エージェント結合方式に着目した分散協調システムの負荷分散への適用

加藤健　　渡辺尚　　水野忠則

静岡大学工学部

概要

分散協調システムでは、エージェントは相互に協力や妥協を行ない個々の目標を達成する。この時、エージェント群は共通の通信プロトコルを必要とする。ここではエージェント結合方式の観点から通信プロトコルを構築する。この方式は問題を持つエージェントと、問題解決能力を持つエージェントの双方からエージェントの結合を実現する枠組である。本稿ではこれを負荷分散モデルに適用し、その有効性を検討する。

1　はじめに

分散協調システムは不完全な知識を持つエージェント群からなるシステムである。これらエージェントは各々の状態に応じて相互に協力や妥協を行ない個々の目標を達成する。ここでエージェントが同一の環境で動作する場合には次のようなことを考慮する必要がある。まずエージェントが互いに協力を行なう場合には、エージェント間で何らかの情報の交換が行なわれなければならない。このためエージェントは共通した通信プロトコルを持つ必要がある。また複数のエージェントが同一の環境の中で作業を行ない、その結果環境を変化させる。これによって、エージェントがあらかじめ求めた予測を越えて変化することがある。よってエージェントは他のエージェントを含めた環境と自分自身との関係を踏まえたうえで次に自分のとる行動の決定方法を個々に持たねばならない。また、予測がはずれた場合の対処も必要となる。次にエージェントは他のエージェントに協力を求める場合、自分の持つ問題を明確にする必要がある。例えば自分の持つ問題を解決するために他のエージェントの能力が必要である場合でも、自分の問題のどの部分が自分自身で解決でき、どの部分が解決できないのかを明確にして、問題を分割することによって他のエージェントの負担を減らすことができる。またこれに対応して結果を統合する手段についても考慮する必要がある。

さらに分散協調システムを設計する際には、設計する分散協調システムの目的にそった評価モデルの構築も重要な問題となる。以下に分散協調システムを設計する際の問題点をまとめる。

1. 通信プロトコルの確立
2. エージェントのとる戦略の決定方法
3. 問題の分割や結果の統合の手段
4. 評価モデルの構築

1の通信プロトコルに関する研究としてはエージェント間の契約交渉を基にしたR.G.Smithのコントラクトネットプロトコル[1][2]などが提案されている。2の戦略の決定方法に関する研究としては文献[3]が存在する。ここではエージェントの状態として、協調、妥協、そして競合の3つの状態を仮定し、エージェントが各々の状態に応じて次にとるべき行動を決定する方法について考察している。同様に文献[4]も2の戦略の決定方法について考察している。ここではエージェントが時々の状況に応じて自分や他のエージェントの副目標を訂正することで問題解決活動の効率化を図り、エージェント群全体としての自己組織化を行なっている。また文献[5]では複数の戦略を持つ系におけるエージェントのばらつきを反映した利得関数について考察している。この研究では戦略を複数のタイプにグループ化するような性質を利得関数に付加することによって、多安定系を作ることを可能にする。文献[6]はR.G.Smithのコントラクトネットに基づき通信プロトコルを構築しており、1と2の両方の問題点を扱っている。そして2に関してはエージェントが未処理タスクを他エージェントに依頼した場合の所要時間を推定することで、このタスクを依頼するかどうかについての決定を行なっている。他に4の評価モデルに関する研究としては[7]のTileworldやバベルの塔[8]が存在する。これらは2次元格子平面上に、エージェントやタイル、ブロック、障害物及び穴を配置した、仮

Adaptive load balancing based on distributed coordination methods

Takeshi KATO　Takashi WATANABE

Tadanori MIZUNO

Faculty of Engineering, Shizuoka University

想的な分散協調モデルである。しかしながら、実際のシステムに分散協調システムの枠組を適用した例は少ない。

本研究では1の通信プロトコルに関する問題点を、(1) 問題と問題解決能力の存在を的確に知ること、(2) エージェント間の通信によるオーバーヘッドを最小限にとどめること、そして(3) エージェントの不適切な結合からもたらされる失敗を少なくすることの3つの視点から検討する。そしてこれを問題を持つエージェントと問題解決能力を持つエージェントの結合を行なう方法に帰着して、それらエージェントの効率的な結合を行なう枠組を構築する。本稿ではこの枠組をエージェント結合方式と呼ぶ。また2に対して、エージェントの結合の型を決定する利得関数を提案する。4の評価モデルとしては複数のサーバとディスパッチャから成る負荷分散モデルを採用する。そしてエージェント結合方式を負過分散モデルに適用し、この方式の有効性を検討する。

