確率的コーラムシステムに基づく負荷分散アルゴリズムとその実験的評価

吉村英明[†] 角川裕次[†] 阿江忠[†]

近年ではパーソナルコンピュータやワークステーションの価格が下がり，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．分散処理の中でも負荷分散は重要な問題の 1 つであり，これまでさまざまな研究がなされてきた．本論文では確率的コーラムシステムを負荷情報交換のためのネットワークの構造として用いた負荷分散アルゴリズムを提案し，シミュレーションによって従来の手法との比較を行った結果について報告する．

A Load Balancing Algorithm Based on Probabilistic Quorum System and Its Experimental Evaluation

Hideaki Yoshimura,[†] Hirotsugu Kakugawa[†] and Tadashi Ae[†]

Recently, distributed processing is becoming popular because powerful personal computers, workstations and network equipments are widely available. One of important problems in distributed processing is load balancing, and many algorithms and systems have been proposed. In this paper, we propose a new load balancing algorithm based on probabilistic quorum system as a communication structure in a network. We also evaluate the proposed algorithm by simulation and compare it with other load balancing algorithms proposed so far.

1. はじめに

近年，コンピュータ技術の発展にともない我々の身の回りのコンピュータの形態も分散処理へと変化をとげており，そうした変化が処理の高速化や過去にはできなかったような処理の実現を可能にした．しかしながら，このような分散処理の優れた処理能力を十分に活用するには，それに適した資源の管理が求められる．

そこでこのような分散システムでは，その処理能力を有効に利用するための分散処理が考慮されてきた．そのうちの重要な問題の 1 つである負荷分散は，分散システム内のコンピュータに負荷（仕事）を分散させて処理させることによって，システムの処理効率を向上させるものである．

このような負荷分散のアルゴリズムはいくつか提案されているが，システムの状況に左右されたり，場合によってはあまり効力を発揮できなかったりする．また，分散システムにおける負荷分散では各計算機の負荷状態を把握することが重要であり，既存の負荷分散アルゴリズムではシステムの大きさに比例して，通信量・扱うデータ量が多大なものになり，通信遅延による影響も無視できないものになる．

そこで本論文ではシステムの状態に影響を受けない安定した，比較的通信量の少ない負荷分散アルゴリズムを提案し，その効果をシミュレーションを行って評価した．

1.1 既存の負荷分散アルゴリズム

以下に既存の負荷分散アルゴリズムのうちのいくつかについて簡単に説明する．

Eager らによって提案された送信機始動のアルゴリズム（*Sender-Initiated Algorithm*）[1] では，新しく仕事が発生した計算機は，（仕事が発生したことによって）自身の負荷がシステム内において比較的重いものになれば，他の計算機と通信し比較的負荷の軽い計算機を探す．このとき負荷の軽い計算機を見つけることができれば，その計算機に新しく発生した仕事を依頼する．この作業は適当な計算機が見つかるまで続くが，システム内の他のすべての計算機と通信しても適当な計算機が見つからなければ，新しく発生した仕事は最初に仕事が発生した計算機で処理される．

Shivaratri らによって提案された受信機始動のアル

† 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

ゴリズム（*Receiver-Initiated Algorithm*）[2]では，実行中の仕事が終了した計算機は仕事が終了したことによって自身の負荷が比較的軽いものになれば，他の計算機と通信し比較的負荷の重い計算機を探す．このとき負荷の重い計算機を見つけることができたならば，その計算機が実行中の仕事のうち移動させる適当な仕事を探し，その仕事を自分へと移動させる．また，負荷の重い計算機が見つからなかったり，負荷の重い計算機内に適当な仕事が見つからなかったりした場合は別の計算機と通信し，再び負荷の重い計算機を見つけようとする．システム内に適当な計算機を見つけることができなかったならば，仕事が終了した計算機は残りの実行中の仕事を実行する．

Krueger らが提案した対称始動のアルゴリズム（*Symmetrically Initiated Algorithm*）[3]は，負荷の軽い計算機（Receiver）は負荷の重い計算機（Sender）を，負荷の重い計算機（Sender）は負荷の軽い計算機（Receiver）をそれぞれ見つけようとするアルゴリズムである．よってこのアルゴリズムは，送信機始動のアルゴリズムと受信機始動のアルゴリズムの両方の長所・短所を持ったアルゴリズムといえる．

