マルチエージェントシステムに関する研究

川村，尚生

(Degree)
博士（工学）
(Date of Degree)
2002-03-08
(Date of Publication)
2014-11-04
(Resource Type)
doctoral thesis
(Report Number)
乙2599
(URL)
https://hdl.handle.net/20.500.14094/D2002599

神戸大学博士論文

マルチエージェントシステムに関する研究

平成14年1月

川村　尚生

目次

図目次

表目次

第1章　序論

1980年代から90年代にかけて，コンピュータサイエンスの様々な分野でエージェントあるいはマルチエージェントという言葉が使われるようになった．それらの研究分野を大別すると以下の3種類に分類できる．

分散人工知能 従来の個別知能を考える人工知能研究が行き詰り，人工知能とは，そもそも処理の有限性や情報の部分性を扱うものであることが分かってきた．つまり，知的な処理とは，情報が欠けた状態で，しかも限られた計算時間で答えを出す必要がある[1]．また，Minskyの「心の社会」[2]のように，人間の心そのものがマルチエージェントの協調によって成立しているという考え方もある．そこで，複数のサブシステムの協調により知能を現出させる，分散人工知能に関する研究が多く行われるようになった．ここで「分散」とは，他のサブシステムに関する情報が時間遅れを伴ってしか分からない場合や，さらにはサブシステムの故障や通信回線の故障を考慮することを示す[3]．サブシステムがシステム全体の共通目的だけでなく固有の目的を持つ場合，サブシステムをエージェント，分散人工知能をマルチエージェントシステムとみなすことができる．

ソフトウェアアーキテクチャ コンピュータネットワークの発達に従い，ソフトウェアアーキテクチャは，1つの仕事を複数のコンピュータの共同作業として行う方向へと向かっている．このような分散環境は，ネットワークのスピードやトポロジの変化，構成要素であるコンピュータの更新，追加，削除が時々刻々と起こるオープンシステム[4]である．オープンシステムにおけるソフトウェアは，1つの固定的な形を静的に持つことはできず，多数のコンポーネントから動的に構成されるような仕組が必要になる[5][6]．コンポーネントはソフトウェアの構成部品であるだけではなく，自律した個としても存在しなければならない．人手を介することなく，自分に適した相手を見つけて協調したり，仕様変更に応じて自らを改変するといったことが求められるからである．つまり，コンポーネントはエージェントであり，ソフトウェア全体はマルチエージェントシステムと言える．

ヒューマンインタフェース コンピュータシステムが複雑化するに伴い，ヒューマン

インタフェースの高度化，擬人化が求められるようになってきた．多少の誤りを訂正してくれたり，よく分からなければ聞き直し，さらには，一を聞いて十を知るような柔軟性を持ったインタフェースの要求が高まっている [3]．また，WWWにおける検索エンジンの出力結果のように，コンピュータシステムの出力する情報は膨大な量であり，必要なものを取捨選択し，人間の直観に訴える形式で提示する技術の必要性も高くなってきた．このような，ユーザの要求とコンピュータシステムが提供するサービスを仲介するソフトウェアのことをインタフェースエージェントと言う．インタフェースエージェントの背後には，ウィンドウシステムに代表される直接実行から間接実行へ転換しようという考え方がある．間接実行のアプローチが成功するためには信頼性の保証と代行能力の高さが鍵になる [7]．

エージェントという言葉はこれだけ多岐の分野にわたって使われているので，その定義に対する統一的見解は存在しない [8] [9]．しかし，エージェントが擬人性を伴う以下のような性質を持つことに対する合意はほぼ形成されている [8] [10]．本論文でもエージェントという言葉を，このような性質を持つソフトウェアという意味で使う．

自律性 エージェントは，内部に信念，願望，意図といった心的状態を持ち，外部環境を観察して状況認識を行った上で，自らの意思決定原理/機構に基き動作し，時には外部環境に影響を及ぼす．

協調性 エージェントは，他のエージェントと協調することで単体ではできない仕事を達成したり，それぞれの目的の競合を解消したり，交渉したりする．

適応性 エージェントは，外部環境の変化に適応して，行動計画を再プランニングしたり，自己を再組織化する．

マルチエージェントシステムとは，上記のような性質を持つ複数のエージェントが並行動作し，協調したり競合を解消したりしながら，自らの目的達成を目指すとともに，同時に全体としての目的達成も目指すシステムと言える．

本論文はマルチエージェントシステムの構築に必要な基礎的技術を主題とし，3つの異なる側面からのアプローチを試みるものである．まず，マルチエージェントシステムの構築に必須となる専用言語を提案する．次いで，マルチエージェントシステムにおける分散データ検索効率を解析する．さらに，信頼性の高いエージェント間通信を実現することを目標として，耐故障性通信アルゴリズムを提案する．

本論文は以下のように構成されている．

マルチエージェントシステムはその性質上，従来のプログラミング言語で記述することは非常に困難なので，専用の言語が必要となる．第2章では，マルチエージェ

ントシステムを構築するための，マルチモバイルエージェントシステム記述用言語Maglogを提案する．

マルチエージェントシステムは，その構成が時々刻々と変化する性質を持ち，かつ，個々のエージェントが自律性を持つので，システム全体が保有するデータを管理エージェントが集中的に保有するのではなく，個々のエージェントが分散して保有すると考えるべきである．このとき，エージェントが分散データを検索するときの効率が重要である．そこで，第3章において，データやプログラムをスーパバイザ・エージェントが集中管理している場合と，各エージェントが分散して管理している場合についての相対的評価を，統計的および実験的な手法で行い，分散管理した場合の性能を調べる．また，分散管理方式の設計上の諸点に指針を与える．

マルチエージェントシステムでは，システム内の通信が重要な役目を持ち，様々な用途で用いられる．したがって，システム内の通信をいかに高信頼に行うかということが重要な問題となる．すなわち，システムの構成要素の一部が故障している場合でも，故障していない部分における通信が可能となるような，耐故障性のある通信の実現が望まれる．第4章では，問題をマルチエージェントシステムにおける通信からネットワークシステムにおける通信として一般化し，故障のあるハイパーキューブ上の全対全個別データ通信アルゴリズムと，故障のあるコーダルリング上の全対全ブロードキャストおよび全対全個別データ通信アルゴリズムを提案する．

最後に第5章において本論文をまとめる．

第2章　マルチモバイルエージェントシステム記述用言語Maglog

2.1　緒言

モバイルエージェントシステムとは，ユーザの代理となるプログラム (エージェント) が，ネットワーク上に複数存在するコンピュータを移動しながら，与えられた問題の解決をはかるシステムである．モバイルエージェントシステムは以下のような利点を持ち [11]，次世代の分散システムとして注目を集めている [12]〜[14].

ソフトウェアの保守管理の容易化 コンピュータにはモバイルエージェントの実行環境だけをインストールしておけば，アプリケーションソフトウェアのインストールやアップデートなどはエージェントの移動により実現できる．

ソフトウェアの柔軟な運用 コンピュータの保守や使用環境の変化に合わせて，ソフトウェアを実行させたまま，他のコンピュータ上へ移動させることができる．

分散プログラミングの容易化 クラアント/サーバ方式のプログラムでは必須となる通信プロトコルが不要であり，通信プログラムを作成しなくてもすむため，プログラミングが容易になる．

通信遅延への対処 エージェントが，通信先との通信遅延が小さい場所へ移動することにより，通信遅延が軽減される．

通信路の切断への対処 無線通信や携帯端末を利用した通信など，コストや通信路の特性から通信路の切断を余儀なくされることがある．このような場合に，切断される通信路の向こうにエージェントを移動させておくことにより，通信路が切断された状態でもエージェントの実行を継続することができる．

現在，モバイルエージェントシステムを実装するための言語やフレームワークの研究が活発に行われている．モバイルエージェントシステムの実装では，エージェントの移動方式が重要である．これによって，移動後のエージェントの継続実行のレベルが決定する．エージェントの移動 (マイグレーション) 方式は，移動可能なエージェントの内部情報によって以下の 2 つに分類できる [15]．ここで，エージェント

はコード領域，データ領域，実行状態領域 (スタックやプログラムカウンタなど) の3つから構成されるものとする．

弱マイグレーション コード領域とデータ領域を転送する．エージェントの継続実行については，移動前の実行個所から再開することはできず，特定のメソッドや関数からの再開となるので，プログラムの記述が煩雑になるという欠点を持つ．移動先が一ヶ所に決まっている場合はまだよいが，複数の移動先を持つエージェントを記述しようとすると，エージェントが実行を再開するメソッドの先頭で，移動先や移動元に応じた条件分岐が必要になり，煩雑さは増大する．

強マイグレーション コード領域とデータ領域に加えて，実行状態領域も転送する．エージェントの実行は移動前の実行個所から再開するので，プログラムを自然かつ簡潔に記述できる [16].

両方式におけるプログラムの書きやすさの違いを例で示す．図 2.1 は，hostA を出発したエージェントが hostB と hostC で何らかの処理を行なって hostA に戻ってくるというプログラムの概略を，Aglets(弱マイグレーション) と Maglog(強マイグレーション) で比較したものである．Aglets に比べて Maglog のステップ数は 1/25 になっている．

論理型言語はユニフィケーションと呼ばれる強力なパターンマッチング機構とバックトラック機構を持ち，推論機構やプランニング機構などを記述しやすいため，エージェントの記述に適している．本論文では，論理型言語に基いたモバイルエージェントシステムの実現に焦点を絞る．論理型言語に基くモバイルエージェント記述用言語は，これまでにいくつか提案されており [17]～[19]，強マイグレーションを実現したものもあるが，次に示す2つの機能を両方兼ね備えたものは存在していない．

1. 移動先のデータおよびプログラム (以後，移動先リソースと総称する) を簡単に利用する機能

2. エージェントの戻り先をプログラマに代わって管理する機能

プログラミング言語の存在理由はプログラムの書きやすさである．書けるか否かだけを問うならば機械語であらゆるプログラムを記述することができる．モバイルエージェントシステム記述用言語としての書きやすさは上記の2点に集約されている．モバイルエージェントは移動先リソースを利用するために移動するのであり，移動の簡便さの概念には経路管理の容易さが含まれるからである．

移動先リソースの利用については，以前から，場やフィールドと呼ばれるオブジェクトが様々に提案されている [20]～[23]．しかし，このようなオブジェクトを，複数

```
1   import com.ibm.aglet.*;
2   import com.ibm.aglet.event.*;
3   import java.net.*;
4   public class SimpleAglet extends Aglet
5     implements MobilityListener {
6       URL hostA = null, hostB = null, hostC = null;
7       public void onCreation(Object init) {
8           hostA = new URL("atp://hostA/");
9           hostB = new URL("atp://hostB/");
10          hostC = new URL("atp://hostC/");
11          addMobilityListener(this);
12          dispatch(hostB);
13      }
14      public void onArrival(MobilityEvent event) {
15          if (event.getLocation().equals(hostA)) {
16              // 終了
17          } else if (event.getLocation().equlas(hostB)) {
18              hostBでの処理;
19              dispatch(hostC);
20          } else if (event.getLocation().equlas(hostC)) {
21              hostCでの処理;
22              dispatch(hostA);
23          }
24      }
25  }
```

(a) Aglets(弱マイグレーション)

```
1   (hostBでの処理, hostCでの処理 on hostC) on hostB
```

(b) Maglog(強マイグレーション)

図 2.1: ホストの巡回プログラム例

コンピュータにまたがるユニフィケーション [24] やバックトラック機構と組み合わせた言語は提案されていない．本論文では，両者の組み合わせがプログラムの書きやすさを高めることを示す．

これまで提案されているすべてのモバイルエージェント記述用言語において，エージェントの移動はいわゆる goto 型であり，移動前のコンピュータに戻ることはプログラマに任されている．しかし，移動先が多数になるとこれを管理することは煩雑であり，バグの温床となる．すなわち，戻り先の管理をプログラミング言語側で行うことが重要である．戻り先の管理については，手続型プログラミング言語において，古くから関数やサブルーチンという概念が存在している．手続型プログラミング言語において，goto 文が危険視され [25]，一般的にサブルーチンや関数呼び出しによって実行個所が移されるようになっているのと同様，モバイルエージェントの移動も，プログラミングの容易性の観点から，サブルーチン型の方が望ましい．

本論文では，以下の特長を持つ，マルチモバイルエージェントシステム記述用言語 Maglog(Mobile AGent system based on proLOG の略) を提案する [26]～[33]．

1. 論理型言語 Prolog に基く．
2. 複数コンピュータにまたがるバックトラックとユニフィケーションが可能．
3. 移動先リソース利用のためのフィールドを備える．
4. エージェントの移動が往復で対になる．

本章の残りは以下のように構成されている．第 2.2 節で Maglog システムの構成を，第 2.3 節で Maglog システムの特長を説明する．第 2.4 節で実装の基本方針を述べた後，第 2.5 節で例題プログラムを通じて Maglog システムの有用性を示す．第 2.6 節において議論および他言語との比較を行い，最後に第 2.7 節で本章をまとめる．

2.2 Maglog システムの構成

2.2.1 エージェント

エージェントは，ネットワークで結合されたコンピュータ群において動作する．以後，エージェントが動作するコンピュータのことをホストと呼ぶ．

エージェントはユーザから直接生成されるか，既存のエージェントが

```
create(AgentID, File, Goal)
```

という形式の組み込み述語を実行することにより生成される．ここで，AgentID は出力引数であり，生成されたエージェントの識別子が返される．File および Goal

には，それぞれエージェントのプログラムが格納されたURL，生成されたエージェントが最初に実行すべきゴールを与える．以後，Prologの手続(同じ頭部を持つ節の集合)を，create/2のように，名前(name)と引数の個数(arity)を“/”で区切って表記する．また，この表記のことをPredSpecと呼ぶ．

与えられたゴールを実行し終わったエージェントは消滅する．

組み込み述語create/2を実行したエージェントを「親のエージェント」，それによって生成されたエージェントを「子のエージェント」とみなすことにすると，エージェント群は親子関係の階層構造を形成する，任意のエージェントについて，親をたどることにより，ユーザに直接生成されたエージェントに行き着く．これをユーザエージェントと呼ぶことにする．もちろん，ユーザエージェントはシステムに複数存在し得る．ユーザエージェントが生成されたホストをユーザエージェントおよびその子孫のエージェントの本籍地と呼ぶ．また，ユーザエージェントを生成したユーザを，ユーザエージェントおよびその子孫のエージェントの所有者と呼ぶ．

エージェントの所有者，または祖先のエージェントは

```
kill(AgentID)
```

という組み込み述語によって，任意の時点でAgentIDで指定されるエージェントを消滅させることができる．

組み込み述語kill/2のために，エージェントの位置を追跡する必要がある．これは，エージェントが移動するごとに，本籍地に登録することで実現する．この方法は，[34]で述べられているLogging，Brute-Force，Registryの3つの方法のうち，Registryに近いが，単一のレジストリサーバに処理が集中するという欠点を持たない．

エージェントは組み込み述語create/2が返す，システム全体で一意な識別子によって特定される．エージェントが自身の識別子を得るには

```
get_id(AgentID)
```

という組み込み述語を用いる．また，

```
bind(AgentID, Alias)
```

という形式の組み込み述語を実行することで，別名としてAliasが使用できるようになる．

2.2.2 フィールド

フィールドはエージェントが利用する手続を保持するオブジェクトであり，静的に生成され，移動したり消滅したりすることはない．

フィールドの手続は public なものと private なものに分けられ，エージェントは public な手続だけを実行することができる．private な手続はフィールド固有の名前空間に置かれるので，エージェントから参照することはできないが，public な手続から private な手続を呼び出すことはできる．

手続を public とするには，フィールド内に

```
:-public PredSpec [ , PredSpec ... ].
```

という宣言文を書く．

public な手続は原則として読み出ししかできない．エージェントが節を削除したり追加したりできるようにするには，フィールド内で

```
:-dynamic PredSpec [ , PredSpec ... ].
```

と宣言する必要がある．dynamic な手続は自動的に public になる．

dynamic な手続は一時には高々1つのエージェントが占有できる．占有は，エージェントが参照することで自動的に行われ，エージェントがその手続を参照しなくなったとき，すなわち，その手続以前にバックトラックした場合か，消滅した場合に自動的に解かれる．あるエージェントが，すでに他のエージェントに占有されている手続を参照しようとしたときは，占有が解除されるまで停止する．占有が一定時間解除されない場合に，停止したままにならず，参照を失敗終了させたい場合は，後述の組み込み述語 `within/2` を使う．

2.2.3　エージェント間通信

エージェント間の通信手段として，同期メッセージ通信を用意する．送信には組み込み述語

```
send(AgentID, Message)
```

を，受信には組み込み述語

```
recv(AgentID, Message)
```

を用いる．いずれの述語もメッセージの送受信が完了するまで停止する．メッセージの送受信完了を待つことなく別の動作をしたい場合は，組み込み述語 `create/2` によって子エージェントを生成してメッセージの送受信を任せ，後から結果を受け取るようなプログラムを書けばよい．

送信されたメッセージは，まず受信エージェント固有のメッセージボックスに先着順に格納される．組み込み述語 `recv/2` が実行されたとき，第2引数とメッセージボックス内のメッセージがユニフィケーションによって検索される．

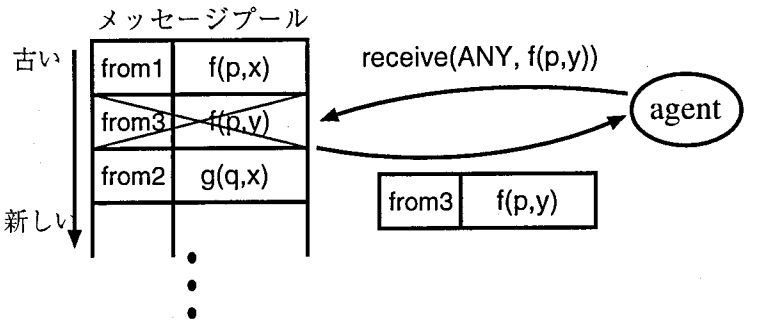


図 2.2: ユニフィケーションによるメッセージ受信

2.2.4 タイムアウトを指定したゴール実行

次の3つ場合に，エージェントのゴール実行がいつまでも完了しないことがあり得る．

1. 移動しようとするホストが稼働していなかったりネットワークに故障が発生している場合の組み込み述語 on/2(これについては後述する)．
2. 相手エージェントと同期できない組み込み述語 send/2 と組み込み述語 recv/2.
3. 他のエージェントが占有している dynamic な手続への参照．

そこで，指定時間内にゴールを実行することを指定する組み込み述語 within/2 を用意する．すなわち，

```
within(Goal, Sec)
```

は，Sec 秒以内に Goal の実行が完了しない場合は失敗する．

2.3 Maglog システムの特長

2.3.1 フィールドによる移動先リソースの利用

エージェントは

```
in(Goal, フィールド指定子)
```

という組み込み述語（中置記法を用いて Goal in フィールド指定子と記述できる）によって，フィールド Field の public な手続のうち，参照すると指定してあるものをスコープに入れて (以後，このことを，フィールドを attach するという) ゴール

Goal を実行することができる．Goal の実行が終われば，Field で定義された手続はスコープから外される (以後，このことを，フィールドを detach するという)．ここで，フィールド指定子とは

1. フィールド
2. import(フィールド指定子，[PredSpec，...])
3. merge(フィールド指定子，[PredSpec，...])

のいずれかであると再帰的に定義する．

フィールドを attach するとき，Field の public な手続は自動的にはスコープに入らない．エージェントは，スコープに入ったフィールドの public な手続を自ら定義した手続と区別なく扱えるため，意図しない手続がスコープに入ることを避けなければならないからである．これは Java におけるクラスやパッケージの参照手段である import 宣言が，クラスが提供する public な部分を一括してスコープに入れることと対照的である．

手続をスコープに入れるためには，フィールド指定子に

```
import(Goal，[PredSpec，...])
```

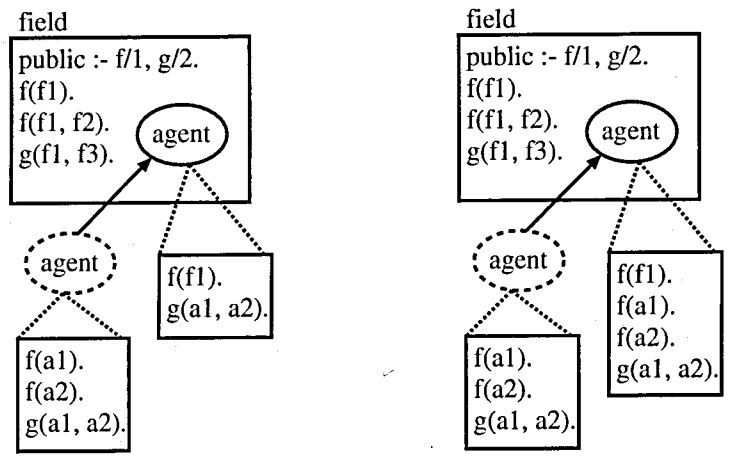
という構造（中置記法を用いて Goal import [PredSpec，...] と記述できる）を指定する．これにより，PredSpec で指定した手続をスコープに入れて Goal を実行することができる．例えば

```
G in F import [p/2]
```

は手続 p/2 をスコープに入れて，フィールド F でゴール G を実行する．

フィールドから import する手続は，原則として，エージェントの手続や，他のフィールドの手続と融合されない．すなわち，同じ手続を構成するものとはみなされない．同じ名前を持つ手続については，後から import されたものだけが有効となる．融合したい場合は組み込み述語 import/2 の代わりに，組み込み述語 merge/2 を使う．組み込み述語 merge/2 は複数のフィールドに分割されたデータベースを統合するときなどに使える．組み込み述語 import/2 と組み込み述語 merge/2 を分けるのは，プログラマが意識しないうちに加わった節によって手続が変化してしまうことを避けるためである．

エージェントは複数のフィールドを同時に複数 attach できる．また，複数のエージェントが同じフィールドを同時に attach することもできる．この様子を図 2.4 に示す．agent-a の p というゴールの呼び出しが，attach するフィールドによって異なる手続を実行することに注意されたい．これはオブジェクト指向言語におけるメソッドの多相性に相当している．



(a) Goal in field import [f/1]. (b) Goal in field merge [f/1].

