アルゴリズム 23－10
(1991. 9. 20)

時刻印を用いないデッドロックフリーな分散相互排除プロトコル
A Deadlock-Free Distributed Mutual Exclusion Protocol Without Using Time Stamp

角川裕次 藤田聡 山下雅史 阿江忠
Hirotsugu KAKUGAWA Satoshi FUJITA Masafumi YAMASHITA Tadashi AE

広島大学工学部
Faculty of Engineering,Hiroshima University

Abstract The distributed mutual exclusion problem is the problem of guaranteeing that a shared resource be exlusively accessed in distributed systems. Extensive protocols for it have been designed. In this paper, we propose a protocol based on the coterie. Our protocol is (1) dead-lock free and (2) starvation free, and uses neither time stamp nor sequence number. The number of messages that the protocol requires per mutual exclusion entrance is exactly $3|q|$ where $|q|$ is the size of the maximal quorum of coterie.

1 はじめに

分散相互排除問題を解決する方法には大別してトークンに基づいた方法 [6],[11] とコーラムコンセンサス (Quorum Consensus) に基づいた方法 [4],[8] がある。前者はシステム内に一つだけトークンと呼ばれるものが存在していて、臨界領域に進むにはそのトークンを入手しなければいけない。臨界領域を出るとそのトークンを、次にトークンを要求しているノードに送り、そしてそのノードは臨界領域に進むことができる。後者では、例えば、臨界領域に入ろうとするノードは他のノードにメッセージを送ってある数以上のノードより許可を得ることができれば臨界領域に進むことができるというものである。この考えを一般化したものがコーラムコンセンサスに基づいた方法である。どの2つのノードのグループ(コーラムと呼ばれる)に対しても、少なくとも一つのノードは両方のコーラムに属しているように、コーラムの集合を作る。コーラムの集合はコータリーと呼ばれる。そして臨界領域に進みたいノードはどれか一つのコーラムに対して、それに属するノード全てより許可を得ればよい。この方式は確かに相互排除を実現している。何故なら各グループは必ず交わりがあり、各ノードは一つのノードに対してのみ許可を与えるからである。もし2つ以上のノードが臨界領域に進むことがあればこれらの事実に反する。

[8] ではノード数を N としたとき、メッセージ数が $2(N-1)$ で排他制御を解決するプロトコルが示されており、[4] では有限射影平面を利用してコーラムの大きさが $\sqrt{N}$ であるコータリーの構成方法と $C\sqrt{N}$, $C=3\sim5$ のメッセージ数で相互排除を解決するプロトコルが示されている。[1] では論理的な二進木を構成して最小 $\log N$ の大きさのコーラムをもつコータリーの構成法が示されている。[10] では、[4] 型のプロトコルのクラスの内、3フェーズ (要求/許可/解放の3フェーズ) のみでデッドロックが生じ

ないプロトコルのクラスについて議論されている。しかしこのプロトコルでは、3フェーズのみで、すなわち [4] のプロトコルで必要とされるデッドロック回避のためのメッセージを用いないようにするために、用いるコータリーに制限をつけてコーラムの大きさの平均がほぼ $N/2$ のものを使っている。

本稿では、3フェーズで、かつ用いるコータリーに何ら制限をつけず、そして時刻印・シーケンス番号を用いないでデッドロック及び飢餓状態の生じないプロトコルが構成可能であることを示す。例えば [4] で用いられたコータリーを用いれば、メッセージ数 $3\sqrt{N}$ で相互排除が可能となる。多くの分散相互排除プロトコルは、時刻印 ([3],[4], [8],[10]) もしくはシーケンス番号 ([9], [11]) を導入することで各相互排除要求に順序をつけてデッドロックを回避している。しかし時刻印・シーケンス番号を記憶するカウンタ変数は無限回カウントすることはできず、いつかは溢れてしまう。(この溢れに対する対策が [11] で示されている。) 我々の示すプロトコルはカウンタ類は一切用いていないので、そのような問題は生じない。

2 では対象とする分散システムのモデルを説明する。**3** ではコータリーの定義を示す。**4** ではプロトコルを示す。**5** では **4** で示されたプロトコルが相互排除を保証し、デッドロック及び飢餓状態が生じないことを証明する。**6** ではプロトコルの評価を行なう。

2 仮定

ここでは対象としている分散システムのモデルを説明する。システムは計算機に相当するプロセスと通信路に当たるリンクより構成される (これらは一般に複数個存在する)。プロセスは無故障であり、リンクは誤りなしのメッセージ通信が可能である。ネットワークトポロジーは完全グラフである。

