解　説

フォールトトレラント分散システム向けアルゴリズム

3. 分散相互排除問題とコータリ†

山　下　雅　史††

1. 分散相互排除問題

互いに通信しながら自律的に動作する**プロセス** (*process*) の集合を，**分散システム** (*distributed system*) と呼ぶ．プロセスは，計算機上で実行されている逐次プログラムをモデル化したものであるが，より一般的に，自律的に動作する計算主体（たとえば，ロボット）と考えてもよい．分散システムはさまざまな族に分類できるが，本稿では，共有変数（共有メモリ）をもたないプロセスから構成される分散システムを検討の対象とする．このようなシステムでは，プロセス間の情報交換は，通信ネットワークを介するメッセージ通信によって行われる．メッセージ転送遅延は常に存在し有限であるが変化する．通信遅延が存在するので，分散システムに属するすべてのプロセスが，ある時点における分散システム全体の状態を共通に把握することはできない．これが，対象の全体像が常に把握できる集中型システムで実行される通常のアルゴリズムの設計との本質的な相違である．

あるプロセスの集合が矛盾のないシステムとして動作するには，プロセスが自らの振舞いをシステムの状況に応じて制約する必要がある．この制約のことを**同期** (*synchronization*) と呼ぶ．二つの異なる同期形態，**競合** (*contention*) と**協調** (*cooperation*) が従来から盛んに検討されてきている（たとえば，参考文献 15) を参照)．☆ 本稿の目的は，典型的な競合問題である相互排除問題を取りあげ，この問題を解決する耐故障分散アルゴリズムについて紹介することである．☆

多くの競合するプロセスの中から一つのプロセスを選出し，そのプロセスに対して特別な権利を与える競合解消問題を**相互排除問題** (*mutual exclusion problem*) という．相互排除問題が出現するのは以下のような場合である．

1. （分散システムに限らず）システム上でプロセスが共有する資源は占有的に使用されることが多い．プロセッサ，メモリ，ファイル，バス，プリンタ，MT 等々，さらに，オペレーティング・システムでよく使われる**臨界領域** (*critical section*) などもその例である．すなわち，ある一つの資源を多くのプロセスが同時に使用しようとしたときには，その中の一つが使用を許可され，他のプロセスはそのプロセスの使用が終わるのを待たねばならない．

2. 自律分散ロボットシステムでは，ロボットが作業空間を共有することが多いが，ある地点を2台以上のロボットが同時に占めることはできない．たとえば，細い道で向き合っている2台の自走ロボットは，どちらかが後退して道を譲る必要があるし，衝突コースに入っている2台の自動操縦航空機は，どちらかがコースを変更して衝突を回避する必要がある．

プロセス間の競合を解消する手段として，たとえば，プロセス識別子の順序関係を優先度として用いることが考えられる．しかし，この場合には，特権を要求しているのに永遠に許可がおりない，いわゆる**飢餓状態** (*starvation*) に陥る優先度の低いプロセスが生じる可能性があり，好ましくない．われわれの目的は，**デッドロック** (*dead-*

† The Distributed Mutual Exclusion Problem and Coteries by Masafumi YAMASHITA (Department of Electrical Engineering, Hiroshima University). 本研究は，一部電気通信普及財団の援助を受けて行われた．

†† 広島大学工学部第2類

☆ 古典的な協調問題に**長さ限定バッファ問題** (*bounded buffer problem*) がある．これは，送信プロセスが保持する十分長い値の列を，固定長のバッファを経由して，受信プロセスに送る問題である．バッファが空のときには，受信プロセスが待機し，バッファがいっぱいのときには，送信プロセスが待機する．

☆ 分散アルゴリズムについては，たとえば，解説 9)，15)，28) を参照せよ．

lock) や飢餓状態に陥るプロセスがないように相互排除問題を解くことである.*

メモリ共有型のシステム上での相互排除問題は，鍵付きの部屋を共同使用する問題にたとえられる．鍵は通常所定の鍵置きに置いてあり，だれもがそこに鍵が置かれてあるかどうか確認できる．部屋の使用希望者は鍵が鍵置きにあることを確認し (TEST)，鍵を取って (SET) 部屋に入る．用事が済めば鍵を元に戻す．このとき，TEST と SET をそれぞれ基本命令だと仮定したり，TEST と SET を不可分の (*indivisible*) 動作で行う TEST andSET と呼ばれる基本命令を仮定したりした，多くの研究がある．しかし，これらの話題は本稿の範囲を外れるので，興味がある読者は，たとえば，Raynal[23] を参照してほしい．われわれが対象とする分散システムでは，“鍵” で表現されているような共有変数は許されておらず，メッセージ通信が唯一の通信手段である．

