2005－AL－103（12）
2005／11／11

グリッドコーラムによる効率的な分散相互排除アルゴリズム

守屋 宣
近畿大学理工学部情報学科

概要: 共有資源への競合を解消する相互排除問題は、分散システムにおける基本問題の一つである。システム内のプロセスに対する負荷を均等に実行する相互排除アルゴリズムとして、コーラムと呼ばれる数学的概念に基づくアルゴリズムに関する研究がされている。本稿では、コーラムに基づくアルゴリズムで、システムに停止プロセスを含むような場合のアルゴリズムの効率に関して考察をする。コーラムタイプとしてグリッドコーラムを採用し、停止プロセスを含むようなシステムでは、通常用いられるコーラムを採用するよりも効率がよくなる相互排除アルゴリズムの提案をする。

Efficient Algorithm by Grid Based Quorum for Distributed Mutual Exclusion

Sen Moriya
Department of Informatics, School of Science and Engineering
Kinki University

Abstract: Designing an algorithm for solving the mutual exclusion problem is an fundamental issue in distributed systems. Some researchers have investigated *quorum* based mutual exclusion algorithms, which can achieve load balance among processes in the distributed system. In this paper, we consider efficiency of quorum based mutual exclusion algorithms in case some processes have failed. We propose an algorithm by *grid* based quorum, and show that the proposed algorithm is more efficient than an algorithm based on a general quorum in the face of some failed processes.

1　はじめに

複数の自律的に動作するプロセスが相互にメッセージ交換を行うことにより通信を行う分散システムを考える。分散システムの基本問題の一つに、多数のプロセスが 1 つの共有資源 (共有ファイル、プリンタ、メモリなど) を使用する際の競合にどう対処するかという問題がある。競合プロセスの中から、1 つのプロセスのみに対して共有資源への特別な権利 (以下、特権とよぶ) をあたえることで競合を解消する問題を**相互排除問題** (*mutual exclusion problem*) という。

この問題を解決する簡単な手法として、リーダ選挙アルゴリズム [5] を用いることが考えられるが、この方法では常に同じプロセスが特権を取得して、特権を希望している他のプロセスが永久に特権を取得できないという飢餓状態に陥るおそれがある。また、ある 1 つのプロセスが調停者となり、そのプロセスが資源の使用を希望するプロセスに特権を与えるような方法 [3] では、調停者になるプロセスに負荷が集中することになり、分散システムにあまり適切ではない。そこで、これらの欠点を解消する分散相互排除アルゴリズムとして、**コーラム** (*quorum*) と呼ばれる数学的概念に基づくアルゴリズムに関する研究がされている [1, 2, 7]。

コーラムの概念に基づく相互排除アルゴリズムのおおまかな流れは次の通りである。コーラムは分散システムに属する全プロセスの部分集合であり、1 個以上のコーラムの集合によりコータリとよぶ集合族を構成する。任意の 2 つのコーラムは、必ず共通プロセスを持つ。特権を必要とするプロセスは、ある 1 つのコーラムに属するすべてのプロセスに対して資源を使用する要望を送信し、そのすべてのプロセスから許可を得た時点で特権を獲得できる。このような、コーラムに基づくアルゴリズムは、Maekawa によって提案された [7] [1] 。文献 [7] では、代表的なコー

[1] 文献 [7] で提案された Maekawa のアルゴリズムはデッドロックの可能性があり、Sanders によって改良版が提案されている [9]。本稿では、Sanders によって提案された改良版の方を、Maekawa のアルゴリズムとよぶ。

ラムの構成法として、有限射影平面 [10] に基づく方法 (FPP コーラム) とプロセスを論理的な 2 次元格子上に配置する方法 (グリッドコーラム) が紹介されている。コーラムに基づく相互排除アルゴリズムは、アルゴリズムの効率はコーラムのサイズに依存する。FPP コーラムとグリッドコーラムは、ともにコーラムサイズは $O(\sqrt{n})$ である。ただし、係数的には FPP コーラムの方がコーラムサイズが小さい一方、コーラムの構成法はグリッドコーラムの方が単純である。そこで、グリッドコーラムの構成法を基に、コーラムサイズをさらに小さくするような構成法 [4, 6] の研究もされている。

コーラムに基づく手法は、コーラム内のプロセスが故障などにより停止した場合でも、そのコーラムでアルゴリズムを継続せずに他のコーラムを選択しなおしてアルゴリズムを継続することが可能である。しかし、コーラムを繰り返し選び直さなければならないなら、アルゴリズムは非効率的になる。文献 [8] などでは、コーラムによる相互排除アルゴリズムで全プロセスが停止していないコーラムを見つけ出すまでの効率を探査複雑度 (*probe complexity*) と定義し、様々なコーラムにおける探査複雑度に関する考察を行っている。

