(仮定 2) より, どのプロセスも等確率で故障すると仮定した場合を考えたが, プロセス P_0 が一時故障を起こしにくいプロセスであれば, その平均ステップ数の値は小さくなる. また, P_{n-1} で生じる一時故障で, 到達までに高々 $2n$ ステップかかるような故障が生じる場合は少ないと考えれば, さらにその値は小さくなる. また, 最悪のスケジューリングを考えても, *FCSSA_2uni* では任意の 1-故障状況から高々 $2n$ ステップで到達する. *DSSA* の最悪到達ステップ数は高々 $5n$ であるので, オーダーでは変わらないが, *FCSSA_2uni* ではプロセス P_0 と P_{n-1} 以外の一時故障による 1-故障状況からの到達には高々 n ステップで到達する.

FCSSA_2uni による 1-故障状況から解状況までの平均到達ステップ数を, 4. 節と同様のシミュレーションプログラムによって解析した. その結果を図 5 に示す. 理論的に *FCSSA_2uni* の実行によって 1-故障状況から解状況までの平均到達ステップ数が高々 6 ステップであることを示したが, 実際にはその半分程度の値に漸近していることが分かる.

6. 双方向状態通信モデルでの FCSSA

6.1 アルゴリズム *FCSSA_bi*

前節の 2-単方向状態通信モデルは一般的なモデルではない. 通常, 状態通信モデルは各プロセスは高々その隣接プロセスの状態が参照できるという仮定がなされる. ここでは, 5.1 節のアイディアを双方向状態通信モデルで実現する. 各プロセス P_i において P_{i-1} がトークンを持つかどうかを確かめるためには P_{i-2} の状態を参照する必要がある. しかし, 双方向状態通信モデルでは 2-単方向状態通信モデルと異なり各プロセス P_i は P_{i-2} の状態を参照することが出来ない. そこで各プロセス P_i に次の 2 つの補助変数 q_i, a_i を与え, その変数の値を参照することで後方プロセス P_{i-1} のトークンの有無をチェックする.

- $q_i \in \{0,1\}$ (ただし, $1 \leq i \leq n-1$): P_i が P_{i-1} に対してトークンの有無を問い合わせているかどうかを表す変数

$$q_i \equiv \begin{cases} 1, & (t(i-1) \text{の成立を問い合わせている状態}) \\ 0, & (\text{問い合わせていない状態}) \end{cases}$$

- $a_i \in \{\perp, 0, 1\}$ (ただし, $0 \leq i \leq n-2$): P_i の P_{i+1} に対する返答を表す変数

$$a_i \equiv \begin{cases} \perp, & (\text{問い合わせを受けていない状態}) \\ 1, & (P_i\text{がトークンを持っている状態}) \\ 0, & (P_i\text{がトークンを持っていない状態}) \end{cases}$$

さらに, a_i の値を与えるための各プロセスに 関数 f_i を与える.

$$f_i \equiv \begin{cases} \perp, & if\ q_{i+1} = 0 \\ 1, & if\ q_{i+1} = 1 \wedge t(i) \\ 0, & if\ q_{i+1} = 1 \wedge \neg t(i) \end{cases}$$

ここで, デッドロック状態を避けるために, P_0 がトークンを持つとき, P_{n-1} がトークンを持っているかどうかに関係なく P_0 は特権を持つようにするので, プロセス P_0 には q_0 をプロセス P_{n-1} には a_{n-1} を与えない. よって, 各プロセスの状態空間は $S_0 = (s_0, a_0)$, $S_i = (s_i, q_i, a_i)$ $(\forall i \in \{1..n-2\})$, $S_{n-1} = (s_{n-1}, q_{n-1})$ であるので, *FCSSA_bi* における状況集合 C は状態空間の n 項組で $C = S_0 \times \cdots \times S_{n-1}$ である.

図 6 は双方向通信リング上での排他制御問題を解く故障封じ込め自己安定アルゴリズム *FCSSA_bi* である[6]. $St_k[\cdot]$ は文の名前を表す. *FCSSA_bi* において, P_i が特権を持つとは, $S_i[2]$ の条件部が成立しているときをいう.