本稿では，分散システムを，通信リンク内でメッセージの順序が入れ代わることがなく，したがって，メッセージは送信された順序で受信される **FIFO システム**であると仮定する.**

2. 論理時計と Lamport のアルゴリズム *LAM*

分散システムには，システム全体を管理するための特別なプロセスは存在しない．そこで，相互排除においても，すべてのプロセスが平等に特権の認可に責任を負うことが求められる．このようなアルゴリズムは，Lamport によって初めて与えられた[14]．

簡明のため，分散システムに属する各プロセスが実時間を参照できる（すなわち，正確な局所時計を保持している）と仮定する．このとき，プロセス i が特権を要求した時刻 T とその識別子 i の対によってプロセス間の競合を解消し，相互排除を解決できる[14]．***

[アルゴリズム *LAM*]

1. 特権を得たいと望むプロセス i は，自分自身も含めてすべてのプロセスに現時刻 T_i を時刻印としてもつメッセージ req(T_i, i) を送信する．

2. プロセス i から req メッセージを受信したプロセス j は，j の管理する待ち行列 $QUEUE_j$ にそのメッセージを送信時刻順 (T_i 順) に挿入し，プロセス i に現時刻 T_j を時刻印としてもつメッセージ reply(T_j, j) を返す．

3. i の req が $QUEUE_i$ の先頭（すなわち，到着した要求の中では最古）にありしかも (T_i, i) よりも新しい時刻印をもつ reply メッセージをすべてのプロセスから受信したときに限り，i は特権を許可される．

4. 特権を解放するときには，i は自分の req メッセージを $QUEUE_i$ から除去し，他のプロセスに release(i) メッセージを送信する．

5. j が i の release(i) メッセージを受信したときには，i の req メッセージを $QUEUE_j$ から除去する． □

LAM のアイデアを説明する．各プロセスは，分散システムに属するプロセスによる特権要求を時刻印が古い順に待ち行列 (*queue*) に保持しており，時刻印の古い順に特権を与える．そこで，特権を要求しているプロセスは，自分の要求が自分の保持する待ち行列の先頭に位置し，しかも，それが “今後到着するものも含めて” 最古のものであることが保証できれば，その時点で，特権を獲得できたことにすれば，これで相互排除が実現できたことになる．*LAM* は，この問題を reply メッセージを使って解決する．プロセス i がプロセス j からの reply を獲得したとすると，i が req(T_i, i) を送信した時刻 T_i が，j がそのメッセージを受信した時刻 T' よりも遅くなることはなく，また T' が，req(T_i, i) に対する応答 reply (T_j, j) の送信時刻 T_j よりも遅くなることはないので，$T_i < T_j$ が成立する．したがって，FIFO システムの仮定から，T_i より古い時刻印をもつ j の特権要求は，もしもそれが存在するならば，i が reply(T_j, j) を受信した時点で，すでに i に到着している．

しかしながら，メッセージ転送遅延が存在するので，各プロセス i が保持している局所時計 C_i を実時間に同期させることは，多くの場合不可能

---

* 飢餓とデッドロックの相違は，デッドロックは，デッドロックに陥っている分散システムの内部では決して解消できないのに対して，飢餓は，周りの状況が好転することによって，飢餓状態から脱出できる可能性が常に存在するところにある．

** FIFO システムではないシステムでも，メッセージにシーケンスナンバを打つことで，実際上容易に FIFO システムを実現できる．

*** 本稿では，各プロセスが固有の識別子をもつと仮定する．プロセス識別子は，二つ以上のプロセスが同時に特権を要求した場合に，これらの要求間に優先順位をつけるために用いる．プロセスが固有の識別子をもたない場合の議論は，たとえば，参考文献 29) を参照せよ．

である．Lamportは，アルゴリズム *LAM* が正しく動作するために局所時計が満たすべき本質的な要件は，それらが実時間を正確に刻むことではなく，$T_i<T'$ かつ $T'<T_j$ を満足するように C_i と C_j が同期していることであることを観察し，* このような時計を，**論理時計** (*logical clock*) と呼んだ．この条件を正確に述べると以下になる．a をプロセス i の事象とするとき，$C_i(a)$ は a が i で生起した（局所）時刻を表すとする．

1. プロセス i において事象 a が事象 b より早く生起したならば，$C_i(a)<C_i(b)$.

2. 事象 a をプロセス i のあるメッセージの送信事象，そして，事象 b をそのメッセージを受信するプロセス j での事象とするならば，$C_i(a)<C_j(b)$.