2 エージェント結合方式

著者らは、エージェントを効率的に結合させるための手段として問題解決における先行順位と情報交換方式に基づく枠組を提案してきた [9]。この方式ではエージェントを問題を持つエージェントと問題解決能力を持つエージェントに分類し、それぞれ依頼者、そして解決者と呼ぶ。依頼者と解決者をそれぞれ次に定義する。

- 依頼者
  問題解決の過程においてエージェントが問題を持つにもかかわらず、その問題を解決できない状況が起こり得る。その時、解決能力を持つ他のエージェントに問題を依頼することを決定するエージェントが依頼者である。
- 解決者
  問題解決の過程においてエージェントは他のエージェントの持つ問題を解決する状況が存在する。この場合に、他のエージェントの問題を解決するエージェントが解決者である。

ここでは問題の種類が単一のものばかりでなく、異なる種類の問題を扱う場合についても考察する。そのため問題の種類に応じてクラス化を行なう。問題の種類が単一である場合は、クラスが1つであると考える。この場合解決者は問題のクラスに対応して1種類の問題解決能力を持てば、全ての問題を処理することが可能である。クラスが2つである場合、解決者は問題のクラスに応じて2種類の問題解決能力を持つ必要がある。1種類の問題解決能力しか持たない解決者は、それに応じたクラスの問題だけを処理する。つまり、クラス1の問題解決能力しか持たない解決者はクラス1の問題を処理し、クラス2の問題解決能力しか持たない解決者はクラス2の問題を処理する。自分の持つ解決能力と異なるクラスの問題を配送された場合、解決者は問題の処理に失敗する。

図1はエージェントの結合例を表している。結合1では問題を持つエージェントAが、自分の持つ問題を解決できるかどうか他のエージェントBやCに聞いている。結合2では問題を持つエージェントDが問題解決能力を持つエージェントCに問題を依頼している。エージェント結合方式では図1に示すように問題解決能力を持つエージェントCが同時にエージェントAやエージェントDなどの複数のエージェントから問題を依頼される場合や、図1の例以外にも、自分の問題を他のエージェントに依頼し、かつ他のエージェントの問題を請け負う場合など様々な状況が起こり得る。

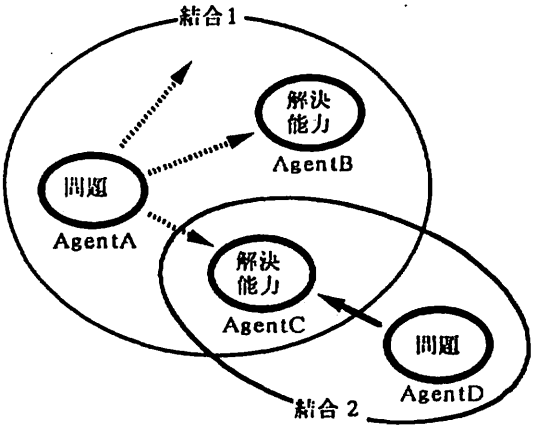

図1 エージェントの結合例

2.1 エージェントの結合

エージェント結合方式は、以下に述べる先行順位と情報交換方式を2つの軸として合計4つの型を構成する。

2.1.1 問題解決における先行順位

- 依頼者先行
  問題を持つエージェントからエージェントの結合を開始する型である。この型の利点は問題が発生した場合即座に問題解決を開始することが可能となることである。
- 解決者先行
  問題解決能力を持つエージェントからエージェントの結合を開始する型である。この型の利点は問

題解決能力を持つエージェントがアイドル状態になった時に、他のエージェントに問題を要求することでエージェント群全体の動作効率を高める点にある。

2.1.2 問題解決における情報交換方式

- 命令型
  過去の問題解決活動の事例から問題の受渡しを決定し、決定の際にはエージェント間の通信を必要としない枠組を命令型と呼ぶ。この型の利点はメッセージのトラフィックを増大させずにエージェントの結合を実現する点である。ただし交渉型と比較して動的に変化する環境下では、不適切なエージェントの結合を行なう可能性が高いという欠点を持つ。
- 交渉型
  問題の受渡しの決定をする前に他のエージェントに関する情報を得る枠組を交渉型と呼ぶ。この型の利点はエージェント群の環境の動的な変化に適応してエージェントを結合する点である。欠点は命令型と比較してエージェント間の通信のコストが大きいことである。

エージェント結合方式は以上で説明した先行順位と情報交換方式に基づき合計4つの型を構成する。これを図2にまとめる。

		情報交換方式	
		命令型	交渉型
先行順位	依頼者先行	過去の事例より能力を持つエージェントを判断して直接に問題を依頼	解決能力を持つエージェントの情報を獲得してから問題を依頼
	解決者先行	過去の事例より問題を持つエージェントを予想して直接に問題を要求	問題を持つエージェントの情報を獲得してから問題を要求する

