

特別研究報告書

大規模マルチエージェントシステムにおける  
エージェント配置

指導教員 石田 亨 教授

京都大学工学部情報学科

宮田 直輝

平成18年2月10日

# 大規模マルチエージェントシステムにおけるエージェント配置

宮田 直輝

## 内容梗概

近年、多数のエージェントを扱う大規模マルチエージェントシステムが開発されている。大規模マルチエージェントシステムは、より多くの利用者にサービスを提供するシステムを構築することや、より複雑な系を対象とするシミュレーションを行うことを目的として構築される。

大規模マルチエージェントシステムにおける問題点の1つは、エージェント数の増加に従ってシステムの性能が低下する事である。例えば、利用者の数が増加するにつれて、利用者にリアルタイムなサービスを提供することが困難になる。また、対象とする系が大きくなるにつれて、シミュレーションを行うために必要となる時間は増加する。このような問題に対しては、複数のエージェントサーバを接続してシステムを分散するアプローチをとることができる。複数のエージェントサーバからなる分散マルチエージェントシステムを構築する際に問題となるのは、以下の2点である。

### 1. システムの性能を改善するためのエージェント配置

分散マルチエージェントシステム上でエージェントを分散配置する際に考慮すべき点に、エージェントの計算負荷とエージェント間のインタラクションが挙げられる。あるエージェントサーバに多くのエージェントが集中すると、そのエージェントサーバの計算負荷が高まりシステムのボトルネックとなる。そのため、エージェントの計算負荷を各エージェントサーバに分散しなければならない。次に、エージェントが分散したことによるインタラクションコストの増加を抑えなければならない。異なるエージェントサーバに存在するエージェント間で発生するインタラクションは、サーバ間でメッセージを交換しなければならない分、同じエージェントサーバに存在するエージェント間で発生するインタラクションよりもコストが大きい。そこで、頻繁にインタラクションを行うエージェントを同じサーバに配置し、インタラクションコストの増加を抑える必要がある。

### 2. エージェントサーバを分散する際の実装方法

エージェントの配置を決定する手法を適用する際の実装方法も、重要な課題である。分散化されたシステム上では、中心となる機能を必要とせずに

各サーバが独立して動作することが必要である．中心となる機能が存在する場合，その機能がシステムのボトルネックになる可能性があるなど，システムを分散したことによる利益を損なう．従って，中心となる機能を用いずに適用手法を実装する必要がある．

以上の問題を解決するために，本研究では分散マルチエージェントシステムにおけるエージェント配置の問題を，エージェント配置問題として定式化する．そしてエージェント配置問題に対する近似解を基にエージェントの配置を決定するアプローチを取る．本手法の適用例として，スタンドアロン型のマルチエージェントシステムである大規模誘導システムを取り上げ，そのシステムを分散化する実装を行う．

本研究における主な貢献は，以下の2点である．

1. インタラクションコストを小さくするエージェント配置手法の確立  
エージェントの配置問題を，各エージェントサーバの計算負荷が閾値より小さく，かつエージェントサーバ間のインタラクションコストを最小にするエージェント配置を求める問題に定式化した．エージェント配置問題の最適解を求めることは困難であるため，ネットワーク分割問題に対する近似アルゴリズムである $k$ -way FM法のアイデアを用いて，エージェント配置問題に対する近似アルゴリズムを定めた．シミュレーションにより，この手法を適用することで先行研究である Comet アルゴリズムよりもインタラクションコストが速く減少し，反復後のインタラクションコストが約 10%改善することが示された．
2. 中心となる機能を必要としない分散マルチエージェントシステムの実装  
スタンドアロン型の大規模誘導システムを分散化し，複数のエージェントサーバを接続する実装について述べた．大規模誘導システムに対して，他のサーバと連携するための機能とシステム内部の監視を行う機能の実装を行うことで，各エージェントサーバが独立して本手法を用いてエージェントの配置を決定できる実装を行った．

上記2点の貢献により，分散マルチエージェントシステム上でエージェントを分散配置する際に本手法を用いることで，インタラクションコストの増加によるシステムの性能低下を抑えることが可能であることが示された．そして，中心となる機能を必要とせずに，本手法を用いたシステムを実装できることが示された．

## Placing Agents in Massively Multi-Agent Systems

Naoki MIYATA

### Abstract

Recently, large-scale multi-agent systems dealing with massive agents are being developed. Large-scale multi-agent systems enable the construction of systems that provide services to more number of users or conduct simulations of more complex systems.

One of the problems with large-scale multi-agent systems is that an increase in the number of users leads to deterioration of system performance. For example, as the number of users increase, the system has more difficulty in providing real-time services to great number of users. Moreover, as systems performing simulations become more complex, more time is required to conduct the simulation. One way to solve this problem is to connect multiple agent servers to construct a distributed multi-agent system. When a distributed multi-agent system is constructed, the following problems must be considered.

1. Agent placement to improve the performance of the system

When agents are distributed to distributed multi-agent systems, both the load of agents and the interaction cost among agents have to be taken into account. If many agents concentrate on one agent server, that agent server may create a bottleneck in the system. Therefore, the load of agents needs to be distributed among multiple agent servers. Increase in the interaction cost due to the distribution of agents has to be also held down. The inter-interaction cost among different servers is greater than the intra-interaction cost within a single agent server due to the load of exchanging messages among agent servers. Given this fact, the agents which frequently interact with each other must be placed on the same agent server to reduce the interaction cost among agent servers.

2. Implementation of distributed multi-agent systems

How to implement distributed multi-agent systems is also an important issue. Each server on distributed multi-agent systems has to run independently without central functions. If central functions exist, the benefits from distributed computing are lost because the functions may create a

bottleneck. Consequently, distributed multi-agent systems have to be implemented without central functions.

To solve the above two problems, this paper formalizes the problem of placing agents on distributed multi-agent systems as an agent placement problem. This paper provides an approximate algorithm to the agent placement problem. I take up a massive navigation system, which is a stand-alone multi-agent system, as an application example to apply my technique, and demonstrate how the distributed system is implemented. The main contributions of this paper are the following.

1. Proposed an agent placement technique with consideration to load and interaction cost of agents

I formalized the problem of placing agents as an agent placement problem, in which the load of each agent server is lower than the threshold and the interaction cost among agent servers is minimized. This paper provides an approximate algorithm to the agent placement problem by means of using the idea of k-way FM algorithm which is an approximate algorithm used in network partition problem. Approximate algorithm was selected because calculating the optimum solution of agent placement problem is difficult. Simulation result showed that the proposed technique reduced the interaction cost faster than other existing techniques such as Comet algorithm, and that after many iterations, the proposed technique improved the interaction cost about 10% more compared to the previous technique.

2. Implemented a distributed multi-agent system without central functions

In this paper, the implementation of a stand-alone massive navigation system that connects multiple agent servers in a distributed manner was stated. I added a function to relate other servers and another function to monitor the system, and this allowed each agent server to independently determine the placement of agent.

With the above two contributions, it was proven that interaction cost can be reduced to improve system performance when the proposed technique was applied to distributed multi-agent systems. Furthermore, the system not needing central functions was shown that it can be implemented.