[6] P_{n-1} においては $St_{n-1}[4]$ は恒偽であるとする

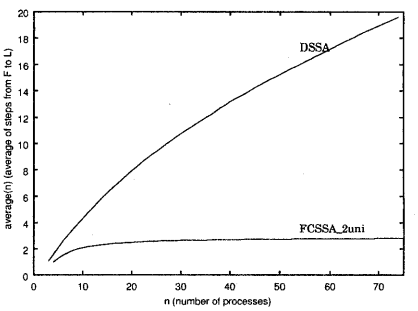

図 5: プロセス数に対する平均到達ステップ数 (上は *DSSA*, 下は *FCSSA_2uni* による実行)

$[FCSSA\_bi]$
Algorithm for process P_i $(1 \leq i \leq n-1)$:
$St_i[1] : t(i) \wedge (q_i = 0 \wedge a_{i-1} = \perp) \Rightarrow q_i := 1;$
$St_i[2] : t(i) \wedge (q_i = 1 \wedge a_{i-1} = 0) \Rightarrow s_i := s_{i-1},$
$q_i := 0;$
$St_i[3] : \neg t(i) \wedge (q_i \neq 0) \Rightarrow q_i := 0;$
$St_i[4] : a_i \neq f_i \Rightarrow a_i := f_i;$
Algorithm for process P_0 :
$St_0[2] : t(0) \Rightarrow s_i := (s_{i-1} + 1) \mathbf{mod} n;$
$St_0[4] : a_0 \neq f_0 \Rightarrow a_0 := f_0;$

図 6: FCSSA on bidirectional ring

アルゴリズム $FCSSA\_bi$ において，P_i $(0 \leq i \leq n-1)$ の状態変数 s_i の値が変わるのは，$St_i[2]$ の条件部が成立して，かつその動作が実行されたときのみ (つまり，P_i が特権を放すときのみ) である．また，$St_i[1], St_i[2], St_i[3]$ の条件部は排反で，互いが同時に成立することはない．また，$St_i[4]$ の条件部は他の 3 つの文の条件文と同時に成立することがあるが，互いの動作により，他方が成立しなくなることはない．

$FCSSA\_bi$ における解状況集合 L とは特権を持つプロセスが高々 1 つ存在し，かつ，その後の任意の実行で特権を持つプロセスが 2 つ以上にならない状況の集合である．解状況は，次の条件を満足する $i \in \{0..n-1\}$ が唯一つ存在する状況である．

$$(\forall k | 0 \leq k < i : \neg t(k) \wedge q_k = 0) \wedge (t(i) \wedge a_{i-1} \neq 1)$$
$$\wedge (\exists j | i \leq j \leq n-1 : (\forall k | i < k \leq j : t(k) \wedge a_{k-1} \neq 0)$$
$$\wedge (\forall k | j < k \leq n-1 : \neg t(k) \wedge q_k = 0))$$

更に，$FCSSA\_bi$ における安定状況集合 L_s とは次の条件を満足する状況の集合である．

- P_0 のみがトークンを持つとき，
  $(\forall i | 1 \leq i \leq n-1 : s_{i-1} = s_i \wedge q_i = 0)$
- P_i のみがトークンを持つとき，
  $s_{n-1} \neq s_0 \wedge (s_{i-1} = s_i + 1) \wedge (a_{i-1} \neq 1)$
  $\wedge (\forall j | 1 \leq j \leq n-1, j \neq i : (s_{j-1} = s_j) \wedge q_i = 0)$

$FCSSA\_bi$ による安定状況集合 L_s は，各状態変数 s_i に関する条件に関しては，$FCSSA\_2uni$, $DSSA$ による安定状況集合と同じで，リングネットワーク上にトークンを持つプロセスが唯一つ存在する状況で安定する．また，各補助変数 q_i, a_i に関しては解状況の条件式と同じである．

安定状況での各 P_i $(i \neq 0)$ の状況遷移図を図 7 に示す．各状態は s_i と s_{i-1} から分かる $t(i)$ の真偽と q_i と a_{i-1} の値の組である．状態を結ぶ有向辺には，状態を遷移する時に実行される文と，その実行により値が変化する変数名を示している．

安定状況における $FCSSA\_bi$ の実行において，P_{i-1} が特権を持つ状況 (say, $c_p^{(i-1)}$) から，P_i が特権を持つ状況 (say, $c_p^{(i)}$) までを考える．P_{i-1} が $St_{i-1}[2]$ の動作によって特権を放すと，P_i はトークンを持つ状況 (say, $c_t^{(i)}$) になる．状況 $c_t^{(i)}$ から状況 $c_p^{(i)}$ まで遷移する間に幾つかのプロセス P_j $(j \neq i)$ において $St_j[4]$ の条件部が成立していて，それが動作する場合がある．しかし，特権がリングネットワーク上を一周するまでの間で考えると，P_j が特権を持ってからちょうど一回の $St_j[4]$ の動作で $a_j = \perp$ となるので，各プロセスがトークンを持ってから特権を持つまでは 3 回スケジュールされる．つまり，状況 $c_t^{(i)}$ から状況 $c_p^{(i)}$ まで到達するまでのステップ数を $step(i)$ とすると，特権がリングネットワーク上を一周するまでを考えると $\sum_{i=1}^{n-1} step(i) = 3(n-1)$ であり，各プロセス P_i $(i \neq 0)$ がトークンを持ってから特権を持つまでのステップ数は平均して 3 ステップである．つまり，$FCSSA\_bi$ において，P_i がトークンを持ってから，q_i, a_{i-1} を介して P_{i-1} がトークンを持たない事を確かめるために，平均 3 ステップがオーバーヘッドとしてかかり，$FCSSA\_2uni$ の実行では，特権が後方に移動するのに 1 ステップで済んだが，$FCSSA\_bi$ の実行では，特権が後方に移動するのに平均 4 ステップかかる．