図2 エージェント結合方式における4つの型

2.2 エージェント結合方式で用いるメッセージ

エージェント結合方式では以下のメッセージを使用する。エージェント結合方式において、問題を問題解決能力を持つエージェントに配送するプロセスには、選択プロセスと配送プロセスの2つのプロセスが存在する。依頼者先行と解決者先行は配送メッセージを除いて、異なるメッセージを使用する。そして依頼者先行と解決者先行はともに、命令型は配送プロセスから、交渉型は選択プロセスと配送プロセスの2つのプロセスから構成される。各プロセスでは以下のメッセージを使用する。

1. 依頼者先行
   (a) 選択プロセス
      i. 問題通知メッセージ
         問題を持つことを他のエージェントにブロードキャストで知らせる時に用いる。
      ii. 問題応答メッセージ
         受けとった問題通知メッセージの内容から判断して、問題を請け負うことを決定した場合にこのメッセージを用いる。
   (b) 配送プロセス
      i. 配送メッセージ
         問題を配送する時に用いる。
2. 解決者先行
   (a) 選択プロセス
      i. 能力通知メッセージ
         自分がアイドル状態であり、問題解決が可能であることを他のエージェントにブロードキャストで知らせる時に用いる。
      ii. 能力応答メッセージ
         受けとった能力通知メッセージの内容から判断して、問題の依頼を決定した場合にこのメッセージを用いる。
   (b) 配送プロセス
      i. 要求メッセージ
         問題を要求する時に用いる。
      ii. 配送メッセージ
         問題を配送する時に用いる。

例として依頼者先行・交渉型は、問題通知メッセージ、問題応答メッセージ、そして配送メッセージの3つのメッセージから構成される。

3 エージェント結合方式の適用例

3.1 負荷分散モデル

動的負荷分散におけるサーバとソースの結合方式に関して、本稿の結合方式と同様なことが文献[10]においても考察されている。この研究ではサーバとソースのイニシアティブに基づいた分類法に従って、負荷分散におけるジョブ配送方式を表1（文献[10]より）のように分類している。この分類法のソースイニシアティブとサーバイニシアティブはそれぞれエージェント結合方式の先行順位における依頼者先行と解決者先行に対応している。この研究ではさらにエージェントの持つ情報のレベルについても考察している。しかし、Wangらは情報のレベルを固定し、それらの動的に変化する情報を獲得する手段については述べていない。本研究のエージェント結合方式における情報交換方式は、この研究のレベル5までの情報をダイナミックに獲得することを目的としている。

Level of Information Dependency in Scheduling	Source-Initiative	Server-Initiative
1	server = f (source)	source = f (server)
2	server = f (source, ω)*	source = f (server, ω)
3	server = f (source, ω, sequence state)	source = f (server, ω, sequence state)
4	server = f (source, ω, sequence state, server busy / idle status)	source = f (server, ω, sequence state, source queue emptiness)
5	server = f (source, ω, sequence state, server queue lengths)	source = f (server, ω, sequence state, source queue lengths)
6	server = f (source, ω, sequence state, server queue lengths, departure epochs of completed jobs at servers)	source = f (server, ω, sequence state, arrival epochs of jobs at sources)
7	server = f (source, ω, sequence state, departure epocks of completed and remaining jobs at servers)	source = f (server, ω, sequence state, arrival epochs and execution times of jobs at sources)

* ω is a randomly generated parameter

表1 A Taxonomy For Load Sharing Algorithms
（文献10より）

図3は本稿の負荷分散モデルを表している。ここで問題解決能力が動的に変化する特徴を持つ負荷分散モデルにエージェント結合方式を適用して考察する。この負荷分散モデルは複数のサーバとディスパッチャから構成され、各々のサーバとディスパッチャは1つの待ち行列を持つ。ディスパッチャにはランダムにジョブが到着する。ディスパッチャは到着したジョブをサーバに配送する。このジョブの配送の部分にエージェント結合方式を適用する。エージェント結合方式における依頼者と解決者は、負荷分散モデルにおいてはそれぞれディスパッチャとサーバに対応する。