# 大規模マルチエージェントシステムにおけるエージェント配置

## 目次

第1章	はじめに	1
第2章	分散環境におけるエージェント配置	3
2.1	問題の定式化	3
2.1.1	エージェント配置問題	3
2.1.2	エージェント配置問題の定式化	4
2.2	エージェント配置問題に対する近似解	6
2.3	特徴	8
第3章	エージェント配置シミュレーション	10
3.1	シミュレーション内容	10
3.1.1	WBLB アルゴリズム	11
3.1.2	Comet アルゴリズム	11
3.2	シミュレーション結果	12
第4章	大規模誘導システムへの適用	15
4.1	大規模誘導システム	15
4.2	本手法の適用	19
第5章	おわりに	21
	参考文献	24

## 第1章 はじめに

近年，多数のエージェントを扱う大規模マルチエージェントシステムが開発されている [1]．大規模マルチエージェントシステムを構築することで，より多くの利用者にサービスを行うシステムを構築することや，より複雑な系を対象とするシミュレーションを行うことができる．例えば，数百万規模の交通をマルチエージェントシステムにより再現するシミュレーション [2] や，大規模な利用者を想定したサービスをマルチエージェントシステムによって提供するシステム [3] が開発されている．こうした大規模なマルチエージェントシステムを構築する際の問題を解決するための研究が，現在進められている [4]．社会中心設計 (Society-Centered Design)[5] は，大規模マルチエージェントシステムの評価を行う際に，多くの被験者や車を集めて実験を行うことは困難であるという問題を解決するための設計手法として提案されている．

大規模マルチエージェントシステムにおける問題点の1つは，エージェント数の増加に従ってシステムの性能が低下することである．システムの性能が低下する主な原因は，エージェント数の増加によるシステムのリソース不足である．エージェントの数が増加すると，システムが行わなければならない計算量は増加する．また，利用者が増えれば，アクセスするための通信帯域が更に必要になる．従って，システムの利用者が増加すると，利用者にリアルタイムなサービスを提供することが困難になる．マルチエージェントによるシミュレーションにおいても，対象とする系が大きくなるにつれてシミュレーションを行うために必要となる時間は増加する．

大規模マルチエージェントシステムの性能が低下する問題に対して，エージェントシステムを分散化するアプローチをとることができる．複数のエージェントサーバからなる分散マルチエージェントシステムを構築することで，システムが使用できるリソースを増加させることができる．その結果，システムのリソース不足による性能の低下を解決することができる．こうしたシステムの分散配置は，分散 OS やワークステーションのクラスタ化に適用されており，その有効性が示されている [6]．

しかし，エージェントサーバを分散配置する場合，分散 OS などに適用されてきた手法では不十分である．なぜならば，マルチエージェントシステム上では，エージェント間で発生するインタラクションを考慮する必要があるため

ある．マルチエージェントシステムにおいては，エージェントが必要に応じて他のエージェントとインタラクションを行う．インタラクションを行う相手が異なるエージェントサーバに存在する場合，サーバ間でメッセージを交換してインタラクションを実現する必要がある．そのため，互いに異なるエージェントサーバに存在するエージェント間で発生するインタラクションは，サーバ間でメッセージを交換する分，同じサーバに存在するエージェント間で行うインタラクションよりもコストが大きくなる．インタラクションが頻繁に発生する場合，このようなインタラクションコストの増加を無視することはできない．従って，分散マルチエージェントシステムのインタラクションコストを小さくするために，頻繁にインタラクションを行うエージェント同士を同じサーバに配置する必要がある．

本研究では上記の問題を，各エージェントサーバの計算負荷が閾値より小さく，エージェントサーバ間で発生するインタラクションコストを最小にするエージェントの配置を求めるエージェント配置問題として定式化する．定式化したエージェント配置問題の最適解を求めることは困難であるため，本研究ではエージェント配置問題に対する近似解を求める手法を提案する．本研究では，提案手法によるシステムのインタラクションコストの変化を検証するために，シミュレーションを行った．このシミュレーションでは，エージェント配置に関する異なるアルゴリズムを適用した場合の，インタラクションコストの比較を行った．

続いて，マルチエージェントシステムである大規模誘導システムを複数のエージェントサーバで構成する際の実装について述べる．分散マルチエージェントシステムを構築する際にシステムを中心となる機能が存在すると，その機能がシステムがボトルネックになる可能性があるなど，システムを分散化したことによる利益を損なう．そこで，本研究では大規模誘導システムを取り上げ，中心となる機能を必要とせずに複数のエージェントサーバを接続し，各エージェントサーバが独立してエージェントの配置を決定するための手法を述べる．

本稿は，以下のように構成されている．2章ではエージェントの配置問題を定式化し，その問題に対する近似解を求める手法を定める．3章では，本手法を用いることによるインタラクションコストの変化を検証するためのシミュレーションについて述べる．4章では，本研究の応用例として大規模誘導システムを分散化する実装について述べる．5章では，本研究の結論と今後の課題について述べる．

## 第2章 分散環境におけるエージェント配置

複数のエージェントサーバにエージェントを分散することで負荷分散を行う際に、エージェント配置について考慮すべき事項は、エージェントサーバの計算負荷とエージェントサーバ間のインタラクションコストの2点である。システム全体の負荷を分散マルチエージェントシステム上で分散するためには、各サーバの計算負荷が小さくなるようにエージェントを配置する必要がある。一方で、サーバ間で発生するインタラクションコストを小さくするために、インタラクションが頻繁に発生するエージェントを同じサーバに配置する必要がある。本章ではまずこれらの問題を定式化し、それに対する近似解を用いる方法を示す。そして、本研究が提案する手法の特徴について述べる。

### 2.1 問題の定式化

本節では、まずエージェント配置問題について述べ、その定式化を行う。

#### 2.1.1 エージェント配置問題

本研究で考えるマルチエージェントシステムは、ネットワーク上で接続された  $m$  台のエージェントサーバに  $n$  体のエージェントが分散配置されているものとする。このマルチエージェントシステム上では、エージェントは必要に応じて相互にインタラクションを行う。互いに異なるサーバに存在するエージェント間のインタラクションは、互いのエージェントが存在するサーバ間でメッセージを交換することで実現可能である。各エージェントサーバの性能は同じであると仮定し、エージェントサーバ間の接続は全て等価であると仮定する。このようなシステムにおいて達成すべき事柄は、以下の2点である。

1. 各エージェントサーバにエージェントの計算負荷を分散する
2. エージェントサーバ間で発生するインタラクション頻度を少なくする

まず、エージェントの計算負荷を各エージェントサーバに分散する必要がある。エージェントは、存在しているエージェントサーバの計算リソースを消費して動作を行う。1台のエージェントサーバが提供できるリソースは有限であるため、必要となるリソースがエージェントサーバが提供できるリソースを超えると、サーバの性能は低下する。そのため、あるエージェントサーバに対してエージェントが集中すると、そのサーバがボトルネックとなりシステムの性能が低下する。従って、各エージェントサーバの性能が大きく低下しないよう、

表 1: 表記の定義

$a_i$	エージェント $i(i = 1, 2, \dots, n)$
$s_i$	エージェントサーバ $i(i = 1, 2, \dots, m)$
$w_i$	エージェント $a_i$ の計算負荷
$W_i$	エージェント移動前の $s_i$ の計算負荷
$W'_i$	エージェント移動後の $s_i$ の計算負荷
$p(a_i, a_j)$	$a_i$ と $a_j$ の間のインタラクション頻度
$A_i$	エージェント移動前の $s_i$ に存在するエージェント集合
$A'_i$	エージェント移動後の $s_i$ に存在するエージェント集合
$T_h$	エージェントサーバの計算量閾値
$c_i$	Comet アルゴリズムにおける $a_i$ の評価値
$gain_i$	本手法における $a_i$ の評価値
$m(a_i)$	$a_i$ の $gain_i$ が得られるエージェントサーバのインデックス

エージェントを分散配置する必要がある。

次に、エージェントサーバ間で発生するインタラクションコストを抑える必要がある。エージェントが複数のエージェントサーバに分散する場合、エージェント間のインタラクションコストは増加する。あるエージェントが異なるサーバに存在するエージェントに対してインタラクションを行う場合、インタラクションのためのメッセージをエージェントサーバ間で通信する必要がある。つまり、異なるエージェントサーバに存在するエージェント間でインタラクションを行う場合、サーバ間でメッセージを交換する分、同じエージェントサーバに存在するエージェント間で行うインタラクションよりもコストが大きくなる。こうしたインタラクションコストの増加を抑えるために、頻繁にインタラクションを行うエージェント同士を同じサーバに配置し、サーバ間で発生するインタラクションの頻度を少なくする必要がある。

### 2.1.2 エージェント配置問題の定式化

表 1 の表記に基づいて、エージェント配置問題を以下のように定式化する。

エージェントサーバ  $s_i$  の計算負荷  $W_i, W'_i$  は、(1) と (2) に示すように、そのサーバが保持しているエージェント  $a_j$  の計算負荷  $w_j$  の総和で与えられると

する .