また，Shivaratri らによって提案された，安定した対称始動のアルゴリズム（*A Stable Symmetrically Initiated Algorithm*）[2]では，各計算機を負荷の重いもの（Sender），負荷の軽いもの（Receiver），負荷が適当なもの（OK）の 3 つに分類して負荷分散を行う．計算機が自分を Sender と見なしたときには送信機始動のアルゴリズムの要素が作用し，計算機が自分を Receiver と見なせば，受信機始動のアルゴリズムの要素が作用する．よって，それぞれの状況に適したアルゴリズムが実行されることになるので，より効果的なアルゴリズムであると考えられる．

以上に述べたような既存の負荷分散アルゴリズムでは，その効力がシステムの状態に左右されがちである．たとえば，送信機始動のアルゴリズムではシステム全体の負荷が比較的大きいときには，新しく仕事が発生した計算機が負荷の軽い計算機を見つけることが困難であるためにあまり効力を発揮できない恐れがあり，逆に受信機始動のアルゴリズムではシステム全体の負荷が比較的小さいときには，負荷の重い計算機を見つけることは困難であるため，効力を十分に発揮できない恐れがある．また受信機始動のアルゴリズムでは実行中の仕事を移送するので，その実装は難しい．

さらに，このような負荷分散アルゴリズムでは各計算機と通信して負荷情報を把握することが重要であり，通信量をできるだけ抑えることや通信遅延を考慮しなくてはならない．また，システム全体としての負荷状態を集中的に管理しているために，耐故障性などの問題点が考えられる．

1.2 本研究の動機

ところで，分散システムにおいて，名前の割当て問題や相互排除問題などを解決するのにコータリ[6]（*Coterie*）を用いることがある．コータリを用いた例として，コータリを用いた相互排除アルゴリズム[4]やコータリを拡張した構造を用いた資源割当てアルゴリズム[10]，さらに分散システム上での大局的なデータの管理を行うための通信構造[8]などが提案されている．そこで分散システムでの通信構造にコータリを用いて，効率の良い負荷分散法を提案できるのではないかと考えた．

今回新たに提案するアルゴリズムでは，コータリの拡張である確率的コーラムシステム（Probabilistic Quorum System[5]：以下 PQS と書く）という構造を用いて負荷情報を通信することで負荷分散を行う．この PQS を負荷情報の通信に用いることで，通信量を抑えることが可能である．

1.3 本論文の構成

以下，2 章では本研究で用いる確率的コーラムシステム（PQS）の構造について，また 3 章では PQS を用いた情報通信と PQS を用いた負荷分散アルゴリズムについて説明する．4 章では PQS を用いて行ったシミュレーション実験の概要や実験結果について述べる．最後に 5 章で本論文で新たに提案した負荷分散アルゴリズムの特徴と今後の課題について述べる．

2. 諸 定 義

本研究で用いた確率的コーラムシステムはコータリの拡張であるので，まずコータリについて説明する．

定義 1（コータリ[6]）

U を大きさ n の集合とする．以下の 2 つの条件を満たす空でない U の部分集合 Q_i の集合 $\{Q_1, Q_2, \ldots, Q_m\}$ を**コータリ**（*Coterie*）と呼ぶ．

Intersection property

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

Minimality

すべての i と j $(1 \leq i, j \leq m, i \neq j)$ に対して，$Q_i \not\subseteq Q_j$．このとき，各 Q_i をコーラム（*quorum*）と呼ぶ． □

定義 2（確率的コーラムシステム[5]）

U を大きさ n の集合，$\mathcal{Q}$ を U の部分集合の集合としたとき，$\mathcal{Q}$ へのアクセス戦術 w は $\mathcal{Q}$ の構成要

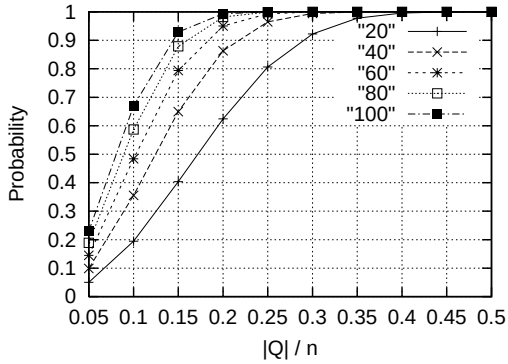

図 1 PQS のコーラムどうしが共通部分を持つ確率
Fig. 1 A probability that two quorums have non-empty intersection.