図 2.3: エージェントのスコープ ((a)`import/2`, (b)`merge/2`)

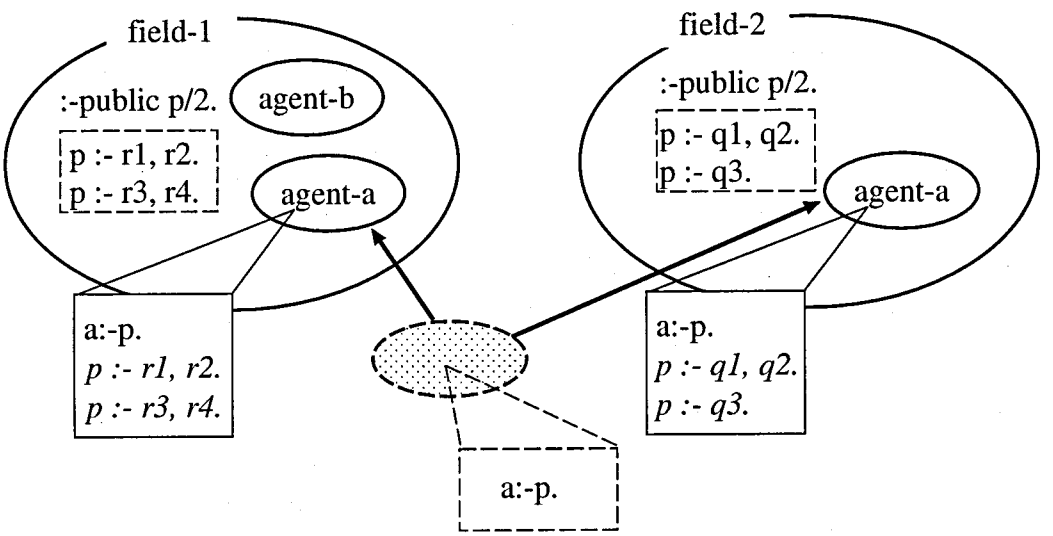


図 2.4: フィールドによる多相性の実現

2.3.2 エージェントの移動

エージェントがattachできるフィールドは，自身が存在しているホスト内のものに限られる．他のホスト内のフィールドをattachするためには，まずそのホストに移動しなければならない．そのためには

```
on(Goal, Host)
```

という組み込み述語（中置記法を用いて `Goal on Host` と記述できる）を用いる．これは，ホスト `Host` に移動してゴール `Goal` を実行することを表している．エージェントの移動方式は強マイグレーションであり，移動後は移動直前の状態から実行を再開する．論理変数の束縛状態もすべて保存されるので，ホスト間のユニフィケーションが簡潔に記述できる．`Goal` の実行が終われば，エージェントは自動的に移動前の場所に戻る．すなわち，Maglogではエージェントの移動単位が往復で対になっていて，プログラマがエージェントの経路管理をする必要はない．このことは，移動先が多数になり，移動経路が複雑になってくると重要な意味を持つ．手続型プログラミング言語においてgoto文が危険視され[25]，サブルーチンや関数呼び出しによってプログラムの実行個所が移されるようになっているのと同様，モバイルエージェントの移動もサブルーチン型の方がよい．

ホスト間のユニフィケーションと往復を単位とする移動の組み合わせが，いかにプログラムを簡潔にするかという例を図2.5に示す．これは，2つのホストに分散されたProlog節データベースを検索するプログラムである．例えばエージェントに `mail(debian,X)` というゴールを与えると，ホスト group-s のフィールド group-f にあるデータベースから debian グループに属する人物を探し，その人のメールアドレスをホスト mail-s のフィールド mail-f にあるデータベースから検索する．

図2.5のエージェントのプログラムにおいて，ゴール `group(Group,Person)` はホスト group-s において，ゴール `mail(Person,Address)` はホスト mail-s において実行されるが，両者に共通する論理変数 `Person` はユニファイされる．ホスト間のユニフィケーションが，普通のPrologプログラムと同様に，同じ名前の論理変数を使うだけで記述できることに注意されたい．実際，ゴール `group/2` と `mail/2` の呼び出しにおいて，`in` 以降を除けばPrologプログラムそのものである．

また，ホスト間のバックトラックも普通のPrologと同様にプログラマが意識することなく行われる．図2.5の例で，ゴール `mail(X,Y)` より以前にバックトラックし，`group(debian,X)` の別解を探す場合には，エージェントは自動的にホスト group-s に移動し，フィールド group-f をattachする．

なお，組み込み述語 `in/2` の第1引数で指定するゴールの中で，組み込み述語 `on/2` を実行することはできない．つまり，あるフィールドをattachしたまま他のホストに移動することはできない．フィールドはホストに属するオブジェクトだからである．

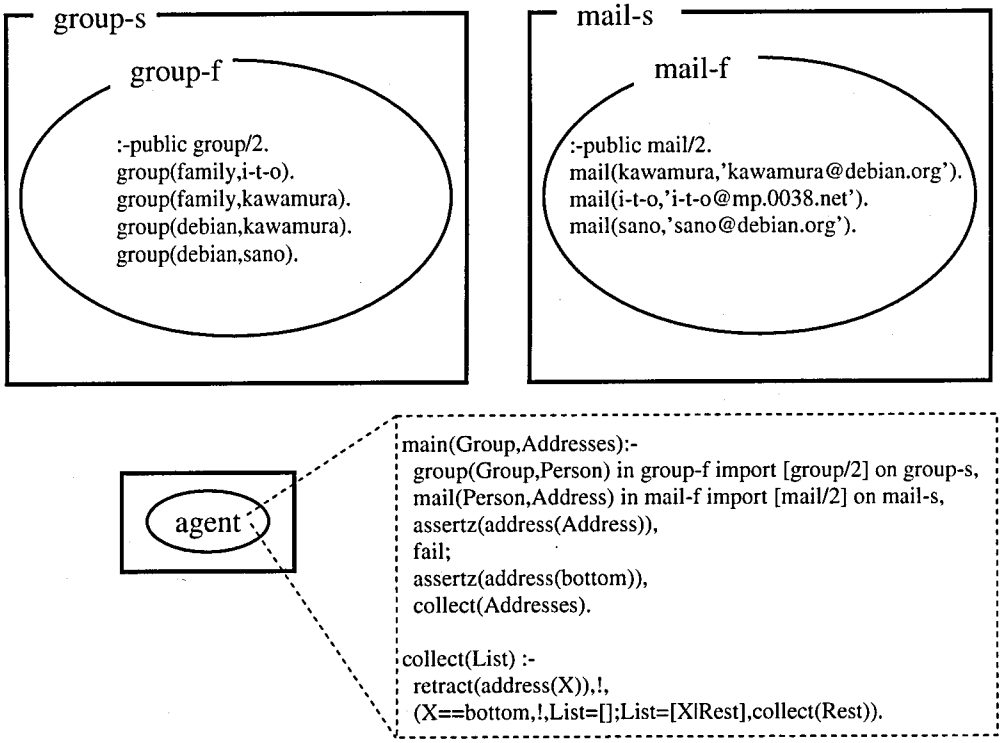


図 2.5: 分散された節データベースの検索

2.4 実装

Maglog システムは，ネットワークで結合されたホスト群において動作する．各ホストにはエージェントサーバが 1 つずつ存在している．エージェントサーバは，エージェントの実行環境を提供する．エージェントが到着した際に，新たなスレッドとして Prolog インタプリタを起動し，エージェントの実行を再開させる．また，エージェントが移動する際の転送を行なうほか，自らを本籍地とするエージェントのメールボックスを管理し，エージェント間のメッセージ配送手段を提供する．

Maglog システムの実装言語としては Java が最適であると考えられる．マルチプラットフォーム，ネットワーク対応，セキュリティ重視といった性質を備えたプログラミング言語は他に見当たらない．また，表 2.1 に示すように，Java 環境には Maglog の実装に必要な様々な要素技術がすでに存在している．

Java 技術に基く Maglog システムの構成を図 2.6 に示す．

ここで，図 2.6 中のマスターサーバは，以下に挙げるシステム全体で一意でなければならないデータを管理するためのものである．

1. エージェントサーバ名

2. エージェントの識別子

表 2.1: Maglog システムの構成要素と Java 技術の対応

Maglog の構成要素	Java 技術
エージェントサーバ	サーブレット
ユーザインタフェースプロセス	Web ブラウザ+FORM
エージェントの移動	RMI

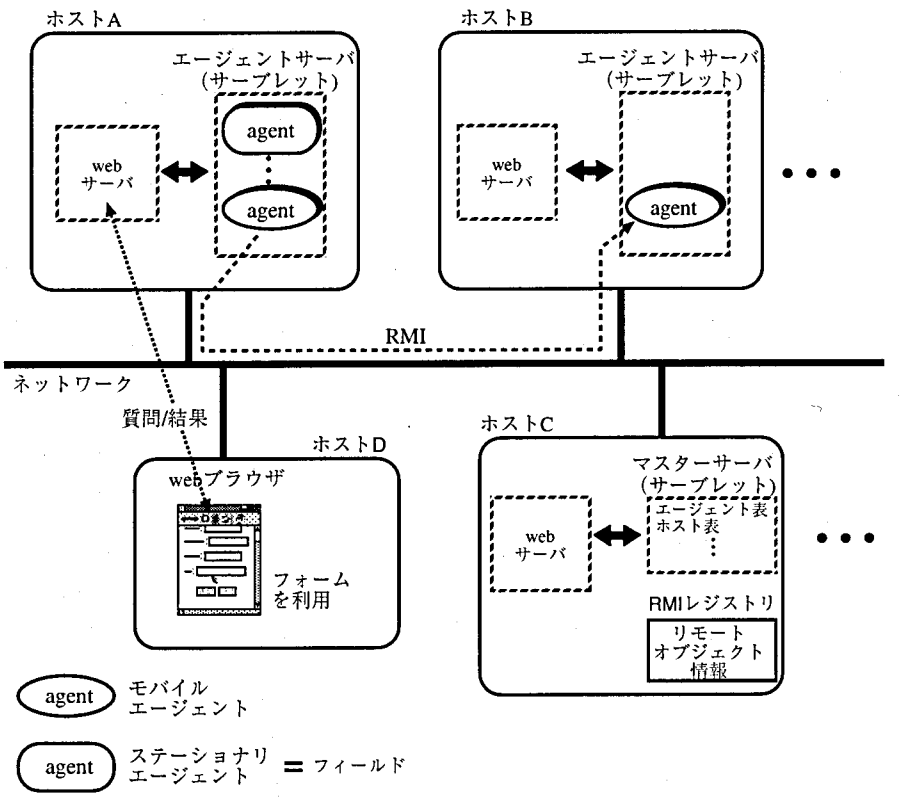


図 2.6: Maglog システムの Java 技術による実装

3. エージェントの別名

4. エージェントの本籍地，所有者情報

エージェントサーバの記述にはWarren Abstract Machine [35] [36] に基く，PrologからJavaへのトランスレータであるProlog Café [37] [38] を用いる．Prolog Caféは，Prologプログラムと Javaプログラムをシームレスに結合させることができる上，Javaの既存のクラスライブラリとの親和性が高く，図2.6に示した構成が容易に実現できるからである．

実装を簡潔にするために，フィールドはオブジェクトとしては作らないものとする．すなわち，フィールドをattachしてゴールを実行することを，そのフィールドを管理するエージェントにゴールの実行を依頼するという形で実現する．

2.5 記述例

簡単な記述例を通じてMaglogの特長を説明する．

2.5.1 スケジューリング

Maglogの特長の一つは，ホスト間のユニフィケーションやバックトラックが可能なことである．そのため，複数のデータベース間での制約を満たすといったことが容易に記述できる．一例として，会議のスケジューリングを行うプログラムを検討する．

複数人の社員がいる会社で，各社員は別々のホスト上に存在する自分固有のフィールドにスケジュールデータを格納しているものとする．各人の空き時間は1時間単位で管理されていて，`free(Day,Hour)` という事実節によって表現されるものとする．また，会議を召集するエージェントは，会議参加者全員の空き時間のうち，もっとも早いものを予約するものとする．

会議のスケジューリングを行うプログラムを付録A.1に示す．この例では，エージェントを

```
create(AgentID, scheduling, main(Day, Hour))
```

のように生成してゴールを与えると，`Day` に1が，`Hour` に11がバインドされる．

複数ホストにまたがる制約の解決は，手続`search/3`において再帰を用いて簡潔に表されていることに注意されたい．このようにPrologプログラミングの常套手段である再帰が使えるのは，ホスト間のユニフィケーションとバックトラックがPrologの拡張として自然に記述できるからである．

会議を召集するエージェントは参加する各人の予定を直接書き換えるが，free/2とschedule/3は予約完了まで占有される．また，占有にはこの例では300秒の実行制限をつけているので，複数のエージェントが並行して別々の会議の予約を行っても問題はなく，スケジュールに矛盾が生じたり，デッドロックが起こることはない．

2.5.2 買物

次に，複数のホストが同一商品を提供している場合に，最安値のものを購入する例を考えてみる．具体的な問題とその解法は次の通りである．host_A，host_B，host_C，host_Dが存在し，host_B，host_C，host_Dは同一の商品を販売している．host_Aのエージェントが3つの子エージェントを生成し，host_B，host_C，host_Dに派遣する．各エージェントは商品の値段を調べ，親エージェントに報告(通信)する．親エージェントは最も安い商品を選んで購入指令を出すとともに，他には不購入の指令を出す．

付録A.2に買物プログラムを示す．各ホストは，marketというフィールド内に，商品の名前と値段をprice/2という事実節として，商品の在庫数をstock/2という事実節として格納しているものとしている．また，商品を購入する子エージェントは，stock/2を直接書き換えるものとする．

この例のように，Maglogでは，子エージェントを生成して仕事をさせるスタイルのプログラムを書くことも容易にできる．1個の巨大なエージェントを移動させるのではなく，機能ごとに分割されたエージェント群を必要に応じて移動させることにより，ネットワークのトラフィックを抑えることができる．

2.6 検討

2.6.1 フィールドのdynamicな手続

フィールドのdynamicな手続とは，エージェントがその構成要素である節を追加したり削除したりできる手続のことである．複数のエージェントが同時に節を追加したり削除しようとした場合に整合性が取れなくなる恐れがあるので，dynamicな手続へのアクセスにおいては排他制御を行う必要がある．

占有時間を短くするためには排他制御を節単位で行うのがよいが，バックトラックを含めた強マイグレーションを実現するためには節単位の排他制御は不可能である．あるエージェントAがバックトラックしたときに実行すべき節を，他のエージェントBが削除してしまったとしたら，エージェントAの実行継続が困難になるからである．

そこでMaglogでは，排他制御を手続単位で行うこととした．すなわち，エージェントがバックトラックによってその手続中の節を参照する可能性がある間は，手続全体が占有される．こうすることで，先に述べたような不都合は発生しなくなる．ここで，不当に長く手続を占有しないことはプログラマに任される．すなわち，バックトラックによってその手続を再び訪れる必要がない場合は，カットを明示的に記述して，それをシステムに伝えることが求められる．

エージェントがdynamicな手続を占有したまま，他のdynamicな手続を参照しようとして停止させられた場合，デッドロックの危険性があるが，組み込み述語`within/2`を用いることで回避できる．すなわち，dynamicな手続への参照は常にタイムアウトを指定して実行するべきである．

2.6.2 ホスト間の移動単位

エージェントの移動単位として往復を対にした場合，goto型の移動に比べてエージェントの転送量が増大してしまう場合がある．たとえば，n個のホストを巡回して元のホストに戻ってくる場合，goto型移動の場合は$n+1$回エージェントを転送すればすむのに対し，往復を対にした場合は$2n$回の転送が必要になってしまう．

この転送量の増大は，実装技術により容易に回避できる．サブルーチン型の移動とは，移動元ホストをリターンスタックに積んで移動先ホストにgoto型移動することに他ならない．したがって，戻ってからそのホストで処理する内容がない場合は，リターンスタックを書き換えずにgoto型移動をすれば，そのホストには戻らないことになる．図2.7(a)はエージェントの最適化前の移動経路を，図2.7(b)は最適化後の移動経路を示している．(b)の場合，エージェントはgoto型移動のシステムとまったく同じ経路を辿ることになる．つまり，Maglogにおけるエージェントのサブルーチン型の移動方式は，効率を犠牲にすることなく，プログラムを書き易くするという利点を持つ．

2.6.3 関連研究との比較

これまでに，モバイルエージェントシステムを記述するための言語やフレームワークは多数提案されている．Aglets [39] [40]，Concordia [41] [42]，MobileSpaces [43]，Voyger [44]，Bee-gent [45]など，その多くがJavaのクラスとして実装されている．これらと，W-Prolog [46]，MINERVA [47]，jProlog [48]などのJava処理系上のProlog処理系を組み合わせると，論理型言語に基くモバイルエージェントシステム記述用言語になるが，このような組み合わせでは強マイグレーションは実現されない．Java VMには，性能上やセキュリティ上の理由から，実行状態領域を参照，改変するた

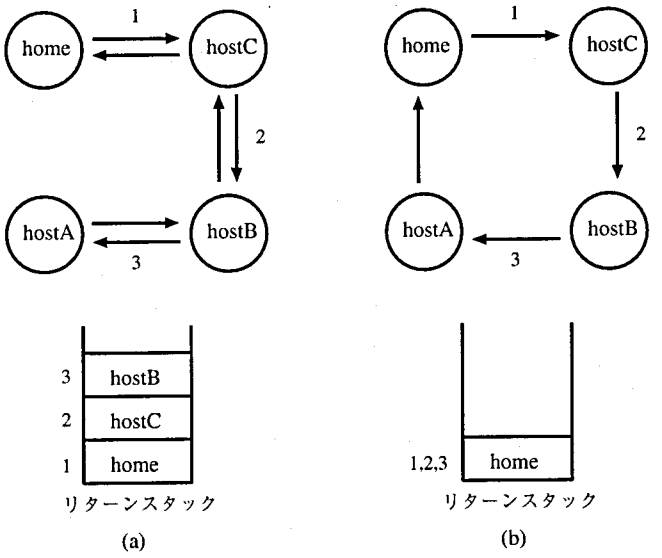


図 2.7: エージェントのホスト間移動 ((a) 最適化前 (b) 最適化後)

めの API が用意されていないので，これらのフレームワークではエージェントの移動方式は弱マイグレーションにならざるを得ないからである．

次に，これまでに提案されている論理型言語に基くモバイルエージェント記述用言語と Maglog を比較する．

Flage [23] [49] は Maglog のフィールドに似た「場」の概念を持つプログラミング言語であり，Quintus Prolog 上に実装されている．エージェントのモデルは非同期並行オブジェクトであり，メタ部分の記述は Prolog をベースとしているが，エージェントそのものは手続き的に記述する．論理型言語としての強マイグレーションは実現されず，ホスト間のユニフィケーションやバックトラックは行えない．

Plangent [17] はモバイルエージェントに知的な振舞いを実現するためのプランニング機構を備えたシステムである．エージェントのアクション定義 (エージェントの状態の事前条件，事後条件，行動内容の組) には Prolog を用いるが，エージェントそのものは Plangent Script 言語という手続型の言語で記述する．プランニング中，すなわち，Prolog が使用されている部分での強マイグレーションは実現されておらず，ホスト間のユニフィケーションやバックトラックは行えない．また，移動先リソース利用のための仕組は用意されていない．

Jinni [18] [50] [51] は Java 上に Prolog 処理系を実装する点で Maglog と類似している．エージェントの移動後にバックトラックが継続できないので，論理型言語における真の強マイグレーションは実現されていない．また，移動先リソース利用のための仕組は用意されていない．

Milog [19] [52] は Java 上に Prolog 処理系を実装する点で Maglog と類似しており，

バックトラックを含めた論理型言語としての強マイグレーションを実現している点もMaglogと同様である．移動先リソース利用のための仕組が用意されていないことが示しているように，Maglogがプログラマ寄りの部分に力点を置いているのに比べて，システム寄りの部分に力点を置いている．

また，ここで取上げたすべての言語において，エージェントの移動はgoto型であり，Maglogのみが往復を対とするサブルーチン型である．

2.7 結言

マルチモバイルエージェントシステム記述用言語Maglogを提案し，その有用性を簡単な例を通して示した．Maglogはエージェントの移動方式が強マイグレーションであること，ホスト間のユニフィケーションやバックトラックが可能であること，移動先のデータおよびプログラムを直接利用できるフィールドという概念を備えていること，エージェントの移動が往復で対になっていることから，エージェントの記述が容易かつ簡潔に行える言語となっている．本論文では実装の基本方針を与えたが，実働する処理系の作成は今後の課題である．

第3章　階層型ネットワークにおける分散データの検索効率の評価

3.1　緒言

コンピュータネットワークが広く使われるようになっており，それに伴い分散システムの重要性もますます大きくなっている．ここで，分散システムとは，ネットワークで相互接続された複数の計算主体上で機能するアプリケーションのことである．

分散システムは次の3種類に大別できる [53].

クラアント/サーバアーキテクチャ クライアントまたはサーバの役割を固定したシステムの相互接続からなるアーキテクチャ．集中システムをネットワークの物理的な広がりに対して拡張したものと言える．

水平分散アーキテクチャ 水平的なシステムの相互接続からなるアーキテクチャ．接続するクライアント相互の理解とモデル化およびインタフェースの厳密な定義が必要．

マルチエージェントアーキテクチャ エージェントの相互接続からなるアーキテクチャ．システムの動的な変化に対処することを目指す．ここでいうシステムの変化には，エージェントの増減だけでなく，ネットワーク構造の変化も含む．必要に応じ一時的に構築される Ad Hoc ネットワーク [54] が動的なネットワーク構造の例である．

マルチエージェントシステムは，その構成が時々刻々と変化する性質を持ち，かつ，個々のエージェントが自律性を持つので，システム全体が保有するデータを管理エージェントが集中的に保有するのではなく，個々のエージェントが分散して保有すると考えるべきである．この場合，あるエージェントが自分の持っていないデータを必要としたとき，そのデータをどのエージェントが持っているかを知る必要がある．一般的な解決法は，ちょうど分散データベースのように，データとそれを持つエージェントの対照表を特別なエージェントが管理することであろう．しかし，エージェントが生成・消滅・移動などを繰り返す状況においては，その管理は事実上不可能である．また，これは結局集中システムとなるので，管理エージェントに対する

負荷集中や，管理エージェントが故障した場合にデータ検索がすべて不可能になってしまうといった集中システムの欠点が再び問題になる．そこで，このような対照表を管理しないデータ検索を考える．すなわち，エージェントは他のエージェントに次々に問い合わせることで，欲しいデータを見つけるものとする．もちろん，その場合にもデータを効率良く取得することが必要である．そのためには，どのエージェントがどういうデータを持っているかという情報を紹介したり口コミで広めたりするような，ヒューリスティックな分散アルゴリズムの開発が望まれるが，まず，対照表を持たないデータ検索がどの程度の検索効率を持っているかを評価する必要がある．

従来のクライアント/サーバアーキテクチャや水平分散アーキテクチャ等の性能評価については，以前から各種の立場で多数の報告がなされている [55]．高速な分散環境のために開発された LAN [56] や分散環境における Kerberos プロトコル [57] の性能評価もあるが，マルチエージェントアーキテクチャにおける分散データの検索効率の評価はこれまでになされていない．そこで本研究では，ネットワーク内の多数のエージェントがデータを分散管理しているときのデータ検索効率の評価を行なう [58]～[61]．

本章では，ネットワークにおいて，データを各クライアントエージェントが分散して保管している場合（以後，分散管理方式と呼ぶ）とスーパバイザエージェントが集中して保管している場合（以後，集中管理方式と呼ぶ）とのデータ検索効率を相対評価する．また，ネットワーク上に分散しているデータから必要なものを効率よく取得する上で，各エージェントがデータを一時記憶することは重要である．このため，一度検索したデータを有限長のキューに保持するようなシステムのデータアクセス効率の時刻変化についても解析を行い，効率的な一時記憶容量について議論する．さらに，ネットワーク構造やデータの検索頻度の違いが検索効率に及ぼす影響について評価する．評価は，統計的手法に加えて，実働している WWW キャッシュシステムから抽出された WWW アクセス記録を基にしたシミュレーションによっても行う．システムトポロジとしては出来るだけ柔軟性のある階層型ネットワークを対象とする．階層型ネットワークは，インターネットやイントラネットにおいて，最も一般的なトポロジである．

本章は以下のように構成されている．まず第 3.2 節で対象となるシステムを示す．第 3.3 節で集中管理方式と分散管理方式におけるデータ検索効率を解析し，定式化する．第 3.4 節はその評価を行う．次に，第 3.5 節において，実験的評価として WWW アクセス記録によるシミュレーションを行う．最後に第 3.6 節で本章をまとめる．

3.2 対象システム

本章では，ネットワーク内に多数のエージェントが存在する場合に，1つのエージェントが1つのデータを求めて得られるまでに要する平均時間の解析を，集中管理方式と分散管理方式において行う．

複数のエージェントが同時にデータを検索する際の通信路での衝突を解析するためには，いくつかの確率モデルが考えられるが，本章では，集中，分散の2方式の相対評価を得ることを最大の目標としているので，確率モデルを静的に，かつ単純化して最大競合状態を想定し，次の仮定を設ける．

- 各エージェント a_i は所望のデータ d_i の検索を一斉に行う．d_i を保持しているエージェント $a_j(i \neq j)$ は，d_i 検索メッセージの応答メッセージとしてデータ d_i そのものを a_i に対して送信する．d_i を保持しないエージェントは d_i 検索メッセージを無視する．また，すべてのエージェントがデータを取得するまで，次のデータ検索は行われない．また，ネットワーク上に存在しないデータの検索は行われない．

集中管理方式においては，唯一のスーパバイザエージェントがすべてのデータを保持しており，他のエージェントは，スーパバイザエージェントに要求することで求めるデータを得ることができる．

分散管理方式においては，データは各エージェントに同数ずつ分散されており，エージェントがあるデータを必要とするとき，まず自分が保持しているかどうかチェックする．自分が保持していないとき，まず自分の所属するサブネットワーク，次に近傍のサブネットワーク…と，データが見つかるまで順に検索範囲を拡大していくものとする．

また，一度検索して見つけたデータを有限長のキューに保持することで，二度目以降の検索が省略できると考えられるので，キューの有無によって，データを求めて得られるまでに要する時間がどのように変化するかについても，分散管理方式において解析する．

以上のような解析の視点を具体的にするために，上記の仮定に次の仮定を加える．

- 各エージェントは均一の処理能力を持つ．したがって，データ検索にかかる時間として，エージェント内部でのデータ検索のための処理時間を無視しても解析結果には影響しない．

- データはすべて同一サイズである．

- エージェント a_i からのデータ d_i の検索メッセージと，それに応答したデータ d_i をアトミックなメッセージとする．あるエージェントが要求したデータを

保持しているエージェントは，そのデータを送る．保持していないエージェントは要求を単に無視する．したがって，エージェントからのデータ要求とそれに対する返事をアトミックなメッセージとしても，解析結果には影響しない．

また，階層型ネットワークに対して以下のような条件を設ける．これらの条件と先の3つの仮定は，統計的解析のために設けるが，いずれも現実的なものであり，集中管理方式と分散管理方式の性能比較を行う上で，議論の一般性を失わせるものではない．

