アルゴリズム 82−10
(2002. 1. 24)

確率的コーラムシステムを用いた資源割り当てアルゴリズム

A Distributed Resource Allocation Algorithm with Probabilistic Quorum Systems

吉村 英明　角川 裕次
Hideaki Yoshimura, Hirotsugu Kakugawa
広島大学大学院 工学研究科 情報工学専攻
Information Engineering, Graduate School of Engineering, Hiroshima University

概 要

近年，コンピュータやワークステーションの高性能化・低価格化に伴って，それらの性能を十分に活用するために，これらを複数台ネットワークで接続して互いに協調して処理を進める分散処理が広がりつつある．こうした分散処理では，性能を十分に活用できる反面，各計算機が正しい処理を行なうためにはさまざまな問題が存在する．資源を複数の計算機が獲得しあう資源の割り当て問題は，そのうちの重要な問題の一つであり，これまでさまざまな研究がなされてきた．

本研究では分散システム内における各計算機間の通信構造として，確率的コーラムシステムを用いた資源割り当てアルゴリズムを提案する．また，シミュレーションによって，提案したアルゴリズムの動作について評価を行なう．

Recently, with highly-efficientizing and low-pricing of a computer or a workstation, the distributed processing which connects these by network and cooperates mutually is popular. In such distributed processing, while a performance is fully utilizable, in order to each computer processes rightly, various problems exist. The resource allocation problem which two or more computers gain resources is one of the important problems of them, and various researches have been studied.

In this paper, we propose a resource allocation algorithm which uses probabilistic quorums system as a structure of information exchange between computers. And, we evaluate about performance of the proposed algorithm by simulation.

1 はじめに

複数の計算機がネットワークによって結合されている分散システムでは，相互排除問題や資源の割り当て問題などさまざまな問題が存在し，それらを解決する手法がいくつか提案されているが，その中にはコータリ (Coterie) を用いたものがいくつか存在する．コータリ [1] とは，もともと分散相互排除問題に対して Garcia-Molina らによって提案されたものであるが，コータリを拡張することで他の様々な問題を解決する手法も提案されている．確率的コーラムシステム (Probabilistic Quorum Systems)[2] とは，そのようなコータリを拡張したものの1つである．本研究では，容量を持ち，その容量を分割して利用することが可能な1つの資源をシステム内の複数の計算機が共有する場合の資源の割り当て問題を，通信構造に確率的コーラムシステムを用いて解決するアルゴリズムを提案し，その動作をシミュレーションを用いて評価する．

1.1 関連研究

分散システムにおける様々な問題を，コータリを用いて解決する研究がこれまでなされてきている．[3] では，コータリ (coterie) を用いて相互排除問題を解決する分散アルゴリズムが提案されている．またいくつかの研究では，様々な問題を解決するためにコータリを拡張したものが用いられている．[4] では，システム内で資源を同時に利用できる計算機の数を k 個以下に制限する k 相互排除問題を，コータリを拡張した (k-coterie) を用いて解決するアルゴリズムが提案されている．また [5] では，コータリの拡張である local coterie を用いて，システム内に存在する複数の資源を相互排他的に割り当てるアルゴリズムが提案されている．さらに，分散システムにおけるオブジェクトの読み書き問題などをコータリを拡張した共コータリを

用いて解決するアルゴリズム [6] が提案されている．一方 [2] では，コータリを拡張したものの 1 つとして，確率的コーラムシステムが提案されている．

2 諸定義

本研究で用いる確率的コーラムシステム，およびコータリについて説明する．

2.1 コータリ

定義 1：コータリ (*Coterie*)[1]
U をシステムを構成するノードの集合とする．以下を満たすグループの集合 S は U のコータリである．

1. $G \in S$ は，$G \neq \phi$ でかつ $G \subseteq U$ である．
2. (Intersection property) もし $G, H \in S$ ならば，G と H は少なくとも一つは共通のノードを持つ．
3. (Minimality) $G \subset H$ を満たすような $G, H \in S$ は存在しない．

このとき，G，H を *quorum*(コーラム) と言う． □

コータリを用いて通信を行なう場合，通信される情報はコーラムを単位として伝達・収集される．このときコーラムどうしは必ず共通部分を持つので，情報を収集するときには，伝達された全ての情報を得ることができる．従ってコータリは，システム全体の正確な情報を必要とするアプリケーションでの通信構造として用いることができる．