素への確率の分布を表し，

$$\sum_{Q \in \mathcal{Q}} w(Q) = 1$$

を満たす．

ここである定数 ϵ $(0 < \epsilon < 1)$ に対して，

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

が成立するとき $\langle \mathcal{Q}, w, \epsilon \rangle$ を**確率的コーラムシステム**（*Probabilistic Quorum System*）という． □

PQS の構成法の例[5)]

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． □

上記の PQS においてランダムに選ばれた 2 つのコーラムの共通部分の大きさの期待値は l^2 であり，l の値を適当に選ぶことによって，ランダムに選ばれた 2 つのコーラムが共通部分を持たない確率を減らせることができる[5)]．本論文ではこの方法で構成された PQS を用いることとする．

図 1 はシステム全体 U の大きさ $n = 20$，40，60，80，100 それぞれに対して，コーラムの大きさとコーラムどうしが共通部分を持つ確率を示したグラフである．このグラフでは，横軸がシステム全体の大きさに対するコーラムの大きさの割合 $|Q|/n$ を，縦軸が一様かつランダムに選ばれた 2 つのコーラムが共通部分を持つ確率を表している．

図 1 からシステム全体の大きさ n が大きいほど，コーラムの大きさを比較的小さく設定しても，ランダムに選ばれたコーラムどうしはある程度高い確率で共通部分を持つことが分かる．

3. PQS による負荷分散アルゴリズム

PQS : $\mathcal{Q} = \{Q_1, Q_2, \cdots\}$ を用いて負荷情報を通信する手順の概要を以下に述べる．

- 各計算機 p_i は，自身の負荷に変化が生じるたびに $Q_1, Q_2, \cdots$ の中からランダムにあるコーラム Q_i を選び，Q_i に含まれるすべての計算機に自身の負荷情報を送信する．
- 他の計算機群の負荷情報を知りたい計算機 p_j は，あるコーラム Q_j を選び，Q_j に含まれるすべての計算機からシステム全体の負荷情報を受信する．
- このとき Q_i と Q_j が共通部分を持つならば，$Q_i \cap Q_j$ の計算機を経由して p_i の情報が p_j に届く．

ここで，ランダムに選ばれた 2 つのコーラムは高い確率で共通部分を持つので，どの計算機も高い確率で大局的な負荷情報を知ることができる．PQS を用いたものはコーラムの大きさを変更することで通信量を変更することができる．このときコーラムの大きさを小さくすることで通信量を少なくすることが可能だが，その場合にはランダムに選ばれた 2 つのコーラムが共通部分を持つ確率も減少する．つまり入手できるシステム全体の負荷情報の正確さが損なわれ，古い負荷情報を入手する確率が増える．本研究では，システム全体の正確な負荷状況を入手するために大きなコストを払うのではなく，情報の正確さを多少犠牲にすることで，コストを下げながら十分な範囲内での負荷分散が可能となるのではないかと考えた．

3.1 アルゴリズム

ここで PQS を用いて負荷情報の通信を行い，負荷分散を行った場合の負荷分散アルゴリズムを以下に示す．ただし，ここで各計算機は複数のプログラム（仕事）を同時に処理できるものとする．その際，プログラムは並行に実行されているため処理時間は同一の計算機で処理されているプログラムの数に反比例する．なお，実行中のプロセスの移送（*migration*）は，本アルゴリズムでは取り扱わないこととする．

各計算機 i $(1 \leq i \leq n)$ は，以下の局所変数を持つ．

- $load_i$：計算機 i の負荷の値を表したもの．
- $info_i[1 \ldots n]$：計算機 i が知っている範囲での各計算機の負荷情報．
- $stamp_i[1 \ldots n]$：$info_i[1 \ldots n]$ の情報が発生した時点の時刻印．なお，本論文では時刻印として実時間を用いることとする．
- t_i：現在の時刻を表す．

また，各計算機が送信・受信するメッセージは以下の 4 種類である．

- *QUERY*：計算機が他の計算機にシステムの負荷情報を問い合わせる際に，他の計算機へと送信するメッセージ．
- *REPLY*：*QUERY* に対する応答として，計算機が自分の知っている負荷情報を他の計算機に伝える際に送信されるメッセージ．
- *JOB*：仕事が発生した計算機が他の計算機へと仕事を依頼する際に送信されるメッセージ．
- *INFO*：計算機が自身の負荷情報に変化が生じたとき，新しい負荷情報を他の計算機に伝える際に送信されるメッセージ．

