効率的な資源割当てのための Local Majority Coterie

程 子学[†] 和田 裕[††]
橋本 悟[†] 野口正一[†††]

分散システムにおける資源配分問題は，分散システムの構成のために最も重要な研究課題の１つである．資源配分問題の解決のためには，排他制御の実現に加え，デッドロックの状態と飢餓状態を回避する分散アルゴリズムが必要とされる．この問題に関しては，今までさかんな研究がなされてきており，様々な解法が提案されてきた．本論文では，既存の解法より，メッセージ通信量が少なく，耐故障性に優れた Local Majority コテリーの定義をあたえ，それを構成する方法を提案する．次に，Local Majority コテリーを用いてデッドロックや飢餓状態を回避しながら資源配分問題を解決するアルゴリズムを提示する．最後にそのアルゴリズムのメッセージ複雑度の考察を行う．

Local Majority Coterie for Efficient Allocation of Resources

Zixue Cheng,[†] Yutaka Wada,[††] Satoru Hashimoto[†]
and Shoichi Noguchi[†††]

To solve the resource allocation problem in distributed systems, many studies have been performed so far, and various solutions for the problem have been proposed. It is necessary to guarantee the mutual exclusive use of a resource in a solution for the resource allocation problem. Furthermore, this solution should avoid both deadlock and starvation. In this paper, we define the Local Majority Coterie that is more efficient in term of message complexity and robuster to network failures than existing ones, and propose a method to create a Local Majority Coterie. Then, we show an algorithm, which guarantees mutual exclusive access to a resource, deadlock-free, and starvation-free, to solve the distributed resource allocation problem, by using Local Majority Coterie. Finally we discuss the message complexity of the algorithm.

1. まえがき

分散システムにおける資源配分問題は，分散システムの構成のために最も重要な研究課題の１つである．その概要は次のようにまとめられる．分散システムには，複数の資源と複数のプロセスが存在する．各プロセスは任意の資源の一部を要求し，要求された資源を獲得したときにこれを利用する．各資源は，排他的に利用されることが要求されている．つまり，同時に２つ以上のプロセスに配分することができない．また，プロセスがお互いのすでに獲得している資源を要求しあい，いずれかのプロセスも資源の利用ができないデッドロック（deadlock）の状態と，あるプロセスが永久に資源を獲得できない飢餓状態（starvation）の双方を回避する分散アルゴリズムが必要とされる．

分散資源配分問題に関しては，今までさかんな研究がなされてきており，様々な解法が提案されてきた．その中に，効率や耐故障性をあげるために，資源配分のためのアルゴリズムの構成にコテリーの概念を導入する方法があり，コテリーを用いた相互排除問題のための分散アルゴリズム[4]が提案されている．さらに，コテリーを資源配分問題に適用したローカルコテリーを用いた分散アルゴリズム[2]が提案された．本論文では，Kakugawa ら[2]のローカルコテリーの構成法よりもクォーラムサイズについてより優れて，耐故障性がより高い Local Majority コテリーの構成法を提案する．そして，Local Majority コテリーを用いてデッドロックや飢餓状態を回避しながら資源配分問題を解決するアルゴリズムを提示し，最後に Local Majority

† 会津大学コンピュータ理工学部
School of Computer Science and Engineering, University of Aizu
†† 会津大学大学院コンピュータ理工学研究科
Graduate School of Computer Science and Engineering, University of Aizu
††† 仙台応用情報学研究振興財団
Sendai Foundation for Applied Information Sciences

コテリーを用いたアルゴリズムの通信量についての考察を行う．

相互排除問題の解法に利用されるコテリーは，システムにおけるすべてのプロセスから構成される．一方，ローカルコテリーについては，各プロセスは，自分と資源を競合するプロセスからコテリーをローカルに構成する．それにより，資源を競合しないプロセスが互いに干渉することなく資源を利用する可能性は生まれる．ただし，Kakugawa ら[2]の構成法によるローカルコテリーは，配分の独立性（allocation independence[5]）が保証できない．つまり，資源を競合しないプロセスどうしが互いの資源配分のための排他制御に干渉することがある．本論文でも，Kakugawa ら[2]の構成法と同様に，配分の独立性（allocation independence）が保証できず，資源を競合しないプロセスどうしが互いの資源配分のための排他制御に干渉する事がある．配分の独立性（allocation independence）を保証するローカルコテリー（Sharing Structure Coterie）の定義およびそのようなローカルコテリーの存在条件は，Sung ら[5]により与えられている．