$$W_i = \sum_{a_j \in A_i} w_j \quad (1)$$

$$W'_i = \sum_{a_j \in A'_i} w_j \quad (2)$$

ここで , エージェントサーバ  $s_1$  が過負荷であり , 他のエージェントサーバ  $s_i (i = 2, 3, \dots, m)$  は過負荷でないとする . そして , 過負荷のエージェントサーバから過負荷でないエージェントサーバにエージェントを移動することで , 各エージェントサーバの計算負荷を閾値以下にすることができるものとする . つまり , 初期条件として (3) , (4) , (5) が満たされているものとする .

$$W_1 > T_h \quad (3)$$

$$W_i < T_h (i = 2, \dots, m) \quad (4)$$

$$\sum_{1 \leq i \leq m} W_i < mT_h \quad (5)$$

このとき , 過負荷のエージェントサーバ  $s_1$  から他のエージェントサーバ  $s_i (i = 2, 3, \dots, m)$  にエージェントを移動して , 負荷の分散を行うことを考える . エージェントサーバ  $s_1$  から  $s_i$  に移動するエージェントの集合を  $M_{1,i}$  とおくと , エージェントを移動した後の各サーバが保持するエージェント集合  $A'_i$  は , (6) 式で与えられる .

$$A'_i = \begin{cases} A_1 - \sum_{2 \leq j \leq m} M_{1,j} & (i = 1) \\ A_i \cup M_{1,i} & (i = 2, 3, \dots, m) \end{cases} \quad (6)$$

上記を用いて , エージェントサーバの負荷を均等にし , かつエージェントサーバ間で発生するインタラクション量を抑えるという目的は (7) , (8) 式として定式化することができる .

$$\min \sum_{a_i \in A'_k \wedge a_j \in A'_l (k < l)} p(a_i, a_j) \quad (7)$$

$$s.t. \quad W'_i < T_h \quad (i = 1, 2, \dots, m) \quad (8)$$

(7),(8) 式を満たす  $M_{1,i} (i = 2, 3, \dots, m)$  が , エージェント配置問題における最適なエージェント移動集合である .

$|A_1|$  が大きい場合，全ての配置の中から (7) 式を満たすエージェントの移動を決定することは困難である．なぜならば， $|A_1|$  体のエージェントに対して，エージェントの移動集合の選び方は  $m^{|A_1|}$  通り存在するからである．そこで次節で，エージェント配置問題に関する近似アルゴリズムを定める．

## 2.2 エージェント配置問題に対する近似解

本研究では，エージェント配置問題の近似解を求めるための手法に，k-way FM 法 [7] のアイデアを利用する．k-way FM 法は，Fiduccia と Mattheyses が提案した近似アルゴリズム (以下 FM 法) [8] を複数のグラフへの分割に拡張したアルゴリズムである．FM 法はネットワーク分割問題に対する近似アルゴリズムであり，このアルゴリズムを複数の分割に拡張したものが k-way FM 法である．k-way FM 法では，与えられた初期分割から分割の均衡を崩さないように各節点の移動を繰り返し行い，その繰り返しの中でネットワーク分割のコストが最も良い場合を選択する手順を反復する．

過負荷になったエージェントサーバ  $s_1$  は，以下の情報が取得できるとする．

- 他のサーバの計算負荷  $W_i (i = 2, 3, \dots, m)$
- 内部エージェントの計算負荷  $w_i$  (ただし， $a_i \in A_1$ )
- 内部エージェント間のインタラクション頻度  $p(a_i, a_j)$  (ただし， $a_i, a_j \in A_1$ )
- 各内部エージェントと外部サーバ間のインタラクション頻度

FM 法を適用するにあたり，エージェント  $a_i$  に対する評価値として  $gain_i$  を (9) 式で定義する．

$$gain_i = \max_{a_i \notin A'_j \wedge W'_j + w_i < T_h} \sum_{a_k \in A'_j} p(a_i, a_k) - \sum_{a_l \in A'_1} p(a_i, a_l) \quad (9)$$

$gain_i$  は，均衡を崩さずに  $a_i$  を移動できるエージェントサーバの中で， $a_i$  が最もインタラクションを行うサーバに移動した場合の，インタラクションコストの減少量を表す． $a_i$  に関して， $gain_i$  が得られるエージェントサーバのインデックスを  $m(a_i)$  とする．

本手法の具体的なアルゴリズムを，Algorithm 1 に示す．

本手法は，大きく 2 つのフェーズからなる．1 つめのフェーズは初期配置フェーズであり，2 つめのフェーズは初期配置からさらにインタラクションコストを減少させるフェーズである．

以下，全てのエージェントとは，初期状態において過負荷のエージェントサー

---

**Algorithm 1** 本手法のアルゴリズム

---

**Require:**  $W_1 > T_h, W_i < T_h (i = 2, 3, \dots, m), \sum_i W_i < mT_h$

/\* initial placing \*/

**while**  $W_1 > T_h$  or  $\max_i gain_i > 0$  **do**

    select  $a_i (a_i \in A'_1 \wedge gain_i = \max_{a_j \in A'_1} gain_j)$

    move  $a_i$  to  $M_{1,m(a_i)}$

**end while**

/\* improve cost loop \*/

**while** *mincost* updates **do**

    Unmark all agents

**while** Unmarked and movable agents exist **do**

        Compute  $gain_i$  of each unmarked agents

        Find  $a_i$  with max  $gain_i$  from unmarked agents

        Move  $a_i$  into  $M_{1,m(a_i)}$  and mark

**end while**

    Find index that minimize *cost* in the above loop and the minimize cost is set to *tmpmincost*

**if** *mincost* > *tmpmincost* **then**

*mincost* = *tmpmincost*

**end if**

**end while**

**return**  $M_{1,m(a_i)} (i = 2, 3, \dots, m)$ , which gives *mincost*

---

バ  $s_1$  に存在しているエージェントを指す．

初期配置フェーズでは，制約条件 (8) を満たすようにエージェントの配置を決定する． $s_1$  が過負荷の状態では，計算負荷に関する制約条件である (8) 式は満たされていない．そこで (8) 式を満たすような配置で，インタラクションコストが小さい配置を初期配置として求める．そのために， $s_1$  の計算負荷が閾値よりも大きく，かつ  $gain_i > 0$  を満たすエージェントがいる限り， $A'_1$  に含まれて  $gain_i$  が最大のエージェント  $a_i$  を  $M_{1,m(a_i)}$  に対して移動する．

次のフェーズは，初期配置フェーズから得られたエージェントの配置を更新することで，インタラクションコストをさらに小さくするフェーズである．印がつ

いていないエージェントの中で， $gain_i$  が最も大きいエージェント  $a_i$  を  $M_{1,m(a_i)}$  に移動して印をつける．全てのエージェントに印がつくか，または移動可能なエージェントが存在しなくなるまでこの操作を反復する．そしてその反復の中で，最もシステム全体のインタラクションコストが小さな配置を選択し，一時最小コストとして記録する．最小コストが一時最小コストによって更新されれば，全てのエージェントの印を外し，反復を繰り返す．最小コストが更新されなければ，その最小コストの配置を解として返す．

以上の操作で得られた  $M_{1,i}(i = 2, 3, \dots, m)$  に基づいて，エージェントの移動を行う．

本研究では，以上の手順をネットワーク分割問題に対する局所最適解を求める k-way FM 法のアイデアを用いて定めた．このアルゴリズムのアイデアを利用した動機として，エージェント配置問題がネットワーク分割問題と類似している点が挙げられる．ネットワーク分割問題とは，ネットワークで接続された節点を指定数の集合に分割し，分割した集合間をまたぐネットワークのコストの総和を最小にする問題である．この問題は，エージェントを複数のエージェントサーバに分割し，エージェントサーバ間のインタラクションコストを最小にするエージェント配置問題に類似している．そのため，ネットワーク分割問題に対する近似アルゴリズムである k-way FM 法のアイデアを利用することができた．ただし，k-way FM 法は，局所最適を探索することでネットワーク分割問題における近似解を求めるため，その解の精度を評価することはできない．従って，本手法も同様に求められる近似解の精度を保証することはできない．そこで，本手法を適用することによるシステムのインタラクションコストの変化を検証するために，シミュレーションを行った．このシミュレーションについては，3章で述べる．

## 2.3 特徴

本節では，本手法の分散性，評価値，エージェントの集合による移動という観点に基づき，本手法の特徴を考察する．

### アルゴリズムの分散性

本手法を用いる際には，各エージェントサーバが独立に動作することが可能である．本手法を過負荷のエージェントサーバに対して適用する場合，必要となる情報は，1) 内部エージェントの計算負荷，2) 内部エージェント間のインタラ

クション, 3) 内部エージェントと外部サーバのインタラクション, 4) 外部サーバの計算負荷の4項目である. 1), 2), 3) に関しては, 各エージェントサーバが, 内部のエージェントを監視することで取得することができる. 4) に関して, 周辺のサーバに対する問い合わせで取得することができる. 従って, 本手法は分散配置された各エージェントサーバが, エージェントの移動に関して独立に動作することを可能にする. この特徴は, エージェントの移動に関して中心となるシステムが必要ではないという点で, 分散マルチエージェントサーバにおいて有効である.