- 最下層のサブネットワーク（以後，階層1のサブネットと呼ぶ）には w 個のエージェントが存在する．
- 階層1のサブネットが w 個集まって，上位のネットワーク，つまり w^2 個のエージェントからなる階層2のサブネットを構成する．この結合が h 回繰り返されることにより，h 個の階層を持つネットワークが構成される．したがって，このとき，ネットワーク上に存在するエージェントの個数は w^h となる．図3.1は，$w=3$，$h=2$ のネットワークである．
- 同時に1メッセージしか通信路上を伝送されない．
- エージェント間の通信時間は，衝突がない限り，その距離によらず一定である．

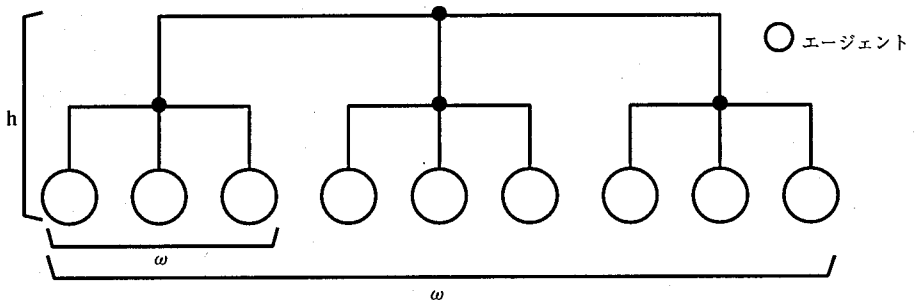


図 3.1: 対象ネットワーク例

3.3　集中管理方式と分散管理方式の定式化

同時に複数のメッセージが通信路を流れることはできないので，複数のエージェントが同時にメッセージを送ろうとした場合は，即座にメッセージを送ることのできるエージェントは1つのみで，残りは通信路が空くのを待たなければならない．1

ステップを，エージェント a_i がデータ d_i の検索メッセージを送信し，d_i を受信するのに要する時間間隔と定義する．1エージェントが，データを検索しようとしてから得られるまでにかかる平均所要ステップ数（以後，ステップ数と呼ぶ）の期待値を，集中管理方式と分散管理方式で比較してみよう．

3.3.1 集中管理方式

ネットワーク上の任意のエージェントをスーパバイザエージェントとする．同時には1エージェントしかスーパバイザエージェントにアクセスできないので，w^h 個の各エージェントがそれぞれ必要とするステップ数は，0から w^h-1 のいずれか異なる値をとると考えることができる．したがって，エージェント a_i がデータ d_i を獲得するのに必要とするステップ数の期待値を E_c とすると次式で表される．

$$E_c = \frac{\sum_{k=0}^{w^h-1} k}{w^h} = \frac{w^h-1}{2} \tag{3.1}$$

3.3.2 分散管理方式

この方式では，エージェント a_i は，階層1，階層2，…，階層 h の順に，その階層に属するエージェントに向かってデータ d_i の検索メッセージを送り，データを検索する．同一階層内のサブネットの各エージェントは，マルチキャストによって一斉にメッセージを受信できる．つまり，階層 $(l-1)$ のサブネット内の全エージェントに d_i 検索メッセージを送信後，1ステップ経ても d_i が送られてこないとき，次に階層 l のサブネットに向けて d_i 検索メッセージを送信する．以下，同様である．

エージェント a_i がほしいデータ d_i をすでに所有している確率を P_0，自分が属する階層1のサブネット内でデータが得られる確率を P_1，階層2のサブネットのうち，自分が属する階層1のサブネットを除いたサブネットで得られる確率を $P_2\cdots$ とすると，それぞれの場合における，d_i が得られるまでの所要ステップ数の期待値 $E_d(l)$ $(l=0,\ldots,h)$ が求められる．

1. $l=0$ の場合

   自分がすでに所有するデータを得るのに要するステップ数は0なので

$$E_d(0) = P_0 \times 0 = 0 \tag{3.2}$$

2. $l=k(k>0)$ の場合

a_i が k 回目の試みとして，つまり階層 k のサブネットにメッセージを送出しようとしたときに競合する他のエージェントとは，それぞれが所属する $k-1$ 以下の階層のサブネット内に求めるべきデータが存在しないエージェントである．したがって，競合するエージェント数が j になる確率 $Q_{k,j}$ は

$$Q_{k,j} = \binom{w^k - 1}{j} \left(1 - \sum_{i=0}^{k-1} P_i\right)^j \left(\sum_{i=0}^{k-1} P_i\right)^{(w^k-1)-j} \tag{3.3}$$

となる．また，$j+1$ 個のエージェントが同時にメッセージを送出しようとした場合には競合が発生するので，$j+1$ 個のエージェントのそれぞれがかかるステップ数は，1から $j+1$ のいずれかの互いに異なる値となる．したがって，$E_d(k)$ は次式で表される．

$$E_d(k) = \sum_{m=1}^{k} \sum_{j=0}^{w^m-1} Q_{m,j} \frac{\sum_{i=1}^{j+1} i}{j+1} \tag{3.4}$$

以上より，分散管理方式のステップ数 E_d は次式で表される．

$$\begin{aligned} E_d &= \sum_{l=0}^{h} P_l E_d(l) \\ &= \sum_{l=1}^{h} P_l \sum_{m=1}^{l} \sum_{j=0}^{w^m-1} \binom{w^m - 1}{j} \left(1 - \sum_{i=0}^{m-1} P_i\right)^j \left(\sum_{i=0}^{m-1} P_i\right)^{(w^m-1)-j} \frac{\sum_{i=1}^{j+1} i}{j+1} \end{aligned} \tag{3.5}$$

3.4　統計的性能評価

前節において，集中管理方式と分散管理方式のそれぞれの期待値 E_c，E_d が定式化された．各エージェントがキューを持つときは，分散管理方式の期待値 E_d の変数 P_l が時間経過によって変化するため，E_d も変化する．本節では，エージェントがキューを持たないときと持つときに分け，具体的に P_l を求め，E_d を評価する．

3.4.1　分散管理方式においてエージェントがキューを持たない場合

ここでは，各エージェントが他のエージェントから確保したデータを保持するキューを持たないシステムの場合を評価する．

データの分布が均質な場合

データ d_j をどのエージェントが所有しているかは，すべてのエージェントについて等確率とする．すなわち，すべてのエージェントについて，データ d_j を所有する確率は，$1/w^h$ である．任意のエージェントから見て，階層 l のサブネットのうち，自分が属する階層 $(l-1)$ のサブネットを除いたサブネットに属するエージェント数は $w^l - w^{l-1}$ であるから，P_l は次式で表せる．

$$P_l = \begin{cases} \frac{1}{w^h} & (l=0) \\ \frac{w^l - w^{l-1}}{w^h} & (l=1,\ldots,h) \end{cases} \tag{3.6}$$

この P_l を用いて式 (3.5) が計算できる．

h を固定してエージェント数を変化させたときの結果を図 3.2 に，w を固定してエージェント数を変化させたときの結果を図 3.3 にそれぞれ示す．凡例で concentrated と記されている直線が集中管理方式の場合である．

図 3.2 では，h が小さければ小さいほどステップ数が小さいという結果になっている．一方．図 3.3 では，w が 2 の場合より 8 の場合の方がステップ数は小さいが，8 の場合より 22 の場合の方がステップ数が大きくなっている．これは，エージェント数を固定したときに，最小の h と最大の w の組合せ，すなわち h が 1 で w が総エージェント数の場合がステップ数を最小にするわけではないことを示唆している．

w が大きくなることは一度にマルチキャストできるエージェント数が増えることを意味し，h が大きくなることはシステム全体に並行して送られるメッセージ数が増えることを意味しており，w と h の増減はトレードオフの関係にある．各 P_l が与えられれば，エージェント数を固定したときに，最もステップ数を小さくするような w と h の組合せが存在するはずであるが，本章で議論している比較的小規模なネットワークでは w と h の組み合わせが限られており，最適構造の議論はできない．

データの分布が非均質な場合

データを複数のエージェントに分散配置するとき，物理的に近いエージェントが同じデータを必要とするように配置することが考えられるので，ここではそのようなデータ分布を扱う．すなわち，a_i と階層的に近いサブネットに d_i がより高い確率で存在することを表現するために，$P_l(l=0,\ldots,h-1)$ が幾何分布に従うものとし，次式を与える．

$$P_l = \begin{cases} p(1-p)^l & (l=0,\ldots,h-1) \\ 1 - \sum_{k=1}^{h-1} P_k & (l=h) \end{cases} \tag{3.7}$$

この P_l を用いて式 (3.5) が計算できる．

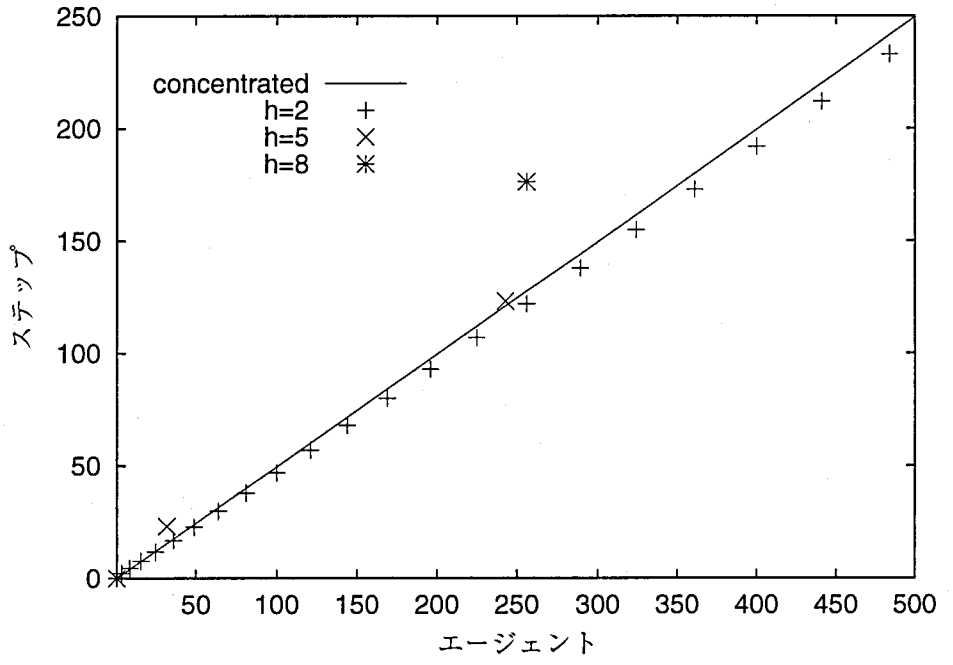


図 3.2: hを固定したときのステップ数の推移

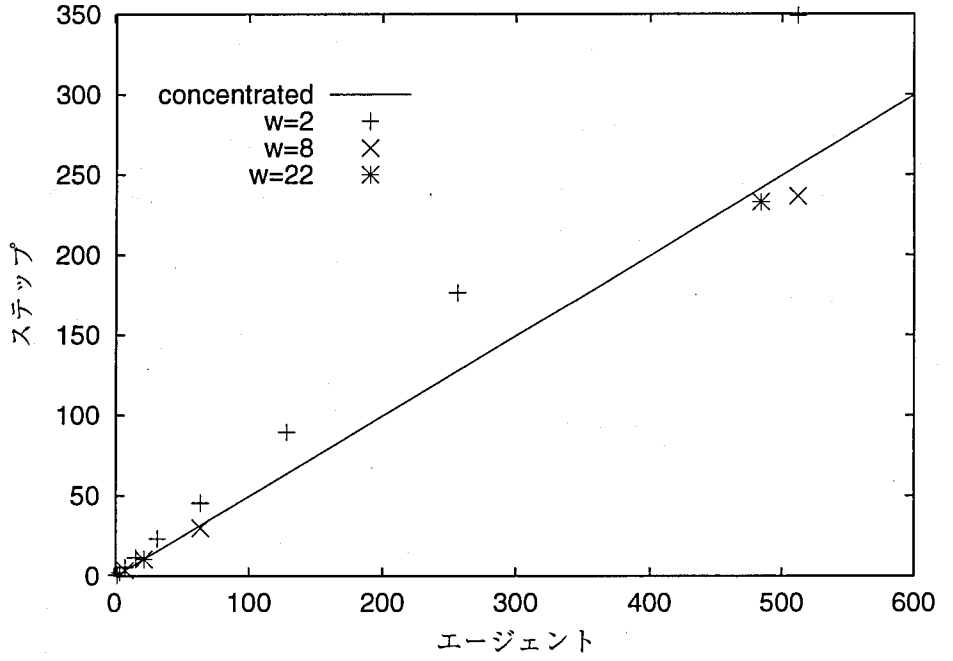


図 3.3: wを固定したときのステップ数の推移

$p = 0.1$, $p = 0.3$, $p = 0.5$ のそれぞれについて，$h = 5$ として w を変化させた結果を図3.4に示す．参考のために，第3.4.1節で示した等確率の場合 (凡例で uniform と記されている点の集合) と集中管理方式の場合 (凡例で concentrated と記されている直線) もあわせて示している．

分散管理方式においては，幾何分布の初期値が大きくなるにつれてステップ数が減少していく．各エージェントにとって，求めるデータが近傍で得られる確率が大きくなるので，小さいステップ数でデータが得られるからである．

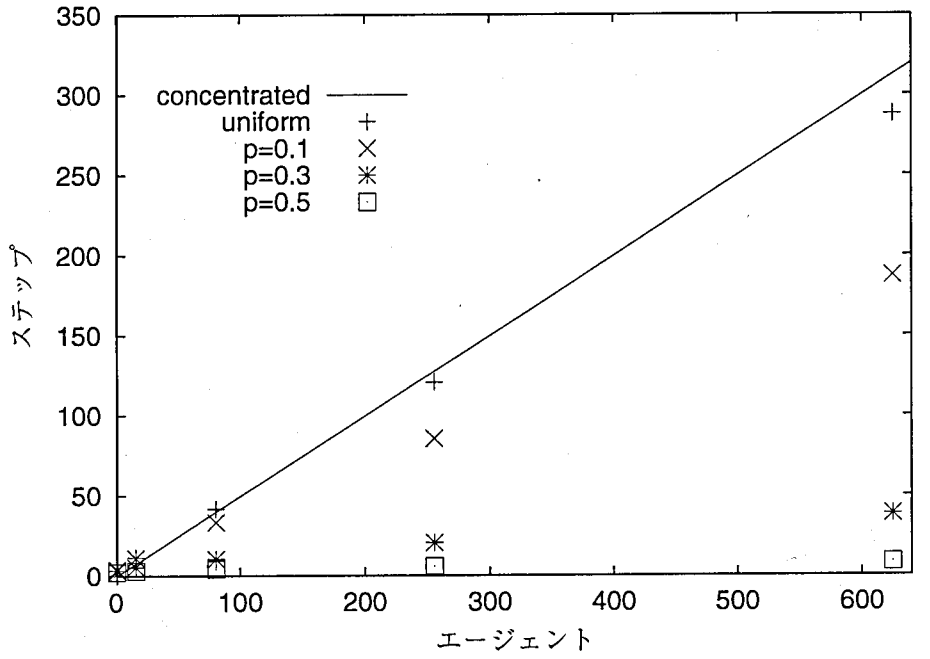


図 3.4: P_l が幾何分布に従う場合のステップ数の推移

3.4.2 分散管理方式においてエージェントがキューを持つ場合

ここでは，各エージェントが他のエージェントから確保したデータを保持するキューを持つシステムの場合を評価する．第3.2節での仮定に加え，さらに次の仮定を設けて解析を具体化する．

- n 個のデータが各エージェントに同数ずつ分散されている．
- 各エージェントは長さ q のキューを持ち，検索したデータをキューの末尾に追加する．このとき，もともと自分が持っていたデータについても，それが検索されたならばキューに保持するものとする．キューが一杯になったときにデー

タを保持する必要が生じた場合，最も古いデータを捨てる．また，同じデータを重複して保持することはなく，古い方を捨てる．

- エージェント a_i は任意のデータ d_j について，確率 $p(a_i, d_j)$ で検索する．

準備

時刻 t においてエージェント a_i がデータ d_i を検索しようとしたとき，実は長さ q の自分のキューに d_i を保持している場合があるので，その確率 $p(t, a_i, q, d_i)$ を求め，P_l を定式化する．

a_i が d_i を自分のキューに保持しているためには，時刻 t 以前に d_i を検索しているはずであるから，最後に検索した時刻 $\tau(<t)$ が存在する．このとき，τ の1ステップ後の $\tau+1$ から $t-1$ までの間に，d_i 以外の $q-1$ 種類以下のデータしか検索しなければ，d_i がキューに保持されている．

1. a_i が時刻 $\tau+1$ から $t-1$ までの間に d_i 以外の1種類のデータのみしか検索しない確率 $p(\tau, t, a_i, q, d_i, 1)$ は

$$p(\tau, t, a_i, q, d_i, 1) = \sum_{d_\beta \in D} \{p(a_i, d_\beta)\}^{t-\tau-1} \tag{3.8}$$

となる．ただし，

$$D = \{d_\beta | 1 \leq \beta \leq n, \beta \neq i\} \tag{3.9}$$

とする．

2. a_i が時刻 $\tau+1$ から $t-1$ までの間に d_i 以外に2種類のデータしか検索しない確率 $p(\tau, t, a_i, q, d_i, 2)$ は，2個のデータ d_{β_1}, d_{β_2} の $(t-\tau)$-重複組合せを用いて，以下のようになる．

$$p(\tau, t, a_i, q, d_i, 2) = \sum_{d_{\beta_1}, d_{\beta_2} \in D} \sum_{\alpha_1, \alpha_2} \frac{(t-\tau-1)!}{\alpha_1! \alpha_2!} \{p(a_i, d_{\beta_1})\}^{\alpha_1} \{p(a_i, d_{\beta_2})\}^{\alpha_2} \tag{3.10}$$

ただし，$\alpha_m (m = 1, 2)$ は自然数で，式 (3.11) 及び式 (3.12) を満たし，$\beta_m (m = 1, 2)$ は自然数で，$\beta_1 \neq \beta_2$ とする．

$$\alpha_1 + \alpha_2 = t - \tau - 1 \tag{3.11}$$

$$1 \leq \alpha_m < t - \tau - 1 \quad (m = 1, 2) \tag{3.12}$$

$$\beta_1 \neq \beta_2 \tag{3.13}$$

3. 一般に，a_iが時刻$\tau+1$から$t-1$までの間にd_i以外にk種類のデータしか検索しない確率$p(\tau,t,a_i,q,d_i,k)$は，k個のデータ$d_{\beta_1},d_{\beta_2},\cdots,d_{\beta_k}$の$(t-\tau)$-重複組合せを用いて，以下のようになる．

$$\begin{aligned} p(\tau,t,a_i,q,d_i,k) &= \sum_{d_{\beta_1},d_{\beta_2},\cdots,d_{\beta_k}\in D}\ \sum_{\alpha_1,\alpha_2,\cdots,\alpha_k}\frac{(t-\tau-1)!}{\alpha_1!\alpha_2!\cdots\alpha_k!} \\ &\quad\times\{p(a_i,d_{\beta_1})\}^{\alpha_1}\{p(a_i,d_{\beta_2})\}^{\alpha_2}\cdots\{p(a_i,d_{\beta_k})\}^{\alpha_k} \end{aligned} \tag{3.14}$$

ただし，$\alpha_m(1\le m\le k)$は自然数で，式(3.15)及び式(3.16)を満たし，$\beta_m(1\le m\le k)$は自然数で，互いに異なるものとする．

$$\sum_{m=1}^{k}\alpha_m = t-\tau-1 \tag{3.15}$$

$$1\le\alpha_m<t-\tau-1\quad(1\le m\le k) \tag{3.16}$$

$$\beta_1\cdots\beta_k\quad\text{は互いに異なる．} \tag{3.17}$$

以上の議論より，時刻tにおいて，a_iがd_iをキューに保持している確率$p(t,a_i,q,d_i)$が次式で表現できる．

$$p(t,a_i,q,d_i)=\begin{cases} 0 & (q=0)\\ p(a_i,d_i) & (q=1)\\ \displaystyle\sum_{\tau=1}^{t-1}p(a_i,d_i)\sum_{k=1}^{q-1}\sum_{d_{\beta_1},d_{\beta_2},\cdots,d_{\beta_k}\in D}\ \sum_{\alpha_1,\alpha_2,\cdots,\alpha_k}\frac{(t-\tau-1)!}{\alpha_1!\alpha_2!\cdots\alpha_k!} & \\ \quad\times\{p(a_i,d_{\beta_1})\}^{\alpha_1}\{p(a_i,d_{\beta_2})\}^{\alpha_2}\cdots\{p(a_i,d_{\beta_k})\}^{\alpha_k} & (q>1)\end{cases} \tag{3.18}$$

これを用いて，P_lが以下のように求められる．

1. $l=0$の場合

エージェントa_iがd_iを検索しようとしたとき，すでにd_iを持っているのは，d_iをもともと所有しているか，他のエージェントから得たd_iをキューに保持しているかのどちらかである．

a_iがもともとd_iを所有していず，かつ，時刻tにおいてd_iをキューに保持していない確率$r(t,a_i,q,d_i)$は次式のようになる．

$$r(t,a_i,q,d_i)=\left(1-\frac{1}{w^h}\right)(1-p(t,a_i,q,d_i)) \tag{3.19}$$

$r(t,a_i,q,d_i)$を用いて，P_0は次のように表される．

$$P_0=1-r(t,a_i,q,d_i) \tag{3.20}$$

2. $l = k(0 < k < h)$ の場合

階層 $k-1$ までのサブネットで d_i が見つからず，階層 k に属するいずれかのエージェントが d_i を持つ．したがって，P_k は次のようになる．

$$P_k = \{r(t, a_i, q, d_i)\}^{w^{k-1}} \left(1 - \{r(t, a_i, q, d_i)\}^{w^k - w^{k-1}}\right) \tag{3.21}$$

3. $l = h$ の場合

最上位の h 未満の階層で d_i が見つかっていない場合，最上位の階層 h には必ずデータが存在するので，P_h は次のようになる．

$$P_h = \{r(t, a_i, q, d_i)\}^{w^{h-1}} \tag{3.22}$$

計算値

t を 1 から $+\infty$ に向けて $+1$ ずつ増加させ，t の各ステップごとに $0 < \tau < t$ なる τ について式 (3.18) を計算すれば，各 t に対して得られた P_l を用いて式 (3.5) の E_d が計算できる．ここでは，データの分布が均質な場合について計算する．本節での仮定によりデータ数 n が与えられているので，式 (3.18) の $p(a_i, d_i)$ は式 (3.23) で与えられ，式 (3.20)～(3.22) から求めた P_l を式 (3.6) の代わりに用いることができる．

$$p(a_i, d_i) = \frac{1}{n} \quad (1 \le i \le w^h, 1 \le j \le n) \tag{3.23}$$

$w = h = 3, n = 108$ とし，q を 0，2，4，6 と変化させたときの式 (3.5) の値を図 3.5 に示す．

時間が経過するに従い，$p(t, a_i, q, d_i)$ は q/n に収束するので，ステップも一定値に向かって収束していく．また，キューが長いほど，ステップは小さくなっている．ステップを小さくするためにはキューを長くすれば良いが，反面，キューが長いということはそれだけ物理的資源を必要とするので，コスト $V(q)$ として次式のような定義をしたとき，

$$V(q) = E_d \times q \tag{3.24}$$

$V(q-1)/V(q)(= V'(q)$ とおく$) > 1$ であれば，q を 1 単位長増加させることによるコストパフォーマンスの改善があると考えることができる．$p(t, a_i, q, d_i)$ を q/n で近似した，$w = h = 3, n = 108$ のシステムでのコストパフォーマンスの推移を図 3.6 に示す．$V'(q)$ は，$q = 10$ 付近から $q = 90$ 付近においてはあまり変化しない．したがって，コストパフォーマンスの見地からは，q の値としては 10 程度が妥当であることが示唆されている．