2.2 確率的コーラムシステム

定義 2：アクセス戦術 w[2]
U の部分集合の集合である集合システム $\mathcal{Q}$ に対するアクセス戦術 w とは，$\mathcal{Q}$ の各要素 Q へアクセスする確率の分布を表し，w は $\sum_{Q \in \mathcal{Q}} w(Q) = 1$ を満たす． □

定義 3：確率的コーラムシステム [2](以下:PQS)
$\mathcal{Q}$ を集合システム，w をアクセス戦術，ϵ を $0 < \epsilon < 1$ 満たす定数としたとき，もし以下の条件が成立するならば，$\langle \mathcal{Q}, w, \epsilon \rangle$ は PQS であるという．

$$\sum_{Q,Q':(Q \cap Q') \neq \phi} w(Q)w(Q') \geq 1 - \epsilon$$

□

PQS を用いて通信を行なう場合，コーラムどうしは高い確率で共通部分を持つので，通信された情報を確率的に得ることができる．従って PQS は，システム全体の情報を必ずしも必要としないが，情報を通信することで効率を向上できるアプリケーション等での通信構造として用いることができる．

例:PQS の構成法の例 [2]
U を大きさ n の集合とし，ある定数 $l \geq 1$ に対して $\mathcal{Q} = \{Q \subseteq U : |Q| = l\sqrt{n}\}$, $\forall Q \in \mathcal{Q}, w(Q) = \frac{1}{|\mathcal{Q}|}$, $\epsilon = e^{-l^2}$ とするとき，$\langle \mathcal{Q}, w, \epsilon \rangle$ は PQS である． □

本研究では，このようにして定義された PQS を，システム内での各計算機間の通信構造として用いる．

3 モデル

3.1 システムモデル

本研究では，計算機集合 $U = \{p_1, p_2, \cdots, p_n\}$ と容量 M であり，容量を分割して利用できる 1 種類の資源 r が存在する分散システムを考える．このとき，システムのネットワークトポロジは完全グラフであり，各計算機間には双方向の通信リンクが存在する．

各計算機は唯一の識別子を持ち，その処理能力はそれぞれ異なるものであり，仕事の処理に要する時間は予測できないものとするが，発生する仕事は有限時間で終了することが保証されているものとする．また，各計算機は実時間を参照できるような大局的な時計を持たないものとする．

各計算機はメッセージ交換を通じてのみ通信できるものとする．各計算機はメッセージを送信する際に，自身の論理時計を参照し，そのときの時刻をタイムスタンプとしてメッセージに付けて送信する．それぞれ

の計算機は，受信したメッセージを溢れることなく保持できる十分な長さ (有限) のキューを持つものとする．また，送信されたメッセージは必ず有限時間以内に送信先へと届き，各リンクではメッセージは送られた順番で通信される．

システム内の全ての計算機は資源 r へとアクセスすることが可能であり，各計算機では r を必要とする仕事が発生する．資源 r は，容量を分割して複数の計算機に割り当てられることが可能である．また，システム中の各計算機や通信リンクは故障しないものとする．

3.2 分割できる資源の獲得問題

各計算機 $p_i \in U$ は以下のような動作に従うとする．

```
while(true)
  begin
      資源 r の容量を獲得
          < 仕事を実行 >
      資源 r の容量を解放
  end
```

このとき，資源の獲得問題を以下のように定義する．

定義4：資源の獲得問題

システム全体の状態の集合を $\pi = \{c_0, c_1, .., c_i, ..\}$ で表す．このとき，c_i において割り当てられている資源の容量を $V(c_i)$ としたとき，$f_t(c_i)$ を，時刻 t において $V(c_i) > M$ ならば $f_t(c_i) = 0$, $V(c_i) \leq M$ ならば $f_t(c_i) = 1$ となるような関数とし，$P_t(c_i)$ を時刻 t においてシステムが c_i となる確率とすると，資源の割り当て問題は，全ての $t(t = 0, 1, 2, ..)$ に対して以下を保証する．

