人　工　知　能　93−13
（1994．3．9）

確率学習オートマトンと遺伝的アルゴリズムによる動的負荷分散

棟朝雅晴　高井昌彰　佐藤義治
北海道大学 工学部

Abstract

分散計算システムは自律した計算機が比較的通信遅延の大きい通信ネットワークを介して相互結合されたシステムである。分散計算システムの性能向上のためには、それぞれの計算機における負荷の一様化をはかることが重要である。分散制御型の動的負荷分散アルゴリズムは計算の実行中にそれぞれの計算機で負荷状態の観測を行ない、負荷の重い計算機から軽い計算機へタスクの転送を行なうことで負荷を一様化する。本論文では、確率学習オートマトンと遺伝的アルゴリズムを用いることで効率的なタスク転送要求の送出法を学習する分散制御型の動的負荷分散アルゴリズムを提案する。本手法では、それぞれの計算機ごとにタスク転送をどの計算機に対して要求するかを記述した文字列からなる集団を用意する。そして、その集団に対して確率学習オートマトンと遺伝的アルゴリズムの操作を用いた学習を適用することで効率的な転送要求の送出を行なう。シミュレーション実験により提案する手法の有効性が確かめられた。

A Dynamic Load Balancing Scheme Using Stochastic Learning Automata and Genetic Algorithms

Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato
Information and Graphics Sciences, Faculty of Engineering, Hokkaido University
North 13, West 8, Kita-Ku, Sapporo 060, Japan.

Abstract

A distributed computing system is a collection of autonomous computers loosely connected via a communicating network whose latency is relatively large. In improving the system performance, it is important to keep the load of each processor even. On a distributed dynamic load balancing algorithm, each processor observes their load state and sends a task in order to balance their loads. In our scheme, we use a population of strings each of which stands for a set of processors to which requests of a task migration are sent, and genetic operations and stochastic learning are applied in order to realize efficient task migrations. Through empirical investigations using a simulator, we show the effectiveness of our scheme.

1 はじめに

自律した計算機が通信遅延の比較的大きいネットワークを介して結合されたシステムを分散計算システム (Distributed Computing Systems, DCS) と呼ぶ [1]。DCS において計算機の利用率を向上させるために、負荷分散アルゴリズムに関する研究が数多くなされている。負荷分散アルゴリズムは大きく静的負荷分散、動的負荷分散に分類することができる。さらに動的負荷分散は一台の計算機で集中して負荷情報の管理とタスク転送の決定を行なう集中制御型と、それぞれの計算機で独立して負荷情報の管理とタスク転送の決定を行なう分散制御型に分けられる。

これまでに、分散制御型の動的負荷分散において遺伝的操作を導入した手法に関する研究が行なわれている [8]。この手法においては、タスク転送をどの計算機に対して要求するかを記述した文字列を遺伝的アルゴリズムにおける個体とする。そして、複数の個体からなる集団をそれぞれの計算機ごとに用意し、それに対して遺伝的アルゴリズムの基本操作を適用する。この手法は、非一様負荷の場合にその有効性が確かめられているが [9]、一様負荷の場合においては改善が見られていない。さらに、過去 N回の要求の成否により適合度の評価を行なっていることから、Nが小さい時には過去の情報が反映されず、Nが大きい時には急激な負荷の変化に対応できないという問題点がある。

本論文ではこの問題を解消するため、適合度評価に確率学習オートマトンを導入し、一様負荷の場合においても有効なタスク転送要求の送出法を学習する分散制御型の動的負荷分散アルゴリズムを提案する。本手法では、各個体の持つ適合度値を確率ベクトルとみなし、確率学習オートマトンによる確率的山登り法を適用する。これにより、過去の情報を反映しかつ迅速な適合度評価を行なうことができる。

以下、第 2 章では動的負荷分散アルゴリズムに関して、関連する従来の手法である sender-initiated algorithm[2, 3] と、それに遺伝的アルゴリズムの操作を導入したアルゴリズム [8, 9] を紹介する。第 3 章、第 4 章では、確率学習オートマトンと遺伝的操作について説明する。第 5 章では提案する手法である GeSLA 動的負荷分散アルゴリズムの詳細を述べる。第 6 章では UNIX-WS 上に実現されたシミュレータによる評価実験の結果により、GeSLA の有効性を示す。