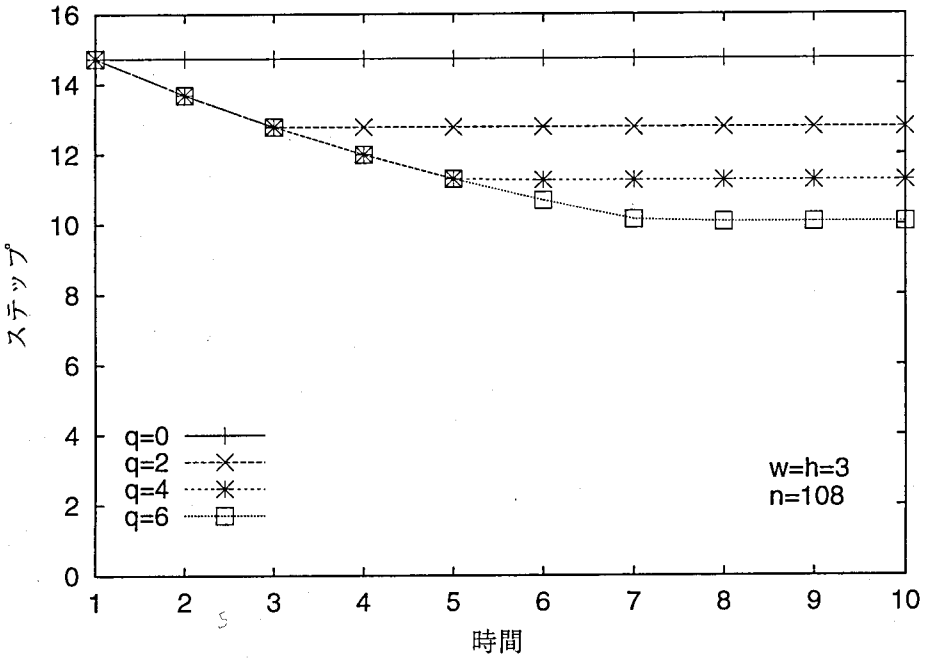


図 3.5: エージェントがキューを持つ場合のステップ数の時間推移

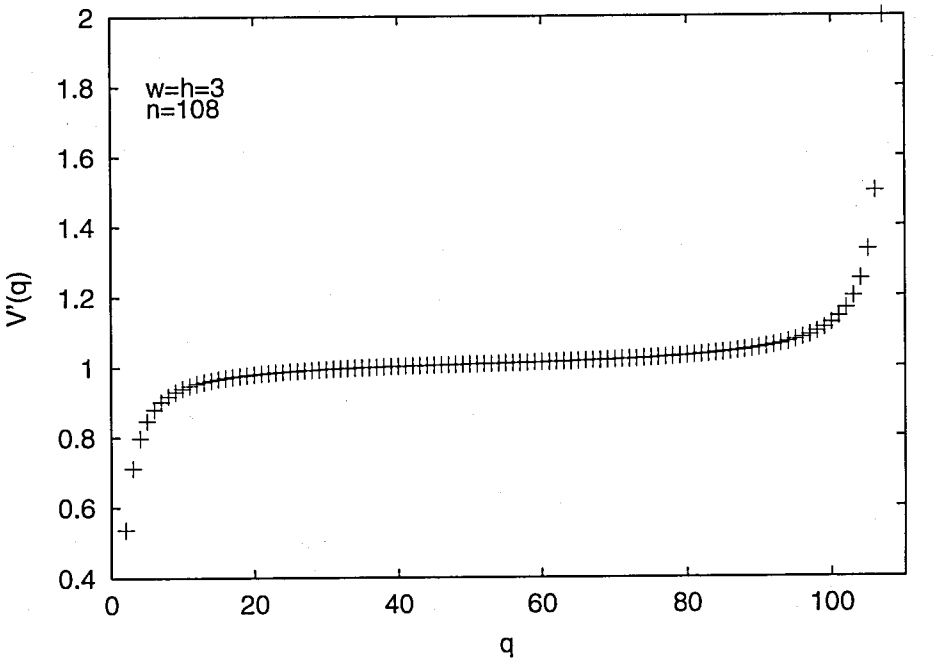


図 3.6: q と $V'(q)$ の関係

3.5　実験的性能評価

3.5.1　実験方法

エージェントがデータを探し始めてから得るまでに要するステップ数はネットワーク全体でのデータアクセスパターンに依存する．統計的評価においては，

- 各エージェント a_i は所望のデータ d_i の検索を一斉に行う．d_i を保持しているエージェント $a_j(i \neq j)$ は，d_i 検索メッセージの応答メッセージとしてデータ d_i そのものを a_i に対して送信する．d_i を保持しないエージェントは d_i 検索メッセージを無視する．また，すべてのエージェントがデータを取得するまで，次のデータ検索は行われない．また，ネットワーク上に存在しないデータの検索は行われない．

という仮定を置いたが，この仮定を除き，WWWキャッシュシステムに残されたアクセス記録から抽出された，現実のデータアクセスパターンを用い，シミュレーション実験を行った．

WWWキャッシュシステムは，WWWサーバとWWWクライアント間の通信を中継すると同時にデータのキャッシングを行う．次回以降同一データのアクセスがあった際にはそのキャッシュデータを用いることにより，ネットワークのトラフィックを軽減することを目的としている [62]．

集中管理方式はスーパバイザエージェントのみがキャッシングを行うシステムに，分散管理方式はすべてのエージェントがキャッシングを行うシステムに相当する．

以下のような手順で，2方式のシミュレーションを行う．

1. アクセス記録から必要な数のWWWクライアントを選んでエージェントとする．例えば，$w = 3, h = 2$ ならば9個のWWWクライアントが選ばれる．その他のWWWクライアントに関する記録は無視する．

2. 図3.7に示すように，選択したWWWクライアントを階層型ネットワーク内の各エージェントに割り当てる．

3. アクセス記録に従ってデータ検索をシミュレートし，ステップ数を評価する．ただし，対象としている階層型ネットワークの外へのデータアクセスに要する時間は無視する．これは，WWWキャッシュシステムがWWWサーバにデータを要求して得るまでの時間に相当する．

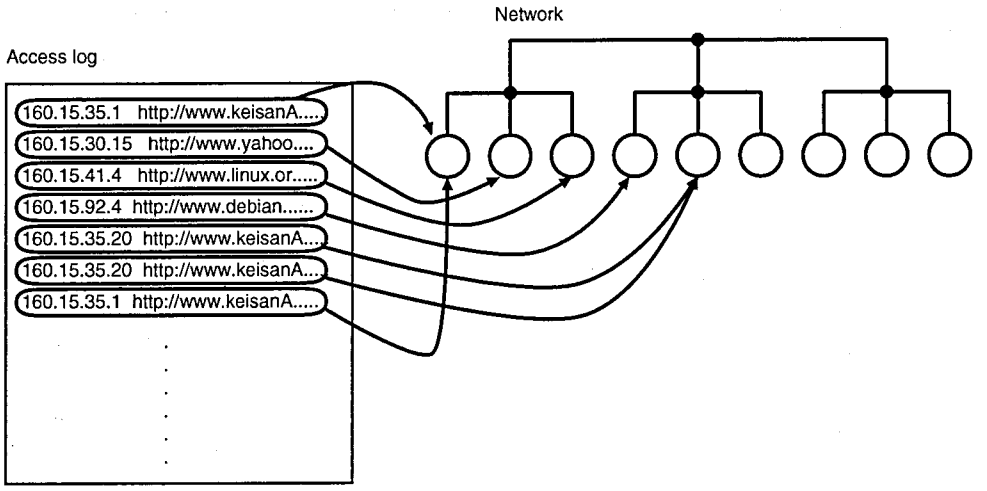


図 3.7: WWW クライアントのエージェントへの割り当て

3.5.2 実験結果

鳥取大学工学部知能情報工学科の WWW キャッシュシステムの 1997 年 9 月から 1999 年 10 月までのアクセス記録を用いて実験を行った.

3.5.3 キューの長さとステップ数の関係

w^h 個の WWW クライアントをランダムに選んでエージェントとし，エージェントのキューの長さ q を変化させながらシミュレーションを行った．$w = h = 3$ のときの結果を図 3.8 に示す．凡例で concentrated と記されたものが集中管理方式の結果で，distributed-A から distributed-E までは分散管理方式の結果である．選択された WWW クライアントによって多少結果は異なるが，いずれの場合でも，q が小さいときは集中管理方式の方がステップ数が小さく，q が大きくなるに従って分散管理方式のステップ数が減少し，$q = 100$ 前後で集中管理方式よりも小さくなっている．

3.5.4 データアクセス頻度とステップ数の関係

次に，データアクセスの頻度とステップ数の関係を調べた．図 3.9 は一週間のデータアクセス回数が 15783 回の週における平均ステップ数で，図 3.10 は一週間のデータアクセス回数が 102387 回の週における平均ステップ数を q を変化させながら調べた結果である．凡例で concentrated と記されたものが集中管理方式，distributed と記されたものが分散管理方式の結果である．データアクセス頻度が小さいときは，

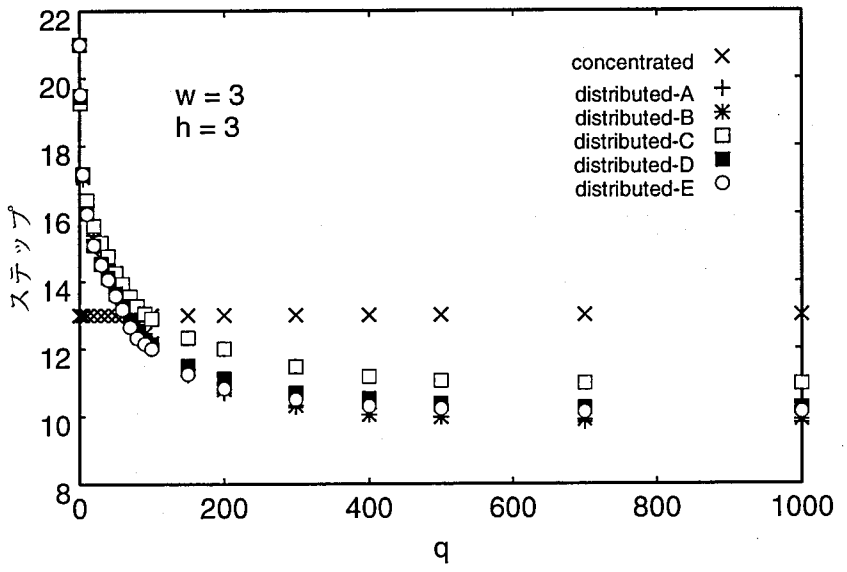


図 3.8: q とステップ数の関係

集中管理方式も分散管理方式も同様の結果を示しているが，データアクセス頻度が大きい場合は，q が大きくなれば分散管理方式のステップ数が集中管理方式のそれを下回る．つまり，データアクセス頻度が大きい場合はキューが有効利用されていることが分かる．

データアクセス頻度とステップ数の関係を詳しく調べるために，エージェントが任意の時間にデータを検索しようとする確率 (以後，検索確率と呼ぶ) というパラメータを導入する．

$w = 3$ で $h = 2$ のときの結果を図 3.11 に示す．凡例で concentrated と記されたものが集中管理方式，distributed と記されたものが分散管理方式の結果である．この場合，$q = 0$ においては集中管理方式の方が分散管理方式に比べて常にステップ数が小さいが，検索確率が 15%を越えた場合，$q = 100$ や $q = 500$ においては分散管理方式の方が集中管理方式よりステップ数が小さくなっている．

3.5.5 ネットワーク構造とステップ数の関係

ネットワークの構造とステップ数の関係を調べるために，64 個の WWW クライアントを選び，w と h を変化させて実験を行った．図 3.12 に示すように，w が大きいほどステップ数が小さくなった．第 3.4.1 節で述べたように，w が大きくなることは一度にマルチキャストできるエージェント数が増えることを意味し，h が大きくなることはシステム全体に並行して送られるメッセージ数が増えることを意味しており，w と h の増減はトレードオフの関係にある．この規模のネットワークにおい

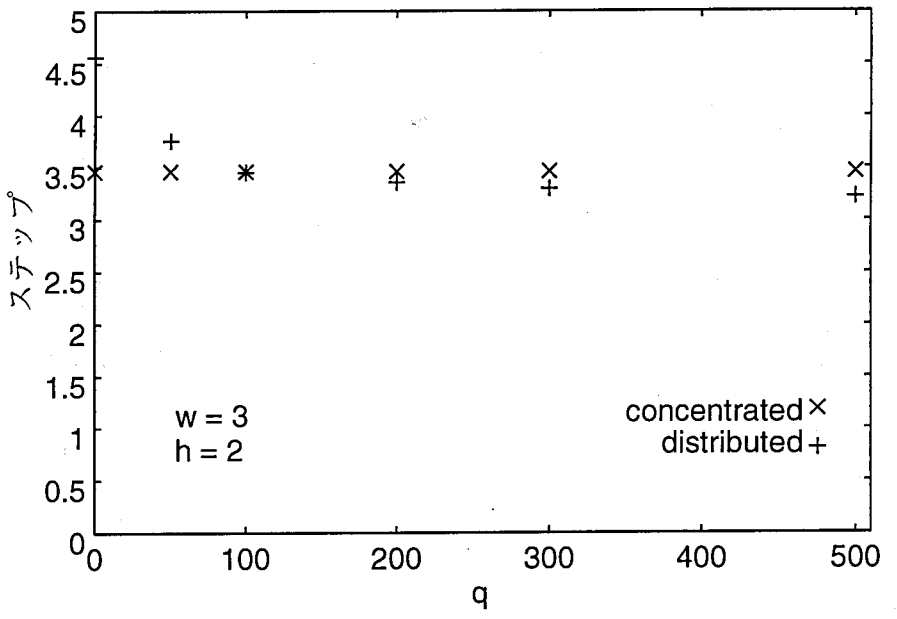


図 3.9: データアクセス頻度が低い場合 (1 週間に 15783 アクセス) の q とステップ数の関係

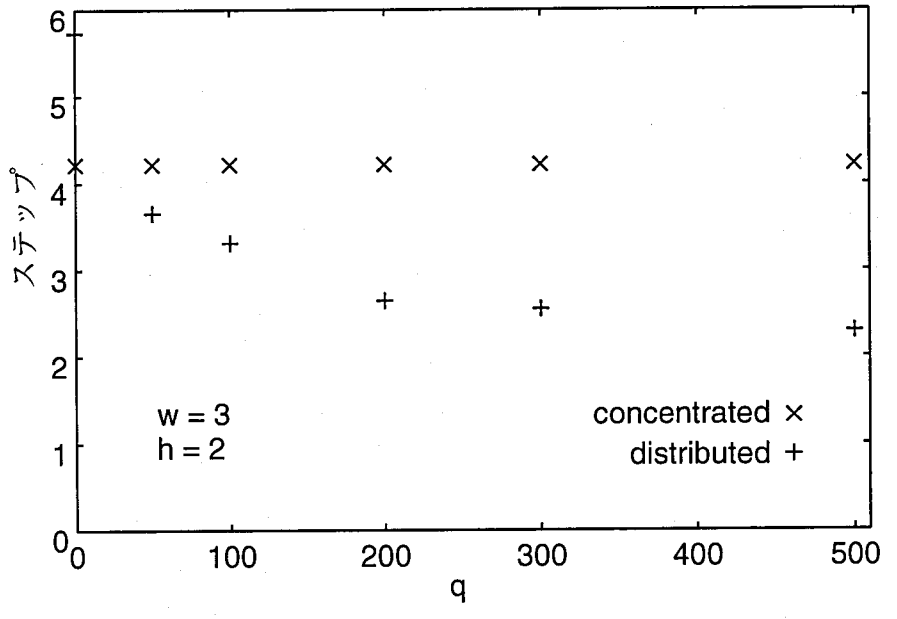


図 3.10: データアクセス頻度が高い場合 (1 週間に 102387 アクセス) の q とステップ数の関係

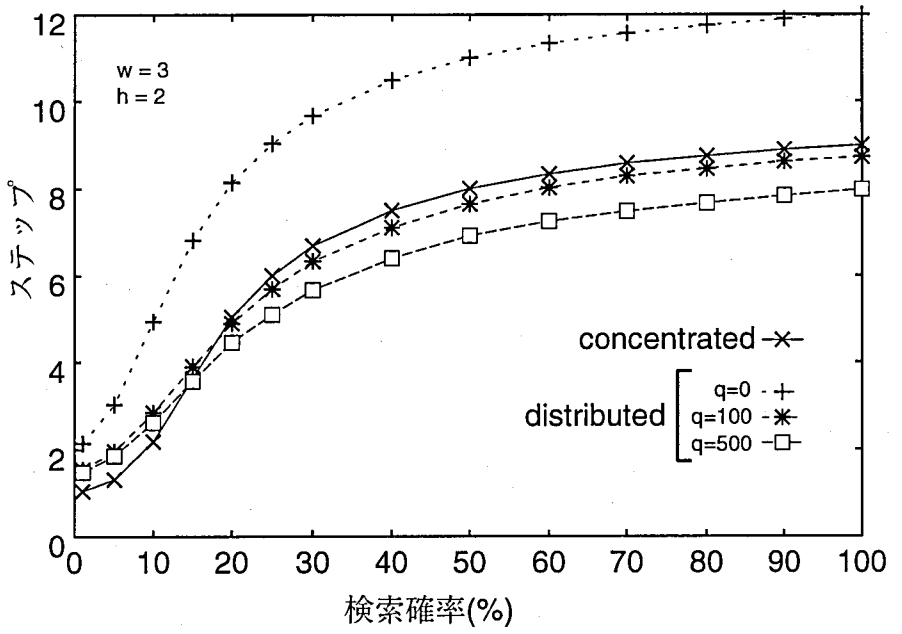


図 3.11: 検索確率とステップ数の関係

ては，w の増加による利得が h の増加による利得を上回るものと考えられる．

3.6 結言

階層型ネットワーク上で，分散管理方式と集中管理方式の2つのデータ管理方式におけるデータアクセスの効率特性を明らかにした．各エージェントが必要とするデータをどのエージェントが所有するかの確率が均等である場合においても，分散管理方式は集中管理方式と比べて遜色ない効率を持つが，各エージェントが必要とするデータを，そのエージェントに近いエージェントほど高い確率で所有する場合には，分散管理方式は集中管理方式より高い効率を持つことが分かった．また，分散管理方式において，キューを持つ各エージェントが，すべてのデータを等確率で検索する場合におけるデータアクセス効率の時刻変化を解析し，その特性を明確にした．例えば，エージェント数27，データ数108のシステムにおいて，長さ10のキューが高いコストパフォーマンスを持つように，エージェント数，データ数とキューの大きさの最適関係が明示された．さらに，実働しているWWWキャッシュシステムから抽出されたWWWアクセス記録を基に，データ検索効率のシミュレーション実験を行った．シミュレーション結果は統計的評価により得られた結果と同じ傾向を示しており，第3.5節の最初で述べた，統計的評価を得る際に課した制約が，結果の有効性を左右しないことを示している．また，データの検索が頻繁に行われる

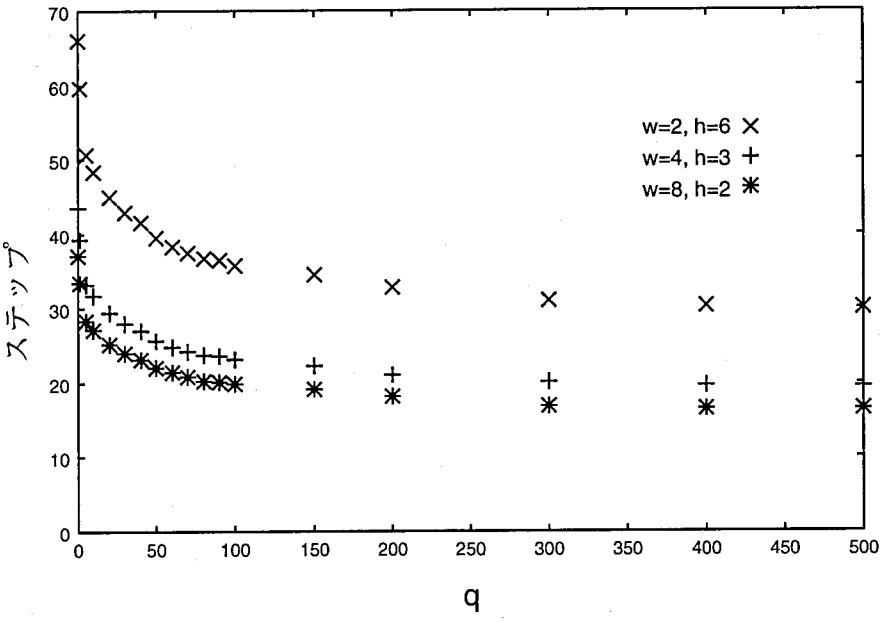


図 3.12: ネットワーク構造とステップ数の関係

ほど分散管理方式が優位に立つことが確かめられた.

本章の階層型ネットワークは総エージェント数が w^h で与えられる. このようなネットワークは階層型ネットワークとしての一般性を失ってはいないが, 総エージェント数が比較的小さい場合は, w や h の取り得る値が限られるので, 分散管理方式での最適構造の議論はできない. そのような議論を可能にするためには総エージェント数を大きくする必要があるが, その場合は, 統計的評価における計算や, 実験的評価におけるシミュレーションに要する時間が膨大なものになる. そのため, 現時点では, エージェント総数が 1000 を越えるような規模についての解析結果は得られていない. 今後, 解析方法を検討し, 分散管理方式での最適構造の議論をすることを目指したい. それによって, 例えば WWW キャッシュシステム [63] の効率的な分散配置方法といった問題の解決に指針が与えられることが期待できる.

第4章　並列コンピュータネットワークにおける耐故障性通信

4.1　緒言

マルチエージェントシステムでは，システム内の通信が重要な役目を持ち，次のように様々な用途で用いられる．

- エージェント間の直接通信

  協調や競合解消のためにエージェント同士が直接対話する．1対1通信の他にも，契約ネットプロトコル [64] のように，ブロードキャストやマルチキャストが用いられる場合もある．

- エージェントと掲示板オブジェクト間の通信

  掲示板オブジェクトとは，エージェント間の間接通信を実現するためのオブジェクトで，黒板アーキテクチャ [65] [66] における黒板や並列プログラミングモデル Linda [67] [68] におけるタプルスペースがその例である．

- エージェントの活動の場であるエージェントサーバ間の通信

  エージェントサーバ間で，モバイルエージェントをメッセージとして転送するといった用途に用いられる．

- エージェントとエージェントサーバ間の通信

  エージェントが，エージェントサーバが管理する各種リソースにアクセスするといった用途に用いられる．

マルチエージェントシステムにおける通信に関しては様々な観点からの研究が行われているが，本研究では，耐故障性通信に焦点を絞る．すなわち，システムの構成要素の一部が故障している場合でも，故障していない部分における通信を可能にすることを考える．ここで，本研究では，問題をマルチエージェントシステムにおける通信から一般化して，ネットワークシステムにおける通信として考察し，その成果をマルチエージェントシステムに還元することを目指す．

これまでに，耐故障性通信がよく研究されているネットワークとして，並列計算機のプロセッサ同士を結合するインターコネクションネットワークがある．メッシュ結合[69]，キューブ結合[70]，リング結合[71][72]，非同期通信モードスイッチング[73]など，様々なトポロジを持つインターコネクションネットワークに関する報告が多数なされている．本章ではネットワークトポロジとして，ハイパーキューブとコーダルリングを対象とし，それぞれの上で耐故障性を持つ通信アルゴリズムを提案する[74]～[76]．

4.2 ハイパーキューブ上の耐故障性通信

本節ではハイパーキューブ上の全対全個別通信アルゴリズムを論じる．ハイパーキューブネットワークは，非常に多数のノードからなる超並列コンピュータにおいて，費用効果が高く現実的なアプローチである．ハイパーキューブはこれまでにSIMDマシンとして数多く研究がなされており[77]～[81]，多くの実験および商用マシンが作られてきた[82]．最近の開発事例として，6000以上のノードを接続したnCUBEシステム[83]が挙げられる．

ノード間の通信時間は計算時間と比べて非常に高価なものとなる可能性があるため，システムを高性能にするために，効率のよい通信方式を確立することが重要となる．JohnssonとHoは文献[82]において以下の4種類の通信プリミティブを提案している．

1. 一対全ブロードキャスト．1つのノードだけが他のすべてのノードに対して同一のデータを送信する．

2. 一対全個別通信．1つのノードだけが他のすべてのノードに対してそれぞれ異なるデータを送信する．

3. 全対全ブロードキャスト．すべてのノードが同時に他のすべてのノードに対して同一のデータを送信する．

4. 全対全個別通信．すべてのノードが同時に他のすべてのノードに対してそれぞれ異なるデータを送信する．

これまでに，ハイパーキューブのためのアルゴリズムが多数提案されているが[84]～[86]，それらはルーティングや1対全通信，あるいは全対全ブロードキャストについてのものである．全対全個別通信もまた，多くのアプリケーションが存在する重要な通信方法である．JohnssonとHoは無故障n次元ハイパーキューブ上の全対全個別通信が$O(2^n)$で実行できるアルゴリズムを提案したが[82]，これまでに故障

を含むハイパーキューブ上の全対全個別通信アルゴリズムは提案されていない．本研究は，n 次元のハイパーキューブが $\lfloor n/2 \rfloor$ 以下の故障を含む場合の全対全個別通信アルゴリズムを与えるものである．