論理時計 C_i は次のようにして実現できる[14]．

1. 受信事象以外の事象 e がプロセス i で生起した場合，その直後に C_i に1を加える．

2. ●プロセス i がメッセージを送信するときには，時刻印 $ts=C_i$ をメッセージに付加して送信する．
   ●i が時刻印 ts をもつメッセージを受信したときには，C_i を $\max\{C_i, ts\}+1$ まで進める．

アルゴリズム *LAM* は，すべてのプロセスが平等に特権の認可に責任を負う（reply を許可メッセージとみなす）という意味で分散的な解であるが，一方，あるプロセスに特権を認可するには，すべてのプロセスの合意が必要である．したがって，たった一個のプロセスが故障しただけでアルゴリズムは進行せず，故障に対しては非常に弱い．そこで，耐故障性という観点から，少々のプロセスが故障しても残されたプロセスの間で相互排除を正常に続けられるような，分散アルゴリズムが考案されている．このアルゴリズムはコータリと呼ばれる構造を用いる．

3. コータリを用いた相互排除アルゴリズム

3.1 コータリ

Uをある固定された有限集合とする．相互排除アルゴリズムに用いるときには，Uをプロセスの集合と取る．以下の2条件を満たす空でないUの部分集合（コーラム (*quorum*) と呼ばれる）Q_i の集合 $\{Q_1, Q_2, \cdots, Q_N\}$ を**コータリ** (*coterie*) という[7]．

1. すべての対 i と j $(1\leq i, j\leq N)$ に対して，$Q_i\cap Q_j\neq\emptyset$.

2. すべての対 i と j $(1\leq i, j\leq N,\ i\neq j)$ に対して，$Q_i\not\subseteq Q_j$.

以下の各例では，U を有限集合，その要素数 $|U|=n$ とする．

例 1 サイズが $k=\lfloor n/2\rfloor+1$ である U のすべての部分集合から構成される集合 *Majority* はコータリである（参考文献 27）参照）． □

例 2 $x\in U$ を任意の要素とする．このとき，*Singleton*$=\{\{x\}\}$ はコータリである（参考文献 2）参照）． □

例 3 U の各要素 x に自然数 $\nu(x)$ を対応させる．$\sigma=\sum_{x\in U}\nu(x)$ とし，$k=\lfloor\sigma/2\rfloor+1$ と定める．$\mathcal{C}=\{Q\mid\sum_{x\in Q}\nu(x)\geq k\}$ は条件1を満たすが，条件2を満たさないので，一般にはコータリではない．しかし，$\mathcal{C}$ から，ある Q_i に対して，$Q_i\subseteq Q_j$ であるような Q_j を順次取り除いてできる集合 *Vote* はコータリである．任意の $x\in U$ に対して $\nu(x)=1$ と定めると，*Vote* は *Majority* に一致し，ある一つの要素 $x\in U$ に対して $\nu(x)=1$ と定め，それ以外の要素 $y\in U$ に対して $\nu(y)=0$ と定めると，*Singleton* に一致する．この方式で定義されるコータリを**票数割当可能コータリ** (*vote assignable coterie*) と呼ぶ．（参考文献 4）参照）． □

例 4 Uが論理的に2次元格子を形成しているとする．すなわち，$U=\{x_{ij}\mid 1\leq i\leq l, 1\leq j\leq m\}$ とする．任意の $1\leq i\leq l$ と $1\leq j\leq m$ に対して，$Q_{ij}=\{x_{hk}\mid h=i$ または $k=j\}$ によってコーラムを定めると，*Grid*$=\{Q_{ij}\mid 1\leq i\leq l, 1\leq j\leq m\}$ はコータリである（参考文献 16）参照）． □

例 5 U が論理的に根付き2分木 T を形成しているとする．すなわち，各要素 x に対して，左右二つの子 x_L と x_R が決まっている．（x_L と x_R のいずれかあるいは両方が存在しない場合もある．）x を根とする部分木 T_x に属する要素の集合を U_x とする．以下のように，各要素 x に対して，集合 U_x のコータリ $\mathcal{C}(U_x)$ を再帰的に構成する．

1. x が葉（すなわち，x_L も x_R も存在しない

* T_i は C_i で計った時刻であり，T', T_j は C_j で計った時刻である．なお，参考文献 14）では，*LAM* に限らずより一般的に，論理時計が分散アルゴリズムで本質的な役割を果たすことを論じている．

とき) $C(U_x)=\{\{x\}\}$ とする.