2 動的負荷分散

動的負荷分散アルゴリズムは、タスクを負荷の重い計算機から負荷の軽い計算機へ転送することで計算機間の負荷の一様化をはかり、システム全体の性能向上 (タスクの平均応答時間の最小化) を目的としている。負荷情報の観測とタスク転送の決定をどのように行なうかによって、集中制御型と分散制御型に分類することができる。

集中制御型の動的負荷分散では、一つもしくは少ない数の計算機で集中して負荷情報の管理とタスク転送の決定を行なう。この方法はタスク転送の決定が比較的正確に行なわれるが、そのためには負荷情報の収集に多くの通信量を必要とする。これとは対照的に、分散制御型の動的負荷分散では負荷情報の管理とタスク転送の決定は、それぞれの計算機において行なわれ、ネットワーク全体を管理する計算機は存在しない。この手法の利点としては、負荷情報の観測が局所的であるために、それに要する通信量が少ないという点があげられるが、全体を管理していないために、非効率的なタスクの転送が行なわれる可能性がある。

分散制御型の動的負荷分散アルゴリズムの中で最も基本的なものの一つである sender-initiated algorithm[2, 3] は、タスク転送の送り手、すなわち負荷の重い計算機からタスク転送要求を送出する手法である。このアルゴリズムでは、ランダムに選ばれた計算機に対してタスク転送の要求を送出し、要求先の計算機の負荷の軽いことが分かった場合には、そこに対してタスクを転送する。そうでない場合には、負荷の軽い計算機が見い出されるか、ある一定回数の限度を越えるまで、繰り返し転送要求を行なう。このアルゴリズムでは計算機ネットワーク全体の負荷が重い場合には要求が送出されてもそれが受理される可能性が少なく、無駄な負荷転送要求が度々送出され、全体の効率が低下する可能性がある。

遺伝的操作を sender-initiated algorithm に導入することで無駄な負荷転送要求の送出を防ぐことを目的とした手法が提案されている [8, 9]。この手法では、タスク転送要求の送出先を記述した文字列を遺伝的アルゴリズムにおける個体とし、それからなる集団を遺伝的操作の対象とする。遺伝的操作として、uniform crossover と mutation を用いて、要求の送出先となる計算機群を適切なものとする。各個体の淘汰について

は ranking selection [4] を用いている。適合度値は過去 N回の要求送出の成功、失敗により決定される。シミュレーション実験によると [9]、非一様負荷の場合には sender-initiated algorithm に比べて有効であるという結果が出ているが、一様負荷の場合については有意な改善は見られていない。これは、適合度評価において過去 N回の要求送出の結果を用いているので、Nを小さくした場合には過去の情報を用いることができず、Nを大きくした場合には負荷の急激な変動に対応できないためと考えられる。

3 確率学習オートマトン

確率学習オートマトン (Stochastic Learning Automata, SLA) は 6 つ組 $\{x, \phi, \alpha, \boldsymbol{p}, A, G\}$ で定義される [5]。ここで、$x \in \{0,1\}$ はオートマトンへの入力の集合で、$\phi = \{\phi_1, \phi_2, \cdots, \phi_s\}$ は内部状態の集合、$\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$ $(r \leq s)$ は出力の集合、$\boldsymbol{p}(n)$ は時刻 n の内部状態を決定する確率ベクトル$\boldsymbol{p}(n) = (p_1(n), p_2(n), \cdots, p_s(n))^t$、$A$ は$\boldsymbol{p}(n)$ から $\boldsymbol{p}(n+1)$ を生成するアルゴリズムであり、$G : \phi \rightarrow \alpha$ は出力関数である。SLA の学習の枠組を図 1に示す。ペナルティー確率の集合 $c = \{c_1, c_2, \cdots, c_r\}$ は環境がオートマトンからの出力α_iを受けとった時に失敗を意味する $x = 1$ を返す確率を示している。その値 x がオートマトンに入力され、その入力にしたがって学習が行なわれる。

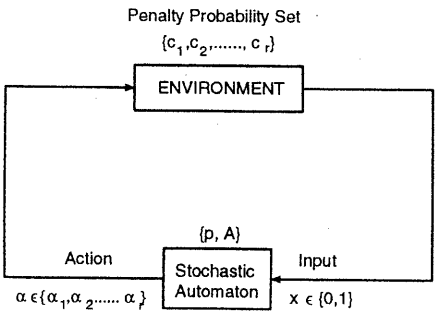

図 1: A Stochastic Learning Automaton

SLA に関して多くの学習アルゴリズムが提案されている。ここでは、提案する負荷分散手法で用いている学習アルゴリズムである、linear reward-inaction scheme[5] について説明する。