故障を含むネットワークにおける通信アルゴリズムでは，無故障のノードにとって故障位置が既知であるか否かが重要な問題となる．故障位置が不明の場合に実行できるアルゴリズムは実時間処理が可能な点が優れているが，各ノードが個別データのコピーを複数のパスで送るという枠組にならざるを得ないので，システム全体の効率は悪くなる．そこで，本研究では故障位置はすべての無故障ノードにとって既知であると仮定する．ノードまたはリンクの故障は滅多に起こらないものと考えられるので，この仮定は非現実的ではない．

通信アルゴリズムは1ポートモデルにおいても多ポートモデルにおいても実現できるが，本研究では多ポートモデルだけを考える．多ポートモデルは効率を犠牲にせずに1ポートモデルでシミュレートできるからである．すなわち，ノードの全リンクは同時にパケット送受信ができるものとする．また，隣接ノードへの通信は1単位時間を要するものとする．

本節の残りは以下のように構成されている．第4.2.1節は後の議論のための準備を行う．第4.2.2節において故障ノードを含むハイパーキューブのための全対全個別通信アルゴリズムを提案する．第4.2.3節では，アルゴリズムをリンク故障に対して拡張する．

4.2.1 準備

n 次元ハイパーキューブ Q_n が $N = 2^n$ 個のノードを持つものとする．各ノードには0から $N-1$ までの番号を付ける．ノード番号 $i(0 \leq i \leq N-1)$ は二進法で $i_{n-1}i_{n-2}\cdots i_1 i_0$ と表せる．$0 \leq j \leq n-1$ のとき，$i_{n-1}\cdots i_{j+1}\bar{i_j}i_{j-1}\cdots i_0$ を $i^{(j)}$ と表すことにする．ハイパーキューブではノード i はすべての j についてノード $i^{(j)}$ に接続されている (隣接しているという)．図4.1は $N = 16$ のハイパーキューブである．本論文では，Q_n は第0次元から第 $n-1$ 次元までの n 次元を有するものとする．

Q_n が k 個の次元 $d_1, d_2, \cdots, d_k$ で分割されたとき，2^k 個の Q_{n-k} ができる．ノード番号の二進法表現 $i_{n-1}i_{n-2}\cdots i_1 i_0$ において，すべての $i_j \in \{i_{n-1}, i_{n-2}, \cdots, i_1, i_0\} - \{i_{d_1}, i_{d_2}, \cdots, i_{d_k}\}$ が同じ値を持つノードの集合をパートナーセット (以後，PSと表記する) と呼ぶと，2^k 個のノードからなる 2^{n-k} 個のPSが存在し，k 次元ハイパーキューブ Q_k を形成する．

2個の l 次元ハイパーキューブ Q_l において，それらが分割されたと考えられる次元について隣接しているノードの対を対応ノードと呼ぶ．ノード i がノード j と d 次元において対応しているとき，それらのノードは第 d ビットが異なる値を持つ．

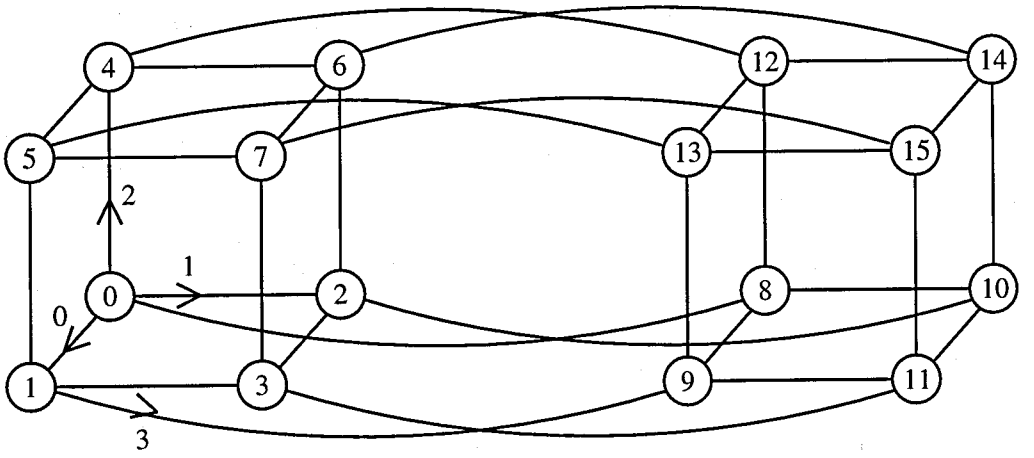


図 4.1: 4次元ハイパーキューブ Q_4

例 1. Q_4 が第0次元と第2次元で分割されたとき，2^2 個の PS，{0,1,4,5}, {2,3,6,7}, {8,9,12,13}, {10,11,14,15} が存在する．

0個以上の故障ノードを含むサブキューブを故障サブキューブと呼ぶ．

ハイパーキューブは以下の3つの性質を持つ [87]．

性質1は，各PSが高々1個の故障ノードしか含まないようにするための，Q_n の分割次元数を与える．

性質 1. $f \leq \lfloor n/2 \rfloor$ 個の故障ノードを含む Q_n は各PSが高々1個の故障ノードしか含まない $(n-f+1)$ 次元サブキューブに分割できる．

性質2と3は，高々1個の故障ノードを含む故障PSがどのように無故障PSと組み合わされるかを示している．

性質 2. Q_n の故障ノード数を $f \leq n$ とすると，すべての故障ノードがそれぞれ異なる無故障ノードと対応するように Q_n を分割できる次元 d が存在する．

性質 3. $f \leq \lfloor n/2 \rfloor$ 個の故障ノードを含む Q_n を任意の f 個の次元で分割した場合，すべての故障PSはそれぞれ異なる無故障PSと隣接する．

上記の性質から，Q_n を $2^{n-(f+1)}$ 個の Q_{f+1} に分割でき，そのうちの $2^{(f+1)}$ 個が故障サブキューブで，残りが無故障サブキューブであることが保証される．

4.2.2 ノード故障に対する全対全個別通信アルゴリズム

以下では，まず故障ノードが1個の場合と2個の場合について全対全個別通信アルゴリズムを開発し，それらを基に故障ノードが f 個の場合に拡張する．

故障ノードが1個の場合のアルゴリズム A_1

Q_{n-1}: Q_n をある次元で分割した $(n-1)$ 次元サブキューブで，故障ノードを含まない．

Q^f_{n-1}: Q_n をある次元で分割したもう片方の $(n-1)$ 次元サブキューブで，故障ノードを含む．

ステップ 1: 送信ノードも受信ノードも Q_{n-1} 内のデータについて全対全個別通信(以後，AAPCと表記する)を実行する．
同時に，送信ノードが Q^f_{n-1} 内で受信ノードが Q_{n-1} 内のデータを，各送信ノードから Q_{n-1} 内の隣接ノードに送る．

ステップ 2: ステップ1で Q^f_{n-1} から送られたデータについて，正しい受信ノードにデータが送られるように Q_{n-1} においてAAPCを実行する．

ステップ 3: 送信ノードが Q_{n-1} 内で受信ノードが Q^f_{n-1} 内のデータについて，各受信ノードの Q_{n-1} 内の隣接ノードにデータが送られるように Q_{n-1} においてAAPCを実行する．

ステップ 4: ステップ3で配られたデータについて，Q_{n-1} 内の各ノードから Q^f_{n-1} 内の隣接ノードにデータを送る．
同時に，送信ノードも受信ノードも Q^f_{n-1} 内のデータを，各送信ノードから Q_{n-1} 内の隣接ノードに送る．

ステップ 5: ステップ4で Q^f_{n-1} から送られたデータについて，各受信ノードの Q_{n-1} 内の隣接ノードにデータが送られるように Q_{n-1} においてAAPCを実行する．
同時に，Q_{n-1} 内の各ノードにそれらのデータが到着した順に Q^f_{n-1} 内の隣接ノードにデータを送る．

例 2. 図4.2に3次元ハイパーキューブに A_1 を適用した例を示す．Q_3 は第2次元で分割されるものとする．図中の1ステップ以降では，各ステップにおいて各ノードに移動または到着したデータのみを記している．ここで，i_j は受信ノードが i で受信ノードが j のデータを表す．

A_1 の所要時間

無故障 Q_n における最適AAPCアルゴリズムは 2^n-1 単位時間かかる．A_1 の所要時間は各ステップの所要時間の総和である．各ステップの所要時間は次のようになる．

ステップ 1: 送信ノードも受信ノードも Q_{n-1} 内のデータについての AAPC の所要時間は $2^{n-1}-1$. 送信ノードが Q^f_{n-1} 内で受信ノードが Q_{n-1} 内のデータは 2^{n-1} 個あるので，これらを Q_{n-1} 内に送るには 2^{n-1} 単位時間かかる．したがって，ステップ 1 の所要時間は $\max(2^{n-1}-1, 2^{n-1}) = 2^{n-1}$.

ステップ 2: Q_{n-1} において AAPC を実行するので所要時間は $2^{n-1}-1$.

ステップ 3: Q_{n-1} において AAPC を実行するので所要時間は $2^{n-1}-1$.

ステップ 4: Q_{n-1} から Q^f_{n-1} へ，Q^f_{n-1} から Q_{n-1} へ送るべきデータは 2^{n-1} 個あるので，所要時間は 2^{n-1}.

ステップ 5: Q_{n-1} において AAPC を実行する所要時間は $2^{n-1}-1$ で，それが完了した後，Q_{n-1} から Q^f_{n-1} へ送るべきデータが各ノードについて高々1 個残っているので，所要時間は 2^{n-1}.

以上より A_1 の所要時間は $5\cdot 2^{n-1}-2$ となる．

故障ノードが 2 個の場合のアルゴリズム A_2

性質 3 により，$f=2$ 個の故障 PS がそれぞれ無故障 PS と隣接するように，Q_n を f 個の次元で分割することができる．故障 PS と無故障 PS を組み合わせるように $(f+1)$ 次元のサブキューブを構成する．それらを Q^1_{f+1} と Q^2_{f+1} とする．Q^1_{f+1} と Q^2_{f+1} はそれぞれが故障ノードを 1 個しか含まない．

Q^1_{f+1} と Q^2_{f+1} をそれぞれ含む 2 個の Q_{n-1} を，Q^1_{n-1} と Q^2_{n-1} とする．

A_2 は次の 2 つのフェーズからなる．

[フェーズ 1] アルゴリズム A_1 により Q^1_{n-1} と Q^2_{n-1} で AAPC を実行する．

[フェーズ 2] Q_n を Q^1_{n-1} と Q^2_{n-1} に分割する次元を d とする．Q^2_{n-1} 内の故障ノードと第 d 次元で隣接する Q^1_{n-1} 内のノードを n_1，Q^1_{n-1} 内の故障ノードと第 d 次元で隣接する Q^2_{n-1} 内のノードを n_2 とする．

ステップ 1: n_1 と n_2 が送信ノードであるデータを Q_n 内のすべての無故障ノードに送る．

ステップ 2: 送信ノードが Q^1_{n-1} 内で受信ノードが Q^2_{n-1} 内のデータと，送信ノードが Q^2_{n-1} 内で受信ノードが Q^1_{n-1} 内のデータを第 d 次元のリンクによって交換する．ただし，第 1 ステップで送ったデータは除く．
同時に，データが到着した順にアルゴリズム A_1 により Q^1_{n-1} と Q^2_{n-1} で AAPC を実行する．

例 3. 図4.3に4次元ハイパーキューブに A_2 を適用した例を示す．Q_4 は第0次元と第2次元で分割されるものとする．フェーズ2において，$d=3$，$n_1=1$，$n_2=14$ である．

A_2 の所要時間

A_2 の所要時間は以下の所要時間の総和である．

[フェーズ 1] Q^1_{n-1} または Q^2_{n-1} における A_1 の所要時間 $5\cdot 2^{n-2}-2$.

[フェーズ 2]

ステップ 1: n

ステップ 2: 最初のデータ交換に要する時間1に Q^1_{n-1} または Q^2_{n-1} における A_1 の所要時間を加えればよいので，所要時間は $1+5\cdot 2^{n-2}-2=5\cdot 2^{n-2}-1$.

以上より A_2 の所要時間は $5\cdot 2^{n-1}+n-3$ となる．

故障ノードが f 個の場合のアルゴリズム A_f

A_2 を故障ノード数が f の場合に一般化する．

性質3により，f 個の故障PSがそれぞれ無故障PSと隣接するように，Q_n を f 個の次元で分割することができる．故障PSと無故障PSを組み合わせるように $(f+1)$ 次元のサブキューブを構成する．それらを Q^j_{f+1} $(j=1,2,\cdots,2^{n-(f+1)})$ とする．このうちの f 個のサブキューブはそれぞれが故障ノードを1個しか含まない．

A_f は次の2つのフェーズからなる．

[フェーズ 1] アルゴリズム A_1 により f 個の故障 Q^j_{f+1} で AAPC を実行する．同時に，無故障 Q^j_{f+1} においても AAPC を実行する．

[フェーズ 2] まず各 Q^j_{f+1} 間でデータ交換を行い，次に各 Q^j_{f+1} 内で AAPC を実行する．

ステップ 1: 故障ノードに対応するノード (故障ノード数が2個の場合の n_1 と n_2 に相当する) が送信ノードであるデータを Q_n 内のすべての無故障ノードに送る．

ステップ 2: 各 Q^j_{f+1} 間でデータ交換を行う．ただし，第1ステップで送ったデータは除く．
同時に，データが到着した順に Q^j_{f+1} 内で AAPC を実行する．

A_f の所要時間

A_f の所要時間は以下の所要時間の総和である．

[フェーズ 1] 故障 Q^j_{f+1} における A_1 の所要時間 $5\cdot 2^f-2$ で，無故障 Q^j_{f+1} における AAPC の所要時間は $2^{f+1}-1$ であるから，所要時間は $5\cdot 2^f-2$.

[フェーズ 2]

ステップ 1: 故障ノードの配置パターンにより多少異なるが，$O(f)$ なので，他のフェーズやステップの所要時間に比べると無視できる．

ステップ 2: 各 Q^j_{f+1} について，他の Q^j_{f+1} にデータを送るのに要する時間は $5\cdot 2^f-2$ なので，所要時間は $(5\cdot 2^f-2)\cdot(2^{n-(f+1)}-1)$.

以上より A_f の所要時間は，ほぼ $2^{n-1}(5-2^{1-f})$ となる．2^{1-f} も，ある程度の f については，5と比べて無視できるほど小さくなるので，A_f の所要時間は故障数 f によらず $5\cdot 2^{n-1}$ とみなすことができる．

4.2.3　リンク故障に対する全対全個別通信アルゴリズム

本節では，リンク故障に対する全対全個別通信アルゴリズムを開発し，所要時間を推定する．

故障リンクが m 個の場合のアルゴリズム B_m

Q_n に $m=\sum_{i=0}^{n-1} m_i \le \lfloor n/2\rfloor$ 個のリンクが故障しているとする．ここで，m_i は第 i 次元にある故障リンク数である．同一ノードに属する故障リンク数の最大値を μ とすると，すべてのリンクが無故障である次元が少なくとも μ 個存在することが知られている [87]．これらの次元を無故障安全次元と呼ぶ．Q_n をこれら μ 個の次元で分割したとすると，2^μ 個の $Q_{n-\mu}$ が得られる．さらに，無故障である $2^{n-\mu}$ 個の PS, Q_μ が得られる．文献 [87] で与えられた次の性質が 2^μ 個の $Q_{n-\mu}$ について成り立つ．

性質 4. $Q_{n-\mu}$ 中のノードに属するすべての故障リンク f_i について，d_i において f_i と対応する無故障リンクが存在するような，固有の次元 d_i が存在する．ここで，対応するリンクとは，同一次元にあり，そのアドレス (すなわち，両端のノードのアドレス) が d_i のみ異なるリンク対のことである．

故障リンク数が m 個の場合のアルゴリズム B_m は次の2つのフェーズからなる．

[フェーズ 1] 各 $Q_{n-\mu}$ において，受信ノードが同じ $Q_{n-\mu}$ に属するデータについて AAPC を実行する．もしも AAPC 中の最短経路上のリンク f_i が故障していたら，他の $Q_{n-\mu}$ に含まれる f_i に対応するリンクによって迂回する．

[フェーズ 2] 各 $Q_{n-\mu}$ において，受信ノードが他の $(2^\mu-1)$ 個の $Q_{n-\mu}$ に属するデータについて，受信ノードが含まれる $Q_{n-\mu}$ に最も近いノードにデータが集まるように AAPC を実行する．そこから正しい受信ノードまでの転送は無故障安全次元のリンクだけを使って行える．

例 4. 図 4.4 に 4 次元ハイパーキューブに B_m を適用した例を示す．図中で太線で示されているリンクが故障している．フェーズ 1 の 2 つのステップは同時に実行できる．フェーズ 2 の (2-1), (2-2), (2-3) 中の各ステップは同時に実行できるが，(2-1), (2-2), (2-3) 自体は逐次的に実行しなければならない．

B_m の所要時間

まず，フェーズ 1 の所要時間を検討する．$Q_{n-\mu}$ が故障リンクを含まない場合，AAPC の実行時間は $2^{n-\mu}-1$ である．故障 $Q^f_{n-\mu}$ 中の高々 μ 個の故障リンクにデータを流す場合は，他の $Q_{n-\mu}$ 中の対応リンクを用いる．$Q^f_{n-\mu}$ 中の μ 個の故障リンクが連続している場合でも，これら μ 個の故障リンクが $Q_{n-\mu}$ または $Q^f_{n-\mu}$ における AAPC に及ぼす影響は，故障リンクを迂回しているデータが優先されるという条件の下で，2 と推定できる．各 $Q_{n-\mu}$ での AAPC は同時に実行できるので，フェーズ 1 の所要時間は $2^{n-\mu}-1+2=2^{n-\mu}+1$ である．

フェーズ 2 では，距離 $d(d=1,2,\cdots,\mu)$ が等しい異なる $Q_{n-\mu}$ に対してデータを送るようにすれば，2^μ 個の $Q_{n-\mu}$ において AAPC を同時に実行できる．送信側の $Q_{n-\mu}$ から受信側の $Q_{n-\mu}$ へのデータ転送も同時に行える．したがって，フェーズ 2 の所要時間は $\mu\cdot 2^{n-\mu}+1+\sum_{i=1}^{\mu} i=\mu\cdot 2^{n-\mu}+\mu^2/2+3\mu/2$ となる．

以上より，B_m の所要時間は $(\mu+1)2^{n-\mu}+\mu^2/2+3\mu/2+1$ である．

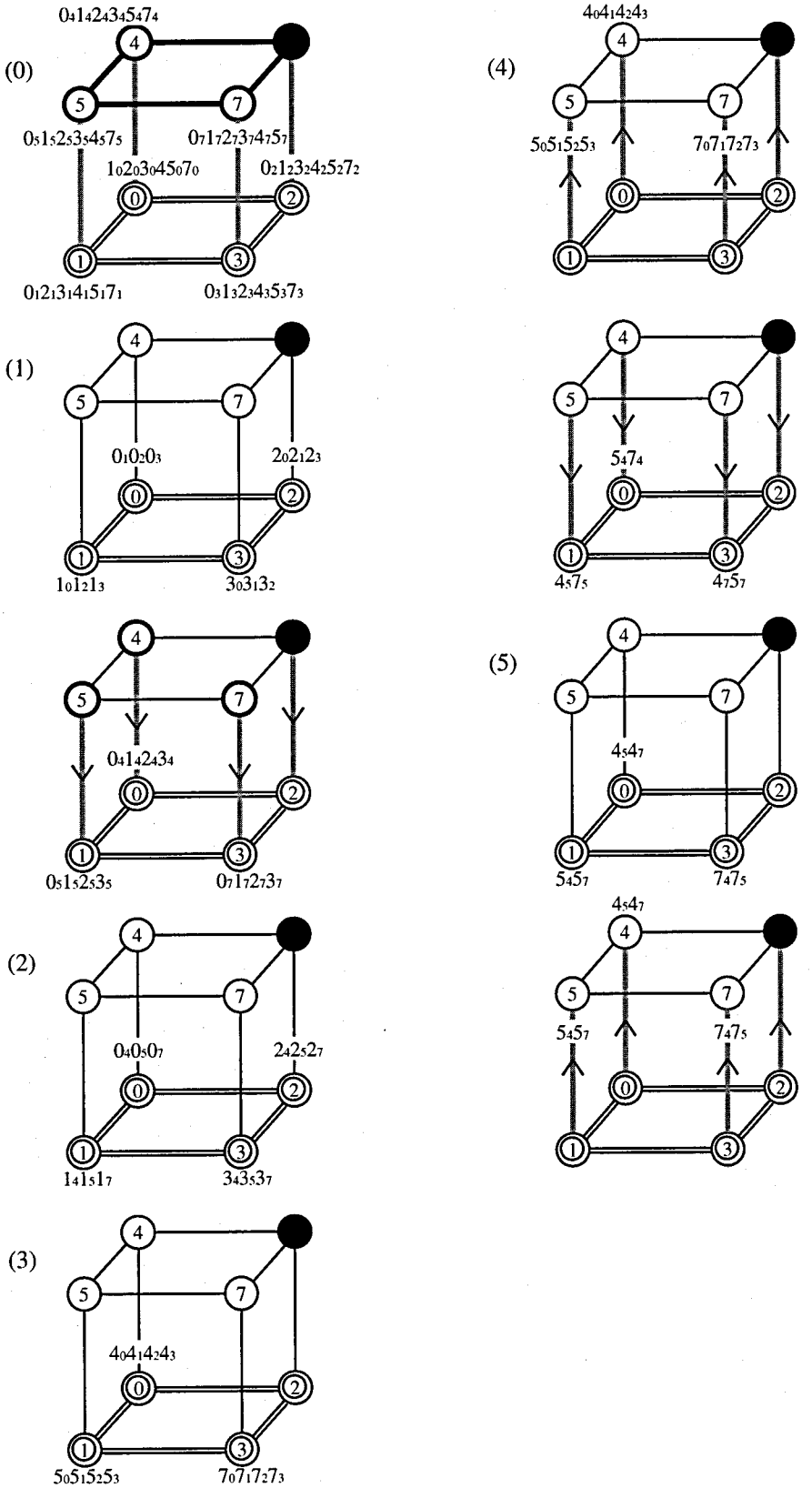
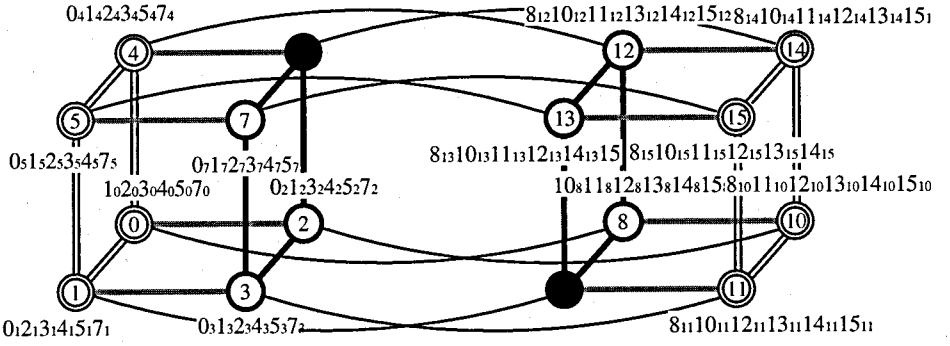
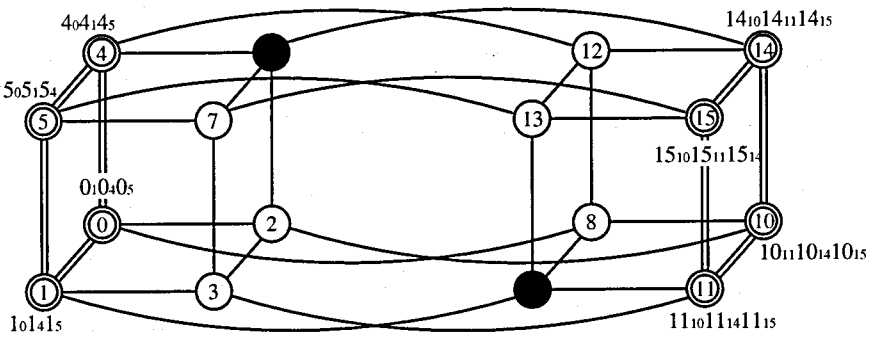
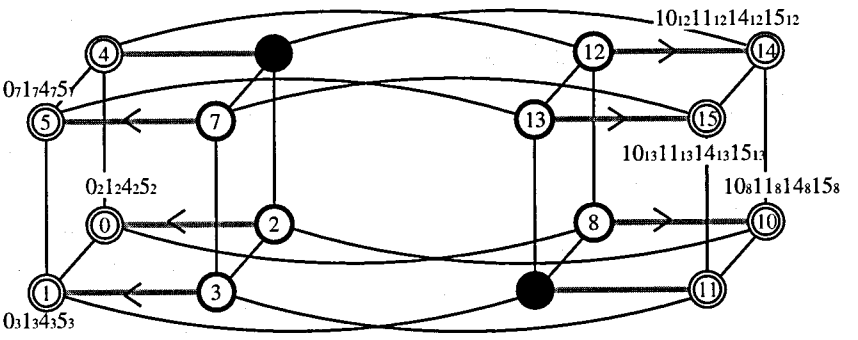