ここで、一般のコーラムによる相互排除アルゴリズムでは、コーラムを選択する戦略的な規則が考えられないなら、コーラムはランダムに選択しなければならず、コーラム選択の繰り返しよる非効率化を回避するのは困難である。一方、グリッドコーラムでは、任意の 2 つのコーラムが 2 プロセス以上の交点を持ち、どのプロセスがそれらの交点プロセスであるかが容易にわかる。また、一般のコーラムでは 1 プロセスでも停止していたらコーラムとして機能しないのに対し、グリッドコーラムではいくつかのプロセスが停止していてもコーラムとしての機能を維持することができる場合がある。よって、グリッドコーラムでは、全プロセスが停止していないコーラムを見つけ出すまでの探査複雑度ではなく、いくつかのプロセスが停止しているとしてもコーラムとしての機能を維持しているものを見つけ出す効率が重要となる。本稿では、グリッドコーラムのこれらの性質を利用して、特権を要求するプロセスが効率的にコーラムの選択を行って特権を獲得できるようなアルゴリズムの考察を行う。

2　分散システムのモデル

分散システムをプロセス集合 $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ と、任意のプロセス間でメッセージ交換により通信を行うための通信リンクからなるものとする。ただし、$\mathcal{P}$ に属するすべてのプロセスが活動し続けてるとは限らず、故障などにより一部のプロセスは停止しているかもしれない。

本稿では、停止しているプロセスを**停止プロセス**と呼び、停止プロセス以外のプロセスを**活動プロセス**と呼んで区別する。活動プロセスは、他のプロセスが活動プロセスであるか停止プロセスなのかは知らないものとする。活動プロセスは他の活動プロセスと相互にメッセージ交換により通信できる。その通信は FIFO 信頼通信、すなわち、あるプロセス P_i から他のプロセス P_j へ送信されたメッセージは、有限時間以内に送信順に P_j が受信するものとする。また、停止プロセスへ送信されたメッセージは、エラーメッセージとして送信プロセスが受信する。

多くの競合するプロセスの中から 1 つのプロセスを選出し、そのプロセスに対して特権を与える競合解消問題を**相互排除問題**とよぶ。本論文では、相互排除問題を以下の 2 条件により定義する。

- 任意の時刻において、特権を保持するプロセスの数は高々1 つである。
- 特権を持つプロセスは必ず有限時間以内にその特権を解放すると仮定すると、特権を要求しているプロセスが永久に特権を得られないことはない。

相互排除問題を解くアルゴリズムの効率は、プロセスが特権を要求するアルゴリズムを開始してから、そのプロセスが特権を獲得するまでの時間により評価する。特に、他のプロセスが特権を保持も要求もしていないときに、プロセスが特権を要求するアルゴリズムを開始してから特権を獲得するまでの時間を、**応答時間**と呼ぶ。

3　一般のコーラムアルゴリズム

3.1　コーラム

効率的に相互排除問題を解く手法として、コーラムと呼ばれる数学的理論に基づいた解法がよく知られている。プロセス集合 $\mathcal{P}$ に対し、$\mathcal{P}$ の空でない部分集合 Q_i を M 個構成し、集合族 $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_M\}$ が次の条件 (i)(ii) を満たすとき、各 Q_i を**コーラム**とよび、$\mathcal{Q}$ を**コータリ**よぶ。

(i) 任意の $i, j (1 \leq i, j \leq M)$ に対して、$Q_i \cap Q_j \neq \emptyset$

(ii) 任意の $i, j(1 \leq i, j \leq M)$ に対して、$Q_i \not\subseteq Q_j$

3.2 一般のコーラムのための相互排除アルゴリズム

コーラムに基づく相互排除アルゴリズムとして、Maekawa のアルゴリズムが知られている [9]。Maekawa の相互排除アルゴリズムを基に、停止プロセスを含むシステムに対応できるようにした、一般のコーラムのための相互排除アルゴリズムを以下に述べる。

Step1: 特権を必要とするプロセス P_i は、コータリに属するあるコーラム Q を選択し、Q に属するすべてのプロセスに特権を要求するメッセージ `req` を送信する。ここで、

- 初めて Step1 を実行するときは、あらかじめ決めておいた適当なコーラムを Q として選択する。
- 後述する Step3 から戻って Step1 を実行するときは、決められた**選択規則 R** に従って未選択のコーラムの 1 つを Q として選択する。先述した `req` は未送信のプロセスのみに送信する。