一般に、オートマトンよりα_iが出力され、その成功、失敗を環境からの入力 $x(n)$ として受けとった場合、確率学習オートマトンの学習は以下で示される確率的山登り法により実現される。

$$p_i(n+1) = p_i(n) + \sum_{j \neq i} f(p_j(n)), \qquad (1)$$
$$p_j(n+1) = p_j(n) - f(p_j(n)),\ \forall j \neq i$$
$$\text{if } x(n) = 0.$$

$$p_i(n+1) = p_i(n) - \sum_{j \neq i} g(p_j(n)), \qquad (2)$$
$$p_j(n+1) = p_j(n) + g(p_j(n)),\ \forall j \neq i$$
$$\text{if } x(n) = 1.$$

ここで、$f(\cdot), g(\cdot)$ は非負の連続関数のうち、$p_k(n) \in (0,1)$ であるとき $(k = 1, 2, \cdots, r)$ に $p_k(n+1) \in (0,1)$ なる条件を満たすものである。

linear reward-inaction scheme は、上で示した $f(\cdot), g(\cdot)$ が以下で定義されるものをいう。

$$f(p) = ap,\ g(p) \equiv 0. \qquad (3)$$

ここで、パラメータ a の値は $0 < a < 1$ である必要がある。この方法は、ペナルティー確率が変化しない環境に対して、最適な確率ベクトルに収束することが知られている [5]。

4 遺伝的操作

遺伝的アルゴリズム (Genetic Algorithms, GA)[6] は生物の進化の仕組みをモデルにした探索手法である。GA は適合度関数と呼ばれる目的関数の最大化問題を近似的に解く手法である。GA で使用される文字の集合をアルファベットと呼び、そのアルファベット中の文字から構成される文字列として個体が表現される。探索の対象となる空間上の点は文字列として表現される個体の形にコーディングされる。個体の示す解空間上の点における目的関数値がその個体の持つ適合度値となる。個体からなる集団を用意し、それに対して基本操作である selection(淘汰), crossover (交叉),

mutation(突然変異) を繰り返し適用することで近似最適解を求める。この基本操作をここでは遺伝的操作と呼ぶ。

selection は、適合度の大きい個体の数を増やし、小さい個体の数を減らすことにより探索に方向性を持たせる操作である。selection には多くの手法が存在するが、本論文で使用する方法は、個体を適合度の大きい順番に並べて、その順位の低い個体を除去するという ranking selection の一形態を用いている。

GA で最も重要な操作である crossover は、2 つの個体の部分列を交換する操作である。この操作についても多くの手法が提案されているが、本論文で使用する uniform crossover[7] は、どの部分列を交換するかを記述したテンプレートをランダムに生成し、それを用いて交叉を行なう。テンプレートは $\{0,1\}$ からなる文字列 $s = s_0 s_1 \cdots s_{n-1}$ で、$s_i = 1$ の場合に位置 i において文字の交換を行なう。これを示したのが以下の図 2である。

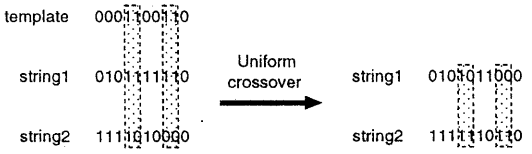

図 2: uniform crossover

mutation は個体中の文字をある確率で別の文字に変化させる操作である。この操作の目的は、初期集団中に含まれない文字を生成することで安定した探索を実現することである。

GA を理解する上で重要な概念にスキーマがある。スキーマとはアルファベット中の文字と $*$ (don't care symbol) からなる文字列で、解空間のある部分空間を示すものである。例えば、アルファベットが $\{0,1\}$ であるとき、$\{0,1,*\}$ から構成される文字列がスキーマであり、スキーマ $H = 01**10$ は個体の集合 $\{010010, 010110, 011010, 011110\}$ を指し示している。GA はこのスキーマの組合せにより探索を行なうアルゴリズムであると言える。