図 4.2: 1個の故障ノードを含む3次元ハイパーキューブへの適用例

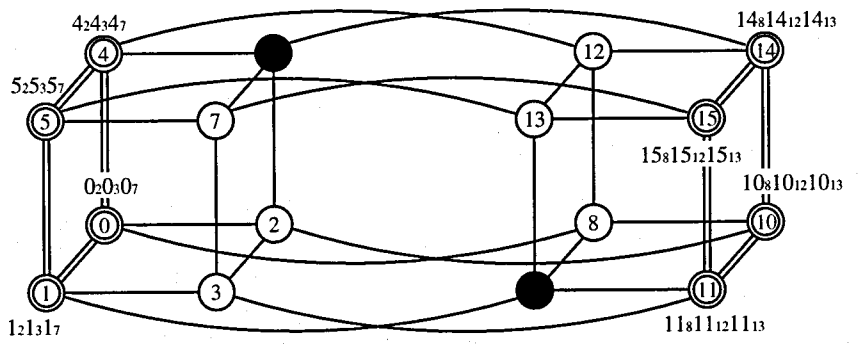

0414243454747 8121012111213121412151281410141114121413141511
0515253545757 0717273747571 8131013111312131413158151015111512151315141510
1020304050701 0212324252721 1081181281381481581081101110121013101410151010
0 2 8 10
1 3 11
0121314151771 0313234353732 8111011121113111411511

(1-1)

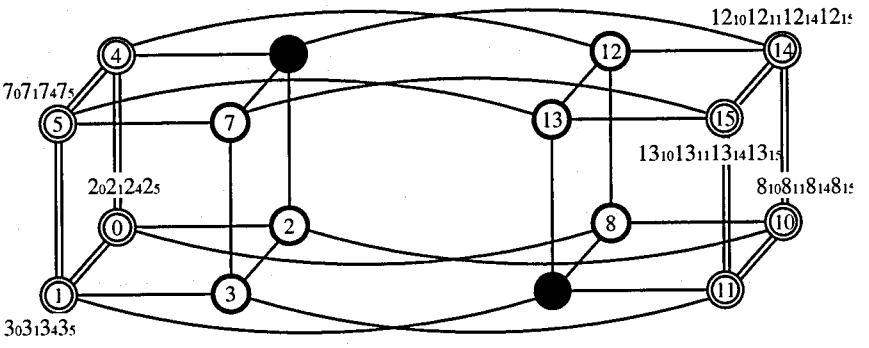

4041445 14101411141515
4 12 14
5051545 7 13 15
15101511151415
0104005
0 2 8 10
10111014101515
1 3 11
1011415 11101114111515


10121112141215121
4 12 14
0717475171 7 13 15
10131113141315131
0212425252 10811814815858
0 2 8 10
1 3 11
0313435353

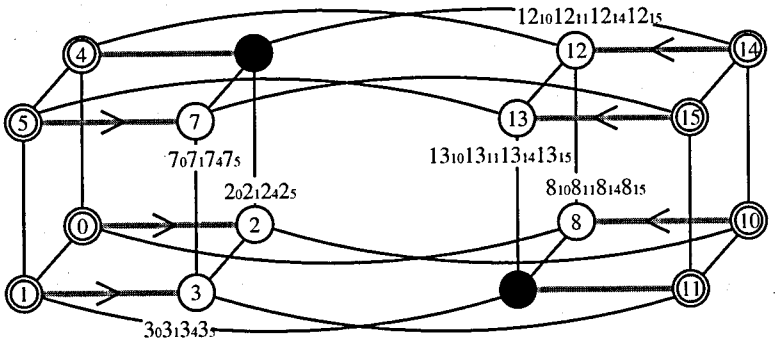
(1-2)

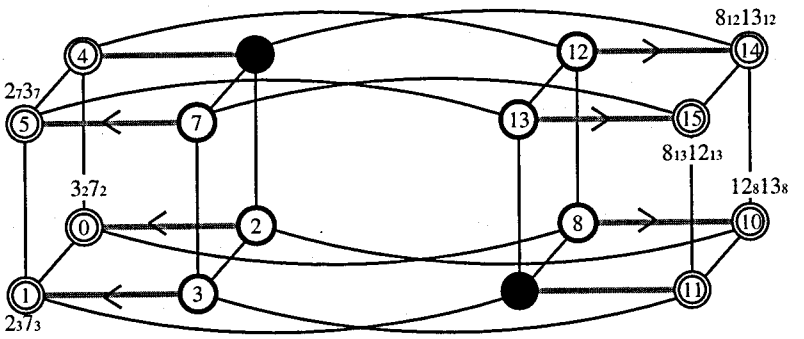

4243475 14814121413
4 12 14
5253575 7 13 15
15815121513
0203007 10810121013
0 2 8 10
1 3 11
1213177 11811121113

(1-3)

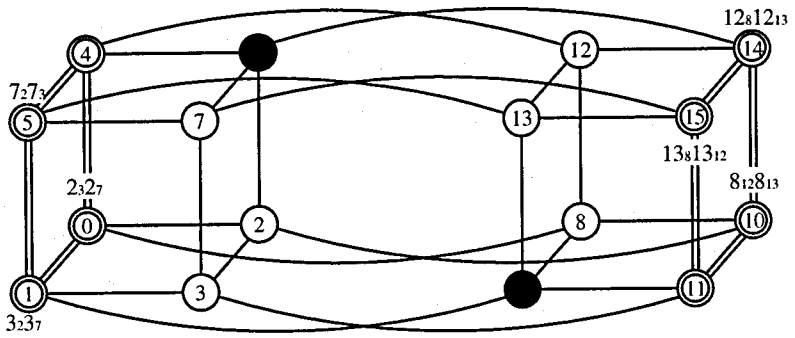

12₁₀12₁₁12₁₄12₁₅
7₀7₁7₄7₅
13₁₀13₁₁13₁₄13₁₅
8₁₀8₁₁8₁₄8₁₅
2₀2₁2₄2₅
3₀3₁3₄3₅

(1-4)


12₁₀12₁₁12₁₄12₁₅
7₀7₁7₄7₅
13₁₀13₁₁13₁₄13₁₅
8₁₀8₁₁8₁₄8₁₅
2₀2₁2₄2₅
3₀3₁3₄3₅


8₁₂13₁₂
2₇3₇
8₁₃12₁₃
3₂7₂
12₈13₈
2₃7₃

(1-5)


12₈12₁₃
7₂7₃
13₈13₁₂
2₃2₇
8₁₂8₁₃
3₂3₇

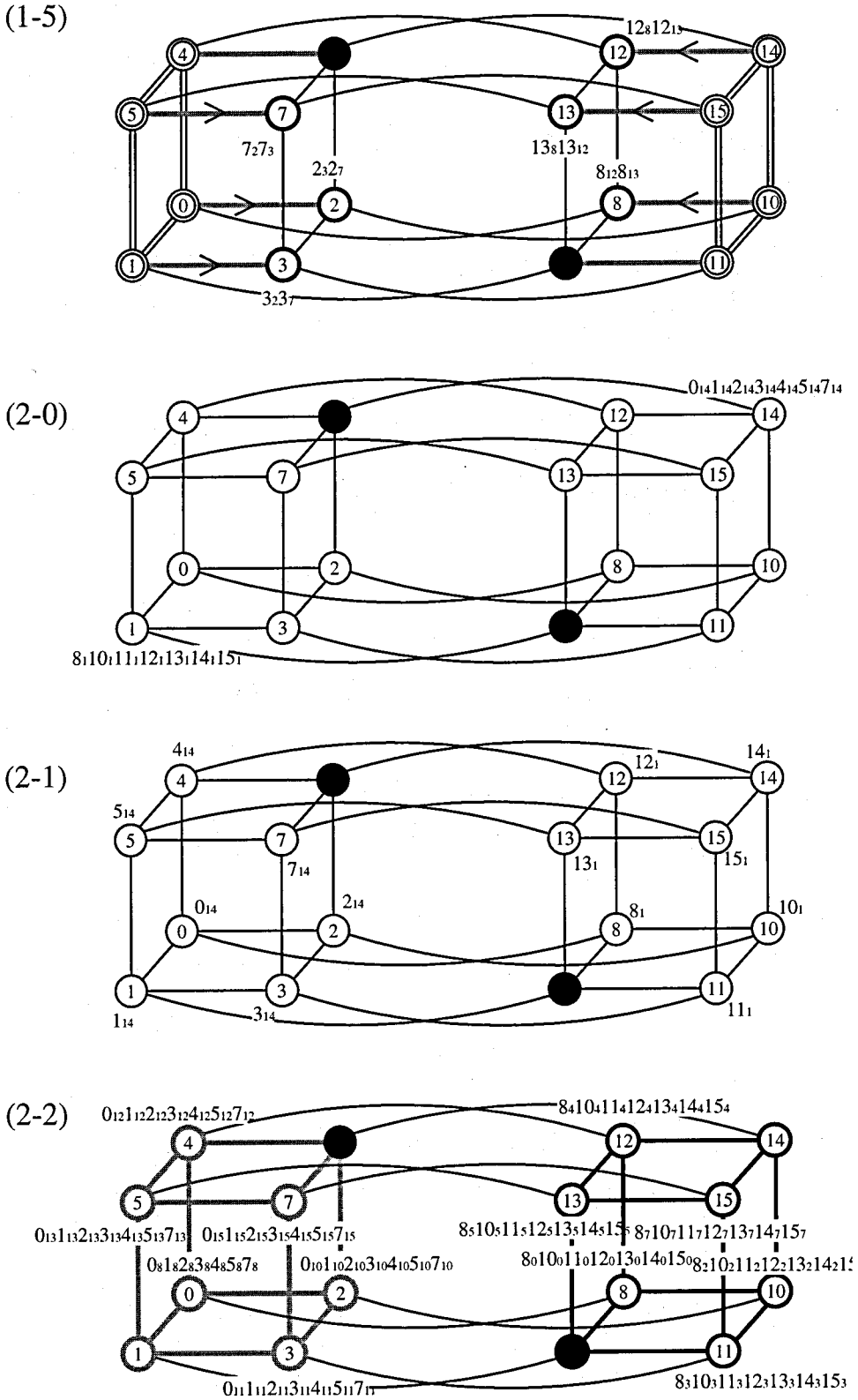
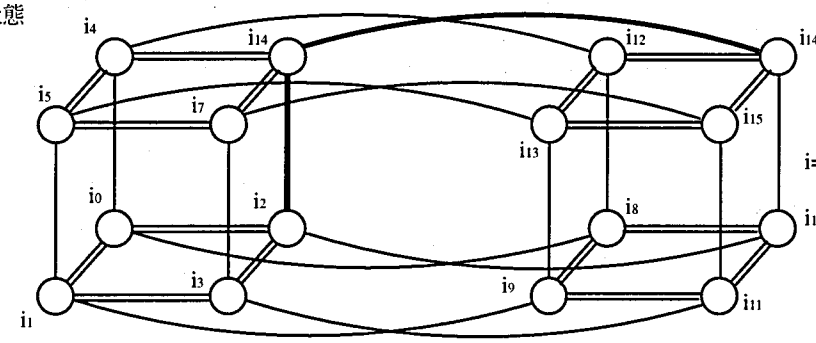
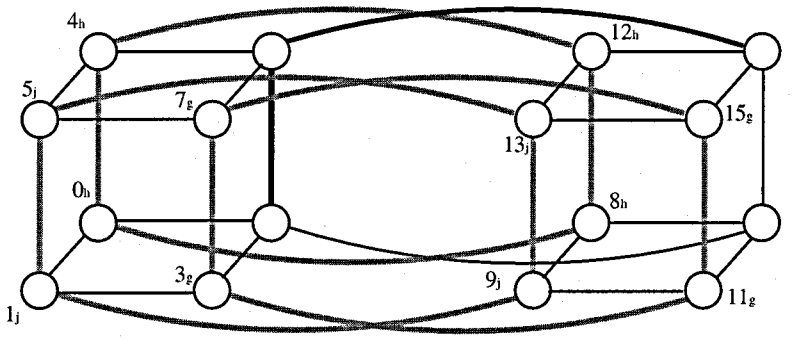


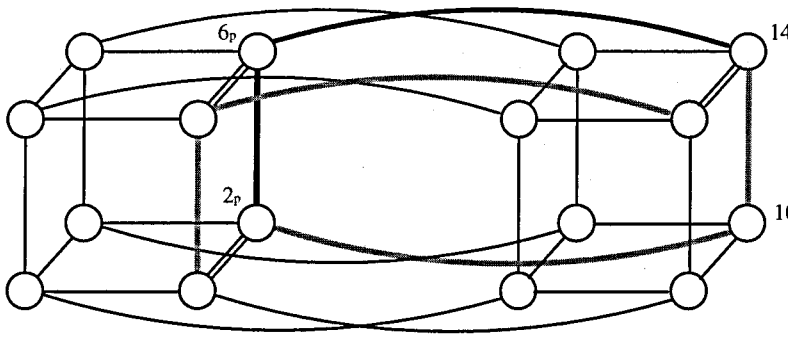
図 4.3: 2個の故障ノードを含む4次元ハイパーキューブへの適用例

(0) 初期状態

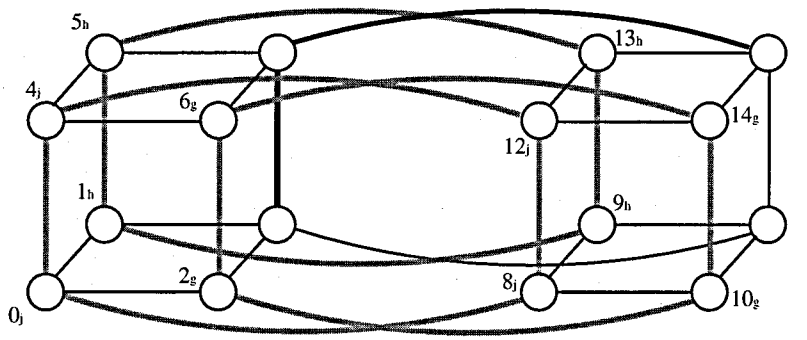


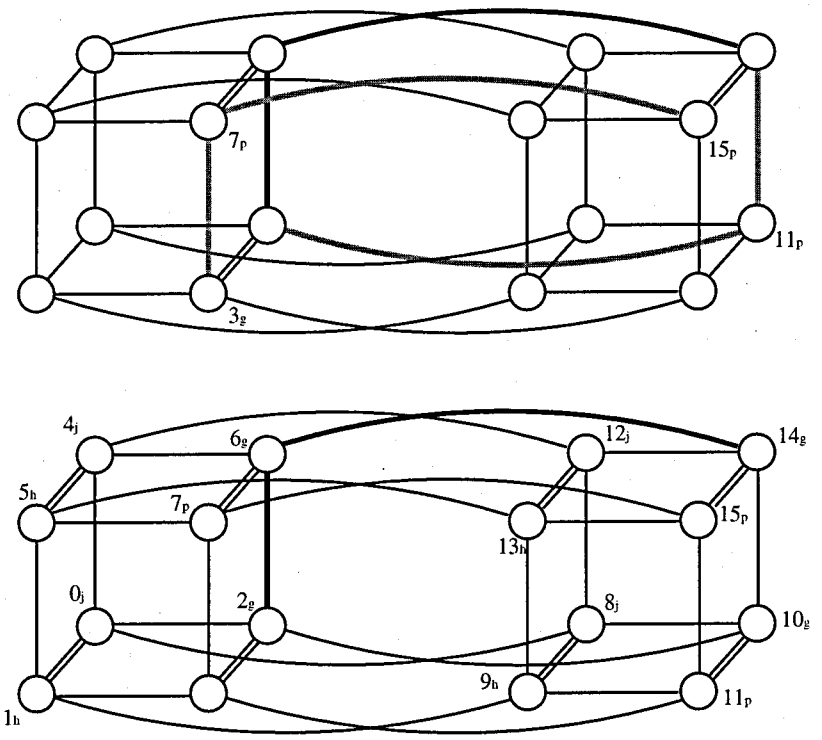
(1) 各Q_2でのAAPC



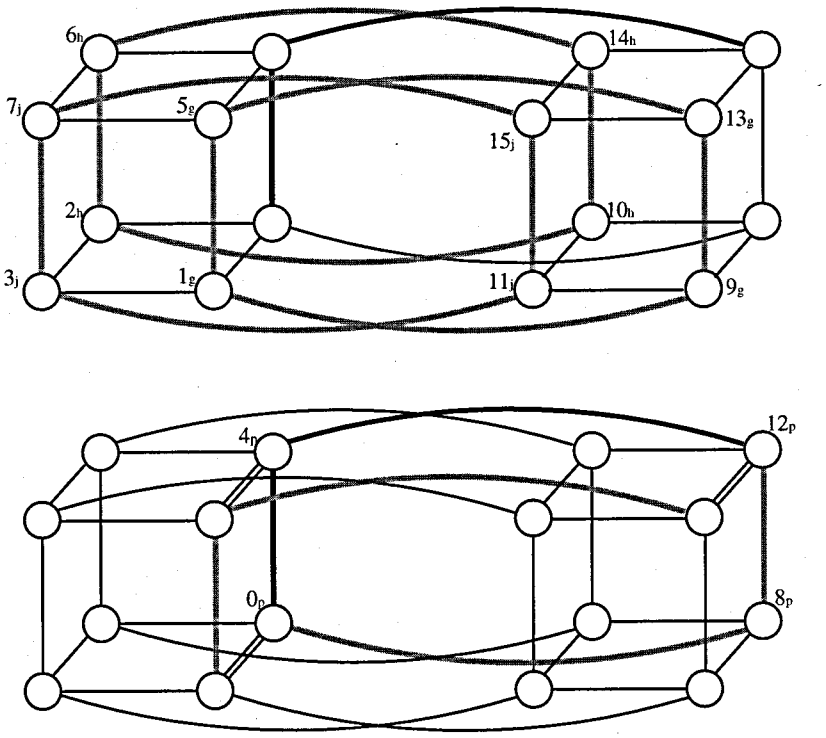


(2-1) 距離1のQ_2間のAAPC





(2-2) 距離1のQ_2間のAAPC



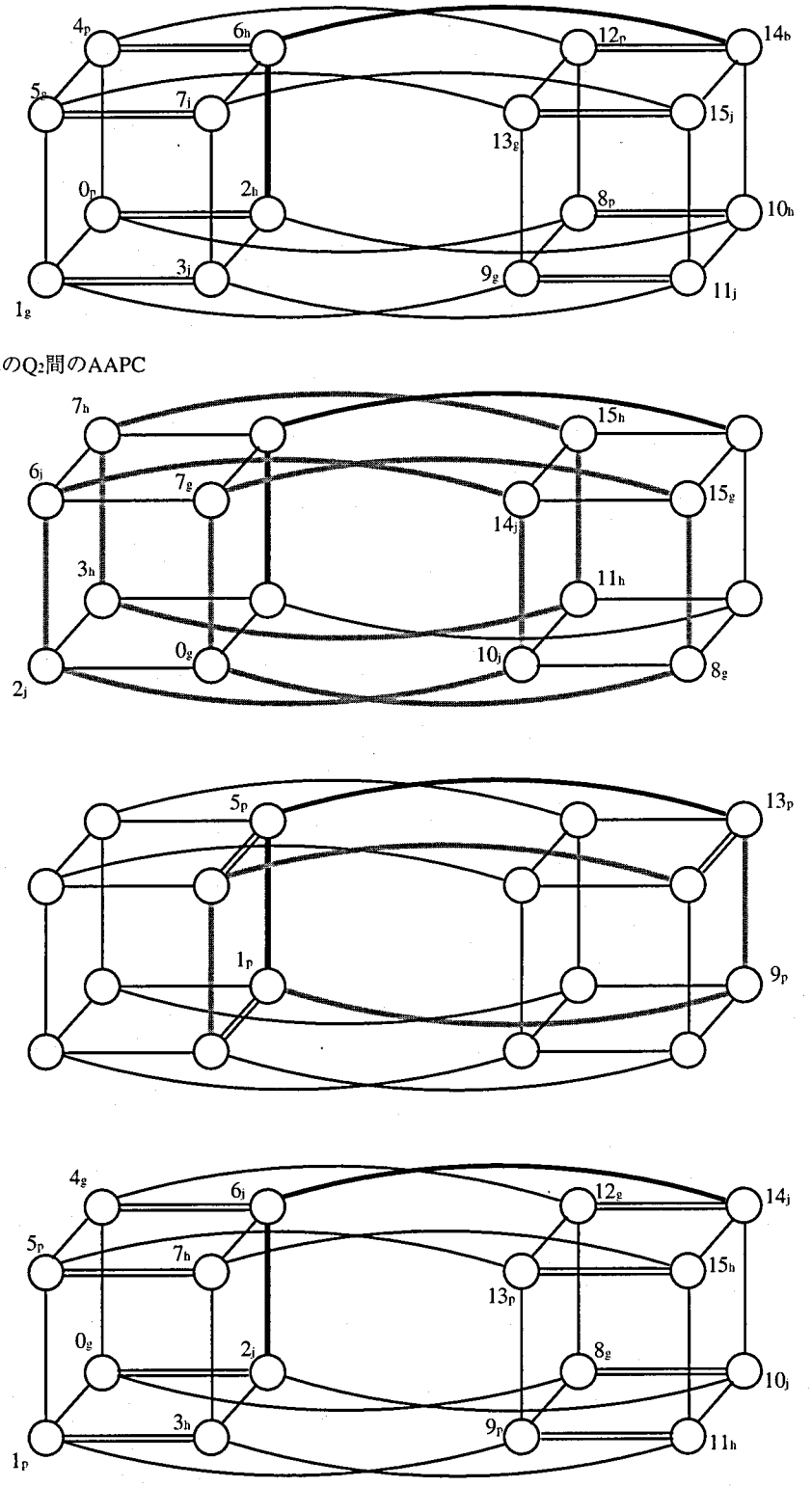


図 4.4: 2個の故障リンクを含む4次元ハイパーキューブへの適用例

4.3 コーダルリング上の耐故障性通信

インターコネクションネットワークには様々な種類があるが，ループ型のネットワークは最もよく使用される型の一つである．中でも，ノードをリング状に結合したシングルループネットワークは，ネットワークインタフェースや制御ソフトウェアが単純な点が優れており，頻繁に使用されるトポロジである．しかしながら，シングルループネットワークはノードまたはアークが故障した場合に信頼性を失う性質がある．この欠点は冗長なリンクを付加することで回避できる．耐故障性通信のためのネットワークアーキテクチャがいくつか存在しているが [88]，コーダルリング [89] はそのようなネットワークの一つであり，これまでに様々な観点からの研究が行われている．コーダルリングは次数3の正規グラフに属しており，ネットワークの直径，すなわちメッセージパスの最大長が解析されていて，グラフを対称にすることで単純な分散アルゴリズムによってメッセージ経路を決定できることが明らかにされている．コーダルリングはネットワーク中に多数のループを含んでいるため，適応的 [90] リング診断が可能であり，故障診断に適している [91]．さらに，故障のあるコーダルリングにおいて，最短側路を見つけるという観点からの分析も行われている．また，コーダルリングの整列能力 [92]，無故障コーダルリング [93] および故障を含むコーダルリング [94] における BPC 置換の能力も研究されている．

前節で述べた4種類の通信プリミティブに対するアルゴリズムは，これまでにハイパーキューブ [87] [95] [96]，メッシュ [95] [97]～[99]，トーラス [99]～[102]，Banyan [103] など，様々なネットワークに対して提案されているが，コーダルリングのための通信アルゴリズムはまだ提案されていない．本研究では，4種類の通信プリミティブのうち，全対全ブロードキャストと全対全個別通信アルゴリズムをコーダルリングに対して提案する．

通信アルゴリズムは1ポートモデルにおいても多ポートモデルにおいても実現できるが，本研究では多ポートモデルだけを考える．多ポートモデルは効率を犠牲にせずに1ポートモデルでシミュレートできるからである．すなわち，ノードの全リンクは同時にパケット送受信ができるものとする．

本節の残りは以下のように構成されている．第 4.3.1 節は後の議論のための準備を行う．第 4.3.2 節と第 4.3.3 節において，それぞれ，全対全ブロードキャストおよび全対全個別通信アルゴリズムを提案する．第 4.3.4 節では，ハイパーキューブにおける同種のアルゴリズムとの比較によって提案するアルゴリズムの性能評価を行う．

4.3.1　準備

コーダルリングは，n個のノードと2種類のリンク(n個のアークと幾つかのコード)から構成される．ノード数nは偶数を仮定し，ノードはリングに沿って$0,1,2,\ldots,n-1$と番号を付ける．アークは$(i, i+1 \bmod n), i = 0,1,2,\ldots,n-1$のリンクである．本論文では，通例通り，もう一種のリンク(コード)について，奇数番ノード$i(i = 1,3,\ldots,n-1)$は$(i-\omega) \bmod n$のノードと連結され，偶数番ノード$j(j = 0,2,\ldots,n-2)$は$(j+\omega) \bmod n$のノードと連結される場合に限定するものとする．ωをコード長と呼び，正の奇数とする．議論の一般性を失うことなく，$\omega \leq n/2$と仮定する．ノード数nが与えられると，コード長ωに応じて多数のコーダルリングが存在する．図4.5は$n = 18$で$\omega = 5$のコーダルリングである．リンクには単方向も双方向もあり得るが，本論文では双方向リンクのみを扱う．

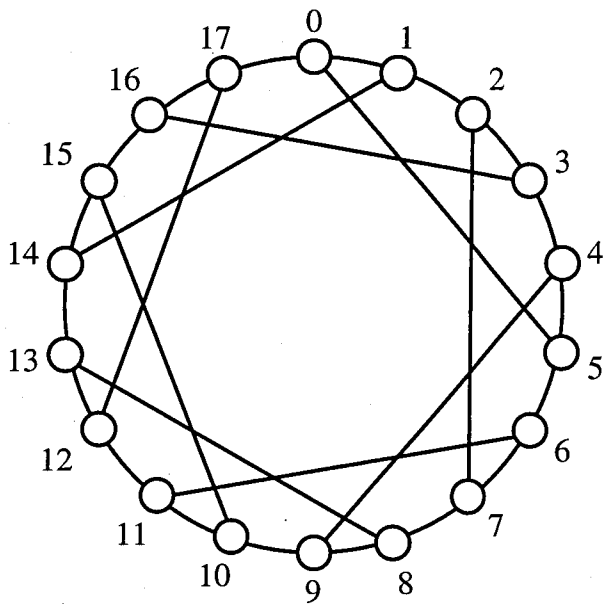


図 4.5: $n = 18$, $\omega = 5$のコーダルリング

コーダルリングの構造は2の倍数のノードをつけ加えることによって拡張できるが，ノード数が増加すると最小直径を与える最適コード長は変化する．直径が最小値である$\sqrt{n}$に近づくときは，コード長も$\sqrt{n}$に近いことが知られている．

ノード数の変化に対して，コーダルリングの性質をもたせるべく連結方法を定式化する試みが[104]において報告されている．コーダルリングでは多くのノード対に対して複数の最短経路が存在する．この性質を最大限に利用した分散ルーティングアルゴリズムも[69]において報告されている．

コーダルリングのための通信アルゴリズムは1ポートモデルにおいても3ポートモデルにおいても実現できる．1ポートモデルでは，ノードは同時に高々1つのリンクでパケットが送信でき，高々1つのリンクでパケットが受信できるモデルである．一方，3ポートモデルでは，ノードは同時にすべてのリンクでパケットが送受信できる．一般に，1ポートモデル用のアルゴリズムは3ポートモデル用のアルゴリズムより単純になるが，ここでは扱わない．すなわち，本論文では3ポートモデルだけを論じる．3ポートモデルは1ポートモデルによって効率を犠牲にすることなくシミュレートできるからである [87].