2. コテリーとローカルコテリー

2.1 分散システムの資源配分モデル

分散システムの資源配分モデルは，二部グラフ $G\big(V, E\big)$ で表される．ここで V は，分散システムのプロセスの集合 P と資源の集合 R の和集合 $(V = P \cup R)$ である．また E は，プロセスと資源についての隣接関係を示す辺の集合を示し，辺で結ばれた隣接する資源についてプロセスは使用を要求する．このとき，あるプロセス $p \in P$ について，p が使用を要求する資源の集合を A_p で表す．各プロセスはそれぞれが局所的な時刻系を持つため，プロセス間の時間の整合性は論理時計の概念[3]を導入して解決する．あるプロセスが資源の使用を要求する場合，それぞれの排他制御を行うプロセスにおいて合意が得られたときにはじめて資源の使用が許可されるものとする．資源の使用を許可されたプロセスは資源を使用するが，それは有限時間内に終わるものとする．

分散システムにおいてプロセスは，資源の使用を要求する要求プロセスとしての役割を果たすだけでなく，資源配分のための排他制御を行う制御プロセスとしての役割も果たす．制御プロセスが同時に複数の要求プロセスに対して資源の使用を許可することはない．各制御プロセスは自らの保持する局所的な資源配分の情報のみを元に排他制御に関する決定を下す．排他制御を行うためのメッセージは直接他のプロセスに送られるものとし，そのメッセージの転送時間は不定であるが，有限時間内に必ず届くものとする．

2.2 コテリー

コテリーを用いた分散アルゴリズム[4]は，排他制御を行う制御プロセスの組合せをうまく選ぶことによりメッセージ通信量の削減を図ったものである．

プロセス集合 P の部分集合を要素とする集合，つまり集合族 $\mathcal{C}$ について，その任意の集合要素 $Q \in \mathcal{C}$ が以下の条件を満たすとき，$\mathcal{C}$ をコテリー（coterie），Q をクォーラム（quorum）と呼ぶ[1),2)]．

- 空集合であるクォーラムは存在しない．
  $(\forall\, Q \in \mathcal{C},\ Q \neq \emptyset)$
- クォーラムどうしは必ず 1 つ以上の共通要素を持つ．
  $(\forall\, Q_i,\ Q_j \in \mathcal{C},\ Q_i \cap Q_j \neq \emptyset)$
- あるクォーラムの部分集合であるクォーラムは存在しない．
  $(\forall\, Q_i,\ Q_j \in \mathcal{C},\ Q_i \not\subseteq Q_j)$

コテリーを用いた分散アルゴリズムでは，クォーラム Q は任意の要求プロセスに対してその制御プロセス集合になる．このとき，Q を P に含まれるすべてのプロセスの過半数集合（Majority）とすると，クォーラムを容易に求めることができる．プロセス集合 P に含まれる全要素数を $|P|$ とすると，その過半数集合の要素数 maj は $maj = \lfloor |P|/2 \rfloor + 1$ で表される．Majority コテリーにおいて，排他制御のために問い合わせる制御プロセスの数はすべてのプロセスが制御プロセスとなる場合よりも少なくなるため，排他制御を行うためのメッセージ量は削減される．また，任意の 2 つの制御プロセス集合は互いに共通する制御プロセスを持つので，いずれの制御プロセス集合に資源の問合せをしても，それぞれの集合が資源の使用を許可する要求プロセスは複数にはならないため，排他制御が実現される．

2.3 ローカルコテリー

ローカルコテリーを用いた分散アルゴリズム[2]は，相互排除問題に適用したコテリーを資源配分問題に適用させたものである．ローカルコテリーは各プロセス p_i ごとに固有のコテリー $\mathcal{C}_{p_i}$ を構成するが，そのクォーラムはプロセス p_i が要求する資源について，それを競合するすべてのプロセスの集合から求められる．ローカルコテリーは以下のような条件を満たす[2]．