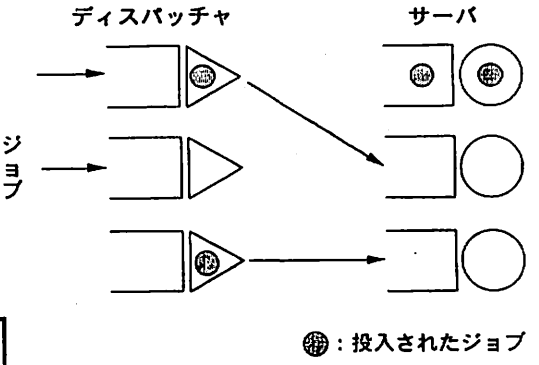

図3 負荷分散モデル

3.1.1 シミュレーション仮定

エージェント結合方式の有効性を示すため、この方式を負荷分散モデルに適用し、シミュレーション実験を行なう。実装には九州大学で開発された分散協調モデル記述言語Cellula/C [11]を使用する。Cellula/Cは協調計算モデルCellulaをC上で具体化したプログラミング言語である。協調計算モデルCellulaは複数のセルから構成され、それぞれのセルはパターン照合通信を行なうことで情報を交換する。ここでは、エージェントがメッセージを受けとった場合の処理機能を1つのセルに割り当てる。エージェント結合方式では依頼者先行から解決者先行、命令型から交渉型へ移るにつれてメッセージ数が多くなるので、依頼者先行・命令型から解決者先行・交渉型まで、それぞれ8から12個のセルを使用している。

ここでは以下の設定に基づき負荷分散モデルのシミュレーション実験を行なう。

サーバとディスパッチャの数 サーバとディスパッチャの数はそれぞれ5つである.

ジョブ到着率 ディスパッチャへのジョブの到着率はポアソン分布に従う.

サーバ処理能力 サーバの処理能力は1（個/秒）とする.

サービス量 ジョブのサービス要求量は平均1（秒）の指数分布である.

通信遅延時間 ディスパッチャとローカルサーバ間の通信遅延時間は0（秒）とする. ディスパッチャとリモートサーバ間の通信遅延時間は実験ごとに値を設定する.

ジョブクラス ジョブのクラスは1つであるか、または2つである。

3.1.2 ジョブ配送時の4つの結合方式の例

2章で示した4つのエージェント結合方式、すなわち(1)依頼者先行・命令型、(2)依頼者先行・交渉型、(3)解決者先行・命令型、(4)解決者先行・交渉型を負荷分散モデルに適用して、それぞれを考察する。

- 依頼者先行・命令型

依頼者先行・命令型は、問題を持つエージェントである依頼者が現在自分が持っている問題の解決能力を備えていると考えられるエージェントを過去の事例より判断し、配送メッセージを用いて直接に問題を配送する型である。この型を負荷分散モデルに適用した場合のディスパッチャとサーバのアルゴリズムの一例を以下に示す。

ディスパッチャ：

1. ディスパッチャにジョブが到着する時にローカルサーバがアイドル状態であれば、ジョブをローカルサーバに配送する。
2. ローカルサーバがビージー状態であれば、ジョブをリモートサーバに配送する。リモートサーバが複数存在する場合、全てのリモートサーバに一通りジョブを配送するまで毎回異なるリモートサーバを選択してジョブを配送する。リモートサーバへのジョブの配送が一巡すれば、再び最初に選択したリモートサーバへジョブを配送する。(巡回配送)

サーバ：

1. ジョブを受け取った時サーバは、アイドル状態であればジョブを実行する。ビージー状態であればジョブを待ち行列に格納する。
2. ジョブの実行を終了すると、元のディスパッチャに返送する。

ディスパッチャとサーバはジョブが到着するたびに、以上のアルゴリズムを繰り返す。

- 依頼者先行・交渉型

依頼者先行・交渉型では、依頼者は自分の持つ問題の解決が可能かどうかを知るために、他の全てのエージェントに問題通知メッセージを用いてブロードキャストで通知する。問題通知メッセージを受け取ったエージェントは現在の状態から解決能力を計算し、この内容を問題応答メッセージを用いて依頼者に返す。問題応答メッセージを受け取ると、依頼者は各々のメッセージの内容から判断してもっとも適切であると考えられるエージェントに問題を配送する。この時には、配送メッセージを用いる。これを負荷分散モデルに適用した場合のディスパッチャとサーバのアルゴリズム例を以下に示す。

ディスパッチャ：

1. ディスパッチャにジョブが到着する。
2. ディスパッチャは全てのサーバにジョブの存在を通知し、返答を待つ。
3. ディスパッチャは返答の内容から、もっともジョブの数の少ないサーバを選択して、ジョブを配送する。

サーバ：

1. サーバはディスパッチャからメッセージを受けとると、現在実行しているジョブと待ち行列に存在するジョブの数を合わせて、ディスパッチャに返答する。
2. ディスパッチャからジョブを受け取った時アイドル状態であれば、ジョブを実行する。ビージー状態であればジョブを待ち行列に格納する。
3. ジョブの実行を終了すると、元のディスパッチャに返送する。

ディスパッチャはジョブが到着するたびに、以上のアルゴリズムを繰り返して実行する。サーバはディスパッチャから現在の状態を問われるたびに以上のアルゴリズムを繰り返す。