2. x が唯一の子プロセス x' をもつとき, $C(U_x)=C(U_{x'})$ とする.

3. それ以外の場合, $C(U_x)=C_1\cup C_2\cup C_3$ とする. ここに, $C_1=\{\{x\}\cup Q \mid Q\in C(U_{x_L})\}$, $C_2=\{\{x\}\cup Q \mid Q\in C(U_{x_R})\}$, そして, $C_3=\{Q_L\cup Q_R \mid Q_L\in C(U_{x_L}),\ Q_R\in C(U_{x_R})\}$ である.

T の根を x_{root} とすると, $Tree=C(U_{x_{root}})$ は U のコータリである (参考文献 1) 参照). □

例 6 コータリは有限射影平面 (*finite projective planes*) によっても与えられる.* U のいくつかの部分集合 L_i の集合 $\mathcal{P}$ を考える. U の部分集合 $L_i=\{a_1, a_2, \cdots, a_k\}$ を k 個の点 $a_1, a_2, \cdots, a_k$ を通る直線と呼ぶことにし, これが以下の3条件を満足するとき有限射影平面と呼ぶ.

1. 任意の異なる2点を通る直線はただ一つ存在する.

2. 異なる2直線はただ1点で交わる. すなわち, 任意の i と $j(i\neq j)$ に対して, $|L_i\cap L_j|=1$.

3. 4点でどの3点も1直線上にないものが存在する.

有限射影平面の直線 L_i の集合を FPP とすると, FPP はコータリとなる(参考文献 16) 参照). □

3.2 前川のアルゴリズム *MAE*

コータリを用いる相互排除アルゴリズムは Maekawa によって初めて提案された[16),17)]. しかし, 参考文献 16), 17) に述べられているアルゴリズムには, デッドロックの可能性があることが Sanders によって指摘されている[24)]. 本節で説明するアルゴリズム *MAE* は Sanders による改良版である. なお, *MAE* の中で用いられる時刻印 T_i は Lamport の論理時計によって生成されるものとする.

アルゴリズム *MAE* はおおよそ以下のような方針に基づいて相互排除を実現する. 特権を獲得するためには, コータリのあるコーラムに属するすべてのプロセスから許可を得る必要がある. また, 一度あるプロセスに許可を与えるとそのプロセスの許可なしでは許可を取り消すことはできない. したがって, すべての二つのコーラムは少なくとも一つのプロセスを共有するので (条件1), 二つのプロセスが同時に特権を獲得することはない. *MAE* が複雑となるのは, デッドロックや飢餓を避けるつぎのような工夫をしているからである. すなわち, あるプロセス Q から許可を得たとしても, Q がより優先度の高いプロセス P から許可を要請されたときには, その許可を P に譲渡する.

[アルゴリズム *MAE*]

あるコータリ C を使う.

1. **[要請]** 特権を望むプロセス i はメッセージ $\mathrm{req}(T_i, i)$ を, あるコーラム $Q\in C$ に属するすべてのプロセスに送信する.

2. **[ロック]** $\mathrm{req}(T_i, i)$ を受信したならば, j は他のプロセスが j をロックしているかどうか調べる.* j が他のプロセスによってロックされていないならば, i にメッセージ locked を返す.

3. **[照会]** 他のプロセス k が j をロックしていたときには, j は $\mathrm{req}(T_i, i)$ を待ち行列 $QUEUE_j$ に挿入する. ($QUEUE_j$ にはいくつかの req が時刻印順に保持され, j をロックする順番を待っている.)

   (a) もしも k の req または $QUEUE_j$ に保持されている req のどれかが, T_i よりも古い時刻印をもつならば, j は i にメッセージ failed を返す.

   (b) そうでなければ, j は k にメッセージ inquire を送信し, k が獲得しようと思っているコーラム $Q'\in C$ に属するプロセスの少なくとも一つから failed を受信したかどうかを照会する. ただし, 他の req に関して j がすでに inquire を k に送信しており, しかもそれに対する応答を受信していないときには, inquire を再度送信する必要はない.

4. **[照会処理]** inquire を送信してきたプロセス j に, もしもすでにメッセージ release を送信しているならば, k は j からの inquire を無視する. そうでなければ, k が自分の req に対する failed をあるプロセスから受信しているならば, メッセージ relinquish を j に返す.

5. **[譲渡処理]** j が k の relinquish を受信した

---

* 有限射影平面はある種の対称的 2-デザインである. 有限射影平面およびデザインについては, たとえば, 永尾 [参考文献 18), 第1章] (1974) を参照されたい.