- すべてのローカルコテリーは空集合ではない．
  $(\forall\, p_i \in P,\ \mathcal{C}_{p_i} \neq \emptyset)$
- 資源を競合するプロセスのクォーラムどうしは必ず 1 つ以上の共通要素を持つ．
  $(\forall\, p_i, p_j \in P,\ A_{p_i} \cap A_{p_j} \neq \emptyset \Rightarrow$
  $\forall\, Q_i \in \mathcal{C}_{p_i},\ \forall\, Q_j \in \mathcal{C}_{p_j},\ Q_i \cap Q_j \neq \emptyset)$

- あるローカルコテリーのクォーラムについて，部分集合であるようなクォーラムが存在しない．
  ($\forall\ p_i \in P,\ \forall\ Q_{i1},\ Q_{i2} \in \mathcal{C}_{p_i},\ Q_{i1} \neq Q_{i2} \Rightarrow Q_{i1} \not\subseteq Q_{i2}$)

Kakugawa らが与えたローカルコテリーの構成法[2)]では，ローカルコテリーは 1 つのクォーラムのみからなる（$\forall\ p_i \in P,\ |\mathcal{C}_{p_i}| = 1$）．また，そのクォーラムは，$p_i$ と競合するすべてのプロセスからなる．各プロセスのローカルコテリーに属するクォーラムがただ 1 つであるため，耐故障性についての考慮が不十分であり，またそのクォーラムは競合するすべてのプロセスからなるため，排他制御のための効率についての考慮も不十分である．

3. LM（Local Majority）コテリー

3.1 LM コテリーの構成

本論文で提案する LM（Local Majority）コテリーは，Kakugawa ら[2)]のローカルコテリー定義を満たすものであるが，Kakugawa ら[2)]が示したローカルコテリーの構成法の効率と耐故障性を改善するために，Majority コテリーの構成と同様に，各プロセス p_i の $\mathcal{C}_{p_i}$ は複数のクォーラムによって構成され，各々のクォーラムは競合するプロセスの一部から構成される．

LM コテリーは，次のような手順で構成される．分散システムの各資源 $r \in R$ について，r を競合するプロセスの集合 S_r を構成し，集合 S_r のすべての最小過半数部分集合からなる Majority コテリー $\mathcal{M}_r$ を作る．すべてのプロセス $p \in P$ について，p が利用する各資源 $r \in A_p$ に関する $\mathcal{M}_r$ に対し，各 $\mathcal{M}_r$ に属するクォーラムを 1 つずつ選び，それらの論理和をとる．この論理和を $\mathcal{C}_p$ のクォーラム候補とする．各 $\mathcal{M}_r$ におけるクォーラムのどれを選択するかによっては，作られる $\mathcal{C}_p$ のクォーラムの候補は異なる．すべての可能な各 $\mathcal{M}_r$ のクォーラムの組合せに対し同様に論理和をとり，その集合を $\mathcal{C}_p$ のクォーラムの候補集合 $\mathcal{T}_p$ とする．最後にその候補集合 $\mathcal{T}_p$ から，すべてのクォーラム候補について，その部分集合であるクォーラム候補が存在するようなクォーラム候補を除外し，LM コテリー $\mathcal{C}_p$ が得られる．以下はその手順をアルゴリズムとして表すものである．

when making LM coteries
begin
  for *each* $r \in R$ **do**
    begin
      $S_r := \left\{ p \in P \;\middle|\; r \in A_p \right\}$;
      $maj := \lfloor |S_r|/2 \rfloor + 1$;
      $\mathcal{M}_r := \left\{ m_i \;\middle|\; \forall\ m_i \subseteq S_r,\ |m_i| = maj \right\}$;
    end
  for *each* $p \in P$ **do**
    begin
      $\mathcal{T}_p := \left\{ Q_p \;\middle|\; Q_p = \bigcup_{\forall\ r \in A_p, \exists\ m \in \mathcal{M}_r} m \right\}$;
      $\mathcal{C}_p := \left\{ Q_{pi} \;\middle|\; \forall\ Q_{pi}, Q_{pj} \in \mathcal{T}_p, Q_{pj} \not\subseteq Q_{pi} \right\}$;
    end
end.