PQS による負荷分散アルゴリズム

計算機集合を U，$\mathcal{Q}$ を任意の PQS とする．
各計算機は，以下の一時変数を用いる．

- m：負荷情報の最小値．
- p：負荷情報の最小値を持つ計算機．
- R：*REPLY* が未到着の計算機の集合．

計算機 a で仕事が発生

あるコーラム Q に属する各計算機 x に，負荷情報を問い合わせる：

ある $Q \in C$ をランダムに選ぶ；
各 $x \in Q$ に対して
  send $\langle QUERY \rangle$ **to** x;
$R = Q$;

計算機 x が，$\langle QUERY \rangle$ を計算機 a より受信

自分が知っている範囲内でのシステム全体の負荷情報を計算機 a に送信する：

$info_x[x] = load_x$;
$stamp_x = t_x$;
send $\langle REPLY, info_x, stamp_x \rangle$ **to** a;

計算機 a が，$\langle REPLY, info, stamp \rangle$ を計算機 x より受信

受信した情報をシステム全体の負荷情報としてまとめる：

各 $y \in U$ に対して
if $(stamp_a[y] < stamp_x[y])\{$
  $info_a[y] = info_x[y]$;
  $stamp_a[y] = stamp_x[y]$;
$\}$
$R = R - \{x\}$;
if $(R = \phi)\{$
  /*すべての *REPLY* を受信*/
  $info_a[1], info_a[2], \cdots, info_a[n]$ の中で
  最小の負荷 m を持つ計算機を p と置く；
  send $\langle JOB \rangle$ **to** p;
$\}$

計算機 y が，計算機 a から $\langle JOB \rangle$ を受信

依頼された仕事を実行する．これにともない，計算機 y は新たな負荷の値をあるコーラム Q に送信する：

ジョブの実行を開始する；
ある $Q \in C$ をランダムに選ぶ；
各 $x \in Q$ に対して
  send $\langle INFO, load_y, t_y \rangle$ **to** x;

計算機 y で実行中の仕事が終了

計算機 y は，新たな負荷の値をあるコーラム Q に送信する：

ある $Q \in C$ をランダムに選ぶ；
各 $x \in Q$ に対して
  send $\langle INFO, load_y, t_y \rangle$ **to** x;

計算機 x が，$\langle INFO, l, t \rangle$ を計算機 y から受信

受信した負荷情報を保持する：

$info_x[y] = l$;
$stamp_x[y] = t$;

3.2 メッセージ複雑度

n をシステム内の計算機集合の大きさ，l をある定数とすると，コーラムの大きさは $l\sqrt{n}$ である．1 つの仕事が発生すると，依頼先の計算機を見つけるまでに負荷状態の問合せ（*QUERY*）とそれに対する返答（*REPLY*），仕事の譲渡に関する通信（*JOB*）の 3 ステップの通信が行われ，メッセージ数は $2l\sqrt{n}+1$ となる．さらに，仕事が発生あるいは終了した計算機の負荷状態の変化の通信（*INFO*）に $l\sqrt{n}$ の通信量が必要とされる．これに対して，仕事が発生した際に他のすべての計算機と通信した場合には，負荷状態の問合せとその返答に $2(n-1)$ の通信量が必要となる．これらの結果から定数 l を固定した場合，n が大きいほど PQS を用いたものが有効であると思われる．

4. シミュレーション実験と考察

4.1 シミュレーションの設定条件

まずシミュレーションの設定条件について説明する．今回は，以下の仮定の下にシミュレーションを行った．

- 各計算機の初期状態は負荷がまったくないものとする．
- 各計算機は故障しないものとする．
- 各計算機間の通信路は故障しないものとする．
- 各計算機の処理能力はまったく同じものとする．
- 仕事はシステム内のすべての計算機上でランダムに発生するものとする．
- 各計算機間の通信は 1 対 1 とする．
- 各計算機間の通信遅延時間はすべて等しいものとする（ここで通信遅延時間とは，ある計算機から

メッセージが送信されてから受け手に受信されるまでの時間である）．

- 実行中の仕事の処理時間は，同じ計算機で実行している仕事の数に反比例するものとする．

4.2 シミュレーションの評価基準

次に，今回のシミュレーションの実行結果を評価するために用いた，5 つの評価基準について述べる．

＜負荷の最大値＞ シミュレーションを実行した際の各計算機の負荷の最大値を検出した値を，システム全体の平均値として計算したもの．

＜負荷の平均値＞ シミュレーションを実行した際の各計算機の単位時間あたりの負荷の値を検出した値．各計算機がシミュレーションの間に平均してどの程度の負荷を持っているかを考慮した．