- 解決者先行・命令型

解決者先行・命令型は、問題解決能力を持つにも

かかわらずアイドル状態にあるエージェント（解決者）から他の問題を持つと考えられるエージェントに過去の事例から判断して、要求メッセージを用いて直接問題を要求する型である。要求メッセージを受けとったエージェントは自分の持つ問題を配送メッセージを用いて送る。この型を負荷分散モデルに適用した場合のサーバとディスパッチャのアルゴリズムの例を以下に示す。

サーバ：

1. サーバはアイドル状態にある場合、ローカルディスパッチャの状態を判断する。
2. もしローカルディスパッチャがジョブを保持していれば、ジョブを要求する。ジョブを保持していなければリモートディスパッチャにジョブを要求する。リモートディスパッチャが複数存在する場合、全てのリモートディスパッチャに一通りジョブを要求するまで毎回異なったリモートディスパッチャを選択してジョブを要求する。リモートディスパッチャへのジョブの要求が一巡すれば、再び最初に選択したリモートディスパッチャへジョブを要求する。
3. サーバはジョブが存在しないことをディスパッチャから告げられると、再び1にもどる。ディスパッチャからジョブを配送された場合には、それを実行する。
4. ジョブの実行を終了すると、元のディスパッチャに返送する。

ディスパッチャ：

1. ジョブを要求された時、ディスパッチャはジョブを保持していなければその旨をサーバに通知する。ジョブを保持していれば、それをサーバに配送する。

サーバはアイドル状態となるたびに、以上のアルゴリズムを繰り返して実行する。ディスパッチャはサーバからジョブを要求されるごとに、以上のアルゴリズムを繰り返す。

- 解決者先行・交渉型

解決者先行・交渉型では、アイドル状態にあるエージェント（解決者）は、他の全てのエージェントに自分がアイドルであることを能力通知メッセージを用いてブロードキャストで通知する。能力通知メッセージを受けとったエージェントは現在の自分の持つ問題の概要を能力応答メッセージを用いて解決者に返送する。能力応答メッセージを受けとると、解決者は各々のメッセージの内容から自分の問題解決能力に適した問題を保持すると考えられるエージェントに問題を要求する。問題を要求されたエージェントは、自分の保持する問題を配送メッセージを使用して解決者に送る。これを負荷分散に適用した場合のサーバとディスパッチャのアルゴリズム例を以下に示す。

サーバ：

1. サーバはアイドル状態にある場合、全てのディスパッチャに自身の状態を通知し、返答を待つ。
2. ディスパッチャの返答の内容から、もっともジョブを多く持つディスパッチャを選択して、ジョブを要求する。
3. ジョブが存在しないことをディスパッチャから告げられると、再び1にもどる。ディスパッチャからジョブを配送された場合には、それを実行する。
4. ジョブの実行を終了すると、元のディスパッチャに返送する。

ディスパッチャ：

1. ディスパッチャはサーバからアイドルであることを知らせるメッセージを受けとると、自分が現在保持しているジョブの個数をサーバに返答する。
2. サーバからジョブを要求された時、ディスパッチャはジョブを保持していれば、それをサーバに配送する。ジョブを保持していなければ、その旨をサーバに通知する。

サーバはアイドル状態となるたびに、以上のアルゴリズムを繰り返して実行する。ディスパッチャはサーバからアイドル状態であることを告げられるたびに、以上のアルゴリズムを繰り返す。