また、`req` には、タイムスタンプとして `req` 送信時の Lamport 論理時計 [3] の値を添付しておく。

Step2: `req` を受信したプロセス P_j は、その時点でどのプロセスにも特権を与えていないならば、P_i へ特権を与えるメッセージ `locked` を送信する。P_j があるプロセスに特権を与えているならば、以下のようにする: P_j が特権を与えているプロセスを P_k とし、P_i と P_k から受信した `req` に添付されている論理時計の値を比較する。論理時計の値が小さい方に優先度があるとする。

- P_i の方が優先度が高いならば、P_j は P_k に特権を一度放棄するよう要求する。P_k が特権を放棄する、あるいは資源の使用を終えて特権を返却したら、P_j は P_i へ特権を与えるメッセージ `locked` を送信する。P_k が特権を放棄した場合は、P_k の要求 `req` を P_j の保持する待ち行列に挿入する。
- P_k の方が優先度が高いならば、P_i の要求 `req` を P_j の保持する待ち行列に挿入する。

Step3: P_i が `locked` を受け取ったプロセスの集合が決められた**特権獲得条件 C** を満たせば、P_i は特権を獲得できたことになる。P_i が `locked` を受信したプロセスで、特権を獲得した時点のコーラム Q に含まれていないプロセス P_j が存在するならば、P_i はこの時点で P_j にメッセージ `release` を送信する。

また、Q に属する停止プロセスにより条件 **C** が満たされないことがわかった場合は、Step1 からアルゴリズムをやり直す。

Step4: P_i が資源の使用を終えたならば、Q に属するすべてのプロセスに特権を返却するメッセージ `release` を送信する。`release` を受けとったプロセス P_j は、待ち行列の中に要求があるならば、その中で最も優先度の高いものを取り出して、そのプロセスに `locked` を送信する。

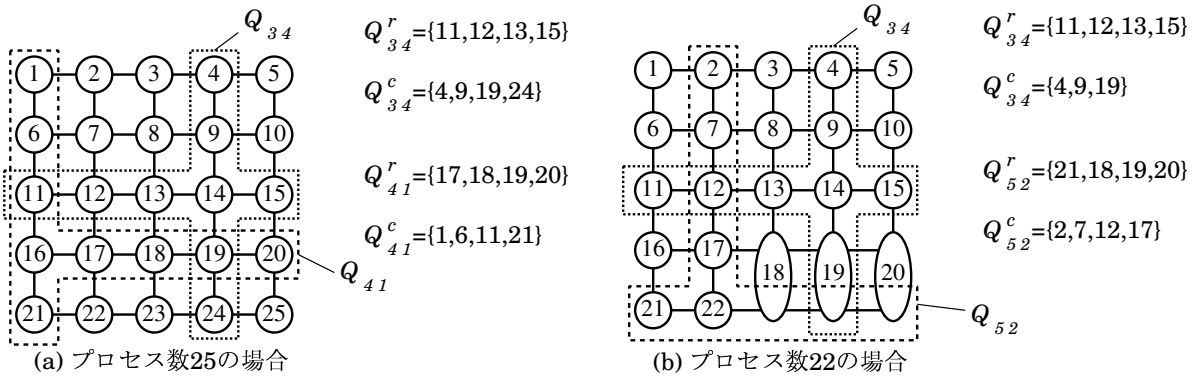
ここで、任意の方法により構成した一般のコータリでは、Step1 のコーラム Q の選択規則 **R** と Step3 の特権獲得条件 **C** は以下のようになる。

選択規則 **R:** 任意の未選択のコーラムを選択する。
特権獲得条件 **C:** コーラム Q に属するすべてのプロセスから `locked` を受信する。

以下では、選択規則、特権獲得条件をこのように設定したアルゴリズムを、**アルゴリズム GENERAL** と呼ぶ。一方、本稿で提案するグリッドコーラムによる相互排除アルゴリズムでは、その構成法を利用し、任意にコーラムを選択するのでなく 4.3 節で述べる選択規則 **R** によりコーラムの選択が可能であり、特権獲得条件 **C** も一部のプロセスから `locked` を受信すればよくなる。

4 グリッドコーラムアルゴリズム

本節で、グリッドコーラムによる効率的な相互排除アルゴリズムについて述べる。



(a) プロセス数25の場合　(b) プロセス数22の場合

図 1: グリッドコーラムの例

表 1: グリッドコーラムの型とプロセス数に対応する行数、列数

コーラムタイプ	プロセス数 (N) と行数 (N_r)、列数 (N_c) の関係
Grid	$(x-1)^2 < N \leq x^2$ を満たす整数 x に対して $N_r = x, N_c = \lceil N/x \rceil$
Grid2	$2(x-1)^2 < N \leq 2x^2$ を満たす整数 x に対して $N_r = x, N_c = \lceil N/x \rceil$
Grid4	$4(x-1)^2 < N \leq 4x^2$ を満たす整数 x に対して $N_r = x, N_c = \lceil N/x \rceil$
Grid8	$8(x-1)^2 < N \leq 8x^2$ を満たす整数 x に対して $N_r = x, N_c = \lceil N/x \rceil$