各プロセスは一意な名前 (プロセス識別子) を持っていて、識別子に対して全順序関係が定義されているものとする。一般性を失うことなくプロセス識別子は自然数と考える。プロセスの実行速度に関しては何の仮定も設けない。プロセス間の情報のやりとりはメッセージ通信で行なわれ、プロセス間の共有変数は存在しない。プロセスに提供されているメッセージ通信のための基本手続きとして、Send 命令と Receive 命令の二つがある。各プロセスには長さに制限のないメッセージ受信のための待ち行列 (メッセージバッファ) がある。Send 命令は、送り先プロセス識別子で指定されたプロセスに指定されたメッセージを送る。メッセージは指定されたプロセスのメッセージバッファの最後に追加される。メッセージ通信にかかる時間 (通信遅延) は有限時間である事しか保証されておらず、遅延時間はプロセスは知ることはできない。Receive 命令が呼び出されると、そのプロセスのメッセージバッファが調べられメッセージバッファの先頭のメッセージが取り出され、それが返される。このとき受けとったメッセージはどのプロセスから送られたのかをプロセスは知ることができるとする。もしメッセージバッファが空であれば、空であることを示す特別の値が返される。

3 コータリー (Coterie)

コータリーの定義を示す。コータリーの持つ性質については [2] を参照されたい。直観的にいえば、コータリーとはコーラムの集合であり、コーラムはプロセスの集合である。コータリーにおいて、2つのコーラムをどのように選んでも必ずそれらには交わりがある。あるプロセスが臨界領域に進むには、あるコーラムに属する全てのプロセスより許可を得れば良い。任意の2つのコーラムは互いに交わりがあるので、2個以上のプロセスは同時に臨界領域に進むことはできない。

定義 1 (コータリー) 集合 U の下のコータリー (Coterie) S とは、

1. $S \subseteq 2^U$, $S \neq \emptyset$
2. **Intersection Property**
   任意の $q_i, q_j \in S$ に対し、$q_i \cap q_j \neq \emptyset$
3. **Minimality**
   任意の $q, r \in S$ に対し $q \not\subseteq r$

プロセス集合 $q \in S$ はコーラム (Quorum) と呼ばれる。 □

4 プロトコル

プロセスの集合を $U = \{p_0, p_1, \cdots, p_{N-1}\}$ とし、この下でのコータリーを S とおく。議論を単純にするために、各プロセス p_i に対して一つのコーラム $R_i \in S$ を対応付ける。特にこの R_i をプロセス p_i の要求集合と呼ぶ。

4.1 プロトコルの概要

各プロセス p_i は次のように動作する。プロセス p_i で相互排除要求が生じると、p_i は R_i の各プロセスに

REQUESTメッセージを送る。このときR_iに属するプロセスのプロセス識別子が小さい順に、メッセージを送りそれより許可メッセージ PERMIT を得るのを待ち、そして次のプロセスに REQUEST メッセージを送る、という動作を繰り返す。そしてR_iのすべてのプロセスより PERMIT メッセージが得られたら臨界領域に進む。臨界領域から出る時は、R_iの各プロセスに RELEASE メッセージを送って臨界領域から抜けることを知らせる。

プロセスp_iがあるプロセスp_jから REQUEST メッセージを受けた時は、その要求をp_iの FIFO キューに入れる[1]。そしてこの時、p_iが他のプロセスに PERMIT を送っていない状況[2]であればp_iはその FIFO キューの先頭より要求を一つ取りだし、その要求を行なったプロセスに対して PERMIT メッセージを送る。p_iが RELEASE メッセージを受けると、FIFO キューを調べる。もしキューが空であれば、何もしない。キューが空でなければ、キューの先頭の要求を取りだして、その要求を行なったプロセスに PERMIT を送る。

4.2 プロトコルの詳細

以下にプロトコルの詳細を示す。排他制御を必要とするときは手続き EnterCS を呼び出す。他のプロセスよりメッセージを受けとった時はメッセージにしたがって RecieptREQUEST, RecieptPERMIT, RecieptRELEASE がアトミックに呼び出される。