3.2 実験結果

図4と図5は各々のアルゴリズムの性能を示している。縦軸はターンアラウンドタイム、横軸はジョブ到着

率を表す。

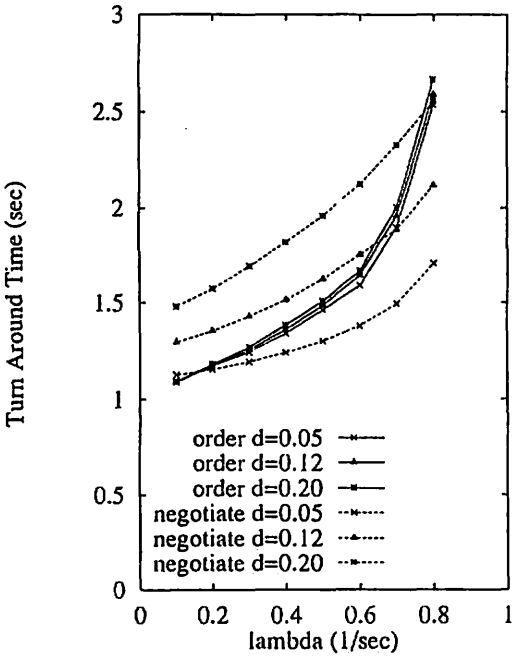

図 4: 依頼者先行方式における命令型と交渉型の性能

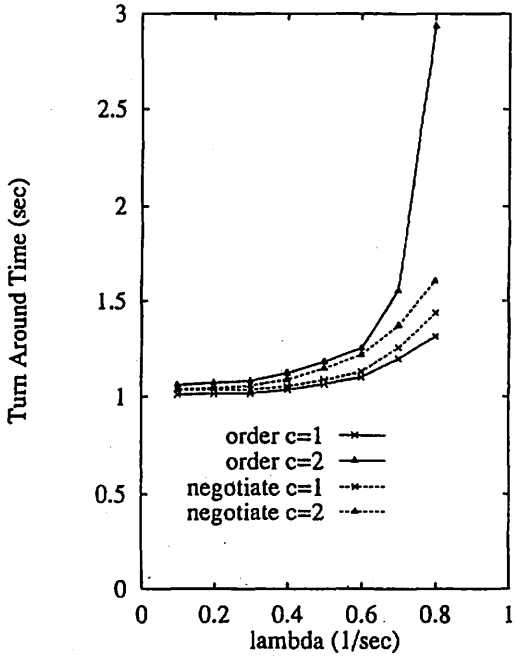

図 5: 解決者先行方式におけるクラス数の影響

図 4 の実験では、ジョブのクラスは 1 つである。この場合サーバはどのジョブを受けとっても、それを解決することができる。ここではエージェント結合方式の依頼者先行において、通信遅延時間 (d) がそれぞれ 0.05 秒、0.12 秒、0.20 秒の場合の命令型 (order) と交渉型 (negotiate) の比較を行なっている。図 4 から、ジョブの到着率が低い場合では命令型の方が性能が良く、高い場合では交渉型の方が性能が良いということが言える。命令型ではローカルサーバがビージーである時、ジョブを巡回させてリモートサーバに配送する。交渉型では準最適なサーバを選択するが、命令型と比較して通信のコストがかかる。これよりジョブ到着率が低い場合ではどのリモートサーバを選択してもアイドルである可能性が大きいため命令型が有効である。しかしジョブ到着率が高い場合ではリモートサーバもビージーである可能性が大きいため交渉に費やされるコストを支払ってもリモートサーバに関する情報を得ることが有効となる。ここで、リモート通信遅延時間がそれぞれ異なる場合の結果を比較してわかることは、遅延が 0.20 秒から 0.12 秒、0.05 秒と小さくなるにつれて交渉型の有効な領域が広くなることである。これは通信遅延時間が小さいほど、命令型に対して交渉型が有効であることを表している。交渉型が命令型より 1 往復分多くの通信遅延を必要としていることはそれぞれ変わりない。しかし、0.05 秒の場合のほうが 0.12 秒や 0.20 秒の場合よりもジョブの処理時間に対する通信遅延時間の比率が相対的に小さい。これがリモート通信遅延時間が小さくなるにつれて交渉型の性能が良くなる原因であると考えられる。

図 5 は解決者先行においてジョブのクラスが 1 つである場合 (c=1) と、クラスが 2 つである場合 (c=2) の命令型と交渉型の比較を行なっている。クラスが 1 つである場合、サーバはどのジョブを受けとっても、それを解決することが可能である。クラスが 2 つである場合、各ディスパッチャに到着するジョブのクラスはランダムに 1 か 2 とする。また、サーバ 1 とサーバ 2 はクラス 1 のジョブに対する解決能力を持ち、サーバ 4 とサーバ 5 はクラス 2 のジョブの解決能力を持つ。サーバ 3 では両クラスのジョブの解決が可能である。ジョブの解決に失敗した場合、サーバはそれをもとのディスパッチャに返送する。ディスパッチャはそのジョブを再び待ち行列に加える。ここでクラスが 1 つである場合の方式を 1 クラス方式、クラスが 2 つである場合の方式を 2 クラス方式と呼ぶことにする。図 5 のサーバとディスパッチャの通信遅延時間は 0.01 秒である。1 クラス方式では、交渉型よりも命令型の方が全体を通して良い性能を示している。

これは解決者先行ではアイドル状態のサーバからジョブを要求するので、依頼者先行と比較すると、命令型において不適切な結合を起こす可能性が非常に低いということが理由としてあげられる。ただしこれは、一度サーバに配送されたジョブは必ずそのサーバにおいて処理が可能である場合にはじめて成立することである。2クラス方式では、命令型よりも交渉型の方が良い性能を示している。命令型も交渉型もクラスを2つにするによって、性能が低下することには変わりない。しかし交渉型と比較して命令型では、クラスの増加が原因でジョブの処理を失敗する可能性が非常に高いために、図5に示すような結果になったと考えられる。これよりサーバがジョブを要求する場合、より適切な判断が必要であるほど命令型より交渉型が有効となると言える。