P: set of processes
R: set of resources
A_p: $A_p = \left\{ r \;\middle|\; p\ requires\ r \right\} \subseteq R$

このようにして求められた LM コテリーは 2.3 節で示されたローカルコテリーの条件を満たす．

3.2 LM コテリーの例

図 1 のような分散システムを考える．この分散システムはプロセス $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ および資源 $R = \{r_1, r_2, r_3\}$ から構成され，プロセス $p_i \in P$ の使用する資源 A_{p_i} は以下のとおりに定められる．

- $A_{p_1} = A_{p_2} = \{r_1\}$
- $A_{p_3} = A_{p_4} = \{r_1, r_2\}$
- $A_{p_5} = \{r_2, r_3\}$
- $A_{p_6} = \{r_3\}$

各資源 $r_j \in R$ に関して，その r_j を競合するプロセスの集合 S_{r_j} は次のとおりである．

- $S_{r_1} = \{p_1, p_2, p_3, p_4\}$
- $S_{r_2} = \{p_3, p_4, p_5\}$
- $S_{r_3} = \{p_5, p_6\}$

各 S_{r_j} 上の Majority コテリー $\mathcal{M}_{r_j}$ は S_{r_j} から作られ，次のとおりである．

- $\mathcal{M}_{r_1} = \{\{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}, \{p_1, p_3, p_4\}, \{p_2, p_3, p_4\}\}$
- $\mathcal{M}_{r_2} = \{\{p_3, p_4\}, \{p_4, p_5\}, \{p_3, p_5\}\}$
- $\mathcal{M}_{r_2} = \{\{p_5, p_6\}\}$

各 p_i に関して，$\mathcal{M}_{r_j}$ からは $\mathcal{C}_{p_i}$ の候補集合 $\mathcal{T}_{p_i}$ が作られ，次のとおりである．

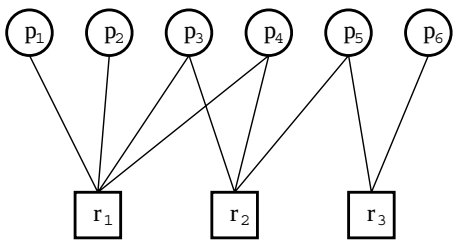

図 1 分散システムの例
Fig. 1 An example of distributed systems.

- $\mathcal{T}_{p_1} = \mathcal{T}_{p_2} = \{\{p_1,p_2,p_3\},\{p_1,p_2,p_4\},\{p_1,p_3,p_4\},\{p_2,p_3,p_4\}\}$
- $\mathcal{T}_{p_3} = \mathcal{T}_{p_4} = \{\{p_1,p_2,p_3,p_4\}\{p_1,p_2,p_3,p_4,p_5\}\{p_1,p_2,p_3,p_5\}\{p_1,p_2,p_4,p_5\}\{p_1,p_3,p_4\}\{p_1,p_3,p_4,p_5\}\{p_2,p_3,p_4\}\{p_2,p_3,p_4,p_5\}\}$
- $\mathcal{T}_{p_5} = \{\{p_3,p_4,p_5,p_6\},\{p_4,p_5,p_6\},\{p_3,p_5,p_6\}\}$
- $\mathcal{T}_{p_6} = \{\{p_5,p_6\}\}$

各 p_i に関して，$\mathcal{C}_{p_i}$ はその候補集合 $\mathcal{T}_{p_i}$ から作られ，次のとおりとなる．

- $\mathcal{C}_{p_1} = \mathcal{C}_{p_2} = \big\{\{p_1,p_2,p_3\},\{p_1,p_2,p_4\},\{p_1,p_3,p_4\},\{p_2,p_3,p_4\}\big\}$
- $\mathcal{C}_{p_3} = \mathcal{C}_{p_4} = \big\{\{p_1,p_3,p_4\},\{p_2,p_3,p_4\},\{p_1,p_2,p_3,p_5\},\{p_1,p_2,p_4,p_5\}\big\}$
- $\mathcal{C}_{p_5} = \big\{\{p_3,p_5,p_6\},\{p_4,p_5,p_6\}\big\}$
- $\mathcal{C}_{p_6} = \big\{\{p_5,p_6\}\big\}$

となる．LM コテリー $\mathcal{C}_{p_3}$，$\mathcal{C}_{p_4}$ から分かるとおり LM コテリーのクォーラムについて，その要素数は一定でない場合がある．本論文では，ある LM コテリー $\mathcal{C}$ における要素数が最大であるクォーラムを Q_{max}，最小であるクォーラムを Q_{min} とする．