* j がロックされているのは, あるプロセスにメッセージ locked を送信し, かつそのプロセスから対応するメッセージ release や relinquish を受信していないときである.

ならば，(1)req(T_k, k) を $QUEUE_j$ に挿入する．その前から $QUEUE_j$ にあった req の中で最古の時刻印をもつ req(T_h, h) を取りだし，そのメッセージを送信したプロセス h に対し locked を送る．(2)$h \neq i$ ならば i に failed を送信する．*

6. **[使用]** i が Q に属するすべてのプロセスから locked を受信したならば，特権を獲得できた．

7. **[解放]** 特権を手放すときには，i は Q に属するすべてのプロセスに release を送信する．

8. **[後処理]** j が i から release を受信したならば，i によるロックを解除する．待ち行列 $QUEUE_j$ が空でなければ，(1)待ち行列 $QUEUE_j$ にある req の中で最古の時刻印をもつ req (T_h, h) を取りだし，プロセス h に対し locked を送る．(2)もしも，受信した release が，あるプロセス l の求めに応じて j が i に送信した inquire に対する応答の代わりであるときには，$h \neq l$ ならば l に failed を送信する．**

待ち行列 $QUEUE_j$ が空ならば何もしない．

3.3 適切なコータリ

アルゴリズム *MAE* が用いるコータリとして，どのコータリが適当か，特に耐故障性およびその尺度の一つである可用性について本節では述べる．

アルゴリズム *MAE* の特徴の一つは，耐故障性が良いことである．トポロジが完全グラフで通信リンクは絶対故障しないような理想的な分散システムでは，少なくともあるコーラムに属するすべてのプロセスが正常であるならば，相互排除アルゴリズムは正常に機能する．したがって，耐故障性の観点から多数のコーラムからなるコータリが望まれる．*Majority* はコーラム数の意味で最良である．しかしながら，すぐ後で述べるように，コーラム数の多いコータリが常に耐故障性に優れているとは限らない．それはプロセスや通信リンクの故障率に依存する．アルゴリズム *MAE* の第2の特徴は，コーラム Q のサイズ $|Q|$ におおよそ比例するメッセージ数で相互排除を達成できる可能性を秘めていることである．したがって，メッセージ複雑度という観点からは，サイズの小さいコーラムを含むコータリが望まれる．*Singleton* はこの意味で最良である．そして，この二つの要望はトレード・オフの関係にある．

二つのコータリを C_1, C_2 とし，これらの間に関係

$$\forall Q_2 \in C_2 \exists Q_1 \in C_1 : Q_1 \subseteq Q_2$$

が成立するとき，C_1 は C_2 に対して**優位** (*dominate*) にあるという．C_1 が C_2 より優位にあれば，あきらかに，C_1 は C_2 より相互排除アルゴリズムに適している．なぜなら，C_2 を用いる相互排除アルゴリズムで，あるプロセスがコーラム Q_2 に属するすべてのプロセスから許可を得ることに成功したとすると，C_1 を用いる相互排除アルゴリズムでも，$Q_1 (\subseteq Q_2)$ に属するすべてのプロセスから許可を得ることができていたことになる．そこで，それ自身よりも優位なコータリが存在しないコータリだけが，最適コータリの候補として残される．このようなコータリを ND (*Non-Dominated*) コータリと呼ぶ．例 1-6 に示したコータリはすべて ND コータリである．

$U = \{x_1, x_2, \cdots, x_n\}$ とし，U の下でのあるコータリを $C = \{Q_1, Q_2, \cdots, Q_m\}$ とする．C に次の規則でブール関数 $fc(x_1, x_2, \cdots, x_n)$ を対応させる．$fc(b_1, b_2, \cdots, b_n) = 1$ となるための心要十分条件は，$X_1 = \{x_i | b_i = 1,\ 1 \leq i \leq n\}$ と定義したときに，ある $Q_j \in C$ が存在して，$Q_j \subseteq X_1$ となることである．このとき，C が ND コータリであるための必要十分条件は，fc が自己双対関数 (*selfdual function*) であることである[10]．Ibaraki と Kameda はこの対応を用いて，ND コータリの構造を，ブール関数の理論として研究し，すべての ND コータリは，例5に示した対応によって，* ある2分木構造と対応することを示した[10]．すなわち，すべての ND コータリは，最も単純な ND コータリである，3要素に対する *Majority* の“組合せ”からできている．なお，Neilson と Mizuno[19),20)] も，単純なコータリから複雑なコータリを構成する問題を提起し検討しているので，そちらも参照されたい．