プロトコル A：

(** **プロセス** p_i**の動作** **)

```
process MutexProcess;
const
  R_i : set of process;
    (* プロセス p_iの要求集合 *)
var
  HaveToken : bool; (* 初期値は true*)
  Q : queue; (* 初期値は null*)
  w : bool;

(** 臨界領域に入る時、この手続きを呼ぶ **)
procedure EnterCS;
var
  j : integer;
begin
  (* R_i ={r_{i,1}, r_{i,2}, ..., r_{i,m_i}} とおく。
        ただし r_{i,l} < r_{i,l+1}とする。 *)
  for j:=1 to m_i do
    begin
      w:=true;
      send(r_{i,j},REQUEST);
      while w do
        ;   (* r_{i,j}より PERMIT を待つ *)
    end

  <臨界領域>

  for j:=1 to m_i do
    send(r_{i,j},RELEASE)
end;

(** メッセージ REQUEST を p_jより受けた時 **)
procedure ReceiptREQUEST;
begin
  Enqueue(Q, p_j);
  if HaveToken then
    begin
      send(Dequeue(Q),PERMIT);
      HaveToken:=false
    end
end;

(** メッセージ PERMIT を受けた時 **)
procedure ReceiptPERMIT;
begin
  w:=false
end;

(** メッセージ RELEASE を受けた時 **)
procedure ReceiptRELEASE;
begin
  if null(Q) then
    HaveToken:=true
  else
    begin
      send(Dequeue(Q),PERMIT);
```

[1]要求に応える方法は FIFO である必要はなく、各プロセスにおいて局所的に飢餓状態が生じない方法で要求に応えることができれば十分である。ここでは議論を単純にするために、FIFO で要求に対するサービスを行なっている。

[2]"PERMIT を送っている状況" とは、あるプロセスに PERMIT メッセージを送ったが、その後まだそのプロセスより RELEASE メッセージを受けとっていないことを指す。プロトコル A の記述において、変数 HaveToken の値が **false** であるときに当たる。"PERMIT を送っていない状況" はその否定である。

```
        HaveToken:=false
    end
end;
```

5 正しさの証明

定理 1 (プロトコル A は相互排除を実現している)

(証明)　プロセスはあるコーラムに属する全てのプロセスより許可を得て臨界領域に進む。また、各プロセスは各時点において高々一つのプロセスにしか許可を送らない。従ってコータリーの Intersection Property により、臨界領域に進むことのできるプロセスの数は各時点において高々一つである。□

定理 2 (プロトコル A はデッドロックを生じない)

(証明)　デッドロックが生じたと仮定する。各プロセスの「許可」を一つの資源と考え、臨界領域に進むことは複数の資源を要求することと見做すことが出来る。デッドロック状態にあるプロセスを $q_0, q_1, \cdots, q_{L-1} \in U$ とし、一般性を失うことなくこれらのプロセスは臨界領域にはないとできる。これらのプロセスは許可の確保と待機の巡回を構成している。すなわち、$q_{i \bmod L}$ は $q_{i+1 \bmod L}$ が得ている許可を求めていてそれが解放されるのを待っている状態にある。より詳しく書けば、次のようになる。各 $i\,(0 \le i < L)$ に対し、q_i に許可を送っているプロセスの集合 a_i は、q_i の要求集合を $R_i = \{r_1, r_2, \cdots, r_m\}$ (但し $\forall j(1 \le j < m)[r_j < r_{j+1}]$) とおけば、ある $t \quad (1 \le t < m)$ が存在して $a_i = \{r_1, r_2, \cdots, r_t\}$ と表せる。このとき、q_i は r_{t+1} に要求を出しているが許可が得られずに待たされている状態にある。そして $q_{i+1 \bmod L}$ は r_{t+1} より許可を得ており、$r_{t+1} \in a_{i+1 \bmod L}$ が成立している。

一般性を失うことなく、$\min \bigcup_{0 \le j < L} a_j \in a_1$ とできる。q_{L-1} に注目すると、ある $r'_1, r'_2 \in R_{L-1}$ $(r'_1 > r'_2)$ が存在して $r'_1 \in a_{L-1}$ かつ $r'_2 \notin a_{L-1}$ かつ $r'_2 \in a_1$ である。すなわち、q_{L-1} は $r'_1 > r'_2$ であるにも関わらず r'_1 の方を先に要求しており、これは矛盾。□

定理 3 (プロトコル A は飢餓状態を生じない)

(証明)　2 つのプロセス $q_a, q_b \in U$ が存在し、プロセス q_a が飢餓状態にあり、かつプロセス q_b が無限回臨界領域の出入りを繰り返していると仮定する。このとき、プロセスの許可の確保と待機の連鎖が生じており、$q_a = q_1, q_2, \cdots, q_L = q_b, \quad 2 \le L \le M$ と表される[3]。ここで各プロセス q_i $(1 \le i < L)$ に対し、それに対応したあるプロセス r_i が存在して q_i は r_i に要求を出しているが許可が得られずに待たされている状態にあり、かつ q_{i+1} は r_i より許可を得ている状態にある。すると、q_{L-1} の要求はプロセス r_{L-1} の待ち行列に入っているにも関わらず q_L の方が無限回サービスを受けることになり矛盾。□