$$t : \sum_{c_i \in \pi} f_t(c_i) P_t(c_i) \geq X$$

ここで，X は $0 < X < 1$ を満たす定数とする． □

4 資源割り当てアルゴリズム

本研究で提案するアルゴリズムは，以下のような方針に基づいて実装される．資源を利用したい計算機は，あるコーラムに属する全ての計算機に資源が利用可能かどうか問い合わせる．問い合わせられた計算機は，自身の知る資源の利用状況から問い合わせてきた計算機が資源を利用できると判断したならば，資源利用の許可と一緒に自分が知っている資源の利用状況を返信する．このようにして，資源を利用したい計算機は問い合わせた全ての計算機からの許可と資源の利用情報を収集し，資源の獲得が可能かどうかを判断する．(可能であれば獲得し，そうでなければ可能になるまで待つ) また各計算機は，資源を獲得・解放する際には，許可をくれた計算機にそのことを知らせる．さらに，アルゴリズムではデッドロックや飢餓状態を回避するために，資源を要求するメッセージに優先度を持たせる．これは，Lamport の論理時計 [10] を用いて実装され，計算機 p_i が資源を要求する際の論理時計を t とすると，p_i はメッセージに，タイムスタンプ t と識別子 i をつけて送信する．こうすることで，資源を要求するメッセージに unique な優先度を持たせ，ある計算機が他の計算機に許可を与えていた場合に，より優先度の高い要求が送られてきたならば，その許可を優先度の高い方へと譲渡することでデッドロックを回避できる．

また各計算機 $p_i(1 \leq i \leq n) \in U$ は，各計算機からの資源を要求するメッセージを，自身が保持する待ち行列 $QUEUE_i$ に，優先度の高い順に格納する．(もし，時刻印が全く等しいものが存在したならば，各計算機の優先度の高いもの (識別子によって決定される) がキューの先頭に近い方へ格納される．)

その他に p_i は，以下の局所変数を持つ．

- $R_i[j : 1..n]$ … 計算機 p_i が知っている範囲での，各計算機 p_j の資源r の利用状況を表す配列 (use_i は，その総量を表す変数)
- $limit$ … 資源の容量 M
- $task_i$ … 計算機 p_i で発生した仕事が必要とする資源の容量
- $stamp_i$ … 計算機 p_i が持つ論理時計が表す時刻印

- $allow_i$ … 計算機 p_i が資源の利用を許可している計算機を表す変数 (初期値は 0)
- $allowstamp_i$ … $allow_i$ の時刻印を表す変数 (初期値は 0)
- $hold_i[j : 1..n]$ … 計算機 p_i が計算機 p_j から受信した $R_j[k : 1..n]$ を一時的に保管するための配列 ($holdall_i$ は，その総計量を表す変数)
- $count_i$ … 計算機 p_i が他の計算機から資源利用の許可を受信した回数を表す変数
- top_i … 計算機 p_i のキューの先頭に入っているメッセージを送信した計算機の識別子を表す変数

4.1 本アルゴリズムで用いるメッセージ

本アルゴリズムでは，各計算機 $p_i(1 \leq i \leq n) \in U$ は以下のメッセージを用いる．

- $REQUEST$ … 各計算機が資源の許可を問い合わせる際に用いるメッセージ．
- $ALLOW$ … $REQUEST$ を受信した計算機が，それに対する返事として，資源の利用を許可する際に用いるメッセージ
- $ACQUIRE$ … 資源を獲得した計算機が，そのことを知らせる際に用いるメッセージ
- $RELEASE$ … 資源を解放した計算機が，そのことを知らせる際に用いるメッセージ
- $REVOKE$ … 既に $ALLOW$ を送信した計算機に，許可を取り消してもらうことを要求する際に用いるメッセージ
- $RETURN$ … $REVOKE$ を受信した計算機が，それに応じる際に用いるメッセージ
- $CHANGE$ … 許可を与えている計算機が，他の計算機に許可を与えている間に自身が知っている資源の利用状況が変わったことを知らせる際に用いるメッセージ

4.2 アルゴリズム

Q を $U = \{p_1, p_2, \cdots, p_n\}$ に対する任意の PQS とする．

- [p_i で資源r を必要とする仕事が発生]

  p_i は，Q から任意のコーラム Q をランダムに選び，Q に含まれる全ての計算機に $< REQUEST, task_i, stamp_i, i >$ を送信する．