上で述べたように，最適コータリの候補は ND コータリに絞ってもよい．しかし，すべての

---

* j は自分をロックする権利を k から i に譲るつもりであったが，i の求めに応じて j が k に inquire を送信した後，T_i より古い時刻印をもつ req(T_h, h) が到着したので，k の権利は h に譲渡された．

** 譲渡処理に対する脚注を参照せよ．

* 正確には，U のある要素が木の二つ以上の頂点に現れることを許す必要がある．

ND コータリが同程度の耐故障性をもつわけではない．代表的な ND コータリである *Majority* の耐故障性は深く研究されてきた．コータリの耐故障性を測る基準の一つに**可用度** (*availability*) がある．* コータリの可用度は，少なくとも一つのプロセスが特権を獲得できる確率である．すなわち，故障しているプロセスと通信リンクを取り除いた後の分散システムのトポロジを G' とするとき，あるコーラム Q が存在して，Q に含まれるすべてのプロセスが G' のある連結グラフの（頂点集合の）中に含まれる確率が可用度である．可用度はプロセスや通信リンクの故障率のみならず，分散システムのトポロジにも依存する．グラフの可用度の計算は，コータリの可用度の計算に帰着でき，前者は $\#P$-困難であることが知られている[3]．したがって，コータリの可用度の解析は一般にはきわめて困難であると予想される．

Barbara と Garcia-Molina[4] は，分散システムのトポロジが完全グラフであり，かつ通信リンクは故障しないと仮定できるとき，プロセスが正常である確率が（プロセスによらず）一律に $p(\geq 0.5)$ ならば，*Majority* はすべてのコータリの中で最大の可用度をもつことを示した．($p<0.5$ のとき，例 2 のコータリ *Singleton* が常に最大の可用度をもつと予想されている．) Tong と Kain[26] と Obradovic と Berman[21] はこのアイデアを拡張し，プロセス i が正常である確率 p_i がプロセスごとに異なる場合でも，うまく関数 $\nu(i)$ を定義すれば，例 3 の票数割当可能コータリ *Vote* が最大の可用度をもつことを示した．

一方，トポロジが完全グラフでない場合には，どの票数割当可能コータリも最適にならないようなトポロジが存在する[21]．最近，Ibaraki, Nagamochi, Kameda[11] は，トポロジが木およびリングの場合を考察し，木の場合には例 2 のコータリ *Singleton* が最適であること，および，リングの場合には，*Singleton* と，リングに属するある三つのプロセス x, y, z から構成される *Majority* コータリのどちらかが最適となることを示した．

メッセージ複雑度という観点では *Singleton* が最適であるのは明らかであるが，コーラム数の点から望ましくない．Maekawa の 2 次元格子に基づくコータリ *Grid* や有限射影平面に基づくコータリ *FPP* では，コーラムのサイズが約 $\sqrt{n}$ である[16]．Agrawal と El Abbadi[1] は最小のコーラムサイズが $\log n$ であるようなコータリ *Tree* と *Tree* を用いた相互排除アルゴリズムを示している．

4. コータリの拡張とその応用

特権を同時に獲得できるプロセス数が最大 k となるように分散システムを制御する問題を，k-相互排除問題と呼ぶ[15]．たとえば，k 個の（区別のない）座席を争う k-相互排除問題を考えてみよう．k 個の座席のそれぞれに名前をつけ，それぞれの座席を争う相互排除アルゴリズムを並列に実行すれば，この k-相互排除問題を解決できる．しかし，この方法では，本来区別のなかった座席を区別し，ある座席を指定して争うことになるので，局所的に混雑する座席が生じるなど無駄が多い．コータリの概念を拡張した k-コータリを用いれば，アルゴリズム *MAE* とよく似たアルゴリズムによって，自然に k-相互排除を解決できる[6),12),13]．有限集合 U の下での k-コータリは，以下の 2 条件を満足するコーラムの集合である．

1. 任意のコーラムの $k+1$ 集合 $\{Q_1, \cdots, Q_{k+1}\}$ に対して，ある二つの要素 Q_i, Q_j が存在して，$Q_i \cap Q_j \neq \emptyset$.

2. 任意の $h \leq k-1$ と，互いに交わらないコーラムから構成される任意の h-集合 $\{Q_1, \cdots, Q_h\}$ に対して，あるコーラム Q が存在して，任意の $1 \leq i \leq h$ に対して，$Q \cap Q_i = \emptyset$.