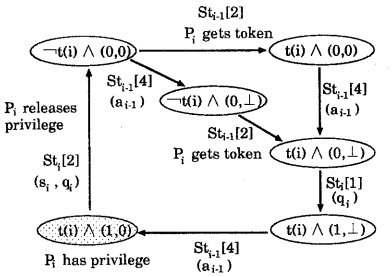

図 7: 安定状況における P_i $(i \neq 0)$ の状態遷移図

次に，$FCSSA\_bi$ が故障封じ込め自己安定アルゴリズムであることを示す．

定理 6.1 アルゴリズム $FCSSA\_bi$ は排他制御問題を解く自己安定アルゴリズムである． □

唯一つのプロセスで一時故障には，状態変数 s_i の内容変質のみでなく，補助変数 q_i, a_i の内容変質も含まれる．まず，P_i で一時故障が生じたとき，状態変数 s_i の値が変質していない場合を考える．

補題 6.2 安定状況 c において P_i で一時故障が生じたとき，状態変数 s_i の値が変質していなければ定数ステップで解状況に到達する． □

次に，P_i で一時故障が生じたときに，状態変数 s_i の値も変質している場合を考える．この状態変数 s_i の値も変質によって変化するのは $t(i), t(i+1)$ の真偽のみであり，P_i, P_{i+1} 以外の各プロセス P_j に対して，$t(j)$ または $\neg t(j)$ の成立は変化しない．$FCSSA\_2uni$ では一時故障によって $t(i) \wedge t(i+1)$ (ただし $0 \leq i \leq n-2$) が成立するような状況になったとき，P_{i+1} は P_i より先に動作することがなかった．しかし，$FCSSA\_2uni$ では，P_{i+1} は補助変数 q_i, a_i を介して $t(i)$ の成立を確かめるので，P_i で一時故障が生じると同時に P_{i+1} が特権を持ち，P_i より先に特権を放すことがある．しかし，これは限られた場合であり，$FCSSA\_bi$ による実行によっても 1-故障状況から解状況までの平均到達ステップ数は定数であることを以下で示す．

補題 6.3 安定状況において P_i $(0 \leq i \leq n-2)$ で一時故障が生じて状態変数 s_i の値が変質し，$t(i), t(i+1)$ が成立する状況になったとき，故障前の状況において $\neg t(i+1)$

が成立していれば[7]，P_{i+1} は P_i より先に特権を持たない．□

補題 6.4 安定状況において P_i $(0 \leq i \leq n-2)$ で一時故障が生じて状態変数 s_i の値が変質し $t(i)$, $t(i+1)$ が成立する状況になったとき，故障前の状況において $t(i+1)$ が成立していれば P_{i+1} が P_i と同時，もしくは先に特権を持つことがある．□

以上の補題 6.3 と補題 6.4 より *FCSSA_bi* の実行による状態変数 s_i の変化は *FCSSA_2uni* の実行による状態変数 s_i の変化と同じである．しかし，*FCSSA_bi* では各 P_i がトークンを持つと，$t(i) \wedge \neg t(i-1)$ が成立していても補助変数 q_i, a_{i-1} を介して P_{i-1} のトークンの有無を調べるため，実際に P_i が特権を持ち，スケジュールされて状態変数 s_i の値を変化させるまで(特権を放すまで)に 4 回スケジュールされる必要がある．つまり，各 s_i の値を変化させるために *FCSSA_2uni* では 1 ステップで実行することを *FCSSA_bi* では 4 ステップかけて実行する．

補題 6.5 *FCSSA_bi* における安定状況 $c \in L_s$ において P_i $(0 \leq i \leq n-1)$ がトークンを持つプロセスとする．P_i で一時故障が生じ状況 c' になったとき，状況 c' は *FCSSA_bi* における排他制御問題の解状況である．□

補題 6.6 , *FCSSA_bi* における安定状況 $c \in L_s$ において P_i $(1 \leq i \leq n-1)$ がトークンを持つプロセスとする．P_0 で一時故障が生じ状況 c' になったとき，状況 c' から解状況へは高々 $4(2n-i)$ ステップで到達する．□