またこれらのシミュレーション実験全体を通して次のことが分かった。それは命令型、交渉型に関係なく解決者先行は、依頼者先行よりも通信の負荷が大きくなることである。依頼者先行ではジョブが到着した時点で、これを配送しようとするが、解決者先行ではサーバはアイドル状態である限り、繰り返してジョブを探し続ける。このため解決者先行ではシステム全体の中にジョブが存在しない場合でも、見つかるまでディスパッチャにジョブを要求し続けるような状況が起こり得る。これを回避する一つの手段として、ジョブを要求する間隔を広げる方法が考えられる。しかしこれは通信の負荷を減少させる反面ディスパッチャに到着したジョブを獲得するタイミングを遅れさせるので、サーバのアイドル時間を増大させる結果をもたらす。つまり解決者先行では、通信の負荷とサーバのアイドル時間はトレードオフの関係にあると言える。

3.3 利得関数による結合方式の選択

シミュレーションの結果から、本稿で示した各々のアルゴリズムはジョブの到着する割合や通信遅延時間の大小によって、有効となる領域が異なることがわかった。ここで各々のアルゴリズムのメッセージ量を比較する。1つのサーバと1つのディスパッチャが存在する時に、サーバとディスパッチャ間で転送されるメッセージ量を1と仮定すると、各々のアルゴリズムにおける1回の結合に必要なメッセージ量は、依頼者先行・命令型では1、解決者先行・命令型では2、依頼者先行・交渉型では3、そして解決者先行・交渉型では4となる。しかし交渉型は通信の際にブロードキャストを行なうので、依頼者先行・交渉型ではサーバの数、解決者先行・交渉型ではディスパッチャの数が増えるにつれて、メッセージ量も増大すると考えられる。これより通信遅延時間が大きくなるほど命令型に対する交渉型の有効性が減少することがわかる。しかしながら交渉型はエージェント間の不適切な結合を減少させるために他のエージェントに関する最新の情報を獲得するので、命令型と比較した場合より適切にジョブの配送が行なうことができる。また依頼者先行と解決者先行を比較した場合も、解決者先行の方がサーバのアイドル時間を減少させる、つまりシステム全体の負荷の割り当てを均等化する可能性を持つが、同時にメッセージ量が多くなるという欠点を持っている。以上の点からジョブの到着率や通信遅延時間などの条件によって各々のアルゴリズムの有効な領域を区別する境界が存在すると考えられる。この境界が示されれば、それぞれの状況に応じて適切なアルゴリズムを選択することが可能となる。本稿ではこうした境界に基づいて適切なアルゴリズムの選択を可能とする利得関数を提案する。

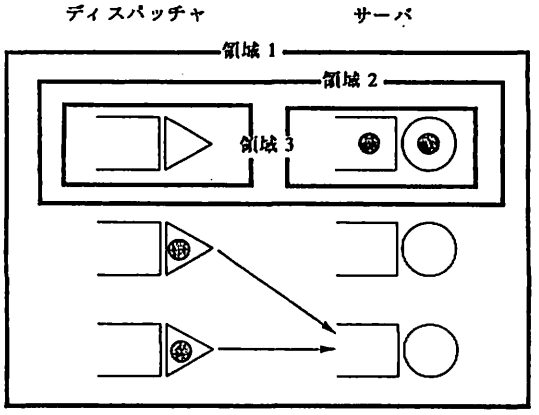

図6 利得関数の有効領域

ここで負荷分散モデルに対応して利得関数を作成する場合、その結果の適用される領域が問題となる。負荷分散モデルでは図6のように3つの領域が仮定できる。領域1はシステムの全体を被覆している。この場合はシステム全体が同一の型でジョブの配送を行なうことになる。ただし、これを実現するためには利得関数から導き出された結果を全てのサーバとディスパッチャに知らせるメカニズムを考案しなければならない。またこの時、特定のエージェント（サーバまたはディスパッチャ）が利得関数を用いることができると仮定すると、そのエージェントだけが他のエージェントに影響を及ぼすことができるので、他のエージェントに対して優先的な立場をしめることになる。逆に全てのエージェントが利得関数を用いることができるとすると、個々のエージェントが利得関数を用いるごとに他の全てのエージェントがそれ

に従うことになり、システム全体の活動が安定しない。領域2は1つのディスパッチャとそのローカルサーバから構成される。負荷分散モデルではディスパッチャとローカルサーバの通信遅延時間を0と仮定しており、ローカルグループ内では情報が瞬時に伝わる。よって利得関数をローカルグループ内に適用することで、負荷分散モデルの実装が簡単化される。この場合、ローカルグループの単位でエージェント結合方式の型が決定される。領域3では、個々のサーバやディスパッチャを一つの単位として考える。この方法は、エージェントが自律的に活動するという分散協調システムの仮定に最も添った形であるが、サーバやディスパッチャの全てが別々の型でジョブの配送を行なうため、システム全体の動作が非常に複雑になると予想される。こうした点を考慮して本稿では利得関数を、領域2の単位に適用して考察する。利得関数gは次のように定義する。