以下では，隣接ノードへの通信は1単位時間を要するものとする．本研究では，コーダルリングの故障としてはノード故障しか考えない．リンクの故障はノードの故障に置き換えることができるからである．さらに，無故障ノード同士の通信が不可能になるような致命的な故障は扱わない．すなわち，1個ないし2個のノード故障について論じる．

定義 1. 以下において，ループの長さとはループ上のノード数である．

(1) $R(1)$ リングは1つのコード (図4.6で (1) と番号付されている) と，このコードによって迂回される ω 個のアーク，および $(\omega+1)$ 個のノードから構成されるループである．$R(1)$ リングの長さ $|R(1)|$ は $(\omega+1)$ である．

(2) $R(2)$ リングは，共通に迂回するアークが1つである2つの連続するコード (図4.6で (1) および (2) と番号付されている) と，$2(\omega-1)$ 個のアーク，および $2w$ 個のノードから構成されるループである．但し，共通に迂回されるアークは含まない．$R(2)$ リングの長さ $|R(2)|$ は $2w$ である．

(3) $R(i)$ リングは，$R(i-1)$ リングから i 番目に付加される新しいコード (図4.6で (i) と番号付されている) によって迂回されるアークを除いたものに，新しく付加するコード1つ，そのコードによって迂回される $(\omega-1)$ 個のアークとノードから構成される．$R(i)$ リングの長さ $|R(i)|$ は $i(\omega-1)+2$ である．コーダルリングには $n/2$ 個の $R(i)$ リングが存在する．最長の $R(i)$ は $R(\lceil(n-2)/(\omega-1)\rceil)$ である．図4.6に $R(i)$ を示す．

(4) 各ループ上の $R(i)^{start}$ と $R(i)^{end}$ とラベル付けされたノードは，ループを見つけるための特別な印である．以下では，ノード i とノード j の距離 (最短パス上のノード数．ノード i は含まずノード j は含む) を $d(i,j)$ と表記する．

性質 1. $n \geq 2(\omega+1)$ のとき，コーダルリングは2個の分離したリング $R(1)$，$R(i)$ を持つ．ここで，$i = \lfloor(n-\omega-3)/(\omega-1)\rfloor \geq 1$ である．

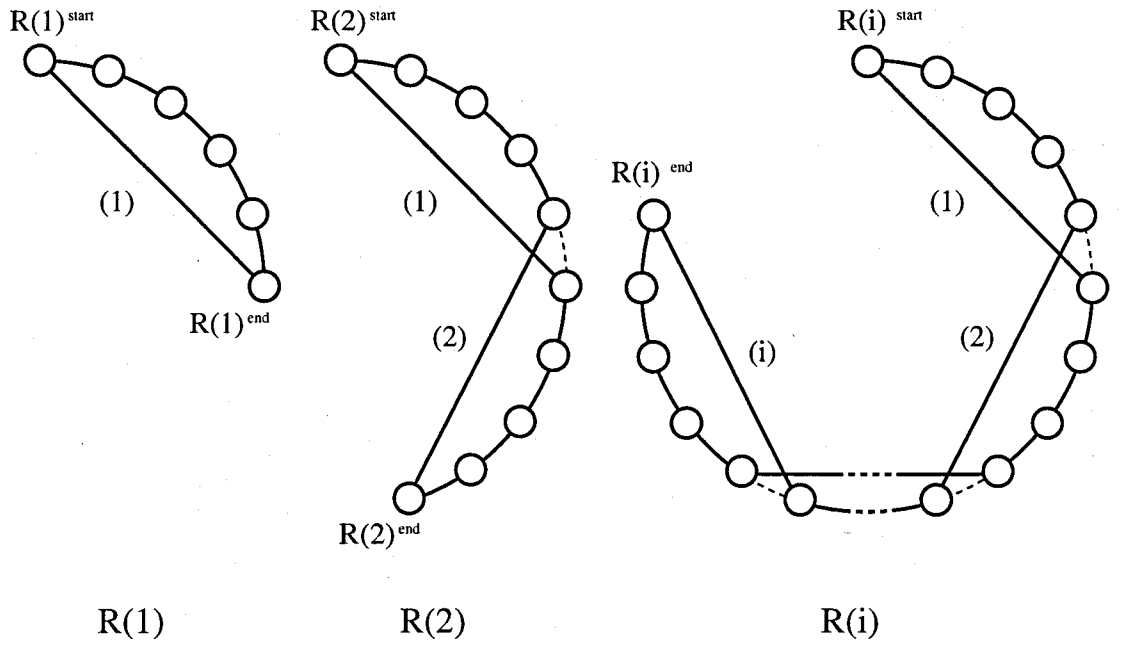


図 4.6: $\omega = 5$ の場合の $R(1)$, $R(2)$, $R(i)$

証明. 自明. コーダルリングは2個の分離したリング $R(1)$ と $R(i+1)$ を持つことはできない. $R(i)^{start}$ がノード $(\omega+2)$ のとき, $R(i)$ リング上の任意のノード j は2個の隣接ノードを以下のように持つ.

1. ノード $j = (\omega+2)$ はノード $j+1$ と $j+\omega$ を持つ.
2. ノード $j = (\omega+2+\alpha(\omega-1))$ はノード $j-1$ と $j+\omega$ を持つ.
3. ノード $j = (\omega+3+\alpha(\omega-1))$ はノード $j+1$ と $j-\omega$ を持つ.
4. ノード $j = (\omega+3+i(\omega-1))$ はノード $j-1$ と $j-\omega$ を持つ.
5. ノード j(その他の場合はノード $j-1$ と $j+1$ を持つ.

ここで, $\alpha = 1, 2, \ldots, i-1$ である. 一例を図4.7に示す. □

性質 2. $n \geq 3(\omega+1)$ のとき, コーダルリングは3個の分離したリングを持つ. それらは2個の連続した $R(1)$ リングおよび $R(i)$ リングである. ここで, $i = \lfloor (n-2\omega-4)/(\omega-1) \rfloor$ である.

証明. 自明. コーダルリングは $R(i)$ の代わりに $R(i+1)$ を持つことはできない. $R(i)^{start}$ がノード $2(\omega+1)+1$ のとき, $R(i)$ リング上の任意のノード j は2個の隣接ノードを以下のように持つ.

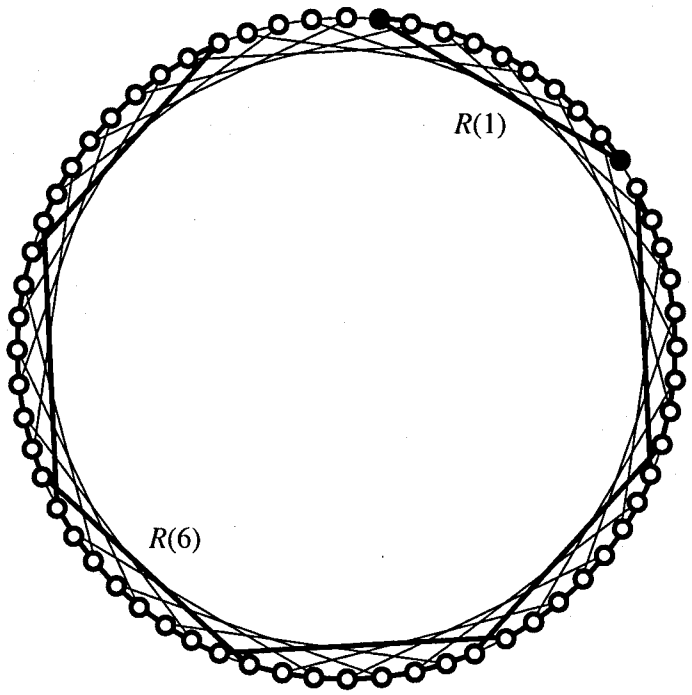


図 4.7: $R(1)$, $R(6)$ を持つ $n=64$, $\omega=9$ のコーダルリング

1. ノード $j=(2(\omega+1)+1)$ はノード $j+1$ と $j+\omega$ を持つ.
2. ノード $j=(2(\omega+1)+1+\alpha(\omega-1))$ はノード $j-1$ と $j+\omega$ を持つ.
3. ノード $j=(2(\omega+1)+2+\alpha(\omega-1))$ はノード $j+1$ と $j-\omega$ を持つ.
4. ノード $j=(2(\omega+1)+2+i(\omega-1))$ はノード $j-1$ と $j-\omega$ を持つ.
5. ノード j(その他の場合) はノード $j-1$ と $j+1$ を持つ.

ここで, $\alpha=1,2,\ldots,i-1$ である. 一例を図 4.8 に示す. □

性質 3. $n\geq 4(\omega+1)$ のとき, コーダルリングは 4 個の分離したリングを持つ. それらは時計回りに $R(1)$, $R(i_1)$, $R(1)$, $R(i_2)$ である. ここで, $1\leq i_1\leq i_2$ である.

証明. 自明. $R(i_1)^{start}$ がノード $(\omega+2)$ のとき, $R(i_1)$ リング上の任意のノード j_1 は 2 個の隣接ノードを以下のように持つ.

1. ノード $j_1=(\omega+2)$ はノード j_1+1 と $j_1+\omega$ を持つ.
2. ノード $j_1=(\omega+2+\alpha_1(\omega-1))$ はノード j_1-1 と $j_1+\omega$ を持つ.
3. ノード $j_1=(\omega+3+\alpha_1(\omega-1))$ はノード j_1+1 と $j_1-\omega$ を持つ.

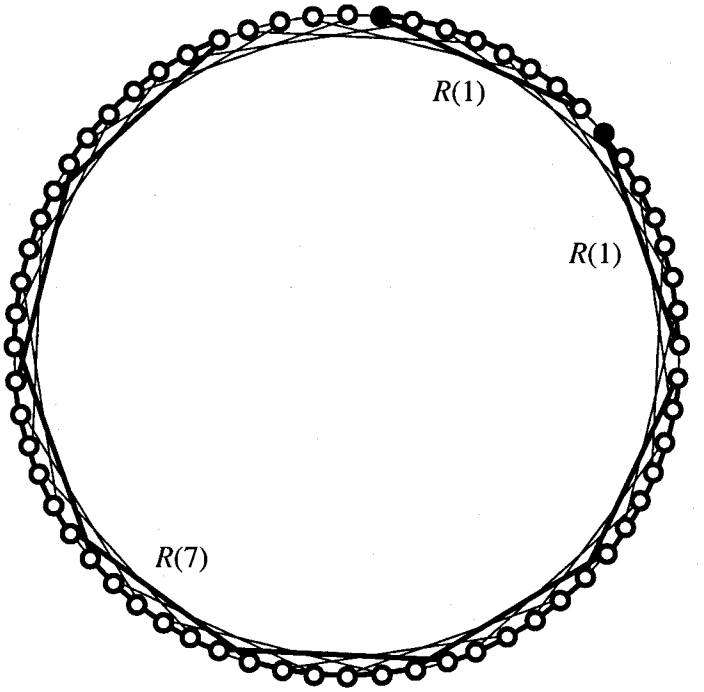


図 4.8: $R(1)$, $R(1)$, $R(7)$ を持つ$n = 64$, $\omega = 9$ のコーダルリング

4. ノード $j_1 = (\omega + 3 + i_1(\omega - 1))$ はノード $j_1 - 1$ と $j_1 - \omega$ を持つ.

5. ノード j_1(その他の場合) はノード $j_1 - 1$ と $j_1 + 1$ を持つ.

ここで, $\alpha_1 = 1, 2, \ldots, \lfloor((\beta - 2)(\omega + 1) - 2)/(\omega - 1)\rfloor - 1$ で, β は $(\beta - 1)(\omega + 1) \leq d(R(i_1)^{start}, R(i_2)^{start})$ を満たす最大の数である.

$R(i_2)$ リング上の任意のノード j_2 は2個の隣接ノードを以下のように持つ.

1. ノード $j_2 = \beta(\omega + 1) + 1$ はノード $j_2 + 1$ と $j_2 + \omega$ を持つ.

2. ノード $j_2 = (\beta(\omega + 1) + 1 + \alpha_2(\omega - 1))$ はノード $j_2 - 1$ と $j_2 + \omega$ を持つ.

3. ノード $j_2 = (\beta(\omega + 1) + 2 + \alpha_2(\omega - 1))$ はノード $j_2 + 1$ と $j_2 - \omega$ を持つ.

4. ノード $j_2 = (\beta(\omega + 1) + 2 + i_2(\omega - 1))$ はノード $j_2 - 1$ と $j_2 - \omega$ を持つ.

5. ノード j_2(その他の場合) はノード $j_2 - 1$ と $j_2 + 1$ を持つ.

ここで, $\alpha_2 = 1, 2, \ldots, \lfloor(n - \beta(\omega + 1) - 2)/(\omega - 1)\rfloor - 1$ である. 一例を図4.9に示す. □

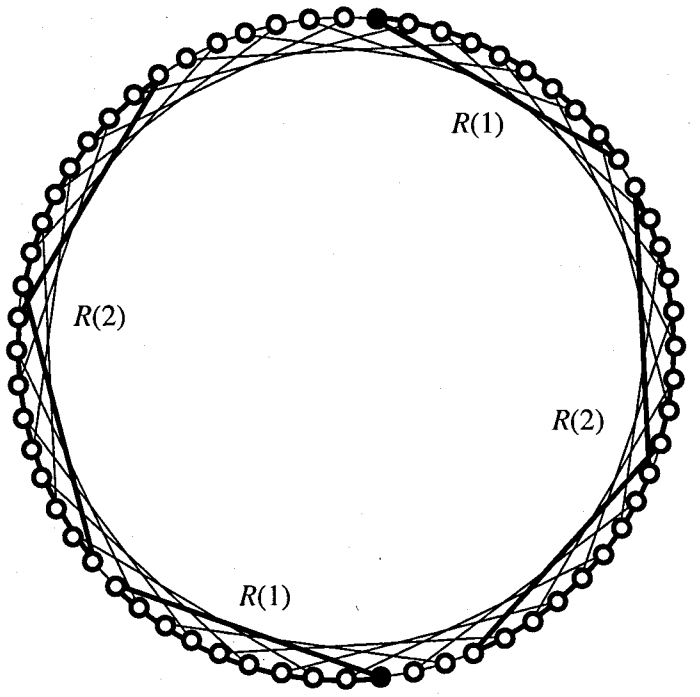


図 4.9: $R(1)$, $R(2)$, $R(1)$, $R(2)$ を持つ $n = 64$, $\omega = 9$ のコーダルリング

4.3.2 全対全ブロードキャストアルゴリズム

本節では全対全ブロードキャストアルゴリズムを論じる.

無故障

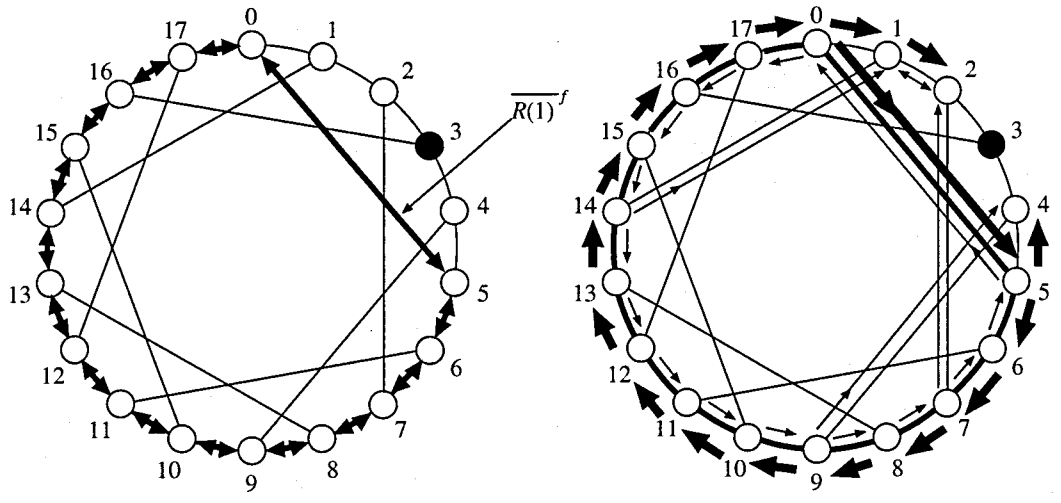
すべてのノードについて, 自分自身のパケットをアークで接続された両側の隣接ノードに送り, 隣接ノードから受け取ったパケットをアークで接続された反対側の隣接ノードに送る. コーダルリングを直径 $n/2$ のシングルループとして使っているので, すべてのノードについて, この操作は $n/2$ 時間で終了する. すなわち, 全対全ブロードキャストが $n/2$ 時間で実行できる.

1 個の故障

故障ノード f を中央に含むような $R(1)$ を $R(1)^f$ とする. また, $R(1)^f$ に含まれないすべてのノードに $R(1)^{start}$ と $R(1)^{end}$ を加えてできる最長無故障シングルループを $\overline{R(1)^f}$ とする. 図 4.10(a) 中の太線が $\overline{R(1)^f}$ の例である.

全対全ブロードキャストは次のようにして $n-(\omega-1)+\lceil(\omega-1)/2\rceil$ 時間で実行できる. $R(1)^f$ 上の $R(1)^{start}$ と $R(1)^{end}$ 以外のすべての無故障ノードについて, $\overline{R(1)^f}$

上の隣接ノードに自分自身のパケットを送る．同時に，$\overline{R(1)^f}$ において，コードを用いて全対全ブロードキャストを開始する．まずそれ自身のパケットについて実行し（$(n-(\omega-1))/2$ 時間かかる），次に $R(1)^f$ から送られたパケットについて実行する（$(n-(\omega-1))/2$ 時間かかる）．これらの操作の間，隣接ノードから受け取ったパケットを反対側の隣接ノードに送る．ノード j がコードによって $R(1)^f$ 上の無故障ノードと結合されている場合，そのノードに対して，大きい番号を持つ方のノードから受け取ったパケットを送る．$R(1)^f$ において，故障ノードの隣接ノードにパケットが送られるのが最後の操作となる（$\lceil(\omega-1)/2\rceil$ は $\max(d(R(1)^{start}, f), d(R(1)^{end}, f))$ である）．図 4.10(b) はブロードキャスト中のパケットの流れの例である．



(a) $\overline{R(1)^f}$ リング　　(b) パケットの流れ

図 4.10: $n = 18$, $\omega = 5$ でノード 3 が故障しているコーダルリング上のブロードキャスト

2 個の故障

2 個の故障ノード f_1 と f_2 間の距離を $d(f_1, f_2)$ とする．ここで，f_1 および f_2 はノード番号も表すものとし，簡単のために $f_1 \leq f_2$ とする．ブロードキャストアルゴリズムおよび通信時間を以下の 4 つの場合に分けて論じる．

1. $1 \leq d(f_1, f_2) \leq \omega$ の場合

   一般性を失うことなく，$1 \leq f_1, f_2 \leq \omega$ なる f_1 と f_2 が $R(1)$ の中に存在すると仮定できる．

   (a) $f_1 + 1 = f_2$ の場合 (図 4.11 に例を示す)

f_1 と f_2 が中央にあるような $R(1)$ を見つける．これら 2 個の故障は 1 個の故障とみなせるので，全対全ブロードキャストアルゴリズムは 1 個の故障の場合とほとんど同じで，$n-(\omega-1)+\lceil(\omega-1)/2\rceil$ 時間で実行できる．

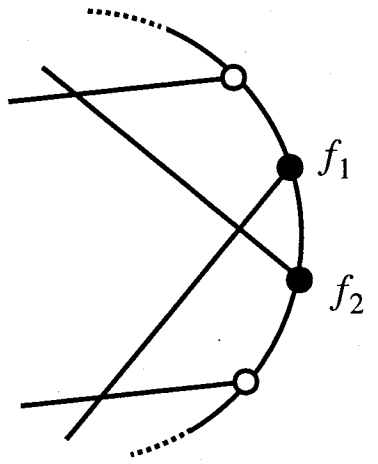


図 4.11: $f_1+1=f_2$ の場合の例

(b) $f_1+2=f_2$ の場合 (図 4.12 に例を示す)
f_1 と f_2 が $R(1)^{start}$ と $R(1)^{end}$ にならないような $R(1)$ を見つける．通信時間は $\max(n-3, n-(\omega-1)+1)$ となる．f_1 と f_2 の間の無故障ノードはコードからしかパケットを受け取れないので，通信時間は前記の 2 つの要素の大きい方になる．