- [p_j が $< REQUEST, task_i, stamp_i, i >$ を受信]

  - $QUEUE_j$ が空であれば，$< REQUEST, task_i, stamp_i, i >$ を $QUEUE_j$ に保持し，このとき，$use_j + task_i \leq limit$ であれば，$allow_j$ を i に，$allowstamp_j$ を $stamp_i$ にセットし，$< ALLOW, R_j[1..n], stamp_j, j >$ を p_i へ返す．
  - $QUEUE_j$ が空でないとき，$< REQUEST, task_i, stamp_i, i >$ を $QUEUE_j$ に保持し，このとき，$< REQUEST, task_i, stamp_i, i >$ よりも優先度の高いメッセージが，$QUEUE_j$ に存在しないならば p_j は $allow_j$ が示す計算機へ $< REVOKE, stamp_j, j >$ を送信する．

- [p_i が $< ALLOW, R_j[1..n], stamp_j, j >$ を受信]

  p_i は $count_i$ の値を 1 つ増やし，$hold_i[j : 1..n]$ を更新する．また，$count_i$ がコーラムの大きさ $|Q|$ に達し，$holdall_i + task_i \leq limit$ ならば，資源r を獲得することを伝えるため，Q に含まれる全ての計算機に $< ACQUIRE, task_i, stamp_i, i >$ を送信し，$R_i[i]$ に $task_i$ を追加して $count_i$ を 0 にセットする．

- [p_i が $< REVOKE, stamp_j, j >$ を受信]

  p_i がまだ資源を獲得していないならば，$count_i$ の値を 1 つ減らし，p_j へと $< RETURN, stamp_i, i >$ を送信して，再び p_j から $ALLOW$ が送信されるのを待つ．(既に資源を獲得した場合には，$RETURN$ を送らない．)

- [p_j が $< RETURN, stamp_i, i >$ を受信]

  p_j は $allow_j$，$allowstamp_j$ を 0 にセットする．

- [p_k が $< ACQUIRE, task_i, stamp_i, i >$ を受信]

  p_k は，$R_k[i]$ に $task_i$ を追加して，$< REQUEST, task_{allow_k}, allowstamp_k, allow_k >$ を $QUEUE_k$ から除き，$allow_k$,$allowstamp_k$ を 0 にセットする．

- [p_j の $allow_j$ が 0 にセットされる]

  もし p_j の待ち行列 $QUEUE_j$ にメッセージ $< REQUEST, task_{top_j}, stamp_{top_j}, top_j >$ が存在し，$use_j + task_{top_j} \leq limit$ であれば，$allow_j$ を top_j,$allowstamp_j$ を $stamp_{top_j}$ にセットして，計算機 p_{top_j} にメッセージ $< ALLOW, R_j[1..n], stamp_j, j >$ を送信する．

- [p_i で仕事が終了]

  仕事が終了し，資源**r** を解放する計算機は，Q に含まれる全ての計算機に $< RELEASE, task_i, stamp_i, i >$ を送信し，$R_i[i]$ から $task_i$ を引く．

- [p_k が $< RELEASE, task_i, stamp_i, i >$ を受信]

  p_k は，$R_k[i]$ から $task_i$ を引く．このとき，既に他の計算機へと $ALLOW$ を送信していたならば，$allow_k$ へとメッセージ $< CHANGE, R_k[i], i, stamp_k, k >$ を送信する．$ALLOW$ を送信してなく，$QUEUE_k$ にメッセージが存在し，use_k が減少したことで $use_k + task_{top_k} \leq limit$ となれば，$allow_k$ を top_k に,$allowstamp_k$ を $stamp_{top_k}$ にセットして，p_{top_k} に $< ALLOW, R_k[1..n], stamp_k, k >$ を送信する．

- [p_i が，$< CHANGE, R_j[k], k, stamp_j, j >$ を受信]

  p_i は，$hold_i[k]$ と $R_j[k]$ が異なるならば，$hold_i[k]$ を更新する．また，更新したことで $holdall_i + task_i \leq limit$ となれば，資源**r** を獲得することを伝えるために，Q に含まれる全ての計算機に $< ACQUIRE, task_i, stamp_i, i >$ を送信し，$R_i[i]$ に $task_i$ を追加して，$count_i$ を 0 にセットする．