4.1 グリッドコーラム

まず、本稿で扱うグリッドコーラムの構成について述べる。集合 $\mathcal{P}$ に属するプロセスが論理的な N_r 行 N_c 列の 2 次元格子を構成しているとする。以降では、P_i を $P_{i_1 i_2}(i_1 = i/N_c + 1, i_2 = i \ (\mathbf{mod}\ N_c)\ )$ とも表すことにし、$\mathcal{P} = \{P_{i_1 i_2} | 1 \leq i_1 \leq N_r, 1 \leq i_2 \leq N_c\}$ とする。任意の $s_1, s_2 (1 \leq s_1 \leq N_r, 1 \leq s_2 \leq N_c)$ に対して、コーラム $Q_{s_1 s_2}$ を 2 次元格子の横方向の 1 行と縦方向の 1 列のプロセスから成る集合 $Q_{s_1 s_2} = \{P_{i_1 i_2} | i_1 = s_1 \vee i_2 = s_2\}$ と定める。そして、集合族 $\mathcal{Q} = \{Q_{s_1 s_2} | 1 \leq s_1 \leq N_r, 1 \leq s_2 \leq N_c\}$ を考察する。このとき、明らかに集合 $\mathcal{Q}$ はプロセス数 $N = N_r \times N_c$ のコータリとなっている (図 1(a))。本稿で提案する手法を説明する上での便宜上、グリッドコーラム $Q_{s_1 s_2}$ に対し、2 次元格子の横方向の行のみに含まれるプロセスの集合を $Q^r_{s_1 s_2} (= \{P_{i_1 i_2} | i_1 = s_1 \wedge i_2 \neq s_2\})$ とし、縦方向の列のみに含まれるプロセスの集合を $Q^c_{s_1 s_2} (= \{P_{i_1 i_2} | i_1 \neq s_1 \wedge i_2 = s_2\})$ と定義する。行と列の交点となるプロセス $P_{s_1 s_2}$ は $Q^r_{s_1 s_2}$ と $Q^c_{s_1 s_2}$ のどちらにも含まない。

なお、プロセス数が整数の積でない場合は、N 個以上の格子点を持つ 2 次元格子を構成し、最後の行に空きを作って、1 行上のプロセスで最後の行の空きを補う (図 1(b))。

また、従来のグリッドコーラムを扱う研究では、コーラムのサイズを最小にするために $N_r = N_c$ として 2 次元格子を正方形とすることを基本としているが、本稿では行数と列数が異なるグリッドコーラムも考察する。本稿で扱うグリッドコーラムの型を表 1 に記す。

以下、プロセス数とコーラムタイプ (2 次元格子の行数 N_r、列数 N_c) はアルゴリズム実行開始前に定まっているものとする。グリッドコーラムの構成は、FPP コーラムなどのコーラムと比べて明らかに単純である。そのため、各コーラムが含むプロセス集合を容易に計算することができる。この特長から、効率的にアルゴリズムを実行できるよう、コーラムを選択することが可能となる。

4.2 グリッドコーラムの性質

提案するグリッドコーラムによる相互排除アルゴリズムを述べる前に、提案アルゴリズムのための補題を述べる。アルゴリズム GENERAL と異なり、提案アルゴリズムでの特権獲得条件 **C** は一部のプロセスから `locked` を受信すればよい。すなわち、停止プロセスから `locked` を受信することは、必ずしも要求されない。そこで、以下の補題で

は、コーラムタイプをグリッドコーラムのタイプとするシステムにおいて、グリッドコーラムから停止プロセスを取り除く場合の議論をする。

補題 1 任意のコーラム $Q_{s_1s_2}$ を考える。システムに停止プロセスが存在しないならば、$Q_{s_1s_2}$ は他のどのコーラム $Q_{t_1t_2}$ とも *2* 個以上の共通プロセスを持つ。さらに、停止プロセスに関して以下のいずれかが成り立ちさえすれば、$Q_{s_1s_2}$ は他のどのコーラム $Q_{t_1t_2}$ とも少なくとも *1* 個の共通プロセスを持っている。

- $Q^r_{s_1s_2}$ と $Q^c_{s_1s_2}$ のうち少なくとも一方は停止プロセスを含まず、$P_{s_1s_2}$ も停止プロセスではない。
- $Q^r_{s_1s_2}$ と $Q^c_{s_1s_2}$ のうち、一方は停止プロセスを含まず、他方は停止プロセスでないプロセスを含む。