＜負荷の差の最大値＞ 発生した仕事が依頼先に分散されるたびに各計算機の負荷を調べて，負荷の最も大きかったものと最も小さかったものとの差を測り，その最大値を検出した値．システム内で最も負荷分散がうまくいっているときの状態は，各計算機の負荷がすべて同一であるような状態と考えられる．そこで，仕事が分散された時点での各計算機の負荷の差を調べることで，各アルゴリズムの負荷分散能力を検出しようとした．

＜負荷の差の平均値＞ 上記と同様に，発生した仕事が依頼先に分散されたときの各計算機の負荷の差を調べて，仕事が分散されるごとの平均値を検出した値．＜負荷の差の最大値＞と同様に，各アルゴリズムの負荷分散能力を検出しようとした．

＜仕事の処理効率＞ 発生した仕事すべてに対して，最初に設定されていた処理時間とシミュレーションで実際に処理された時間との比率を検出した値．発生する仕事には，あらかじめ処理時間が設定されているが，実際に処理された時間とは異なる場合が多い．この比率を検出することで，仕事がどの程度の速さ（割合）で処理できたかを考慮して，各アルゴリズムの処理能力を比較しようとした．

4.3 実験環境

今回の実験には，以下のような環境を持った計算機 1 台を用いて行った．

- OS：Linux カーネル 2.2.16
- CPU：Dual Pentium III 800 MHz
- メモリ：512 MB
- 言語：C 言語
- コンパイラ：gcc 2.95.2
- シミュレーションプログラム：専用の離散事象シミュレータを C 言語にて開発した．

4.4 PQS に関する予備実験

4.4.1 実験の概要

先にも述べたように，PQS を用いたものはその部分集合の大きさによって通信量に差が生じる．この通信量の差によって，通信遅延の影響やデータ量，さらには負荷情報の収集能力にも違いが生じると思われる．そこで，PQS を用いたものではシステム全体のサイズに対してコーラムの大きさをどの程度に設定すればよいのかを検討するために，4.1 節で述べた条件を固定したものに対し，PQS の部分集合の大きさのみを変更させてそれぞれ比較させてみた．このような実験をシステムのサイズ 8 種類（$10, 20, 30, \cdots, 80$）に対して実行し，それぞれのサイズでの最も良いコーラムのサイズを検出した．またこのとき，コーラムの大きさはシステム全体の大きさの 1/2 までとした．これはコーラムの大きさが 1/2 を超えると，任意に選ばれた 2 つのコーラムはつねに共通部分を持ち，システム全体の負荷状態を正確に知ることができるからである．

4.4.2 実験結果

今回の実験では，シミュレーションの各パラメータを以下のように設定して行った．

(1) システムのサイズ（プロセス数）= 10〜80
(2) シミュレーション時間 = 20,000
(3) 仕事の発生間隔 = 4
(4) 発生する仕事の負荷 = 1〜10
(5) 発生する仕事の処理時間 50〜100
(6) 通信遅延時間 = 0.01

ここで，(2) はシミュレーションの振舞いが定常状態になるまでの時間が，全シミュレーション時間の 10^{-2} 以下となるように設定し，(3) はシステム全体での仕事の発生する間隔であり，(4)，(5) については，上で示した範囲内で一様分布に従って値をランダムに発生させた．なお，(3)〜(6) の値に関しては，発生する仕事の例として比較的短い時間（10 秒〜数十分程度）を要する数値計算などを想定した．

また，シミュレーションを実行する回数を 100 回とした☆．

今回の実験では，先に述べた評価基準のなかでも＜負荷の差の平均値＞が最も顕著に違いが見られたので，＜負荷の差の平均値＞の値を参考とした．

図 2 は，プロセス数が（20，40，60，80）のときのそれぞれの＜負荷の差の平均値＞の値を示したものであり，縦軸が評価基準の値，横軸が l の値を示して

☆ これは，今回のシミュレーションで得られる値の期待値を信頼係数 95%で区間推定すると，信頼区間 $[\mu_1, \mu_2]$ に対して $\mu_2/\mu_1 \leq 1.05$ となる．

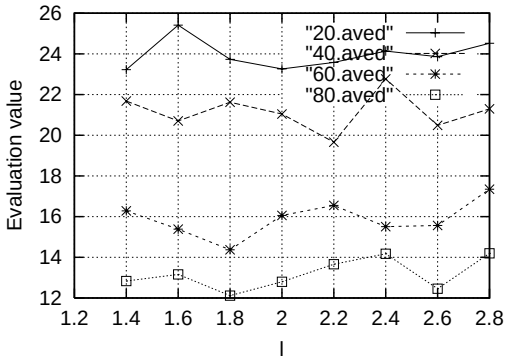

図 2 PQS に関する予備実験の実験結果
Fig. 2 A result of the preliminary experiment about PQS.