補題 6.7 , *FCSSA_bi* における安定状況 $c \in L_s$ において P_i $(1 \leq i \leq n-1)$ がトークンを持つプロセスとする．P_j (ただし，$1 \leq j \leq n-2, i \neq j$) で一時故障が生じ状況 c' になったとき，状況 c' から解状況への平均到達ステップ数は高々 16 ステップである．□

補題 6.8 , *FCSSA_bi* における安定状況 $c \in L_s$ において P_i $(1 \leq i \leq n-2)$ がトークンを持つプロセスとする．P_{n-1} で一時故障が生じ状況 c' になったとき，状況 c' から解状況へは高々 $8n$ ステップで到達する．□

定理 6.9 アルゴリズム *FCSSA_bi* は排他制御問題を解く故障封じ込め自己安定アルゴリズムである．

(略証) 補題 6.6, 6.7, 6.8 より，プロセス数が n のリングネットワークにおいて，

$$average(n) \leq \frac{1}{n}\left(8n + \sum_{i=1}^{n-2} 16 + 8n\right) < 32$$

よって 1-故障状況から解状況までの平均到達ステップ数は定数で押えられる．□

理論的に平均到達ステップ数を定数で押えたが，前節の *FCSSA_2uni* と同様に考えると，実際の平均到達ステップ数はこの値より小さい値になると考えている．

7. まとめ

本稿では，リングネットワーク上で排他制御問題を解く故障封じ込め自己安定アルゴリズムを提案した．

まず，排他制御問題に対する故障封じ込めを定義した．これは従来の故障封じ込めの定義が non-reactive な問題に対するものであり，そのままでは適用できないからである．提案した定義では，スケジューラが公平で，かつ，どのプロセスでも等確率で一時故障を生じるという仮定のもとで，1-故障状況から解状況までの平均的な到達時間が定数であるとした．

次に，本稿での故障封じ込めの定義のもとで，既存の自己安定アルゴリズム *DSSA* が故障封じ込めであるかどうかを確かめた．プロセス数 n に対する平均到達ステップ数，最悪到達ステップ数をシミュレーションプログラムの実行により解析した．この結果を見る限りにおいては，既存の排他制御問題に対する故障封じ込め *DSSA* では 1-故障状況から解状況までの平均到達ステップが定数であるとは考えられない．

これに対し，2 単方向通信リングでの決定性故障封じ込め自己安定アルゴリズム *FCSSA_2uni* と双方向通信リングでの決定性故障封じ込め自己安定アルゴリズム *FCSSA_bi* を提案し，それぞれが自己安定アルゴリズムであり，かつ，1-故障状況から解状況までの平均到達ステップ数が定数で押えられることを示した．*FCSSA_bi* のアイディアは *FCSSA_2uni* と同じであるが，補助変数を介することにより *FCSSA_2uni* と比べ定数時間のオーバーヘッドがかかる．

[7] 故障前は $\neg t(i) \wedge \neg t(i+1)$ または $t(i) \wedge \neg t(i+1)$ のどちらかが成立している．

参考文献

[1] Shlomi Dolev and Ted Herman. "Superstabilizing Protocol for Dynamic Distributed Systems". *In Proc. 2nd Workshop on Self-stabilizing Systems*, pp. 3.1–3.15, 1995.

[2] Eiichiro Ueda, Yoshiaki Katayama, Toshimitsu Masuzawa, and Hideo Fujiwara. "A Latency Optimal Superstabilizing Mutual Exclusion Protocol". *In Proc. 3nd Workshop on Self-stabilizing Systems*, 1997.

[3] S.Ghosh, A.Gupta, T.Herman, and S.V.Pemmaraju. "Fault Containing Self-Stabilizing Algorithm". *In Proc. 15th Symposium on Principles of Distributed Computing*, pp. 45–54, 1996.

[4] S.Ghosh and A.Gupta. "An execrcise in fault-containment: Self-stabilizing Leader Election". *Information Processing Letters,59(5)*, pp. 281–288, 1996.

[5] S.Ghosh, A.Gupta, and S.V.Pemmaraju. "A fault-containing self-stabilizing spanning tree algorithms". *Journal of Computing and Information,2(1)*, 1996.

[6] E.W. Dijkstra. "Self-stabilizing Systems in spite of distributed control". *Communications of the ACM,17(11)*, pp. 643–644, 1974.

[7] 千星 裕. "リングネットワーク上で排他制御問題を解く故障封じ込め自己安定アルゴリズム". 修士学位論文, 大阪大学, Feb. 1998.