(証明) N_r 行 N_c 列のグリッドコーラムから成るシステムについて考える。$s_1 \neq t_1, s_2 \neq t_2$ の場合、$Q_{s_1s_2}$ とコーラム $Q_{t_1t_2}$ の共通プロセスは $P_{s_1t_2} \in Q^r_{s_1s_2}$ と $P_{t_1s_2} \in Q^c_{s_1s_2}$ の 2 個である。よって、$Q^r_{s_1s_2}$ または $Q^c_{s_1s_2}$ の一方の集合に属するすべてのプロセスを $Q_{s_1s_2}$ から取り除いても、$Q_{s_1s_2}$ と $Q_{t_1t_2}$ の共通プロセスは存在する。

また、$s_1 \neq t_1, s_2 = t_2$ の場合、$Q_{s_1s_2}$ とコーラム $Q_{t_1t_2}$ の共通プロセスは、$P_{1s_2}, ..., P_{is_2}(i \neq s_1), ..., P_{N_cs_2} \in Q^c_{s_1s_2}$ の $N_c - 1$ 個と $P_{s_1s_2}$ である。よって、$Q^c_{s_1s_2}$ に属するすべてのプロセス、または、$Q^c_{s_1s_2}$ の任意の真部分集合に属するすべてのプロセスと $P_{s_1s_2}$ を Q_{s_1,s_2} から取り除いても、Q_{s_1,s_2} と Q_{t_1,s_2} の共通プロセスは存在する。

$s_1 = t_1, s_2 \neq t_2$ の場合も同様にして共通プロセスは成立する。 □

補題 1 が成り立つことより、以下を定義する。

定義 1 グリッドコーラム $Q_{s_1s_2}$ から停止プロセスの集合 $\mathcal{P}_F$ に属するプロセスを取り除いたとする。このとき、$\mathcal{P}_F$ が以下のいずれかを満たすとき、コーラム $Q_{s_1s_2}$ は**有効**であるという。

- $Q^r_{s_1s_2} \cap \mathcal{P}_F = \emptyset \quad \wedge \quad (Q^c_{s_1s_2} \cup \{P_{s_1s_2}\}) \not\subseteq \mathcal{P}_F$
- $Q^c_{s_1s_2} \cap \mathcal{P}_F = \emptyset \quad \wedge \quad (Q^r_{s_1s_2} \cup \{P_{s_1s_2}\}) \not\subseteq \mathcal{P}_F$ □

プロセスがあるコーラム $Q_{s_1s_2}$ を選択してアルゴリズムを実行しているとき、$Q_{s_1s_2}$ のいくつかのプロセスが停止プロセスで、$Q_{s_1s_2}$ が有効でないとわかったとする。このとき、停止プロセスの集合により、$Q_{s_1s_2}$ 以外のコーラムの中から、高確率で有効であるようなコーラムを選択できる場合がある。

補題 2 任意のグリッドコーラム $Q_{s_1s_2}$ と停止プロセスの集合 $\mathcal{P}_F$ を考える。ただし、$Q^c_{s_1s_2}$ と $\mathcal{P}_F$ にはただ 1 つの共通プロセスが存在し、$P_{s_1s_2}$ は $\mathcal{P}_F$ に含まれないとする。このときのただ 1 つの共通プロセスを $P_{t_1s_2}$ とする。いま、$Q_{s_1s_2} \cap \mathcal{P}_F$ に含まれるプロセスをコーラム $Q_{t_1s_2}$ から取り除いたとき、$Q_{t_1s_2}$ が有効であるための条件は、$Q^r_{t_1s_2}$ が停止プロセスでないプロセスを少なくとも *1* つ含むことである。

(証明) N_r 行 N_c 列のグリッドコーラムから成るシステムについて考える。いま、$Q_{s_1s_2}$ のプロセスのうち、$P_{ts_2} \in Q^c_{s_1s_2}(t \neq s_1)$ が停止プロセスとし、コーラム $Q_{s_1s_2}$ が有効でないとする。次に、Q_{ts_2} を考える。$Q^c_{ts_2} = \{P_{1s_2}, ..., P_{is_2}(i \neq t), ..., P_{N_cs_2}\}$ のプロセスはすべて活動プロセスである。よって、Q_{us_2} が有効であるには、$Q^r_{us_2}$ が少なくとも 1 個の活動プロセスを含めばよい。 □

補題 2 は、選択したコーラム $Q_{s_1s_2}$ が、縦方向の列 $Q^c_{s_1s_2}$ に停止プロセスを 1 個だけ含む場合についての補題だが、横方向の行 $Q^r_{s_1s_2}$ に停止プロセスを 1 個だけ含む場合についても同様の補題が成り立つ。