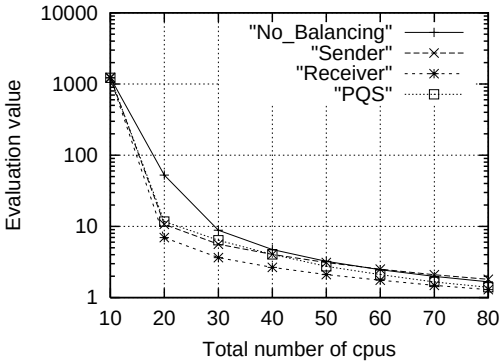

図 4 ＜負荷の平均値＞の実験結果
Fig. 4 A result about average of load.

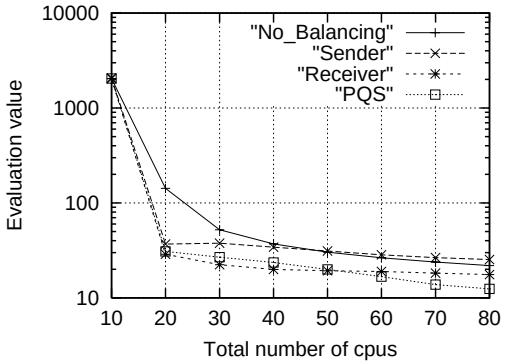

図 3 ＜負荷の最大値＞の実験結果
Fig. 3 A result about maximum of load.

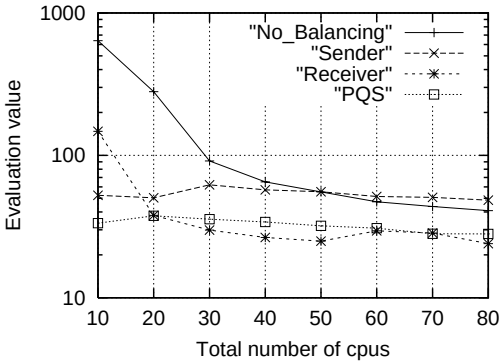

図 5 ＜負荷の差の最大値＞の実験結果
Fig. 5 A result about maximum of the difference of load.

いる．

実験の結果から，l を 2.2 付近に設定したものが良い評価を得ることが判明した．よって，以降の実験では $l = 2.2$ に設定することにした．

4.5 既存の負荷分散アルゴリズムとの比較実験

4.5.1 実験の概要

次に，PQS がどのような効力を持っているのか検討するために，先の実験で得られた結果をもとに $l = 2.2$ に設定して，同様の条件において，負荷分散をまったく行わなかったものや送信機始動のアルゴリズム，受信機始動のアルゴリズムを用いたものと比較した．ここで，受信機始動のアルゴリズムに関しては，実行中の仕事の移動にともなう通信量および通信遅延については考慮しないものとした．

4.5.2 実 験 結 果

図 3，**4**，**5**，**6**，**7** は，4.4.2 項で述べたものと同じ状況における各アルゴリズムの実行結果をそれぞれの評価基準についてまとめたものであり，グラフでは横軸が計算機の総数，縦軸が評価基準の値となっている．また各グラフは，見やすさのために縦軸を対数変換して表示した．今回の実験では，発生する仕事の統計的な総量は CPU 数に関係なく一定としている．つまり，CPU 数が少ない（x 軸の左側）ほど 1CPU あたりの処理すべき仕事が多くなっていることに注意してほしい．なお，図中の 4 つの線はそれぞれ負荷分散をまったく行わないもの（No_Balancing），送信機始動のアルゴリズムを用いたもの（Sender），受信機始動のアルゴリズムを用いたもの（Receiver），PQS を用いたもの（PQS）を表している．

グラフから，他のアルゴリズムなどと比較して，受信機始動のアルゴリズムにはやや劣るものの，負荷分散をまったく行わないものはもちろん，送信機始動のアルゴリズムにも同等かそれ以上の評価を得ることができた．

さらに，シミュレーションのパラメータを変更すると，送信機始動のアルゴリズムや受信機始動のアルゴリズムは得られる評価に違いが生じてしまうが，PQS を用いたものでは上のグラフと同様に安定した評価が得られることが判明した．