#### 評価値

本手法では, エージェントに対する評価値に, エージェントが移動することによるインタラクションコストの変化量を用いている. エージェント  $a_i$  をエージェントサーバ  $s_{from}$  から  $s_{to}$  に移動するときのインタラクションコストの変化量は (10) 式で与えられる.

$$\sum_{a_j \in A'_{from}} p(a_i, a_j) - \sum_{a_k \in A'_{to}} p(a_i, a_k) \quad (10)$$

本手法では, (10) 式に基づき, エージェント  $a_i$  に対する評価値として, 移動しても計算負荷が閾値を越えないエージェントサーバの内,  $a_i$  が最もインタラクションを行うサーバに対して移動した時のインタラクションコストの減少量として,  $gain$  を定義した. 従って,  $gain$  が最大のエージェントは, 移動可能なエージェントの中で, 移動によってインタラクションコストを最も減少させることができるエージェントである. そうしたエージェントを選択することで, より移動後のインタラクションコストを小さくすることができる.

#### エージェント集合による移動

本手法では, エージェントの配置を決定するために, 移動するエージェント集合を考えている. 通常, 大規模なマルチエージェントシステムにおいては, 過負荷なエージェントサーバから他のサーバに負荷を分散するためには, 複数のエージェントを移動することが必要である. 複数のエージェントを移動する場合, 移動するエージェントの集合を考えることで, 単体ごとにエージェントを移動した場合よりも, インタラクションコストを小さくすることができる. 例えば, 評価値に基づいてエージェントとその移動先を順に選択していく場合, 他のエージェントの移動先が決まるにつれて, 最初に選択された移動先よりも, よりインタラクションコストが小さくなるような移動先が見つかる場合がある.

移動するエージェントを集合として考えることで，こうした場合に対応することができる．

## 第3章 エージェント配置シミュレーション

本章では，本手法の有効性を検証するために行ったシミュレーションについて述べる．このシミュレーションは，本手法を含む3つのエージェント配置手法を分散マルチエージェントシステムに対して適用し，インタラクションコストの変化を測定したものである．

### 3.1 シミュレーション内容

このシミュレーションは，複数のエージェントサーバを接続した分散マルチエージェントシステムをシミュレートしたものである．このシミュレーションでは，エージェント数  $n = 10,000$ ，エージェントサーバ数  $m = 5$  としている．各エージェント間のインタラクション頻度  $p(a_i, a_j)$  は，ランダム値を定める．まず初期配置として，各エージェントサーバの計算負荷  $W_i$  が均等になるようにエージェントの配置を行う．この初期配置は，エージェント移動のために適用する各アルゴリズムに対して共通である．そして，シミュレーションの開始後，まず各エージェントの計算負荷  $w_i$  をランダム値で更新する．エージェントが多数存在するため，各エージェントサーバごとに平均となる負荷をランダム値で定め，各エージェントにはサーバの負荷平均値に応じた計算負荷をランダムで与える．エージェントの計算負荷を定めた後，計算負荷が閾値  $T_h$  を超えたエージェントサーバに対して各手法を適用し，負荷の分散を行う．そして，(11) によって与えられる移動後のシステム全体のインタラクションコストの変化を記録する．

$$\sum_{a_i \in A'_k \wedge a_j \in A'_l (k < l)} p(a_i, a_j) \quad (11)$$

以上の操作を1回の反復として，100回反復した際のシステム全体のインタラクションコストの変化を比較する．

エージェント移動の際に適用したアルゴリズムは，以下の3手法である．

- WBLB(workload based load balance)
- Comet アルゴリズム [9]
- 本手法

---

**Algorithm 2** WBLB(workload based load balance)

---

**Require:**  $W_1 > T_h, W_i < T_h (i = 2, 3, \dots, m), \sum_i W_i < mT_h$ compute the load  $w_i$  of each agent**while**  $W_1 > T_h$  **do**    Find  $a_i$  ( $w_i = \max_{a_j \in A_1} w_j$ )    Move  $a_i$  to  $s_j$  ( $W'_j = \min_{k \neq 1} W'_k$ )**end while**

---

---

**Algorithm 3** Comet Algorithm

---

**Require:**  $W_1 > T_h, W_i < T_h (i = 2, 3, \dots, m), \sum_i W_i < mT_h$ compute the credit  $c_i$  of each agent and compute the load on each host;**while**  $W_1 > T_h$  **do**    select  $a_i$  ( $c_i = \min_{a_j \in A_1} c_j$ )    move  $a_i$  to  $s_j$  ( $\sum_{a_k \in A_j} p(a_i, a_k) = \max_{m_k} \sum_{a_l \in A_k} p(a_i, a_l)$ )**end while**

---

### 3.1.1 WBLB アルゴリズム

Algorithm 2 に、WBLB の具体的なアルゴリズムを示す。

WBLB アルゴリズムは、エージェントの計算負荷のみを考慮し、各エージェントサーバの計算負荷が均等になるようにエージェントの配置を行うアルゴリズムである。この手法におけるエージェント移動の手順は、過負荷のサーバに存在するエージェントのうち最も計算負荷が高いエージェントを、最も計算負荷が小さいエージェントサーバに対して移動するものである。過負荷のエージェントサーバの計算負荷が閾値を下回るまでこの操作を反復することで、負荷の分散を行う。

### 3.1.2 Comet アルゴリズム

Algorithm 3 に、具体的なアルゴリズムを示す。

文献 [9] が提案する Comet アルゴリズムは、インタラクション頻度に基づいたエージェント配置アルゴリズムである。Comet アルゴリズムは定期的に各エージェントサーバの情報を収集する。そして、あるエージェントサーバが過負荷になった際に、過負荷のサーバが保持するエージェントの中から移動するエージェントを選択し、どこのサーバに移動するのかを決定する。

Comet アルゴリズムでは，どのエージェントを移動するのかを決定する際に，(12) 式で定義される credit 値と呼ばれる評価値を用いる．

$$c_i = -x_1 w_i + x_2 g_i - x_3 h_i \quad (12)$$

ここで  $g_i$  はエージェント  $a_i$  が同じサーバに存在しているエージェントに対して行うインタラクション頻度であり， $g_i = \sum_{a_j \in A_1} p(a_i, a_j)$  で定義される． $h_i$  は異なるサーバに対して  $a_i$  がインタラクションを行う頻度であり， $h_i = \sum_{a_j \in A_k (2 \leq k \leq m)} p(a_i, a_j)$  で定義される．そして， $x_i (i = 1, 2, 3)$  は各要素に対する重みである．この credit 値はエージェントとそれが属するエージェントサーバとのつながりを表す．Comet アルゴリズムでは，この値が最小のエージェントを移動するエージェントとして選択する．移動するエージェントとして選択したエージェントは，そのエージェントが最もインタラクションを行うエージェントサーバに対して移動する．

以上の操作を，過負荷のエージェントサーバの計算負荷が閾値を下回るまで繰り返すことで，エージェントサーバの負荷を分散する．

### 3.2 シミュレーション結果

図 1 は適用手法ごとに 10 回ずつシミュレーションを行い，その結果の平均値を記録したグラフである．このグラフの縦軸はエージェントサーバ間で発生するインタラクションコストであり，(11) 式で与えられる．横軸は，シミュレーションにおける反復回数を表している．このグラフ中には，WBLB アルゴリズム，Comet アルゴリズム，本手法をそれぞれ適用した場合のインタラクションのコストを反復回数ごとに記録している．