4. LM コテリーを用いた分散アルゴリズムの動作

4.1 アルゴリズムの概要

以下では，LM コテリーを用いる資源割当てを行う分散アルゴリズムを示す．このアルゴリズムは，基本的な部分が Kakugawa らのアルゴリズム[2]と同様な制御を行うが，コテリーの構成が異なるため，効率や耐故障性が改善される．

また，本アルゴリズムは，記述をシンプルにするため，Kakugawa らのアルゴリズム[2]と比べて，以下のように簡略化したものである．

各要求プロセスは，自らに隣接する資源について，任意の複数資源を獲得することができる Kakugawa らのアルゴリズム[2]とは異なり，本論文では，各要求プロセスが資源の使用を要求する場合，自らに隣接するすべての資源についてこれを要求する．

4.2 アルゴリズムの詳細

各プロセス p_i は，次の内部変数を持つ．

- $time_i$—プロセス p_i の論理時刻．
- $queue_i$—プロセス p_i に送られてきた $inqury$ メッセージのタイムスタンプを記憶する配列．
- $permit_i$—$permission$ メッセージを送ったプロセス p_j について p_j の資源要求タイムスタンプを記憶する．p_j から $release$ または $dispose$ メッセージが返されたならばその内容は破棄される．
- $cancel_i$—プロセス p_j に $cancel$ メッセージを送ったかどうかを記憶する．初期値として，$null$ に設定し，プロセス p_j に $cancel$ メッセージを送る場合に，$cancel_i$ に p_j を記録する．p_j から $release$ または $dispose$ メッセージが返されたならばその内容は破棄され，初期値にリセットされる．

次にアルゴリズムの動作について述べる．

このアルゴリズムは，基本的には，前川のアルゴリズム（の Sanders による改良版[6]）と同じであるが，デッドロック回避のための横取り（preempt）のメカニズムは，前川のアルゴリズム（の Sanders による改良版）と異なって，前川のアルゴリズム（の Sanders による改良版）に使われる failed メッセージが使われていない．それにより，複雑度のオーダは変わらないが，メッセージの数が少なくなる．

(1) **プロセス p_i が資源の使用を要求するとき**
任意のクォーラム $Q \in \mathcal{C}_{p_i}$ を選び，資源の使用を終えるまでこのクォーラムのプロセスを要求プロセス p_i の制御プロセスとする．Q に属するすべての制御プロセスに対して $inquiry(p_i, time_i)$ メッセージを送り，その各々からの $permission$ メッセージを待つ（$(p_i, time_i)$ は p_i が $inquiry$ メッセージを発行した時点での論理時刻 $time_i$ を含んだタイムスタンプである）．要求プロセス p_i はすべての制御プロセスから $permission$ メッセージを返されたときに初めて資源の使用を開始する．

(2) **プロセス p_i が資源の使用を終えたとき**
資源の使用を終えたならば，Q に属するすべての制御プロセスに対して $release(p_i, time_i)$ メッセージを送り，資源を解放した旨を通知する．

(3) **プロセス p_i がプロセス p_j から $inquiry(p_j, time_j)$ メッセージを受け取ったとき**
プロセス p_j からの $inquiry$ メッセージを受け取った時点で，まだ他の要求プロセスに対し $permission$ メッセージを送っていない場合（つまり，$permit_i$ にタイムスタンプが格納されていない場合），プロセス p_j に対し $permission(p_i)$ メッセージを送り，$permit_i$ に $(p_j, time_j)$ を格納する．すでにプロセス p_k に $permission$ メッセージを送っていた場合は，p_j のタイムスタンプ $(p_j, time_j)$ を待ち行列 $queue_i$ に入れる．次にプロセス p_j とプロセス p_k のタイムスタンプを比較し，タイムスタンプの大小で優先度を定め，小さいタイムスタンプを持っているプロセスは，大きなタイムスタンプを持っているプロセス

より，優先度が高いと見なす．p_j よりプロセス p_k の優先度が低ければ，p_k に対して，$cancel$ を送っていない場合（$cancel_i = null$ の場合），$cancel_i$ に p_k を入れて，p_k に対し $cancel(p_i)$ メッセージを送り，p_k からの $dispose$ もしくは $release$ メッセージを待つ（$cancel_i \neq null$ の場合，$dispose$ を送信しない）．

(4) **プロセス p_i がプロセス p_j から $permission(p_j)$ メッセージを受け取ったとき**

すべての制御プロセスから $permission$ メッセージを受け取ったならば，資源を使用する．そうでなければ，すべての制御プロセスから $permission$ メッセージが送られてくるのを待つ．