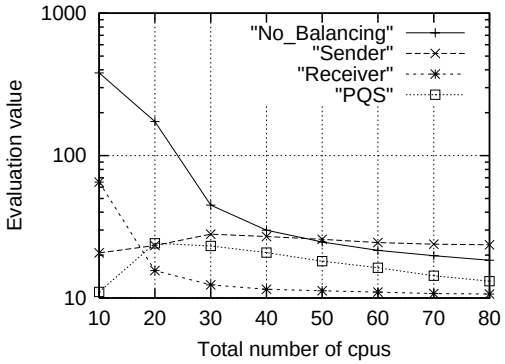

図 6 ＜負荷の差の平均値＞の実験結果
Fig. 6 A result about average of the difference of load.

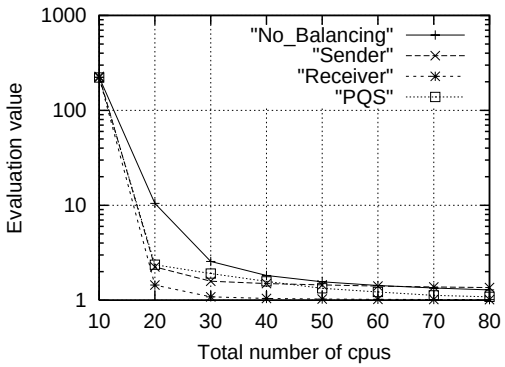

図 7 ＜仕事の処理効率＞の実験結果
Fig. 7 A result about processing efficiency of task.

4.6 考　　察

今回のシミュレーション実験では，まずシステムの大きさに対してコーラムの大きさをどの程度に設定したものが効率的な負荷分散を行えるのかを検証した．その結果，$l = 2.2$ 付近に設定するのが適切であることが判明した．しかしこの値は今回行ったシミュレーションに依存したものであり，通信遅延時間や計算機の処理能力によって変化するものと考えられる．

また，PQS がどのような効力を持っているのか検討する実験を行ったが，＜負荷の最大値＞，＜負荷の差の最大値＞に関して比較的良い評価が得られた理由は，PQS を用いたものはシステムの負荷状態に変化が生じるたびに情報通信を行っているため，仕事が発生した際の情報通信によってシステム全体の比較的新しい負荷状態をつねに把握することができ，その時点である程度適した計算機へと仕事を分散させるので，仕事が発生した際に各計算機間に負荷の差があまり生じないように仕事を分散したためだと思われる．

また＜負荷の平均値＞，＜負荷の差の平均値＞では，PQS を用いた負荷分散がシステム全体にほぼ同等の負荷を与えるように仕事を分散したために，システム全体の平均としては多少の改善にしか至らなかったと考えられる．

＜仕事の処理効率＞については，今回のシミュレーションでは実行中の仕事の処理時間を同じ計算機で実行している仕事の数に反比例したものと設定したので，負荷状態を考慮して仕事を分散させている PQS を用いた負荷分散では必ずしも最適な処理効率を得ることができなかったことが理由として考えられる．

なお，受信機始動のアルゴリズムでは，実行中の仕事を移動させる際に多大な通信が必要であるが，今回の実験では都合上それらの影響を考慮しなかった．実際に受信機始動のアルゴリズムを実装する際には，実行中の仕事についての多大な情報量とそれにともなう通信コストなどを考慮しなければならない．それに対して PQS を用いた場合，各計算機は負荷情報のみを通信すればよいので，実装は比較的容易であると思われる．さらに，送信機始動，受信機始動のアルゴリズムでは，負荷情報を集中的に管理しているために，耐故障性などの観点から問題点が考えられる．しかし，PQS では負荷情報を分散して管理しているので，そのような問題はない．

以上の 2 つの実験から，PQS を用いた負荷分散が送信機始動のアルゴリズムや受信機始動のアルゴリズムと比べてシステムの負荷状況にあまり影響を受けないことが分かった．

5. おわりに

本論文では，分散システムにおいて負荷分散を行う際に必要な負荷情報の通信に確率的コーラムシステムを用いた新たな負荷分散アルゴリズムを提案し，その有効性をシミュレーションによって示した．

このような負荷分散における情報通信には，他のコータリ構造を用いることも可能であると考えられる．たとえば，有限射影平面コータリ[12] を用いた場合には，本論文で提案した PQS を用いた場合よりも少ない通信量で負荷情報の通信を行うことが可能である．ところが，ある計算機が故障して通信を行えないような状況に陥ってしまった場合，その計算機を共通部分として持っていたコーラムどうしはその計算機が回復するまで満足な通信を行うことができない．このとき有限射影平面コータリでは，故障した計算機を含むコーラムを用いるプロセスは十分な情報の交換が行えない．それに対して PQS を用いる場合には，各プロセスは全計算機の一定の割合をランダムに選ぶので，各プロセスに対して比較的安定した情報の交換が