まず図 1 より，WBLB アルゴリズムを用いた場合，インタラクションコストはほとんど変化していない．これは，初期配置が計算負荷のみを考慮して行われ，WBLB アルゴリズムが同様に計算負荷のみを考慮しているためである．

次に図 1 より，本研究の提案する手法を適用することで，Comet アルゴリズムと比べて少ない反復回数でインタラクションコストがより減少する．初期状態からのインタラクションコストの減少量を比較すると，本手法を適用した場合に反復回数が 19 回目までは 2 倍以上，インタラクションコストが減少している．このシミュレーションの反復を 1000 回行った場合，Comet アルゴリズムのインタラクションコストは 11274.35，本手法は 10460.58 となり，初期配置からのインタラクションコストの減少量に 9.3%の改善が見られた．このようにイ

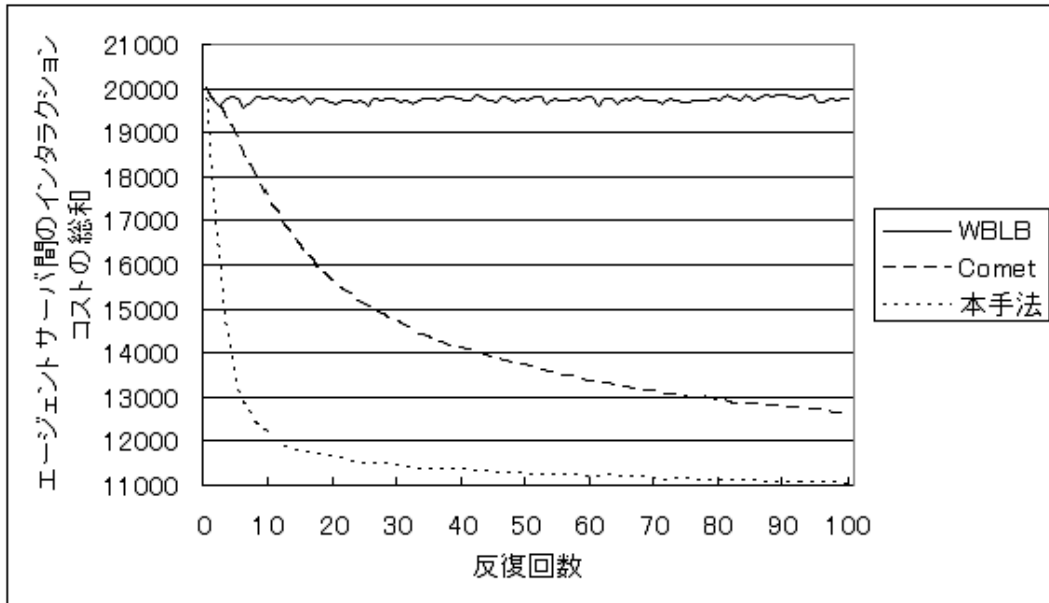


図1: インタラクションコストの変化

インタラクションコストが少ない反復で減少した理由として、次の2点が挙げられる。

まず、本手法の特徴であるエージェントの評価方法が挙げられる。Comet アルゴリズムでは、エージェントの計算負荷と内部・外部のインタラクションを基に評価値を定めている。一方で、本手法では、各エージェントを移動した場合のインタラクションコストの減少量を基に評価値を定めている。こうした評価値の違いによりエージェントの選択が異なる場合の一例を、図2に示す。図2において、(a)と(b)は同じ計算負荷のエージェントであり、それぞれ一定期間内でインタラクションを行う回数が矢印の添え字で表されている。(a)のエージェントを、最もインタラクションを行うサーバBに対して移動する場合、インタラクションのコストは2だけ減少する。一方で、(b)のエージェントをサーバEに移動した場合、インタラクションのコストは6減少する。従って、移動するのであれば、(b)を選択しサーバEに移動した方が、全体のインタラクションコストは小さくなる。本手法において、(a)を移動する場合、サーバBに移動した時に最もインタラクションコストが小さくなるため、 $gain_a = 4 - 2 = 2$ であり、 $m(a_{(a)}) = B$ である。(b)を移動する場合、サーバEに移動した時に最もインタラクションコストが小さくなるため、 $gain_b = 8 - 2 = 6$ であり、 $m(a_{(b)}) = E$ である。従って、本手法では  $gain$  が大きい (b) を選択し、サーバEに対して移動

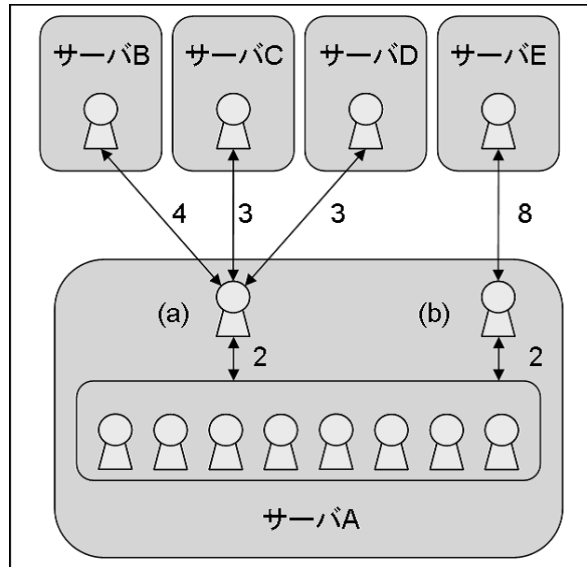


図 2: Comet アルゴリズムと本手法の比較

する．次に Comet アルゴリズムにおける各エージェントの credit 値を考察する．(a) の credit 値は， $c_{(a)} = -x_1 w_{(a)} + 2x_2 - x_3(4 + 3 + 3) = -x_1 w_{(a)} + 2x_2 - 10x_3$  となる．(b) の credit 値は， $c_{(b)} = -x_1 w_{(b)} + 2x_2 - 8x_3$  となる． $w_{(a)} = w_{(b)}$  より  $c_{(a)} \leq c_{(b)}$  であるため，(a) を選択し，サーバ B に対して移動する．従って，Comet アルゴリズムでは (a) が選択される．なぜならば，(a) と (b) のエージェントは同じサーバに対するインタラクションは同じ回数であり，異なるサーバに対するインタラクションは (a) の方が高いからである．このように，どのエージェントを移動するのかを選択する際に，インタラクションコストの減少量に注目して移動するエージェントを選択することで，インタラクションコストを小さくするために最適なエージェントを選択することが有効である．

次に，本手法の特徴である，移動するエージェント集合を決定しているという点が挙げられる．Comet アルゴリズムでは，エージェントに対する評価値を基にして，移動するエージェントとそのエージェントの移動先を順に決定する．一方で，本手法では，過負荷となったエージェントサーバの計算負荷が閾値を下回り，かつインタラクションコストが小さくなるエージェントの移動集合を求める．

以上の特徴が，このシミュレーションにおけるインタラクションコストの差に現れたといえる．

しかし，インタラクションコストが減少する一方で，エージェントの配置を

決定するために必要となる計算量は大きくなる．Comet アルゴリズムは，移動可能なエージェントについて credit 値を求めた後，credit 値の小さなエージェントから移動を決定している．この操作に必要な計算時間は  $O(n \log n)$  である．一方で，本手法は各反復で全てのエージェントの gain 値を再計算すると，エージェントの移動を決定するのに  $O(n^2)$  時間を必要とする．従って，エージェントの配置に必要な計算量は大きくなる．

## 第4章 大規模誘導システムへの適用

本手法を適用する具体的なアプリケーションとして，大規模誘導システムを複数のサーバに対して分散する実装を行った．本章では，スタンドアロン型に実装された大規模誘導システムに対して，本研究が提案する手法をどのように適用したのかについて述べる．

### 4.1 大規模誘導システム

大規模誘導システムは，都市規模の利用者を想定した大規模マルチエージェントシステムである．このシステムの目的は，利用者に対して利用者の特徴に応じた誘導サービスを提供することである．携帯電話を所持した利用者が，GPS によって得た位置情報をシステムに送信することで，利用者の位置や特性に応じた誘導を行う．