補題 3 任意のグリッドコーラム $Q_{s_1s_2}$ と停止プロセスの集合 $\mathcal{P}_F$ を考える。ただし、$Q^r_{s_1s_2}$ と $\mathcal{P}_F$ にはただ 1 つの共通プロセスが存在し、$P_{s_1s_2}$ は $\mathcal{P}_F$ に含まれないとする。このときのただ 1 つの共通プロセスを $P_{s_1t_2}$ とする。いま、$Q_{s_1s_2} \cap \mathcal{P}_F$ に含まれるプロセスをコーラム $Q_{s_1t_2}$ から取り除いたとき、$Q_{s_1t_2}$ が有効であるための条件は、$Q^c_{s_1t_2}$ が停止プロセスでないプロセスを少なくとも *1* つ含むことである。 □

以上より、選択したコーラム $Q_{s_1s_2}$ の縦方向の列、 または横方向の行のどちらかの停止プロセスが 1 個ならば、コーラム $Q_{s_1s_2}$ が有効でない場合でも、次に高確率で有効なコーラムを選択することができる。ここで、プロセスが停止プロセスである確率を π とすると、$Q^r_{s_1s_2}$ がただ 1 個の停止プロセス $P_{t_1s_2}$ を含む場合にコーラム $Q_{t_1s_2}$ が有効である確率は $1 - \pi^{N_c-1}$、$Q^c_{s_1s_2}$ がただ 1 個の停止プロセス $P_{s_1t_2}$ を含む場合にコーラム $Q_{s_1t_2}$ が有効である確率は $1 - \pi^{N_r-1}$ である。本稿で考察する *Grid2*, *Grid4*, *Grid8* では、$N_r < N_c$ である。よって、1 個の停止プロセス $P_{t_1s_2}$ を含む列と 1 個の停止プロセス $P_{s_1t_2}$ を含む行が存在する場合は列を優先し、次は $Q_{t_1s_2}$ を選択するものとする。

4.3 グリッドコーラムアルゴリズム

4.2 節で述べた補題を基に、効率的なグリッドコーラムアルゴリズムを提案する。提案アルゴリズムは、3.2 節で述べたアルゴリズムを基本とし、選択規則 **R** と特権獲得条件 **C** を以下のように変更する。以降、このアルゴリズムを**アルゴリズム GRID** と呼ぶ。

選択規則 **R:** 特権を必要とするプロセス P_i は、あるコーラム Q を選択してアルゴリズムを実行しているとき、エラーメッセージを受信するか否かにより Q に属する各プロセスが停止プロセスか活動プロセスかがわかる。そこで、Q の各プロセスが活動プロセスか停止プロセスかを記憶しておく。わかっている活動プロセス、停止プロセスの情報を基に、選択したコーラム Q だけでなく、他のコーラムに対しても有効かどうかを確認する。

選択していたコーラム Q が有効でなく Step1 に戻ったとき、以下の 1 から 3 の優先順位で、まだ有効でないと判定していないコーラムの中から、次に選択するコーラムを決定する。

1. 停止プロセスが 1 個で他のプロセスがすべて活動プロセスである列が存在するならば、そのような停止プロセスの 1 つ $P_{t_1t_2}$ に対して $Q_{t_1t_2}$ を選択する。
2. 停止プロセスが 1 個で他のプロセスがすべて活動プロセスである行が存在するならば、そのような停止プロセスの 1 つ $P_{t_1t_2}$ に対して $Q_{t_1t_2}$ を選択する。
3. それ以外の任意のコーラムを選択する。 □

特権獲得条件 **C:** プロセス P_i が特権を必要としているとし、P_i が `locked` を受信したプロセスの集合を $\mathcal{P}_l$、P_i が停止プロセスと知っているプロセスの集合を $\mathcal{P}_f$ とする。いま、P_i がコーラム $Q_{s_1s_2}$ を選択してアルゴリズムを実行しているとすると、$\mathcal{P}_l \cup \mathcal{P}_f \supseteq Q_{s_1s_2}$ が成り立ち、$\mathcal{P}_f$ が以下のいずれかの条件を満たすならば、P_i は特権を獲得できる。

- $Q^r_{s_1s_2} \cap \mathcal{P}_f = \emptyset \quad \wedge \quad (Q^c_{s_1s_2} \cup \{P_{s_1s_2}\}) \not\subseteq \mathcal{P}_f$
- $Q^c_{s_1s_2} \cap \mathcal{P}_f = \emptyset \quad \wedge \quad (Q^r_{s_1s_2} \cup \{P_{s_1s_2}\}) \not\subseteq \mathcal{P}_f$ □

5 実験

本節で、提案アルゴリズムの効果を評価するために行った実験の結果を示す。今回の実験では、1 台のホスト上に 150 個または 500 個のスレッドをプロセスとして動作させてシミュレーションを行った。