(5) **プロセス p_i がプロセス p_j から $release(p_j, time_j)$ メッセージを受け取ったとき**

$cancel_i$ に p_j が入っている場合，それを破棄する（$null$ にする）．$permit_i$ に格納されているタイムスタンプ $(p_j, time_j)$ を破棄し，待ち状態を解除する．もし $queue_i$ にプロセス p_i からの $permission$ メッセージを待つ要求プロセスのタイムスタンプが存在すれば，その最も優先度の高い要求プロセス p_k に対して $permission(p_i)$ メッセージを送り，$permit_i$ に $(p_k, time_{p_k})$ を格納する．

(6) **プロセス p_i がプロセス p_j から $dispose(p_j)$ メッセージを受け取ったとき**

$cancel_i$ を $null$ にリセットする．$permit_i$ に格納されているタイムスタンプを待ち行列 $queue_i$ に入れる．$permit_i$ に格納されているタイムスタンプ $(p_j, time_j)$ を破棄する．もし $queue_i$ にプロセス p_i からの $permission$ メッセージを待つ要求プロセスのタイムスタンプが存在すれば，その最も優先度の高い要求プロセス p_k に対して $permission(p_i)$ メッセージを送り，$permit_i$ に $(p_k, time_{p_k})$ を格納する．

(7) **プロセス p_i がプロセス p_j から $cancel(p_j)$ メッセージを受け取ったとき**

もしすべての制御プロセスから $permit$ を受信し，資源の使用を開始しているならば，プロセス p_j からの $cancel$ メッセージを無視する．そうでなければ，$dispose(p_i)$ メッセージをプロセス p_j に送る．

5. アルゴリズムの解析

5.1 正当性の証明

本論文のアルゴリズムは，デッドロックと飢餓状態を回避しながら，各資源へのアクセスの排他制御を行うアルゴリズム，つまり以下のすべての条件を満たすアルゴリズムである．

5.1.1 排他制御の保証

[命題] 制御プロセスは要求プロセスに対して排他的に資源使用の許可を与えるため，競合される各資源排他制御は保証される．

（証明） 4.2 節から，ある制御プロセスに対し 1 つ以上の $inquiry$ メッセージが送られた場合，その制御プロセスはその中の 1 つのプロセスに対してのみ，$permission$ メッセージを送り，$permission$ を受け取った要求プロセスから $release$，$dispose$ メッセージが送られない限り，他の要求プロセスに対して $permission$ メッセージを送ることはない．つまり資源の使用権を与えた要求プロセスが，使用権を（一時的にでも）手放さない限りは，資源の使用を要求する他の要求プロセスに対して使用権を与えることはない．また，3.1 節の方法によって構成される LM コテリーが 2.3 節の条件を満たすため，資源使用の競合を起こしうる要求プロセスどうしについて，各々の制御プロセス集合であるクォーラムの間には必ず共通の制御プロセスがあるため，任意の 2 つの要求プロセスは，資源を競合するならば，各々の制御プロセス集合（クォーラム）のすべての制御プロセスから使用権（$permission$）が（同時に）与えられることはありえない．したがって，各資源の排他制御は保証される．

5.1.2 デッドロックの回避

[命題] 少なくとも最も優先度の高い要求プロセスに，その制御集合のすべてのプロセスから資源使用権が与えられるため，デッドロックは回避される．

（証明） ある要求プロセスに対して，$permission$ メッセージを送り，資源の使用権を与えた制御プロセスに，優先度の高い他の要求プロセスから $inquiry$ メッセージが到着すると，すでに資源の使用権を与えた要求プロセスに対して $cancel$ メッセージを送る．$cancel$ メッセージを受け取った要求プロセスは，資源を使用しはじめていない場合，$cancel$ メッセージに対して，$dispose$ メッセージを送り返し，資源の使用権を一時放棄する．$dispose$ メッセージを受け取った制御プロセスは，最も優先度の高い要求プロセス（$inquiry$ のタイムスタンプが最小）に $permission$ メッセージを送り，資源の使用権を与える．この方法により，少なく