このシステムは，IBM が開発したマルチエージェントフレームワークである Caribbean[10][11] 上で開発された．Caribbean 上では，大量のエージェントを扱うシステムにおいてもメモリ管理やスレッド管理を意識することなく，システムの開発を行うことができる．図3に，Caribbean の概観を示す．Caribbean は，Agent と MessageManager，ServiceObject からなる．Agent はオブジェクト ID によって一意に識別され，自律的に動作することで利用者にサービスを提供するなどの目的を達成する．MessageManager は，Agent にメッセージ機能を提供するためのオブジェクトである．MessageManager によって，Agent が他の Agent に対して非同期のメッセージ送信を行うことや，メッセージを複数の Agent に対して送信することが可能である．ServiceObject は，Agent が実行時に必要となるサービスを提供するオブジェクトである．

大規模誘導システムは，Caribbean が提供するエージェントフレームワーク

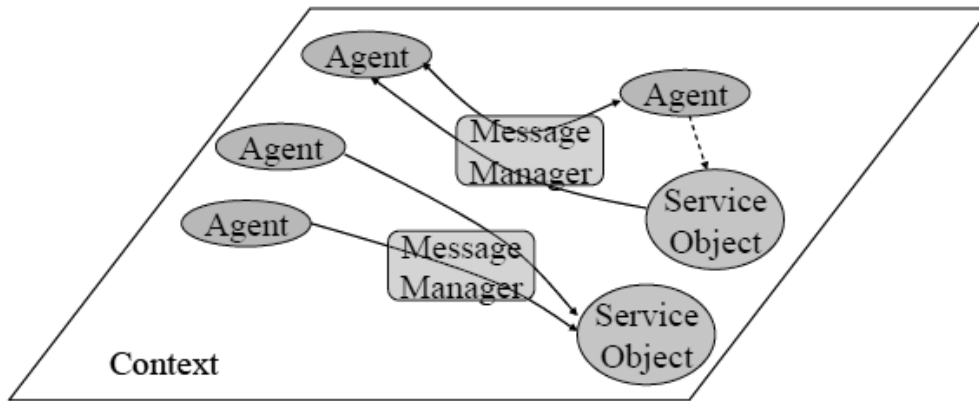


図 3: Caribbean の概観

(Gaku Yamamoto. Agent server technology for managing millions of agents.  
In Ishida, et al. [4], pp. 1-12.)

上で設計・実装されたマルチエージェントシステムである。このシステムについて、図 4 に大規模誘導システムの配置図を示す。大規模避難誘導システムは次の 4 つの要素からなる。

#### 携帯端末

利用者が誘導システムにアクセスする場合、GPS 付きの携帯電話を用いる。GPS によって取得する位置情報を、Web ブラウザを用いて HTTP により送信することで、誘導情報を取得することができる。

#### HTTP サーバ

利用者からの HTTP アクセスを、Java RMI によって誘導サーバに転送する。そして誘導サーバから取得した誘導情報を、Web ページとして携帯端末に送信する。

#### 誘導サーバ

誘導を行うための中心となる要素であり、Caribbean 上で設計・開発が行われている。HTTP サーバから受け取った利用者からのリクエストに対して、誘導情報を提供する。

#### 地図データベース

誘導情報に必要な地図データを管理するデータベースである。誘導システムや、管制アプリケーションから JDBC によって接続される。

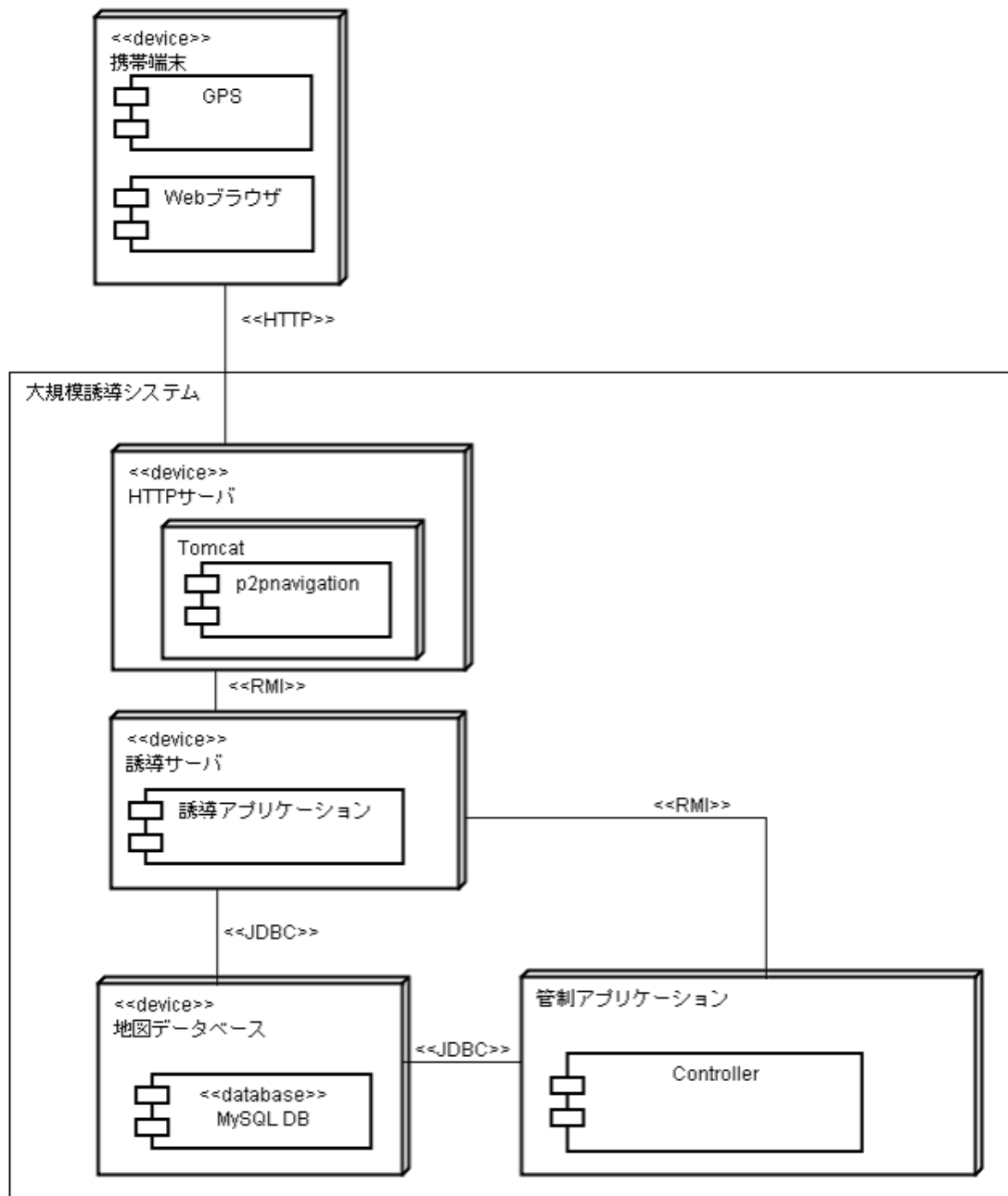


図 4: 大規模誘導システムの配置図  
(SCOPE メガナビゲーションプロジェクト)

### 管制アプリケーション

誘導を行う管制者に対して、誘導システムの情報と地図データベースの情報を基にして、誘導の状況を提示する。誘導システムと Java RMI によって接続しており、管制者から利用者に対して指示を送ることができる。