5.1 実験の概要

ホスト上で動作する全プロセスのうち、ランダムに選択した 1 プロセスのみが特権を要求し、他のプロセスはその特権の要求に対する許可のみを行う。各コーラムのリストはホスト上にあり、特権を要求するプロセスはそのリストにアクセスしてコーラムの選択、`req` メッセージの送信を行う。このとき、アルゴリズム GENERAL と提案手法であるアルゴリズム GRID を、故障率を 0 から 0.2 まで 0.025 毎に変化させて 100 回ずつ実行し、特権を要求するプロセスが特権を獲得するまでの応答時間の期待値を比較した。アルゴリズム GENERAL は係数的にコーラムサイズの小さい FPP コーラムにより実行した (以下、このアルゴリズムを FPP と記す)。また、アルゴリズム GRID はコーラムタイプを *Grid*, *Grid2*, *Grid4*,*Grid8* として実行した (以下、これらのアルゴリズムをそれぞれ Grid, Grid2, Grid4, Grid8 と記す)。プロセス数 150, 500 それぞれで FPP, Grid, Grid2, Grid4, Grid8 を実行したときのコーラムサイズについて表 5.1 に記す。

5.2 結果

プロセス数 150,500 それぞれの場合でアルゴリズムを実行したときの応答時間を図 2 に示し、停止プロセスがない故障率 0 の場合と比較しての応答時間増加率を図 3 に示す。なお、プロセス数 500 の場合の FPP と Grid はすべてのコーラムを選択しても特権を獲得できないケースが 10%以上であったため除外した。故障率が 0 の場合は、どのコーラムでアルゴリズムを選択しても故障プロセスが含まれないため再選択の必要はなく、応答時間はコーラム

表 2: プロセス数 150, 500 の場合のコーラムサイズ

	プロセス数 150 の場合の コーラムサイズ (行数, 列数)	プロセス数 500 の場合の コーラムサイズ (行数, 列数)
FPP	14	24
Grid	24 (13 行 12 列)	44 (23 行 22 列)
Grid2	25 (9 行 17 列)	47 (32 行 16 列)
Grid4	28 (7 行 22 列)	53 (12 行 42 列)
Grid8	34 (5 行 30 列)	70 (8 行 63 列)

サイズに比例することになる。一方、故障率が上昇するに従って、選択したコーラムが停止プロセスを含む確率が高くなるためにコーラムを再選択する必要が生じ、応答時間が長くなる。

コーラム内の全プロセスが活動プロセスであることが要求される FPP は、コーラムサイズは小さいながら故障率の増加の影響を大きく受けている。一方、停止プロセスを含んでいるコーラムでも有効な場合がある各グリッドコーラムによるアルゴリズムでは、故障の影響は比較的小さくなっている。その結果、プロセス数 100 の場合では故障率 0.1 以上、プロセス数 500 の場合は故障率 0.05 以上で、どのグリッドのアルゴリズムも FPP より応答時間が短くなっている。ただし、グリッドのタイプによっても、故障率の影響にかなり差が出ている。プロセス数 150 の場合ではグリッドのどのタイプであっても応答時間の差はあまり見られなかったのに対し、プロセス数 500 の場合はグリッドのタイプによる差がかなり大きくなっている。これは、コーラムの 2 次元格子の行数が少なければ列に停止プロセスを含む確率が小さくなるため、選択したコーラムが有効になりやすいことによる。

6 まとめ

本稿では、停止プロセスを含む分散システムにおける、コーラムによる相互排除アルゴリズムについて考察した。率直なコーラムアルゴリズム GENERAL では、コーラムに属するすべてのプロセスから特権の許可を得なければならないため、多くの停止プロセスを含むようなシステムでは効率よく動作しない。一方、本稿で提案したアルゴリズム GRID では、コーラムとしてグリッドコーラムを使用することで、多くの停止プロセスを含むようなシステムでも効率よく動作する。特に、2 次元格子の行数が少ないコーラムでは、停止プロセスの発生の影響をかなり小さく抑えられる。

本稿で提案したアルゴリズム GRID では、コーラムの再選択をするときに有効なコーラムを選択することに主眼をおいている。しかし、応答時間を小さくするためには、コーラム再選択後に送信する `req` メッセージを減らすようにコーラムを選ぶことも効果があると思われる。また、グリッドコーラムはコーラム集合であるコータリ全体を再構成することが比較的容易であるため、再構成により有効なコーラムを構成する手法も考えられる。そこで、本稿で提案した手法とこれらの手法との応答時間の比較などが今後の課題として挙げられる。

謝辞

本研究を進めるにあたり御討論いただきました大阪大学大学院情報科学研究科 増澤利光教授、角川裕次助教授に感謝いたします。

参考文献

[1] T. Ibaraki and T. Kameda. A theory of coteries: Mutual exclusion in distributed systems. *IEEE Trans. Prallel and Distributed Systems*, Vol. 4, No. 7, pp. 779–794, 1993.