4.3 アルゴリズムの正当性

補題 1: 本アルゴリズムを，コータリを用いて実装したとき，資源**r** は容量 M を越えずに割り当てられる．

証明:資源を要求する計算機はあるコーラム Q を選択し，Q に含まれる全ての計算機から資源を利用する許可を得た場合にのみ，資源を獲得することができる．このとき，許可を与える計算機 ($\in Q$) は同時に複数の計算機へと許可を与えることができないので，コータリを用いたとき，選択したコーラムに含まれる全ての計算機から許可を得る計算機は，システム上にたかだか 1 つである．

また，許可を得る計算機は，資源を利用している計算機が選択したコーラム Q' と共通部分を持つので，資源を利用している全ての計算機の利用状況を獲得することができる．資源を獲得する計算機は，収集した利用状況から資源が獲得できるかどうかを判断するので，コータリを用いたとき，全てのコーラムどうしは共通部分を持ち，資源は容量を越えずに利用されることが保証される． □

補題 2: システム上で構成された PQS において，任意に選ばれた 2 つのコーラムが共通部分を持つ確率を p とし，各計算機が資源の容量の $1/m$ を要求するとき，本アルゴリズムを PQS を用いて実装したときに，資源**r** が容量 M を越えずに割り当てられる確率 X は，$X > p^{m(n-m)}$ となる．

証明:本アルゴリズムを実装した際，システム上の各計算機は資源を獲得したり，解放したりして資源を利用するが，PQS を用いた場合，資源が容量を越えて割り当てられる状況が発生する．このような場合に計算機が資源を解放するとき，割り当てられている資源の容量は減少するので，X が増加する可能性がある．そこで，資源の解放を考えずに，全ての計算機が一斉に資源を要求するような場合のみを考えることで X の最悪の確率を求めることができる．そのときの X は以下のようになる．

$$
\begin{aligned}
X &= A_1 \times A_2 \times \ldots \times A_n \\
A_i &= 1 && (i \leq m) \\
A_i &= 1 - (1 - p^m) \times (1 - p)^{i-m-1} && (m < i \leq n)
\end{aligned}
$$

このとき $p^m = A_{m+1}, p^m < A_i (m+2 < i \leq n)$ であるので，上記より $X = A_{m+1} \times A_{m+2} \times \ldots \times A_n > p^m \times p^m \times \ldots \times p^m = p^{m(n-m)}$ となる．したがって，補題 2 が成立する． □

4.4 deadlock-free

補題 3: 本アルゴリズムが，deadlock-free であることを証明する．

証明:deadlock が生じると仮定する．

今回提案したアルゴリズムでは，deadlock は複数の計算機 (例えば，計算機 a と b) が資源を利用する際に問い合わせた共通の計算機 (例えば，計算機 c と d) が，それぞれ異なる計算機へと許可を与える場合に生じる．資源を利用したい計算機が送信する $REQUEST$ には，そのときの時刻印と識別子が付けられているので，各計算機の $REQUEST$ には優先関係が必ず存在する．また，送信されたメッセージは，有限時間内に必ず送信先へと到着することが保証されている．すなわち，問い合わせられた計算機には，有限時間内に必ず送信された全ての $REQUEST$ が到着する．このとき，それぞれの $REQUEST$ には優先関係が存在するので，優先度の低い要求へと許可を与えていた計算機は，$REVOKE$ を用いることで許可を取り消し，新しく優先度の高い計算機へと許可を与える．よって優先度の高い計算機は，全ての許可を得て，資源を獲得できる．したがって，deadlock は生じない． □