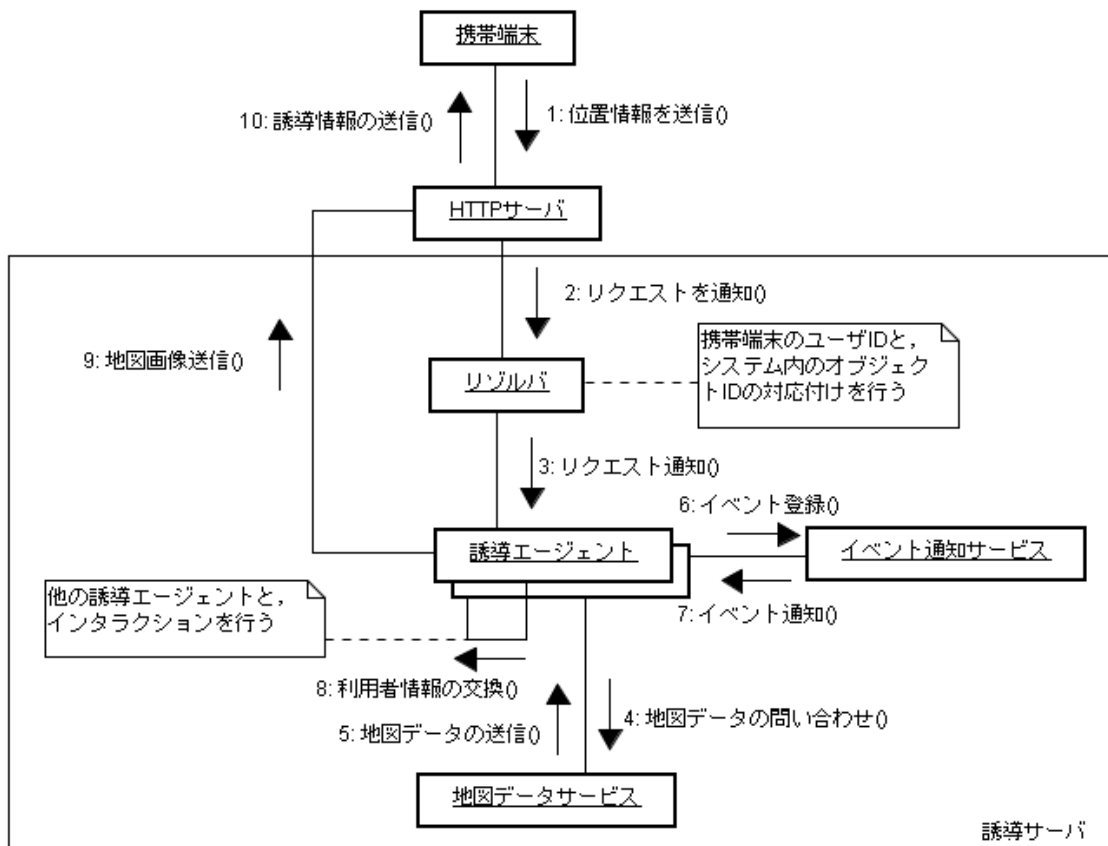


図 5: 携帯端末がアクセスする際のコラボレーション図  
(SCOPE メガナビゲーションプロジェクト)

図 5 に、利用者が誘導システムにアクセスする際のコラボレーション図を示す。

1) 利用者は携帯端末の GPS によって得られた位置情報を、Web ブラウザを用いて HTTP サーバに対して送信する。2) 利用者からのリクエストを受信した HTTP サーバは、Java RMI によって誘導サーバにリクエストを転送する。3) 誘導サーバが HTTP サーバからリクエストを受信すると、まずリゾルバに対してリクエストを送信する。リゾルバでは、携帯端末の利用者 ID とシステム上の誘導エージェントの対応が取られる。リクエストを受信した誘導エージェントは、必要に応じて以下の処理を行う。利用者に対して送信する地図を更新する必要がある場合、4) 地図データサービスに対して地図データ問い合わせる。5) 地図データサービスは外部の地図データベースに接続されており、問い合わせ

に応じて地図データを誘導エージェントに送信する．6) アクセスにより，登録すべきイベントをイベント通知サービスに対して登録する．そして，7) 利用者に対して通知すべき情報を，イベント通知サービスから取得する．8) 他のエージェントの情報が必要な場合，そのエージェントとインタラクションを行うことで情報を交換する．9) 以上の結果，利用者に対する地図画像を作成し，HTTPサーバに対して送信し，10) 利用者に対して誘導情報を送信する．以上の操作で利用者に対して誘導サービスを提供する．

しかし，大規模な利用者を想定する場合，スタンドアロンなシステムでは，利用者の数が増加するにつれてシステムの性能が低下する．システムが扱うエージェントの数に比例してシステムの負荷は増加し．より多くのリソースが必要になる．しかし，1台のエージェントサーバが提供できるリソースは有限であるため，システム性能が低下することは避けられない．性能が低下することで，利用者にリアルタイムなサービスを提供することが困難になる．Caribbeanの性能評価 [12] では，全てのエージェントをメモリ上に置けなくなると，極端に性能が低下する事が報告されている．

そこで本研究では，複数のエージェントサーバを接続することで，システムの負荷を分散するアプローチを取る．複数のサーバにエージェントが分散することで各サーバの負荷が減少し，システム全体の性能が向上する．しかし，エージェントが分散する場合，異なるエージェントサーバ間でインタラクションが発生する．異なるエージェントサーバに存在するエージェント間でインタラクションを実現するためには，まずインタラクションを行う相手の位置を特定し，メッセージを互いに交換する必要がある．そのため，エージェントをシステム上で分散配置することで，システムのインタラクションコストは増加する．そこで，このコストの増加を小さくするために本手法を適用して，エージェントの配置を決定する．

## 4.2 本手法の適用

大規模誘導システムを分散配置するためのアーキテクチャを，図6に示す．大規模誘導システムを分散化し，複数の誘導サーバをネットワーク上で接続できるように拡張するために追加した機能は，1) システムの監視，2) ネットワークサービスの2点である．

システムの監視を行うために，リゾルバにエージェントに対するアクセス頻

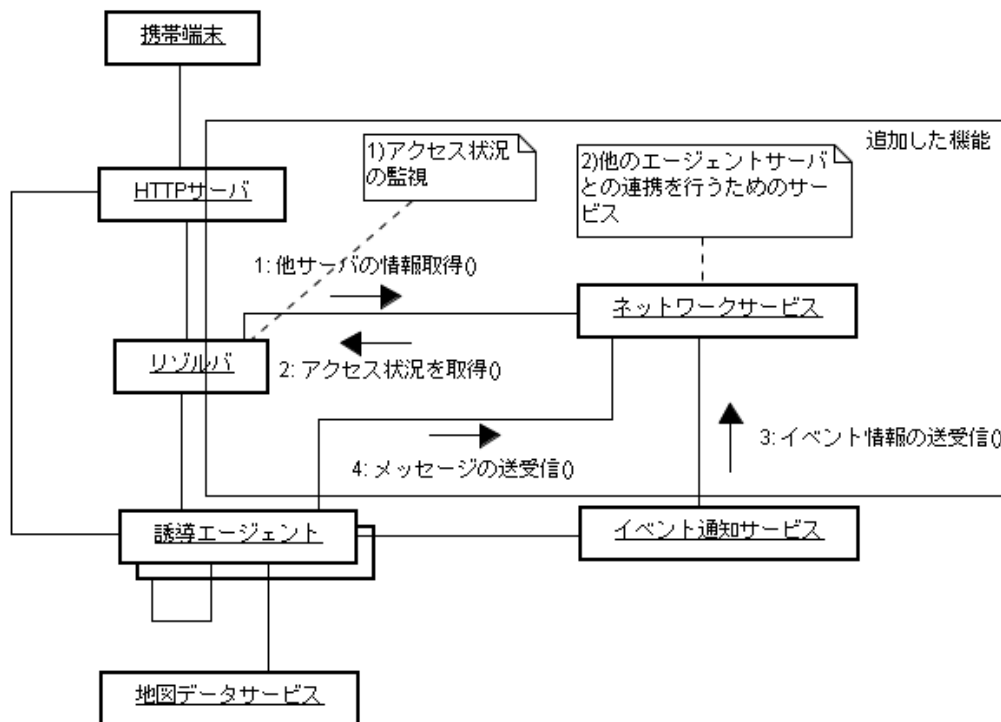


図 6: 大規模誘導システムを分散する際に追加する機能

度とエージェント間のインタラクションに関する情報を保持する機能を追加した。本研究が提案する手法を適用する際に、エージェントの動作頻度と、エージェント間で発生するインタラクション頻度を取得する必要がある。リゾルバは、利用者が誘導エージェントに対してアクセスする場合やエージェント間でインタラクションを行う場合に、利用者 ID と誘導エージェントのオブジェクト ID の対応関係を解決するために利用されている。従って、リゾルバに対する問い合わせを監視することで、エージェントに対するアクセス頻度とエージェント間のインタラクション頻度を取得することが可能である。