$$g_{type}(d,\lambda) = z(d,type) + w(\lambda) + s + F \quad (1)$$

d：通信遅延時間

λ：ジョブ到着率

$type$：エージェント結合方式の型変数

z：各型における平均通信時間を求める関数

w：ジョブのシステム内における平均待ち時間を求める関数

s：サーバにおけるジョブ処理時間

F：ジョブを失敗したためにかかる時間

(1)式はエージェント結合方式の各々の型おけるターンアラウンドタイム、すなわちコストを計算する。各ディスパッチャは次式によって次の結合型を選択する。

$$h(d,\lambda) = \min\{g_1(d,\lambda), g_2(d,\lambda), g_3(d,\lambda), g_4(d,\lambda)\} \quad (2)$$

$$\text{次の型} = f(h(d,\lambda)) \quad (3)$$

(2)式は4つのアルゴリズムのうち最小のコストをもつものを求める。(3)式は最小のコストでジョブを処理するエージェント結合方式の型を導出する。

ここで、例として図7に示すような状況を仮定する。図7では型1と型2は交差点を境にコストが入れかわっている。この交差点におけるコストを境界Aとする。(1)式で計算するコストが境界A以下では型1を、境界A以上では型2を採用する。この利得関数を使用することで、その時々において適切な型を選択することが可能となる。

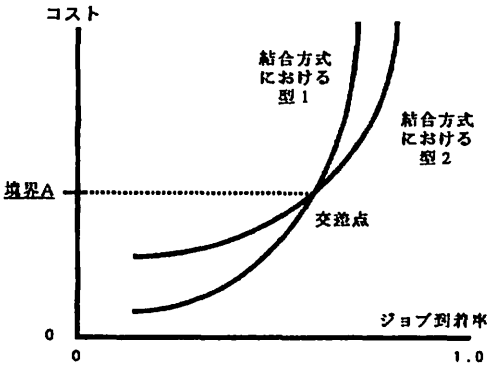

図7 各型の交差点で示される境界

4 まとめ

本稿では、問題を持つエージェントと問題解決能力を持つエージェントを効率的に結合するプロトコルとして、エージェント結合方式を提案した。そしてこれを負荷分散モデルに適用しシミュレーション実験を行なうことで、この方式が有効であることを示した。今後利得関数の詳細な検討を行ない、様々な状況のもとで、エージェントが独自の判断で適切な型を選択する場合の本方式の有効性を考察する予定である。

参考文献

[1] Randall Davis and Reid G. Smith : "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence 20 (1983) pp63-109

[2] Reid G. Smith : "The Contract Net Protocol : High-Level Communication and Control in a Distributed Problem Solver", IEEE TRANS. ON COMPUT., VOL.C-29, NO.12, DEC. 1980

[3] Gilad Zlotkin and Jeffrey S. Rosenschein : "Cooperation and Conflict Resolution via Negotiation Among Autonomous Agents in Noncooperative Domains", IEEE TRANS. ON SYSTEMS, MAN, AND CYBERNETICS, VOL.21, NO.6, NOVEMBER/DECEMBER 1991

[4] 山崎哲哉, 波辺尚 : "格子空間を移動するエージェント群の協調動作について - 「バベルの塔」にお

ける副目標の生成と達成 - ", 情報処理学会研究報告 Vol. 93, No.69 (1993)

[5] 沼岡千里："自律エージェントの集団的戦略変更とその応用", 情報処理学会研究報告 Vol. 93, No.69 (1993)

[6] 小川智之, 小林真也, 木村春彦, 武部幹："依頼による自律的な負荷分散方式の提案", 1993 年電子情報通信学会春季大会

[7] Martha Pollack and Marc Ringuette : "Introducing the Tileworld : Experimentally Evaluating Agent Architectures", In Proceedings of The Eighth National Conference on Artificial Intelligence, pp.183-189, 1990.

[8] 石田亨："バベルの塔：組織指向プラニングに向けて Tower of Babel: Towards Organization-Centered Planning", 第一回マルチ・エージェントと協調計算ワークショップ (MACC '91) 資料 (1991)

[9] 加藤健, 渡辺尚 : "問題解決の主導権に着目した分散協調システム", 情学技報 AI92-49, Vol.92, No.185 (1992)

[10] Yung-Terng Wang and Robert J.T. Morris : "Load Sharing in Distributed Systems", IEEE TRANS. ON COMPUT., VOL.C-34, NO.3, MARCH 1985

[11] 吉田紀彦, 楢崎修二："場と一体化したプロセスの概念に基づく並列協調処理モデル Cellula", 情報処理学会論文誌 Vol.31, No.7 (1990)