行える．

また，分散システムにおける負荷分散に確率的でないコーラムを用いた場合には，コーラムの定義を各計算機が持つ必要があるが，本論文で用いた PQS を用いる場合にはその必要はない．実際のシステムに対する応用例としては，動的に計算機の一覧を作成しそれを基に一定数の計算機をランダムに選んでコーラムとする，という方法が考えられる．つまり，動的に変化するネットワークに対しても対応が容易であるという長所を持っている．

今後の課題としては，各計算機の処理能力に差が生じている場合や各計算機が故障する可能性を持つ場合，通信に多大な時間を割かれてしまうような場合，さらには今回使用されているタイムスタンプの局所時計への対応など，現実に起こりうる状況に対応させた負荷分散法を見つけていくことが考えられる．

謝辞 本研究の一部は，文部省科学研究費補助金（奨励研究 A，課題番号 11780229）の援助を受けている．

参 考 文 献

1) Eager, D.L., Lazowska, E.D. and Zahorjan, J.: Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Trans. Softw. Eng.*, Vol.12, No.5, pp.662–675 (1986).
2) Shivaratri, N.G. and Krueger, P.: Two Adaptive Location Policies for Global Scheduling, *Proc. 10th International Conference on Distributed Computing Systems*, pp.502–509 (1990).
3) Krueger, P. and Livny, M.: The Diverse Objectives of Distributed Scheduling Policies, *Proc. 7th International Conference on Distributed Computing Systems*, pp.242–249 (1987).
4) 山下雅史：分散相互排除問題とコータリ，情報処理学会論文誌，Vol.34, No.11, pp.1350–1357 (1987).
5) Malkhi, D., Reiter, M. and Wright, R.: Probabilistic Quorum Systems, *Proc. 16th International Conference on Distributed Computing Systems* (1997).
6) Barbara, D. and Garcia-Molina, H.: Mutual Exclusion in Partitioned Distributed Systems, *Distributed Computing*, Vol.1, pp.119–132 (1986).
7) Singhal, M. and Shivaratri, N.G.: *Advanced Concept in Operating Systems*, pp.259–294, McGraw-Hill (1994).
8) Nakajima, A.: Fault-Tolerant Distributed Match-Making with Any Resiliency, *IEICE Trans.*, Vol.E 74, No.2, pp.427–434 (1991).
9) Ibaraki, T. and Kameda, T.: Theory of Coteries: Mutual Exclusion in Distributed Systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.4, No.7, pp.770–794 (1993).
10) Kakugawa, H. and Yamashita, M.: Local Coteries and a Distributed Resouce Allocation Algorithms, 情報処理学会論文誌，Vol.37, No.8, pp.1487–1496 (1996).
11) Chang, Z., Naik, K., Tajima, N., Huang, T. and Noguchi, S.: An Efficient Distributed Algorithm for Implelentation of Multi-Rendevous based on l-Chainig-Coterie, 情報処理学会論文誌，Vol.42, No.2, pp.462–473 (1996).
12) Maekawa, M.: A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems, *ACM Trans. Comput. Syst.*, Vol.3, No.2, pp.145–159 (1985).

(平成 12 年 8 月 7 日受付)
(平成 13 年 12 月 18 日採録)

吉村 英明（学生会員）

1978 年生．2000 年広島大学工学部第二類（電気系）情報工学課程卒業．現在同大学大学院工学研究科情報工学専攻博士前期課程に在学中．分散アルゴリズムに興味を持つ．

角川 裕次（正会員）

1967 年生．1990 年山口大学工学部電子工学科卒業．1992 年広島大学大学院工学研究科情報工学専攻博士課程前期修了．1998 年広島大学工学部第二類助教授．2001 年改組により広島大学工学研究科情報工学専攻助教授．分散アルゴリズムならびに教育用ソフトウェアの研究に従事．博士（工学）．

阿江 忠（正会員）

1941 年生．1964 年東北大学工学部通信工学科卒業．1969 年同大学大学院工学研究科博士課程修了．工学博士．現在広島大学工学部教授．研究歴は Device, Computer, AI, Brain（DCAB）．右能コンピュータ開発に興味を持つ．