他のサーバとの連携を行うための ServiceObject であるネットワークサービスが、Caribbean 上のオブジェクトに対して提供する機能として、以下の 4 点が挙げられる。

1. 他サーバの情報を提供

リゾルバがユーザ ID とオブジェクト ID の対応を解決する際、ユーザ ID に対応するエージェントが内部に存在しなければ、そのエージェントがどのサーバに存在するのかを解決する必要がある。ネットワークサービスは他

のサーバにエージェントの問い合わせを行い，エージェントの位置を特定してリゾルバに対して提供する．

#### 2. 定期的にシステム負荷を監視

ネットワークサービスは，定期的にリゾルバに対してアクセス状況を問い合わせ，システムの負荷を監視する．システムが過負荷になっている場合は，リゾルバが収集するアクセス状況を基に本手法を用い，エージェントを他のサーバに移動する．

#### 3. エージェント間のメッセージの送受信

内部のエージェントが，異なるエージェントサーバに存在するエージェントに対してインタラクションを行う場合，エージェントサーバ間でメッセージを交換してインタラクションを実現する必要がある．ネットワークサービスは，他のエージェントサーバに存在するエージェントに対してメッセージを送信するための機能を提供する．

#### 4. イベント情報の送受信

イベント通知サービスに登録されたイベント情報を，定期的にネットワークサービスに登録することで，周辺のエージェントサーバとの情報共有を行う．また，ネットワークサービスが他のサーバから受信したイベント情報をイベント通知サービスに登録することで，内部のエージェントにその情報を利用させることができる．

本研究では以上の機能を実装することで，大規模誘導システムにおいて複数のエージェントサーバを接続可能にした．

## 第5章 おわりに

本研究では，分散マルチエージェントシステムにおけるエージェント配置に関する以下の2点についての議論を行った．

#### 1. システムの性能を改善するためのエージェント配置

エージェントが複数のエージェントサーバ上で分散配置される場合，システムのインタラクションコストが増加する．そのため，エージェントの計算負荷のみを考慮してエージェントの配置を行うと，インタラクションコストが大きくなり，効率的にシステムの性能を改善することができない．従って，分散マルチエージェントシステム上でエージェントを分散配置する際，

エージェントの計算負荷とエージェント間のインタラクション頻度を考慮して、エージェントを配置する必要がある。

## 2. エージェントサーバを分散する際の実装方法

エージェントの配置を決定する手法の実装方法も、重要な課題である。分散化されたシステム上では、中心となる機能を必要とせずに各サーバが独立して動作することが必要である。従って、中心となる機能を用いずに、適用する手法を実装する必要がある。

以上の問題を解決するために、本研究では1の問題をエージェント配置問題として定式化し、それに対する近似解を基にエージェントの配置を決定するアプローチを取った。その手法によってシステムのインタラクションコストがどのように変化するかを検証するため、シミュレーションを行った。そして本手法を適用する具体的なアプリケーションとして、スタンドアロン型のマルチエージェントシステムである大規模誘導システムを分散化する実装を行った。

本研究における主な貢献は、以下の2点である。

### 1. インタラクションコストを小さくするエージェント配置手法の確立

エージェントの配置問題を、各エージェントの計算負荷が閾値よりも小さく、かつエージェントサーバ間のインタラクションコストを最小にする問題に定式化した。エージェント配置問題の最適解を求めることは困難であるため、ネットワーク分割問題に対する近似アルゴリズムである k-way FM 法のアイデアを用いて、エージェント配置問題に対する近似アルゴリズムを定めた。シミュレーション上で本手法と先行研究である Comet アルゴリズムとの比較を行い、ランダムな配置から本研究がより速くインタラクションコストを減少させ、反復を繰り返した後にも、インタラクションコストが約 10%改善することを示した。

### 2. 中心となる機能を必要としない分散マルチエージェントシステムの実装方法の確立

スタンドアロン型の大規模誘導システムを分散化し、複数のエージェントサーバを接続する実装について述べた。大規模誘導システムに対して他のサーバと連携するための機能と、システム内部の監視を行う機能の実装を行うことで、各エージェントサーバが独立してエージェントの配置を決定できるアーキテクチャについて述べた。

上記2点の貢献により，分散マルチエージェントシステム上でエージェントを分散配置する際に本手法を用いることで，インタラクションコストの増加によるシステムの性能低下を抑えることが可能になる．そして，中心となる機能を必要とせずに，本手法を用いたシステムを実装できることが示された．

今後の課題として，提案手法に関するより多くの評価を行う必要がある．本論文では，エージェントサーバ間で発生するインタラクションコストに注目して，提案手法の検証を行った．インタラクションのコストがシステムの性能に与える影響の度合いは，エージェントシステムの適用アプリケーションによって異なる．例えば，エージェントサーバがどのように接続されているのかといった条件や，エージェントの動作にどれほどのインタラクションが含まれるのかによって，インタラクションのコストがシステムの性能に与える影響は異なる．今後は，こうしたインタラクションコストがシステムの性能に与える影響を検討していく．

更に，エージェント配置を決定するための手法を拡張する点が挙げられる．本研究で提案した手法は，過負荷になったエージェントサーバからエージェントを移動する際に，インタラクションコストが小さくなるようなエージェント移動を決定している．しかし，過負荷のエージェントサーバが存在していない場合も，エージェントの移動を行うことで更にシステムのインタラクションコストを減らすことができると考えられる．このようなことを実現するためには，定期的にエージェントの位置を更新するような手法が必要である．今後は，そのような手法を考察していくことも必要である．

## 謝辞

本研究を進めるにあたり，熱心な支援と適切な指導を行っていただいた石田亨教授に厚く御礼申し上げます．

また，日頃より議論していただき，有益な助言を与えてくださいました石田研究室の皆様方に心より感謝いたします．

本研究は，戦略的情報通信研究開発推進制度 (SCOPE) の支援を受けて行われました．

## 参考文献

- [1] Gasser, L. and Kakugawa, K.: MACE3J: fast flexible distributed simulation of large, large-grain multi-agent systems., *AAMAS*, pp. 745–752 (2002).
- [2] *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, IEEE Computer Society (2004).
- [3] Tai, H. and Yamamoto, G.: An Agent Server for the Next Generation of Web Applications, *DEXA '00: Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, Washington, DC, USA, IEEE Computer Society, p. 717 (2000).
- [4] Ishida, T., Gasser, L. and Nakashima, H.(eds.): *Massively Multi-Agent Systems I, First International Workshop, MMAS 2004, Kyoto, Japan, December 10-11, 2004, Revised Selected and Invited Papers*, Lecture Notes in Computer Science, Vol. 3446, Springer (2005).
- [5] Ishida, T.: Society-Centered Design for Socially Embedded Multiagent Systems., *CIA*, pp. 16–29 (2004).
- [6] Ahmad, I. and Ghafoor, A.: Semi-Distributed Load Balancing for Massively Parallel Multicomputer Systems, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 10, pp. 987–1004 (1991).
- [7] Sanchis, L. A.: Multiple-Way Network Partitioning, *IEEE Trans. Comput.*, Vol. 38, No. 1, pp. 62–81 (1989).
- [8] Fiduccia, C. M. and Mattheyses, R. M.: A linear-time heuristic for improving network partitions, *DAC '82: Proceedings of the 19th conference on Design automation*, Piscataway, NJ, USA, IEEE Press, pp. 175–181 (1982).
- [9] Chow, K.-P. and Kwok, Y.-K.: On Load Balancing for Distributed Multi-agent Computing., *IEEE Trans. Parallel Distrib. Syst.*, Vol. 13, No. 8, pp. 787–801 (2002).
- [10] Yamamoto, G.: Agent Server Technology for Managing Millions of Agents., In Ishida et al. [4], pp. 1–12.

- [11] Yamamoto, G. and Tai, H.: Architecture of an agent server capable of hosting tens of thousands of agents, *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, New York, NY, USA, ACM Press, pp. 70–71 (2000).
- [12] Yamamoto, G. and Nakamura, Y.: Architecture and Performance Evaluation of a Massive Multi-Agent System., *Agents*, pp. 319–325 (1999).