4.5 starvation-free

補題 4: 本アルゴリズムが，starvation-free であることを証明する．

証明:starvation に陥る計算機 x が存在すると仮定する．

x は任意にコーラム Q を選び，Q に含まれる全ての計算機 $y \in Q$ にメッセージ $REQUEST$ を送信する．各計算機間の通信リンクは故障しないので，x のメッセージは有限時間内に y によって受信されるが，このとき Lamport の論理時計より，y のタイムスタンプは x のものよりも大きなものとなる．その後 y は，自身のキューの先頭に存在する (優先度の最も高い) メッセージを送ってきた計算機に資源の利用を許可するが，その際に許可を与えられた計算機 z は，y からメッセージ $ALLOW$ を受信するので，z のタイムスタンプは y のものよりも大きくなる．したがって，y において x よりも優先度の高い要求を送っていた計算機は，資源を利用した時点で x よりも大きなタイムスタンプを持つ．これを繰り返すと，x 以外の全ての計算機が x よりも大きなタイムスタンプを持つようになり，いつかは y のキューの先頭に x のメッセージが保持される．全ての $y \in Q$ のキューの先頭に x のメッセージが保持されたとき，x は資源を利用できるので starvation には陥らない．よって矛盾するので，starvation に陥る計算機は存在しない． □

4.6 Message Complexity

補題 5: 本アルゴリズムにおいて，各計算機が資源を要求した際に必要とされる通信量について分析する．ここで，計算機集合の大きさを n，コーラムの大きさを k とする．

<最良の場合>: ある計算機 p の資源の要求 (1回) に対して必要とされる通信量は，$REQUEST$ の送信や $ALLOW$ の受信，$ACQUIRE$ や $RELEASE$ の送信にそれぞれ k の通信量が必要なので $4k$ となる．今回用いた PQS のコーラムサイズは $l\sqrt{n}$ であるので，Message Complexity は $O(\sqrt{n})$ となる． □

<最悪の場合>: 最良の場合の通信量の他に，p の要求よりも優先度の低い要求への許可を取り消すための $REVOKE$，$REVOKE$ を受信したことを知らせる $RETURN$，許可を取り消した計算機に再度許可を与える $ALLOW$，さらに，計算機 p の仕事が終了したことを知らされた計算機が，許可を与えている計算機へ自身の資源についての情報が変化したことを知らせる $CHANGE$ が発生し，それぞれ最大でコーラムの大きさ ($= k$) の通信量が必要とされるので，$4k$ の通信量が追加される．したがって最悪の場合に資源の要求 (1回) に対して必要とされる通信量は，$4k+4k=8k$ となり Message Complexity は $O(\sqrt{n})$ となる． □

5 シミュレーションによる評価

本研究では，提案したアルゴリズムについてシミュレータを用いて評価を行なった．シミュレーションは，分散システム内の複数の計算機が資源を同時に獲得しようとした場合に割り当てられる資源の容量の期待値を算出するものとし，以下の値をパラメータとして評価を行なう．

- 分散システムを構成する計算機の数
- コーラムに含まれる計算機の数 (コーラムサイズ)
- 資源を獲得しようとする計算機の数
- 計算機が要求する資源の容量

5.1 実行結果

図 1，図 2 は，資源の容量が 100 のときの以下の条件におけるシミュレーション結果を表したグラフである．

- システムを構成する計算機数：100
- コーラムサイズ：5 〜 30
- 資源を獲得しようとする計算機の数：2 〜 10
- 各計算機が要求する資源の容量：100

図 1 は，割り当てられた資源の容量についてまとめたグラフで，各計算機が要求する資源の容量が 100 のときのグラフである．グラフでは縦軸が割り当てられた資源の容量，横軸がコーラムサイズとなっていて，グラフ上の各線は，それぞれが資源を獲得しようとする計算機の数を示している．

このとき，各計算機間の通信構造としてコータリを用いた場合，グラフは M(今回は 100) を越えないことに注意されたい．

また，図 2 は，割り当てられた資源の容量とそのときの確率との関係を示したグラフ (グラフは，資源を同時に獲得しようとする計算機の数 $= 5$ のとき) であり，各計算機が要求する資源の容量が 100 のときのグラフである．グラフでは縦軸が確率，横軸が割り当てられた容量となっていて，グラフ上の各線は，それぞれがコーラムサイズを示している．