特権を要求するプロセスは，k-コータリのあるコーラムに属するすべてのプロセスから許可を得たときに特権を獲得できることにすると，定義から，k 個のプロセスまでは特権を獲得でき（条件 2），$k+1$ 個のプロセスが特権を獲得することはない（条件 1）．k-コータリを用いた k-相互排除アルゴリズムが提案され[13]，k-コータリの構成法[6]やその可用性[12]が議論されている．

Garcia-Molina と Barbara は参考文献 7) でコータリを導入したが，本稿で述べてきた分散相互排除とともに，**複製型データベース** (*replicated database*) の並行性制御（たとえば，参考文献 8), 25)）をコータリ導入の動機にあげている．複製型データベースでは，複数のサイトが一つのデータ

* 可用度以外の基準でコータリを評価する試みも最近行われている[22]．

アイテムの複製を保持している．そこで，複製間の一貫性を保つように複製にアクセスする方法が，問題となる．複製を保持するサイトの集合Uの下でのコータリを一つ固定し，データアイテムへの書込みを，あるコーラムに属する各サイトの保持する複製のおのおのへの書込み，データアイテムからの読出しを，あるコーラムに属する各サイトの保持する複製の中で最も新しい複製からの読出しとみなすと，コータリの性質から，任意のコーラムに属するサイトの保持する複製の中には，最新の書込み結果をもつ複製が含まれるので，一貫性が保たれる．上の説明では書込みと読出しに同じコーラム集合を用いたが，書込み用のコーラム集合 $\mathcal{P}$ と読出し用のコーラムの集合 Q に分けることにより，より弱い結合関係によっても一貫性を保持することができる．Fu[5] は，条件

$$\forall G\in\mathcal{P}\,\forall H\in Q : G\cap H\neq\emptyset$$

を満たす構造，**双コータリ** (*bicoterie*)，および，さらにそれに条件

$$\forall G,H\in\mathcal{P} : G\cap H\neq\emptyset$$

を追加した構造，**半コータリ** (*semicoterie*) を導入したが，これらの性質，構成方法，コータリとの関係などが調べられている[10),20)]．

5. おわりに

本稿では，分散相互排除問題を説明し，コータリを用いた分散相互排除アルゴリズムを紹介するとともに，そこで用いられるコータリと耐故障性の関係について述べ，さらに，コータリの拡張とその応用についても触れてきた．最後に，未解決な問題あるいは今後の発展が期待できる問題をいくつか示し，本稿を締めくくる．

1. コータリの設計問題，すなわち，分散システムが具体的に与えられたとき，耐故障性の高いコータリを構成する問題は，トポロジが木やリングなど，特殊な場合を除き未解決である．

2. 可用度の計算複雑度が一般には手に負えない程度に複雑であることはすでに述べた．そこで，可用度に代わる，耐故障性を測るための適切な尺度が望まれる．

3. 分散システムに参加するプロセスが動的に変化しても正しく動く相互排除アルゴリズムの提案，特にコータリの動的な更新問題は，現実的な問題の一つであろう．

4. 相互排除問題を解決する任意の分散アルゴリズムが利用する情報構造はコータリを本質的に含んでいると考えているのだが，どうだろうか．

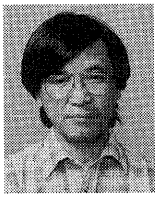
5. k-相互排除を含む，より一般的な分散競合解消問題に利用できるような結合構造を構成できるだろうか．

参考文献

1) Agrawal, D. and El Abbadi, A.: Efficient Solution to the Distributed Mutual Exclusion Problem, *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pp. 193-200 (1989).
2) Alsberg, P. A. and Day, J. D.: A Principle for Resilient Sharing of Distributed Resources, *Proc. 2nd International Conference on Software Engineering*, pp. 562-570 (1976).
3) Ball, M. O.: Computational Complexity of Network Reliability Analysis: An Overview, *IEEE Transactions on Reliability*, *R-35*, 3, pp. 230-239 (1986).
4) Barbara, D. and Garcia-Molina, H.: The Reliability of Voting Mechanisms, *IEEE Transactions on Computers*, *C-36*, 10, pp. 1197-1208 (1987).
5) Fu, A.: *Enhancing Concurrency and Availability for Database Systems, Ph. D. Thesis*, Simon Frase University, Canada (1989).
6) Fujita, S., Yamashita, M. and Ae, T.: Distributed k-Mutual Exclusion Problem and k-Coteries, *Proc. 2nd International Symposium on Algorithms, Lecture Notes in Computer Science 557*, Springer Verlag, Berlin, Germany, pp. 22-31 (1991).
7) Garcia-Molina, H. and Barbara, D.: How to Assign Votes in a Distributed System, *J. ACM 32*, 4, pp. 841-860 (1985).
8) Gifford, H.: Weighted Voting for Replicated Data, *Proc. 7th ACM Symposium on Operating Systems*, pp. 150-162 (1979).
9) 萩原，増沢：分散アルゴリズム，情報処理，Vol. 31, No. 9, pp. 1245-1256 (Sep. 1990).
10) Ibaraki, T. and Kameda, T.: A Boolean Theory of Coteries, *Proc. 3rd IEEE Symp. on Parallel and Distributed Processing*, Dallas, Texas, pp. 150-157 (1991).
11) Ibaraki, T., Nagamochi, H. and Kameda, K.: Optimal Coteries for Rings and Related Networks, *Proc. 12th International Conference on Distributed Computing Systems*, pp. 650-656 (1992).
12) Kakugawa, H., Fujita, S., Yamashita, M. and Ae, T.: Availability of k-Coteries, *IEEE Transactions on Computers* 42, pp. 5, pp. 553-558 (1993).