5 GeSLA 動的負荷分散アルゴリズム

ここでは我々の提案する手法である、GeSLA 動的負荷分散アルゴリズムについて述べる。この手法は、sender-initiated algorithm に、確率学習オートマトンと遺伝的操作を導入することで、効率的なタスク転送要求を送出することを目的としたものである。はじめにその概要を紹介し、つぎに確率学習オートマトンと遺伝的操作による学習の詳細について説明を行ない、最後にアルゴリズムを構成する手続きとメッセージの処理について述べる。

5.1 概要

GeSLA 動的負荷分散アルゴリズムの概要を図 3に示す。図の P_iはそれぞれの計算機を示している。本手法においては、タスク転送要求を複数送出するマルチキャストが導入されている。どの計算機に対して要求を送出するかを $\{0,1\}$ からなる文字列の形にコーディングする。文字列の k番目の文字が 1 のときに計算機 kにタスク転送の要求を送出するものとし、0 のときには送出しないものとする。この文字列が遺伝的操作の対象となる個体となる。各計算機ごとに複数の個体からなる集団を用意し、それに対して遺伝的操作を適用する。それぞれの個体について、適合度 (集団全体における適合度値の和を 1 とする。これにより適合度を確率とみなすことができる) が与えられる。従来手法では過去 N回の要求の成否により適合度の値を求めていたが、GeSLA では SLA による学習により適合度値の更新を行なう。初期集団はすべて 1 からなる個体で構成される。つまり、初期状態ではすべての計算機に対してタスク転送の要求を送出することとなる。

それぞれの計算機では、外部からタスクが到着した場合に負荷の観測が行なわれ、負荷が重いと判断された場合に他の計算機に対するタスク転送手続きが起動される。負荷状態の観測については、タスク待ち行列の長さを採用した。待ち行列の長さが 0 のときに負荷が軽く、3 以上の場合に負荷が重いものとした。まず始めに一つの個体がその適合度値に比例した確率で選択され、その記述内容にしたがって複数の計算機に対しタスク転送要求が送出される。負荷の軽い計算機が少なくとも一つ見い出されたなら要求は成功し、タス

クが実際に転送される。すべての計算機から要求が拒絶された場合にはタスクは転送されず、それが入力された計算機で実行される。タスク転送の手続きが終了したなら、それが成功したかどうかにより、SLA の確率的山登り法により集団内での学習が行なわれ、適合度の値が更新される。適合度値の更新が行なわれた後に一定確率で、遺伝的操作を起動する。集団内に存在する個体は、可能な組合せのうちの一部であるが、これにより新たな個体が生成され、大域的な学習が実行される。遺伝的操作が実行された場合、適合度値の低い個体が除去され、遺伝的操作により生成された個体と置き換えられる。

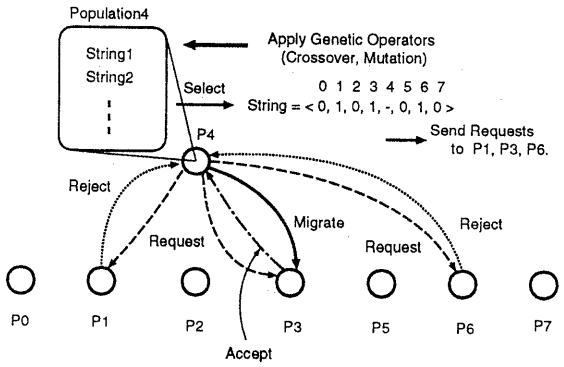

図 3: GeSLA 動的負荷分散アルゴリズムの概要

5.2 手続き間の依存関係とメッセージ処理

GeSLA 動的負荷分散アルゴリズムは、5 つの手続き、Initialization, Check_load, Message_evaluation, String_evaluation, Genetic_operators から構成されている。各手続き間の依存関係を図 4に示す。

Initialization は集団と負荷情報の初期化をする手続きであり、システムの起動時に実行される。Check_load は計算機に外部からタスクが入力された時に起動され、その計算機の負荷状態を観測し、負荷が重いと判断されたならタスク転送要求のメッセージ送出を行なう。Message_evaluation はメッセージの処理を行ない、タスク転送を実行する。String_evaluation では、タスク転送の成否により SLA による適合度値の更新が行なわれる。Genetic_operators では、遺伝的操作である crossover, mutation をある一定の評価回数毎に起動する。