このとき，各計算機間の通信構造としてコータリを用いた場合，資源が M(今回は 100) を越えて割り当てられる確率は存在しない．

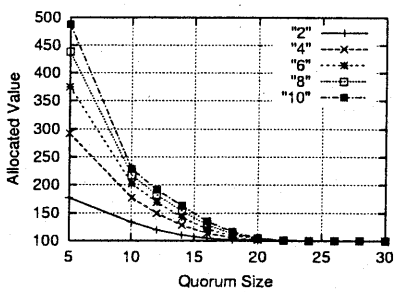

図 1: 割り当てられる資源の容量の期待値について

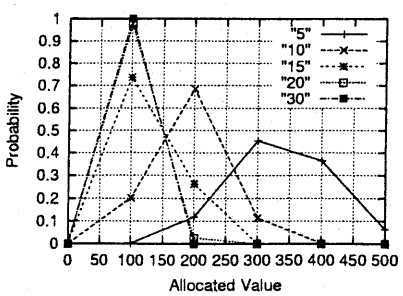

図 2: 各容量毎の確率の分布 (同時に要求する計算機の数： 5)

5.2 考察

シミュレーションでは，複数の計算機が同時に資源を獲得しあう場合について評価した．グラフからコーラムサイズが大きい程，割り当てられる資源の容量は少なく抑えられることが明らかである．

また，計算機が要求する資源の容量を少なくすると，資源が容量 M を越えずに割り当てられる確率は低くなり，コーラムサイズを大きくしても，割り当てられる資源の容量は多少大きくなる．これは容量 M を越えることなく資源を割り当てるためには，資源を利用している計算機全ての利用状況を把握することが必要であり，計算機が要求する資源の容量を少なくすると，容量 M を越えることなく資源を利用できる計算機の数が増加し，確率的コーラムシステムを用いた場合には，そのような確率が比較的小さくなってしまうためであると思われる．

また，今回行なったシミュレーションにおいて，許可を与える計算機が資源の容量 M をあらかじめ少なく見積もっておくことで，計算機が資源の容量を越えて利用する状況を減らすことができると予想される．

おわりに

本研究では，分散システムにおいて，複数の計算機が単一の分割できる資源を利用するような問題を PQS を用いて解決するアルゴリズムを提案し，複数の計算機が同時に資源を要求する場合に資源が割り当てられる容量についてシミュレーションを行って評価した．

今後の課題としては，確率的コーラムシステムを用いた資源の割り当てが保証する確率について詳しく解析を行なうことを考えている．また，許可を与える計算機が資源の容量をあらかじめ少なく見積もった場合について，シミュレーションを用いて評価を行なうことも考えている．

参考文献

[1] H Garcia-Molina,D Barbara: "How to Assign Vote in a Distributed System", in Journal of the Association for Computing Machinary, Vol. 32, No. 4, pp. 841–860,(1985).

[2] Malkhi, D., Reiter, M., Wright, R.: "Probabilistic Quorum Systems", in *Proceedings of the 16th International Conference on Distributed Computing Systems*, (1997).

[3] M. Maekawa: "A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems", in ACM Trans. Comput.Syst., pp. 145–159,(1985).

[4] Hirotsugu Kakugawa, Satoshi Fujita, Masafumi Yamashita, Tadashi Ae: "A distributed k-mutual exclusion algorithm using k-coterie", in Information Processing Letters 49, (1994).

[5] Hirotsugu Kakugawa, Masafumi Yamashita: "Local Coteries and a Distributed Resource Allocation Algorithm", in Transaction of Information Processing Society of Japan, Vol. 37, No. 8, pp. 1487–1495(1996).

[6] 芦原 評, 清水 謙多郎: "分散システムにおける読み書き問題に対する拡張されたコテリーの構成とその応用", 情報処理学会論文誌, Vol. 38, No. 2, pp. 167–179 (1997).

[7] Kerry Raymond: "A DISTRIBUTED ALGORITHMS FOR MULTIPLE ENTRIES TO A CRITICAL SECTION", Information Processing Letters, Vol. 30, pp. 189–193,(1989).

[8] Pradip K. Srimani, Rachamallu L.N. Reddy: "Another distributed algorithm for multiple entries to a critial section", Information Processing Letters, Vol. 41, pp. 51–57,(1992).

[9] Divyakant Agrawal: "Analysis of Quorum-Based Protocols for Distributed (k+1)-Exclusion", IEEE Transaction on Parallel and Distributed Systems, Vol. 8, No. 5, pp. 533–537,(1997).

[10] L Lamport: "Times,Clocks,and the Ordering of Events in a Distributed Systems", Communications of the ACM, 21(7),pp. 558–565,(1978).