とも，要求メッセージの全順序関係より，最小のタイムスタンプを持つ（最も優先度の高い）*inquiry* メッセージを発行した要求プロセスは，そのプロセスのすべての制御プロセスから資源の使用権（*permission* メッセージ）を受け取ることができる．したがって，デッドロックは回避される．

5.1.3 飢餓状態の回避

[命題] 任意の要求プロセスに，有限時間内に必ず資源使用権が与えられるため，飢餓状態は回避される．

（証明） アルゴリズムにおいて，*inquiry* メッセージを発行した要求プロセスは，そのタイムスタンプの全順序関係によって優先順位を定めており，タイムスタンプが小さい順に制御プロセスから *permission* メッセージを受け取る．2.1 節より，資源の使用は有限時間内に終了する．その後資源を使用していたプロセスは，ただちに *release* メッセージをすべての制御プロセスに送り，資源を解放する．*inquiry* メッセージのタイムスタンプが最小のプロセスが資源を使用したあと，2 番目に小さいタイムスタンプが最小となり，その *inquiry* を発行したプロセスは，すべての制御プロセスより，資源使用権が与えられる．さらに，タイムスタンプが最小のプロセスが資源を使用し，その後再び資源の使用を要求する場合を考える．このとき発行された新たな *inquiry* メッセージのタイムスタンプは，他のプロセスがすでに発行したどの *inquiry* メッセージよりも新しい（大きい）．したがって，すべての要求プロセスについて有限時間内に，自分の要求メッセージ *inquiry* のタイムスタンプが最小となり，必ず *permission* メッセージが返される順番が回ってくる．よって，有限時間内に必ず資源の使用権が得られるので飢餓状態の発生はありえない．

5.2 通信量の考察

以下に本論文で示したアルゴリズムの通信量について，最良のケースと最悪のケースを提示する．

最良のケースは，各要求プロセス p_i が LM コテリーの最小クォーラム Q_{min} を制御プロセス集合として選択し，その各制御プロセスで競合を起こさずにすべての資源を獲得した場合である．このとき要求プロセス p_i と Q_{min} の各制御プロセスの間で *inquiry*, *permission*, *release* の 3 つのメッセージが 1 度ずつやりとりされる．したがって，そのメッセージ通信量は $3|Q_{min}|$ である．

最悪のケースは以下のような状況に陥ったときである．ある要求プロセス $p_i \in P$ について，p_i と資源を競合しあうプロセス（要求プロセス）の集合を P_{p_i} とする．このとき P_{p_i} は p_i 自身を含む（$p_i \in P_{p_i}$）．またすべての要求プロセスの選択したクォーラムのサイズは，Q_{max}（LM コテリーの最大クォーラム）である．各要求プロセスは，選択したクォーラムの各制御プロセスに *inquiry* メッセージを送る．各要求プロセスが発行した *inquiry* メッセージは優先度の低い順に制御プロセスに到着する．ある要求プロセス p_i に対してすでに *permission* メッセージを発行した制御プロセス p_k に，より優先度の高い要求プロセス $p_j \in P_{p_i}$ からの *inquiry* メッセージが到着した場合，制御プロセス p_k はアルゴリズムに従って先に資源の使用権を与えた要求プロセス p_i に対して *cancel* メッセージを送り，資源の使用権を一時取り上げる．このとき *cancel* メッセージが前述の要求プロセス p_i に届くタイミングは，p_i が資源の使用を開始する前である．以上のような状況において，P_{p_i} に属するすべてのプロセスが制御プロセス p_k に対して *inquiry* メッセージを送るとする．このような状況下でアルゴリズムに従い排他制御についてのメッセージ通信を行うと，最終的に最も優先度が低いプロセスが資源の使用を終えるまでに送りあうメッセージの総量は $\big(3+6(|P_{p_i}|-1)\big)|Q_{max}|$ になる．ただし，配分の独立性（allocation independenceSung ら[5]）を保証することができないことや p_i と資源を競合するプロセスがシステムの全プロセスとなる場合を考えて，最悪の場合に，メッセージの総量における P_{p_i} は，全システムプロセスの集合 P となる．

LM コテリーはローカルコテリーの構成法の 1 つであるため，各資源の排他的な割当てのためのアルゴリズムは同一のものを利用することができる．いいかえれば，本論文の LM コテリーを Kakugawa らのアルゴリズム[2]に適用しても割当てが正常に動作する．