6 評価

6.1 メッセージ複雑度

本プロトコルが 1 回当たりの相互排除を行なうのに必要なメッセージ数は $3|q|$ である。用いるコータリーには何ら制限は必要ない。それに加えてタイムスタンプ等は一切用いていないのでメッセージ一つ当たりのビット数はかなり小さい。メッセージの送り元プロセス識別子をメッセージに入れないとすると (すなわちメッセージを受けとったリンクよりそのメッセージを送ったプロセスを知ることができるとすると) 用いるメッセージは REQUEST, PERMITE, RELEASE の三種類であることより、1 メッセージの大きさは 2 ビットで済む。従って、1 回の相互排除を行なうのに要するメッセージのビット数の総数は $6|q|$ ビットである。本プロトコルで用いるコータリーに、例えば有限射影平面を用いて構成したコータリー [4] を用いれば $6\sqrt{n}$ ビットとなる。

[10] では、任意の i, j に対して $i \in R_j$ or $j \in R_i$ と要求集合に制約を加えることで 3 フェーズでデッドロックフリーなプロトコルが示されている。しかし、その制約のために要求集合の大きさの平均値は $|R_i| \approx (n-1)/2$ である。また、そのプロトコルはタイムスタンプを用いているので必要となるメッセージのビット数は更に大きい。

6.2 臨界領域に進むのに要する時間

プロセスで相互排除要求が生じてから臨界領域に進むまでに要する時間について考察する。これを評価するに当たって、メッセージの伝わる時間を T とおき、局所計算に要する時間は 0 と仮定をして議論する。本プロトコルでは最良の場合で $2T|q|$ かかることがプロトコルの記述より導かれる。一方、例えば [10] では、最良の場合で $2T$ である。従って、$|q|$

[3] ただし、q_a はその要求集合のどのプロセスよりも許可を得ていない場合も含まれる。

が大きい時は不利である。

7 おわりに

本稿では時刻印を用いないデッドロックフリーな分散排他制御のプロトコルを示した。本プロトコルは、各プロセスに線形順序をつけ、その順序にしたがってコーラムのプロセスより順々に許可を求めることでデッドロックフリーを得ることができた。集中システムにおいても、資源の割り当てに関して、資源に線形順序をつけてその順で資源の要求を行なえばデッドロックが生じないことが知られている。集中システムと同様なアプローチで、分散システムでもデッドロックが回避されることは大変興味深い。トークンに基づいたプロトコルも含め、他のプロトコルでは時刻印またはシーケンス番号を用いており、そのために1回の排他制御要求当たりの必要なメッセージの総ビット数はかなり大きくなっている。本プロトコルは1回当たりの排他制御要求当たりの必要なメッセージの総ビット数の意味で最適なプロトコルではないかと期待されるが、まだその証明はない。それを証明すること、または最適なプロトコルを構成することは今後の課題である。

参考文献

[1] Divyakant Agrawal and Amr El Abbadi. An efficient solution to the distributed mutual exclusion problem. In *Principles of Distributed Computing*, pp. 193–200, August 1989.

[2] Toshihide Ibaraki and Tiko Kameda. Theory of coteries. CSS/LCCR TR90-09, Simon Fraser University, Burnaby, B.C. Canada, 1990.

[3] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, Vol. 21, No. 7, pp. 558–565, July 1978.

[4] Mamoru Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, Vol. 3, No. 2, pp. 145–159, March 1985.

[5] Mamoru Maekawa, Arthur E. Oldehoft, Rodney R. Oldehoft(前川守監訳). オペレーティングシステムの先進的概念. 丸善, 1989.

[6] Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, Vol. 7, No. 1, pp. 61–77, February 1989.

[7] Michel Raynal. *Algorithms for Mutual Exclusion.* North Oxford Academic, 1886. (Translated by D. Beeson).

[8] Glenn Ricart and Ashok K. Agrawala. An optimal algorithm for mutual exclusion in computer network. *Communications of the ACM*, Vol. 24, No. 1, pp. 9–17, January 1981.

[9] Glenn Ricart and Ashok K. Agrawala. Author's response to 'On mutual exclusion in computer networks' by Carvalho and Roucairol. *Communications of the ACM*, Vol. 26, No. 2, pp. 147–148, February 1983.

[10] Mukesh Singhal. A class of deadlock-free maekawa-type algorithms for mutual exclusion in distributed systems. *Distributed Computing*, pp. 131–138, April 1991.

[11] Ichiro Suzuki and Tadao Kasami. A distributed mutual exclusion algorithm. *ACM Transactions on Computer Systems*, Vol. 3, No. 4, pp. 344–349, November 1985.