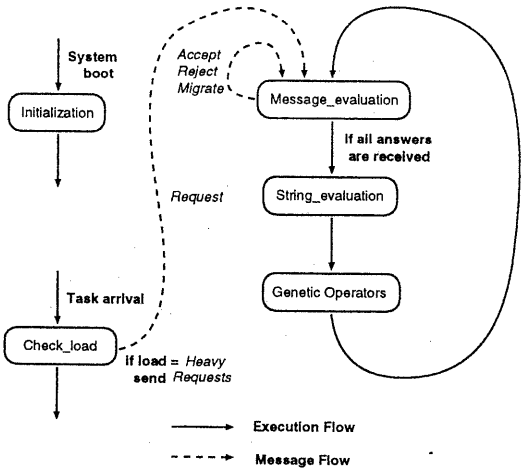

図 4: 手続き間の依存関係

以下の 4 種類のメッセージにより通信が行なわれる。

Request タスク転送の要求

Accept タスク転送要求の受け入れ

Reject タスク転送要求の拒絶

Migrate タスク転送 (転送対象のタスクを含む)

ある計算機で負荷が重くなった場合に、その計算機に存在する集団中から各個体の持つ適合度値に比例した確率で個体を一つ選択し、その個体の記述にしたがって、複数の計算機に対して Request メッセージが送出される。その Request メッセージを受けとった計算機では、その負荷が軽い場合には Accept メッセージを、そうでない場合には Reject メッセージを送り返す。Request メッセージの送り出し元の計算機では、その返事である Accept, Reject メッセージをすべて受けとった時に、少なくとも一つの Accept メッセージがその中に存在する場合にはそのメッセージの送出元である計算機 (複数ある場合にはその中からランダムに選択する) に対して Migrate メッセージを送り、

タスク転送を実行する。すべてのメッセージがRejectである場合には、タスク転送要求は失敗し、タスクはそれが入力された計算機にそのまま渡される。その後に、SLAによる学習操作と遺伝的操作が集団に対して適用される。

5.3 SLAによる学習

集団内に存在する各個体に対して、それが選択される確率 p_i $(i = 0, 1, \cdots m-1)$ として適合度値が与えられる (m は集団内の個体数)。タスク転送要求の結果が戻ってきた場合にその成否により確率 p_iに対してlinear reward-inaction schemeが適用される。負荷の軽い計算機が少なくとも一つ見い出された場合にその要求が成功したとみなされ、式2にしたがって p_i の値を増加させる。式3における a の値は以下を用いている。

$$a = b/k, \text{ where } 0 < b < 1. \qquad (4)$$

ここで、kはタスク転送要求の送出数(つまり選択された個体の文字列中での1の数)である。これにより、要求の送出数が少ない場合により大きな適合度値を割り当て、タスク転送要求に使われる通信コストが少ない個体を有利にする。また、linear reward-inaction schemeを用いているため、タスク転送要求が失敗した場合には p_iの更新は行なわれない。

5.4 遺伝的操作

タスク転送手続きにおける個体選択と確率学習による適合度の更新を行なった後に、ある一定確率で、遺伝的操作のcrossover, mutationを集団に適用する。

crossoverが起動されると、集団から2つの個体がランダムに選択され、その個体のペアに対してuniform crossoverが適用され、新たな個体が生成される。生成された個体の適合度に関しては、それぞれの親となる個体から継承される。crossoverの実行後、集団中で適合度の低い個体が2つ除去され、新たに生成された個体と置き換えられることでselectionが実行される。

mutationが起動されると、集団から一つの個体がランダムに選択され、その個体中の文字のうちの一つが別の文字に変化することで新たな個体が生成される。その個体の適合度は親となる個体から継承される。その後に、集団中で最も適合度の低い個体が除去され、新たに生成された個体と置き換えられることでselectionが実行される。

crossover, mutationが実行された直後では、適合度 p_iがそれぞれの親から継承されるため、$\sum_i p_i \neq 1$ となる可能性がある。そこで、$p_i \leftarrow p_i / \sum_j p_j$ による調整がなされる。

このアルゴリズムにおいて、スキーマはタスク転送をする(もしくはしない)計算機群を示している。例えばスキーマ $H_1 = ***11*$は計算機 P_3と P_4に対して要求を送ることに対応し、スキーマ $H_2 = 0*0***$は計算機 P_0と P_2に対して要求を送らないことを示す。スキーマ H_1と H_2を含む個体が高い適合度を持つ、すなわちタスク要求の成功率が高いものとすると、crossoverによりこの両者が組み合わされ、さらに成功率の高い個体が生成される可能性がある。crossoverの手法としてuniform crossoverが使用されているのは、文字列中における位置が近い計算機が必ずしも同じような負荷状態を持つとは限らないので、one-point crossoverやtwo-point crossoverのような文字の位置関係を使用した手法を用いる必然性がなく、単に部分列を交換すると良いためである。