しかし，同一の分散システムにおいて，LM コテリーのクォーラム（制御プロセス集合）のサイズは，**図 2** に示すように各リソースを共有するプロセスの数が 2 つの場合に限り，Kakugawa ら[2]の構成法によるクォーラムのサイズと同じであるが，一般的に，Kakugawa

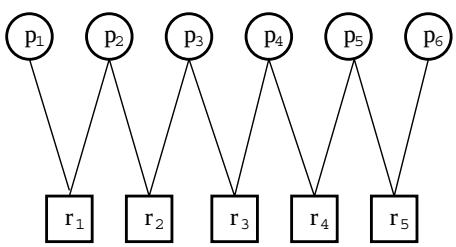

図 2 各ソースが 2 つのプロセスだけに競合される例
Fig. 2 An example where each resourse is competed for by exactly two processes.

ら[2]の構成法によるクォーラムのサイズより小さくなる．したがって，本論文のアルゴリズムは，排他制御のためのメッセージ量は少なくなると予想される．

また，コテリーを構成するクォーラムの数は，図2に示すように各リソースを共有するプロセスの数が2つの場合に限り，Kakugawa ら[2]の構成法によるクォーラムの数と同じであるが，一般的に，Kakugawa ら[2]の構成法によるクォーラムの数より大きくなる．したがって，プロセスの停止故障に対する耐性の向上も予想される．

6. むすび

本論文では，ローカルコテリーの構成法の1つとして，Majority コテリーの考えを導入した LM コテリーを提案し，それを用いた分散アルゴリズムを示した．そして，Kakugawa ら[2]のローカルコテリーの構成法と比較し，LM コテリーを導入することにより，ローカルコテリーの条件を満たしながら，制御プロセスの総数を減らすことができ，これにより排他制御のためのメッセージ量が削減できる．また，各コテリーに属するクォーラムの数が増えることにより，耐故障性が高くなる．今後の課題としては，LM コテリーのクォーラムサイズについての厳密な定量的解析や Availability の計算や allocation independence 性を満たすローカルコテリー構成法の開発等があげられる．

謝辞 本研究に関し多くの有益なご討論，ご助言をしていただいている会津大学コンピュータネットワーク学講座の諸氏と貴重なコメントをくださった査読者に感謝いたします．

参考文献

1) Garcia-Molina, H. and Barbara, D.: How to Assign Votes in a Distributed System, *J. ACM*, Vol.32, No.4, pp.841–860 (1985).
2) Kakugawa, H. and Yamashita, M.: Local Coteries and a Distributed Resource Allocation Algorithm, 情報処理学会論文誌，Vol.37, No.8, pp.1487–1496 (1996).
3) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol.21, No.7, pp.558–565 (1978).
4) Maekawa, M.: A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems, *ACM Trans. Computer Systems*, Vol.3, No.2, pp.145–159 (1985).
5) Sung, S.C. and Manabe, Y.: Coterie for Generalized Mutual Exclusion Problem, *Trans. IEICE*, Vol.E82-D, No.5, pp.968–972 (1999).
6) 亀田恒彦，山下雅史：分散アルゴリズム，pp.126–128, 近代科学社 (1994)

(平成13年 6 月11日受付)
(平成13年10月16日採録)

程 子学（正会員）
昭和32年生．平成5年東北大学大学院工学研究科博士課程修了．同年会津大学講師，平成11年同大学助教授．分散アルゴリズム，遠隔教育，心理エージェントの研究開発に従事．IEEE，ACM，IEICE 各会員．

和田 裕（学生会員）
平成9年会津大学コンピュータ理工学部卒業．同年同大学大学院コンピュータ理工学研究科入学．平成11年同研究科博士前期課程修了．コンピュータ理工学修士．現在，同研究科博士後期課程在学中．分散アルゴリズムに関する研究に従事．

橋本 悟
平成9年会津大学コンピュータ理工学部に入学．現在，同大学在学中．分散アルゴリズムに関する研究に従事．

野口 正一（正会員）
昭和29年東北大学工学部電気工学科卒業．昭和34年同大学大学院博士課程了．工学博士．昭和46年東北大学電気通信研究所教授，平成2年東北大学応用情報学研究センター長．平成5年日本大学教授．平成9年会津大学学長．現在，仙台応用情報学研究振興財団理事長．主として，情報システム構成論，知識処理に関する研究に従事．