13) Kakugawa, H., Fujita, S., Yamashita, M. and Ae, T.: A Distributed k-Mutual Exclusion Algorithm Using k-Coterie, *Information Processing Letters* (submitted).
14) Lamport, L.: Time, Clocks and Ordering of Events in a Distributed System, *Comm. ACM 21*, 7, pp. 558-564 (1978).
15) Lamport, L. and Lynch, N.A.: Distributed Computing: Models and Methods, *in Hand Book of Theoretical Computer Science, volume B: Formal Models and Semantics* (*van Leeuwen, J.*), *Ed.* The MIT Press/Elsevier, Cambridge, Massachusetts/Amsterdam, Netherlands, pp. 1157-1200 (1990).
16) Maekawa, M.: A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems, *ACM Transactions on Computer Systems* 3, 2, pp. 145-159 (1985).
17) Maekawa, M., Oldehoeft, A.E. and Oldehoeft, R.R.: *Operating Systems: Advanced Concepts*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1987 (前川監訳, 佐藤, 星野, 渡辺訳：オペレーティングシステムの先進的概念, 丸善 (1990).
18) 永尾 汎：群とデザイン, 数学選書, 岩波書店 (1974).
19) Neilson, M.L., Mizuno, M. and Raynal, M.: A General Method to Define Quorums, *Proc. 12th International Conference on Distributed Computing Systems*, pp. 657-664 (1992).
20) Neilson, M.L. and Mizuno, M.: Coterie Join Algorithm, *IEEE Transactions on Parallel and Distributed Systems*, 3, 5, pp. 582-590 (1992).
21) Obradovic, M. and Berman, P.: Voting as the Optimal Static Pessimistic Scheme for Managing Replicated Data, *Proc. 9th Symposium on Reliable Distributed Systems* (1990).
22) Papadimitiou, C.H. and Sideri, M.: Optimal Coteries, *Proc. 10th ACM Symposium on Principles of Distributed Computing*, pp. 75-80 (1991).
23) Raynal, M. (Translated by Beeson, D.): Algorithms for Mutual Exclusion, *Scientific Computation Series*, The MIT Press, Cambridge, Massachusetts (1986).
24) Sanders, B.A.: The Information Structure of Distributed Mutual Exclusion Algorithms, *ACM Transactions on Distributed Systems* 5, 3, pp. 284-299 (1987).
25) Skeen, D.: A Quorum-Based Commit Protocol, *Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks*, pp. 69-80 (1982).
26) Tong, Z. and Kain, R.Y.: Vote Assignments in Weighted Voting Mechanisms, *Proc. 7th Symposium on Reliable Distributed Systems*, pp. 138-143 (1988).
27) Thomas, R.H.: A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, *ACM Transactions on Database Systems* 4, 2, pp. 180-209 (1979).
28) 山下：分散アルゴリズムについて, オペレーションズ・リサーチ, 37, 8, pp. 382-385 (1992).
29) Yamashita, M. and Kameda, T.: Electing a Leader When Processor Identity Numbers Are Not Distinct, *Proc. 3rd International Workshop on Distributed Algorithms, Lecture Notes in Computer Science 392*, Springer-Verlag, Berlin, Germany, pp. 303-314 (1989).

(平成5年5月26日受付)



山下 雅史 (正会員)

昭和49年京都大学工学部情報工学科卒業. 昭和52年同大学院修士課程修了. 昭和55年名古屋大学大学院博士課程修了. 工学博士. 豊橋技術科学大学助手, 広島大学工学部助教授を経て, 平成4年より広島大学工学部教授. 並列/分散システムとそのアルゴリズムの研究に従事.