6 シミュレーション実験

GeSLAの有効性を確かめるためのシミュレーション実験を行なった。シミュレーション実験は、C言語を用いてWS上に実現されたシミュレータを用いて行なった。共通の実験条件を以下に示す。

- 計算機ネットワーク：16台の計算機が10Mbpsのネットワークによりバス結合されている。
- 入力されるタスク：平均実行時間100ms、平均のサイズ10KBytesでそれぞれ指数分布に従う。
- sender-initiated algorithmの条件：最大8回までタスク転送要求を行なう。
- GAの条件：集団の大きさが10、遺伝的操作は20回の評価毎に行なう。適合度評価には過去5回のタスク転送要求の成功率を用いる。

- GeSLA の条件：集団の大きさが 10、遺伝的操作は 20 回の評価毎に行なう。また SLA の部分において、式 4のパラメータを $b = 0.2$ とする。
- シミュレーション時間：システムが 10000 個のタスクを処理するまで行なった。

この場合、平均到着割合λはタスクの平均実行時間l(ms)、1ms あたりのタスク到達確率pを用いて、$\lambda = lp$ で定義される。以下では、すべての計算機に対して一定の平均到着割合でタスクが到達する場合と、一部の計算機に集中してタスクが到達する場合について実験を行なった。ここで、Eをタスクが入力されている計算機の数とする。λはシステム全体としてのタスクの平均到着割合であるので、各計算機に対しては$\lambda \times 16/E$なる平均到着割合でタスクが到達することになる。

はじめに、一様負荷の場合 ($E = 16$) について実験を行なった。この場合、すべての計算機に対して同じ平均到着割合λでタスクが到達する。この結果を図 5に示す。これによると、GA のみを用いた手法は、sender-initiated algorithm とほぼ同じ結果となっており、改善が見られない。しかしながら、SLA を導入することにより良い結果を得ることができ、平均応答時間が最も少ない値となっている。

非一様負荷に関する実験として多くの場合が考えられるが、ここでは、$E = 8$, $E = 4$ の場合について実験を行なった。$E = 8$ の場合には、8 台の計算機に対して集中してタスクが到達する、つまり 8 台の計算機に対しては平均到着割合 2λでタスクが到達し、残りの計算機にはまったくタスクが到達しないこととなる。$E = 4$ の場合には、4 台の計算機に対して平均到着割合 4λでタスクが到達し、残りの 12 台の計算機にはタスクが入力されない。

$E = 8$ に関する結果を図 6に、$E = 4$ に関する結果を図 7に示す。これによると、タスクの平均到着割合に関して、ばらつきが比較的少ない $E = 8$ の場合には GA のみによっても比較的良い結果を得ることができているが、ばらつきの大きい $E = 4$ の場合には、GeSLA により格段の性能向上がはかられていることがわかる。どちらの場合も従来の sender-initiated algorithm に比べて、大きな性能向上が見られている。

以上の結果により、GA のみを用いた手法に比べて GeSLA 動的負荷分散が優れており、従来の sender-initiated algorithm に比べて格段に良い結果を得ていることが確かめられた。

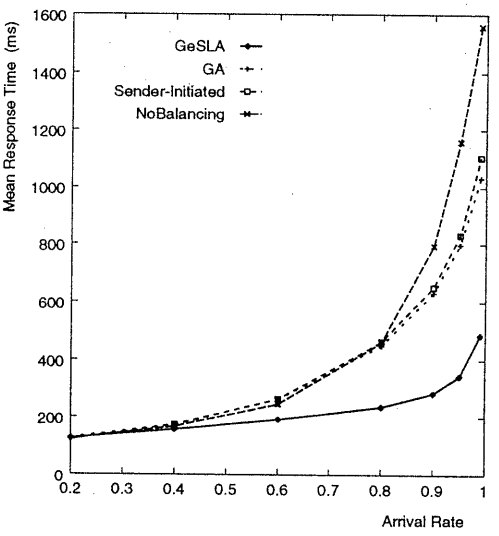

図 5: 平均応答時間の比較 (一様負荷, $E = 16$)