[2] H. Kakugawa, S. Fujita, M. Yamashita, and T. Ae. A distributed k-mutual exclusion algorithm using k-coterie. *Information Processing Letters*, Vol. 49, No. 2, pp. 213–218, 1994.

[3] L. Lamport. Time, clocks and ordering of events in a distributed system. *Communication of ACM*, Vol. 21, No. 7, pp. 558–564, 1978.

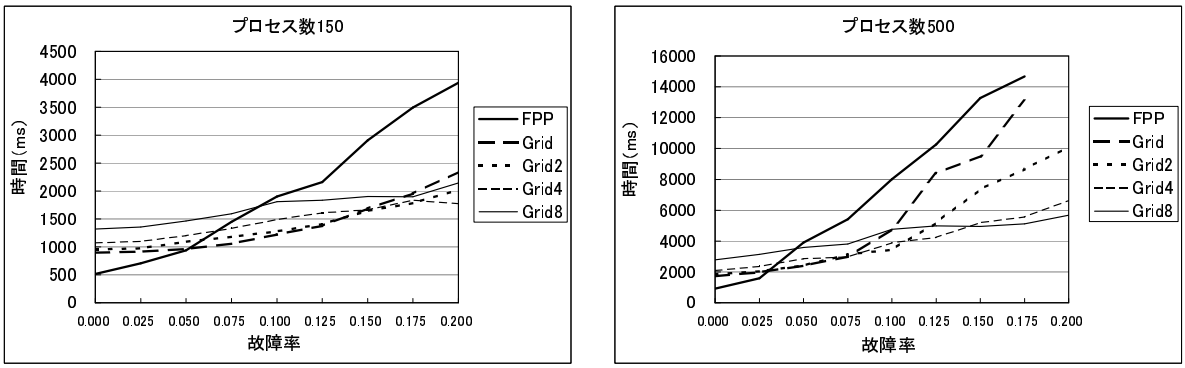


図 2: 応答時間

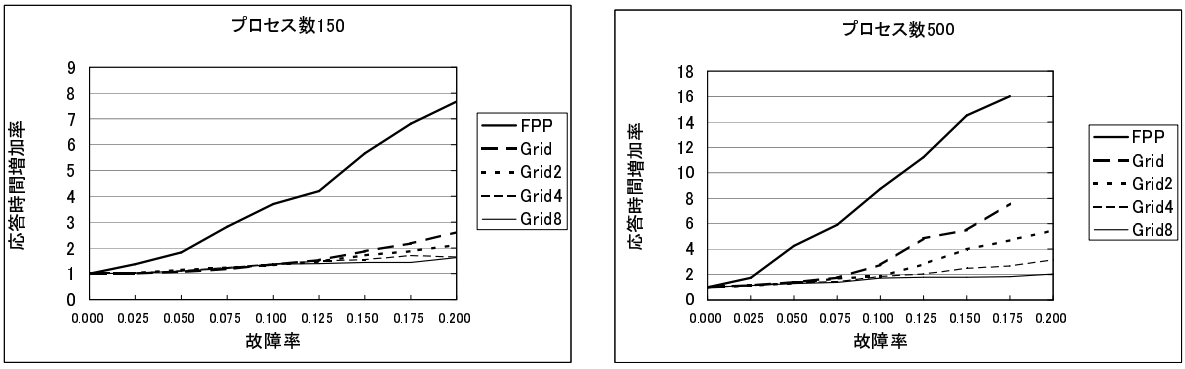


図 3: 応答時間増加率

[4] S.D. Lang and L.J. Mao. A torus quorum protocol for distributed mutual exclusion. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems,*, pp. 635–638, 1998.

[5] G. Le Lann. Distributed system – towards a formal approach. *Information Processing '77*, pp. 155–160, 1977.

[6] W.S. Luk and T.T. Wong. Two new quorum based algorithms for distributed mutual exclusion. In *Proceedings of the 17th International Conference on Distributed Computer Systems*, pp. 100–106, 1997.

[7] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transaction on Computer Systems*, Vol. 3, No. 2, pp. 145–159, 1985.

[8] D. Peleg and A. Wool. How to be an efficient snoop, or the probe complexity of quorum systems. *SIAM Journal on Discrete Mathematics*, Vol. 15, No. 3, pp. 416–433, 2002.

[9] B. A. Sanders. The information structure of distributed mutual exclusion algorithms. *ACM Transaction on Distributed Systems*, Vol. 5, No. 3, 1987.

[10] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.

[11] 亀田恒彦, 山下雅史. 分散アルゴリズム. 近代科学社, 1994.