7 おわりに

本論文では、分散制御型の動的負荷分散アルゴリズムの一つである sender-initiated algorithm に確率学習オートマトンと遺伝的操作による学習を導入した GeSLA 動的負荷分散アルゴリズムを提案し、その有効性を、シミュレータを用いた実験により確認した。従来の GA を用いた手法では改善の見られなかった一様負荷の場合においても、GeSLA は良い性能を示すことが確かめられた。また、タスクが一部の計算機に集中して入力されているという非一様負荷の場合に効率的なタスク転送が行なわれていることを示した。今後の課題としては、タスクの受け入れ側からタスク転送要求を送出する receiver-initiated algorithm の考え方を導入した手法の開発があげられる。

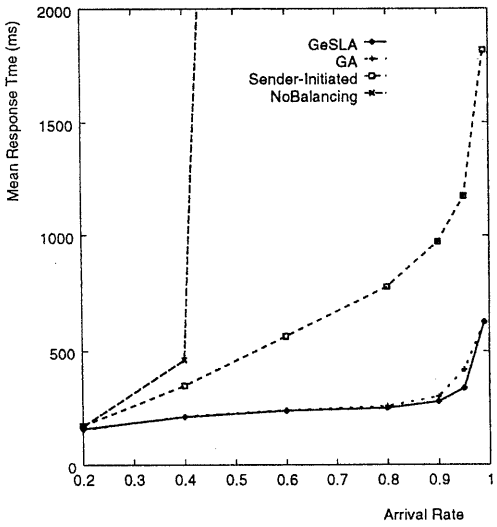

図 6: 平均応答時間の比較 (非一様負荷, $E = 8$)

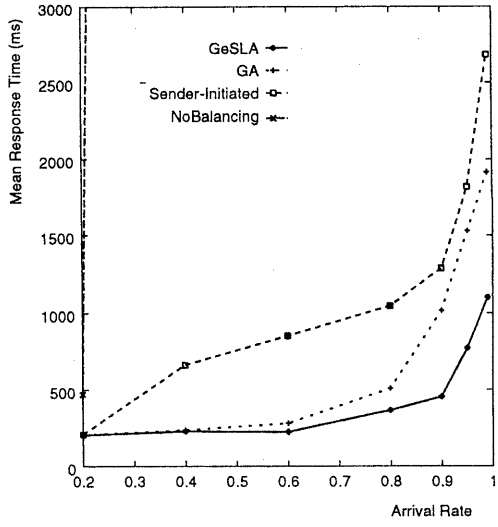

図 7: 平均応答時間の比較 (非一様負荷, $E = 4$)

参考文献

[1] T. T. Y. Suen and J. S. K. Wong: "Efficient task migration algorithm for distributed systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 4, pp. 488–499 (1992).

[2] D. L. Eager, E. D. Lazowska and J. Zahorjan: "Adaptive load sharing in homogeneous distributed systems", *IEEE Transactions on Software Engineering*, Vol. 12, No. 5, pp. 662–675 (1986).

[3] N. G. Shivaratri, P. Krueger and M. Singhal: "Load distributing for locally distributed systems", *IEEE COMPUTER*, Vol. 25, No. 12, pp. 33–44 (1992).

[4] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley (1989).

[5] K. S. Narendra and M. A. L. Thathachar: "Learning automata — a survey", *IEEE Transactions on System, Man, and Cybernetics*, Vol. 4, No. 4, pp. 323–334 (1974).

[6] J. H. Holland: *Adaptation in Natural and Artificial Systems*, University of Michigan Press (1975).

[7] G. Syswerda: "Uniform crossover in genetic algorithms", *Proceedings of the Third International Conference on Genetic Algorithms* (Ed. by J. D. Schaffer), Morgan Kaufmann Publishers, pp. 2–9 (1989).

[8] 棟朝雅晴, 高井昌彰, 佐藤義治: "遺伝的操作を用いた動的負荷分散アルゴリズムの提案", 電子情報通信学会 1993 年秋季大会講演論文集 (6) , p. 88 (1993).

[9] 棟朝雅晴, 高井昌彰, 佐藤義治: "分散制御型動的負荷分散における遺伝的操作の導入", 北海道大学工学部研究報告, No. 167, pp. 127–135 (1994).