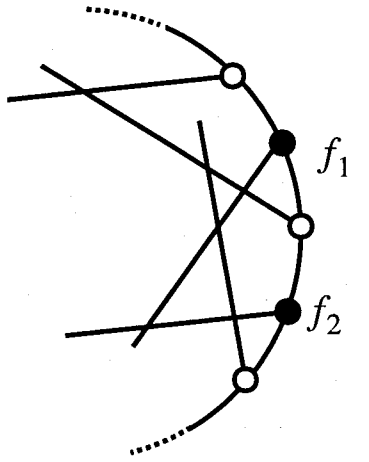


図 4.12: $f_1+2=f_2$ の場合の例

(c) その他の場合

全対全ブロードキャストの実行時間は，以下のように，$(n-\omega-3)/(\omega-1)$ が整数のときとそうでないとき，高々$(\omega-1)i+2+\omega-2$ と $(\omega-1)i+3+\omega-2$ になる．ここで，$i=\lfloor(n-\omega-3)/(\omega-1)\rfloor$ とする．

i. $(n-\omega-3)/(\omega-1)$ が整数の場合

f_1 と f_2 を含む $R(1)$ 上のすべての無故障ノードについて，$R(i)$ 上の隣接ノードに自分自身のパケットを送る．同時に，$R(i)$ において，コードを用いて全対全ブロードキャストを開始する．まずそれ自身のパケットについて実行し $((\omega-1)i+2)/2$ 時間かかる)，次に $R(1)$ から送られた $(\omega-3)$ 個のパケットについて実行する $((\omega-1)i+2)/2$ 時間かかる)．これらの操作の間，隣接ノードから受け取ったパケットを反対側の隣接ノードに送る．ノード j がコードによって $R(1)$ 上の無故障ノードと結合されている場合，そのノードに対して，大きい番号を持つ方のノードから受け取ったパケットを送る．

ii. $(n-\omega-3)/(\omega-1)$ が非整数の場合

図4.7がこの場合の例である．$R(1)$ にも $R(i)$ にも属さない無故障ノード自身のパケットは，$R(1)$ と $R(i)$ 上の異なるノードに対して高々2単位時間で送ることができる．したがって，全対全ブロードキャストの通信時間は i が整数の場合に比べて1単位時間だけ長くなる．

2. $\omega+1=d(f_1,f_2)$ の場合

これは図4.13に示す場合である．ノード j は1つのアークからしかパケット受け取れず，f_1 と f_2 は同じ $R(1)$ を共有することはないので，通信時間は $n-3$ となる．

3. $\omega+1<d(f_1,f_2)\leq 2\omega+1$ の場合

性質2から，コーダルリングが3つの分離したリング，すなわち f_1 と f_2 が別々に含まれるような2個の連続した $R(1)$ および $R(i)$ を持つことが仮定できる．ここで，$i=\lfloor(n-2\omega-4)/(\omega-1)\rfloor$ である．全対全ブロードキャストの実行時間は，以下のように，$(n-2\omega-4)/(\omega-1)$ が整数のときとそうでないとき，高々$(\omega-1)i+6+(\omega-1)$ と $(\omega-1)i+8+(\omega-1)$ になる．

(a) $(n-2\omega-4)/(\omega-1)$ が整数の場合

$R(1)$ 上のすべての無故障ノードについて，$R(i)$ 上の異なるノードに自分自身のパケットを送る．これには高々2単位時間を要する．次いで $R(i)$ において全対全ブロードキャストを開始する．まずそれ自身のパケットにつ

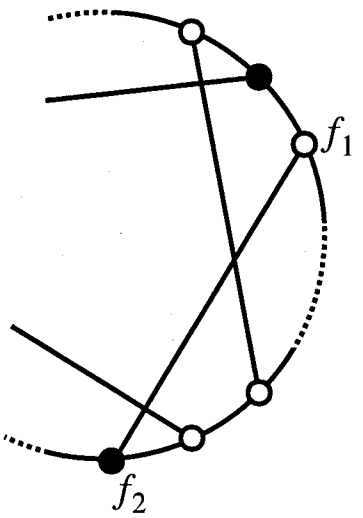


図 4.13: $\omega+1=d(f_1,f_2)$ の場合の例

いて実行し $((\omega-1)i+2)/2$ 時間かかる)，次に $R(1)$ から送られたパケットについて実行する $((\omega-1)i+2)/2$ 時間かかる)．この場合は，全対全ブロードキャストが終了したとき，$R(1)$ に送られるべきパケットが残っている可能性がある．これらのパケットは高々2 単位時間で送ることができる．最後に，$R(1)$ のアーク上でパケットを送るのに高々$(\omega-1)$ 単位時間を要する．したがって，総通信時間は $2+((\omega-1)i+2)+2+(\omega-1)=(\omega-1)i+6+(\omega-1)$ となる．

(b) $(n-2\omega-4)/(\omega-1)$ が非整数の場合

$R(i)$ に属さない無故障ノード自身のパケットは，$R(i)$ 上の異なるノードに対して高々3 単位時間で送ることができる．したがって，総通信時間は $3+((\omega-1)i+2)+3+(\omega-1)=(\omega-1)i+8+(\omega-1)$ となる．

4. $2\omega+1<d(f_1,f_2)$ の場合

f_1 と f_2 をそれぞれ別々に含む $R(i_1)$ と $R(i_2)$ を見つける．ただし，$R(i_1)^{start}$, $R(i_1)^{end}$, $R(i_2)^{start}$, $R(i_2)^{end}$ が無故障ノードになるようにする．$R(i_1)^{start}$, $R(i_1)^{end}$, $R(i_1)$ と $R(i_2)$ 以外のノードに $R(i_2)^{start}$, $R(i_2)^{end}$ を加えてできる，長さ $n-2(\omega-1)$ のシングルループを作り，全対全ブロードキャストを行う．故障が 1 個の場合と同様にして，通信時間が $n-2(\omega-1)+\lceil(\omega-1)/2\rceil$ であることが導かれる．

4.3.3 全対全個別通信アルゴリズム

本節では全対全個別通信アルゴリズムを論じる．

無故障

すべてのノードについて，2個の個別パケットを異なる方向の最も遠いノードにそれぞれ送り，次に2番目に遠いパケットに送り，...，最後にアークで接続された両側の隣接ノードに送る．この様子を図4.14に示す．これらの操作の間，隣接ノードから受け取ったパケットをアークで接続された反対側の隣接ノードに送る．コーダルリングを直径$n/2$のシングルループとして使っているので，全対全個別通信が$\sum_{i=1}^{n/2} i$時間で実行できる．

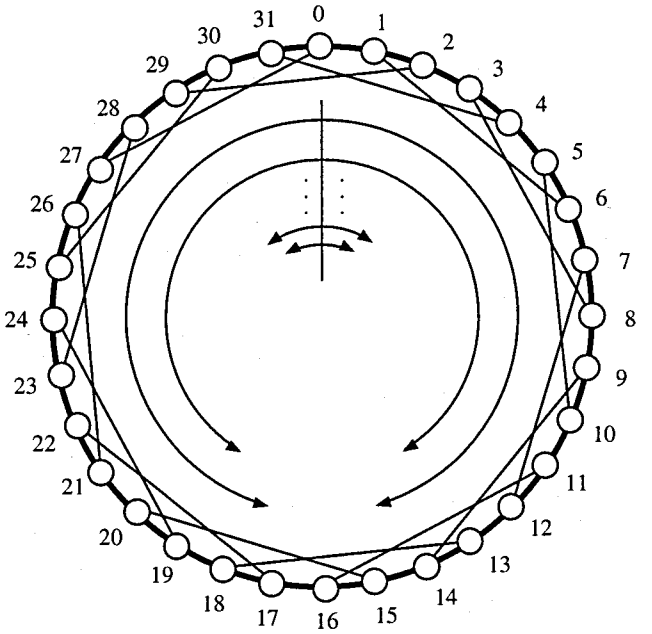


図 4.14: ノード0からのパケット送信

1個の故障

全対全ブロードキャストにおいて故障ノードが1個の場合と同じく，故障ノードfを中央に含むような$R(1)$を$R(1)^f$，$R(1)^f$に含まれないすべてのノードに$R(1)^{start}$と$R(1)^{end}$を加えてできる最長無故障シングルループを$\overline{R(1)^f}$とする．全対全個別通信は以下に示すように，$2\sum_{i=1}^{\alpha} i + 2\sum_{i=\alpha+1}^{\alpha+\beta} i$時間で実行できる．ここで，$\alpha = \lceil (n-(\omega-1))/2 \rceil$で，$\beta = \lceil (\omega-1)/2 \rceil$である．まず，$\overline{R(1)^f}$上のすべての個別パケッ

トについて，すべての無故障ノードに対して全対全個別通信を行う．このとき，各ノードは受信パケットを全対全ブロードキャストにおいて故障ノードが1個の場合と同様に扱う．図4.15に示すように，これには $\sum_{i=1}^{\alpha} i + \sum_{i=\alpha+1}^{\alpha+\beta} i$ 時間を要する．次に，$R(1)^{start}$ と $R(1)^{end}$ を除く $R(1)^f$ 上の無故障ノードから個別パケットを $\overline{R(1)^f}$ 上の隣接ノードに，距離が遠いものから順に送る．隣接ノードは受け取ったパケットを全対全個別通信する．このとき，各ノードは受信パケットを全対全ブロードキャストにおいて故障ノードが1個の場合と同様に扱う．これには $\sum_{i=1}^{\alpha} i + \sum_{i=\alpha+1}^{\alpha+\beta} i + 1$ 時間を要する．1番目の操作の最後のステップと2番目の操作の最初のステップは同時に実行できるので，全対全個別通信の実行時間は上述のようになる．

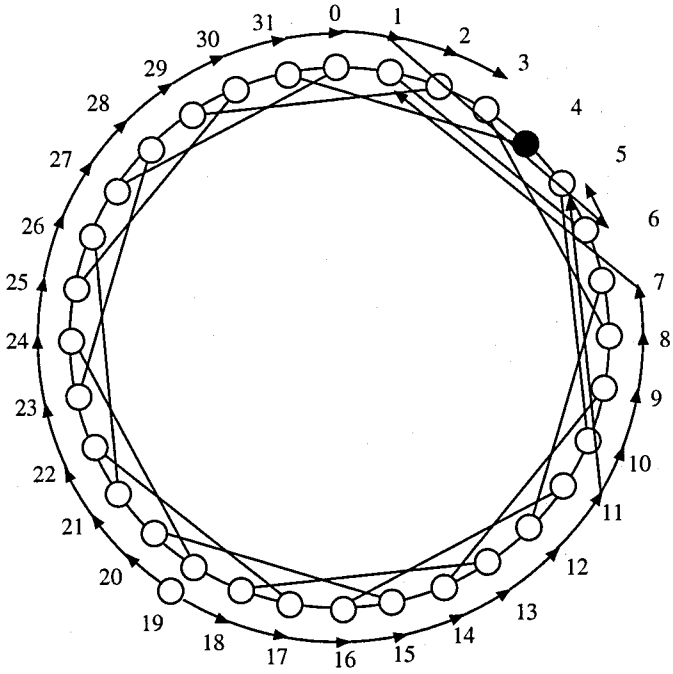


図 4.15: ノード 19からのパケット送信の経路

2個の故障

1. $1 \leq d(f_1, f_2) \leq \omega$ の場合

   (a) $f_1 + 1 = f_2$ の場合

   f_1 と f_2 が中央にあるような $R(1)$ を見つける．これら2個の故障は1個の故障とみなせるので，全対全個別通信アルゴリズムは1個の故障の場合とほとんど同じで，$2\sum_{j=1}^{\alpha} j + 2\sum_{j=\alpha+1}^{\alpha+\beta} j$ 時間で実行できる．

   (b) $f_1 + 2 = f_2$ の場合

まず，$\overline{R(1)^f}$ 上で 4.3.3 節で述べた方法で全対全個別通信を行う．次に，$\overline{R(1)^f}$ のノードと $R(1)^f$ の無故障ノードの間で全対全個別通信を行う．この様子を図 4.16 に示す例で説明する．ノード a が受け取った 2 個のパケットをノード b に 2 ステップ (=2 単位時間) かけて送る．同時に，ノード b から $\overline{R(1)^f}$ の異なる方向に対して 2 パケット送る．したがって，通信時間は $2\sum_{j=1}^{\alpha} j + \sum_{j=1}^{\alpha}(j+2)$ となる．

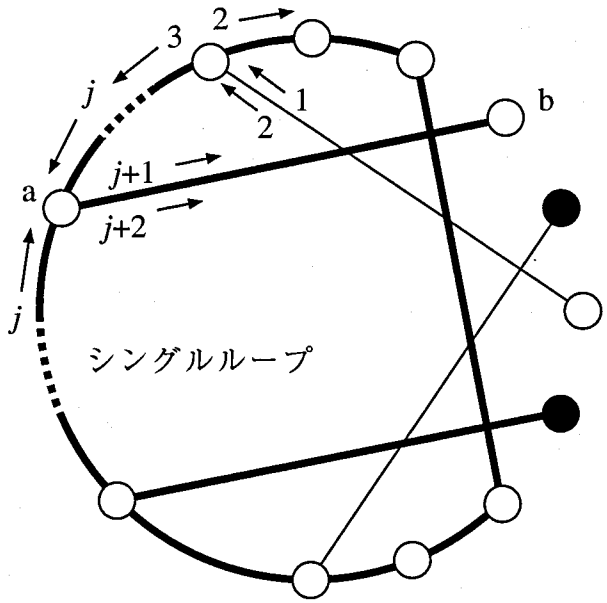


図 4.16: $f_1 + 2 = f_2$ の場合の例

(c) その他の場合

全対全個別通信の実行時間は，以下のように $(n-\omega-3)/(\omega-1)$ が整数のときとそうでないとき，それぞれ，$\sum_{j=1}^{\gamma} j + \sum_{j=1}^{\gamma}(j+2)$ と $\sum_{j=1}^{\gamma} j + \sum_{j=1}^{\gamma}(j+2) + \sum_{j=1}^{\gamma}(j+3)$ になる．ここで，$\gamma = ((\omega-1)i+2)/2$，$i = \lfloor (n-\omega-3)/(\omega-1) \rfloor$ である．

i. $(n-\omega-3)/(\omega-1)$ が整数の場合

コーダルリングは図 4.17 に示すように変更することできる．図 4.15 のシングルループの代わりに $R(i)$ を使うことで，2 番目の場合と同様にして全対全個別通信が実行できる．$R(i)$ リングの長さは 2γ なので，前記の通信時間が得られる．

ii. $(n-\omega-3)/(\omega-1)$ が非整数の場合

図 4.18 がこの場合の例である．$R(i)$ リング上にないすべての無故障ノードから $R(i)$ 上の最も近いノードへの距離は高々2 であるから，通信アルゴリズムは次のようになる．まず，無故障 $R(i)$ リング上で

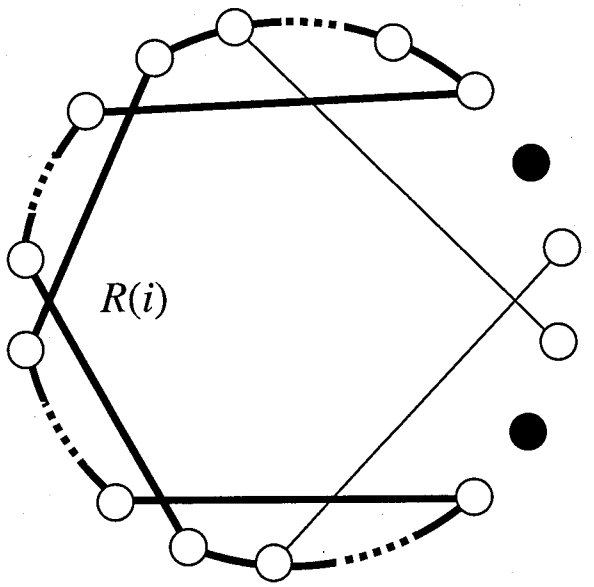


図 4.17: $(n-\omega-3)/(\omega-1)$ が整数の場合

4.3.3 節で述べた方法で全対全個別通信を行う．次に，$\overline{R(1)^f}$ のノードと，$R(1)^f$ の無故障ノードで $\overline{R(1)^f}$ との距離が 1 のものの間で全対全個別通信を行う ($\sum_{j=1}^{\gamma}(j+2)$ かかる)．最後に，$\overline{R(1)^f}$ のノードと，$R(1)^f$ の無故障ノードで $\overline{R(1)^f}$ との距離が 2 のものの間で全対全個別通信を行う (図 4.19 に示すように，$\sum_{j=1}^{\gamma}(j+3)$ かかる)．

2. $\omega+1=d(f_1,f_2)$ の場合

   (a) $(n-\omega-3)/(\omega-1)$ が整数の場合

   $R(i)$ リング上にないすべての無故障ノードから $R(i)$ 上の最も近いノードへの距離は高々3であるから，通信アルゴリズムは距離が 2 の場合のものから容易に導かれ，$\sum_{j=1}^{\gamma} j+\sum_{j=1}^{\gamma}(j+2)+\sum_{j=1}^{\gamma}(j+3)+\sum_{j=1}^{\gamma}(j+4)$ 時間を要する．

   (b) $(n-\omega-3)/(\omega-1)$ が非整数の場合

   $R(i)$ リング上にないすべての無故障ノードから $R(i)$ 上の最も近いノードへの距離は高々4であるから，通信アルゴリズムは $\sum_{j=1}^{\gamma} j+\sum_{j=1}^{\gamma}(j+2)+\sum_{j=1}^{\gamma}(j+3)+\sum_{j=1}^{\gamma}(j+4)+\sum_{j=1}^{\gamma}(j+5)$ 時間を要する．

3. $\omega+1<d(f_1,f_2)\leq 2\omega+1$ の場合

   (a) $(n-\omega-3)/(\omega-1)$ が整数の場合

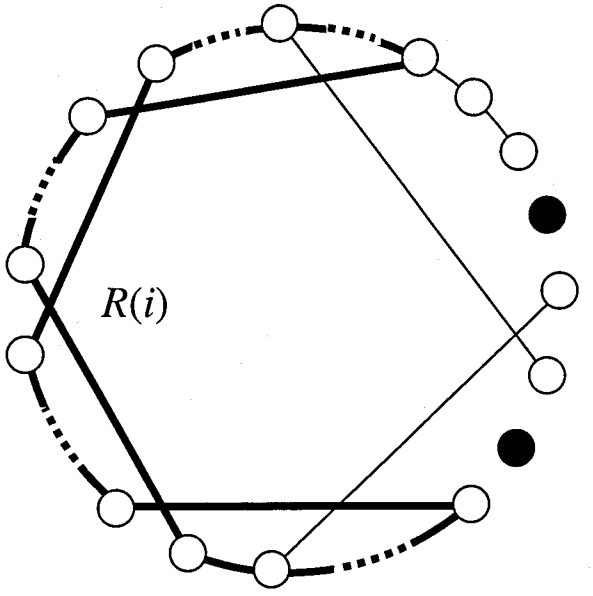


図 4.18: $(n-\omega-3)/(\omega-1)$ が非整数の場合

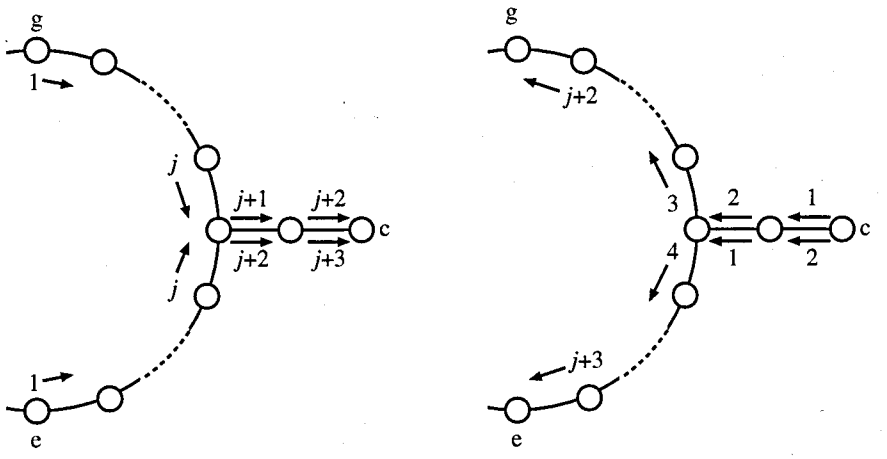


図 4.19: $d(g,c) = d(e,c) = j+2$ のときのパケット伝達の順序

$R(i)$ リング上にないすべての無故障ノードから $R(i)$ 上の最も近いノードへの距離は高々2であるから，通信アルゴリズムは $\sum_{j=1}^{\gamma} j + \sum_{j=1}^{\gamma}(j+2) + \sum_{j=1}^{\gamma}(j+3)$ 時間を要する．

(b) $(n-\omega-3)/(\omega-1)$ が非整数の場合

$R(i)$ リング上にないすべての無故障ノードから $R(i)$ 上の最も近いノードへの距離は高々3であるから，通信アルゴリズムは $\sum_{j=1}^{\gamma} j + \sum_{j=1}^{\gamma}(j+2) + \sum_{j=1}^{\gamma}(j+3) + \sum_{j=1}^{\gamma}(j+4)$ 時間を要する．

4. $2\omega+1 < d(f_1, f_2)$ の場合

$R(i)$ リング上にないすべての無故障ノードから $R(i)$ 上の最も近いノードへの距離は高々1であるから，通信アルゴリズムは $\sum_{j=1}^{\gamma} j + \sum_{j=1}^{\gamma}(j+2)$ 時間を要する．

4.3.4 ハイパーキューブとの比較による性能評価

S.Park と B.Bose は文献 [87] において，n 次元ハイパーキューブにおける全対全ブロードキャストの通信時間が，無故障の場合，故障ノードが1個の場合，故障ノードが f 個の場合，それぞれ $(2^n-1)/n$，$2(2^{n-1}-1)/(n-1)+2$，$2(2^f-1)(f-1)+(2^n-2^{f+1})/(n-f-1)+2$ であることを示している．図 4.20 はコーダルリングとハイパーキューブの通信時間を比較したものである．通信時間と通信リンク数がトレードオフの関係であることが示されている．前述したように，故障のある n 次元ハイパーキューブにおける全対全個別通信の通信時間は，故障ノードが1個の場合，故障ノードが2個の場合，それぞれ $5\cdot 2^{n-1}-2$，$5\cdot 2^{n-1}+n-1$ である．図 4.21 はコーダルリングとハイパーキューブの通信時間を全対全個別通信において比較したものである．

4.4 結言

本章では，まず $\lfloor n/2 \rfloor$ 以下の故障ノードを含む n 次元ハイパーキューブのための，全対全個別通信アルゴリズムを提案した．このアルゴリズムの所要時間は，故障数によらず $5\cdot 2^{n-1}$ とみなすことができる．このアルゴリズムをリンク故障に対して拡張した結果，アルゴリズムの所要時間は，ノードに属する故障リンク数の最大値が μ のとき，$O(\mu\cdot 2^{n-\mu})$ となることがわかった．

次に，故障を含むコーダルリングのための，全対全ブロードキャストおよび個別通信アルゴリズムを提案した．このアルゴリズムは，無故障ノード間の通信ができないような故障を除いて，すなわち2個以下のノード故障に対して耐故障性を持つ．

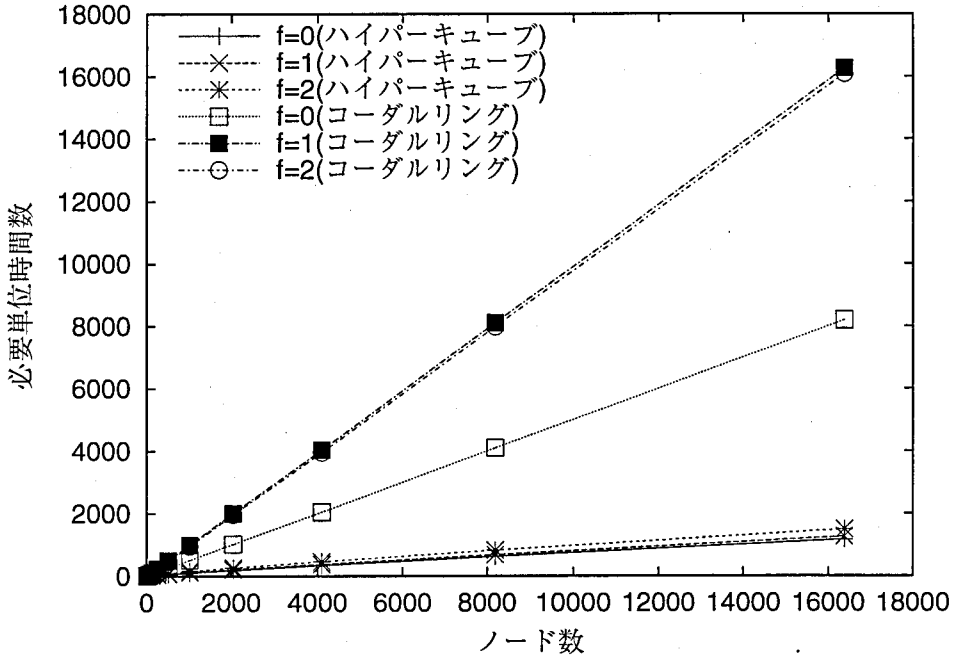


図 4.20: 全対全ブロードキャストの場合のコーダルリングとハイパーキューブにおけるノード数と通信時間の関係

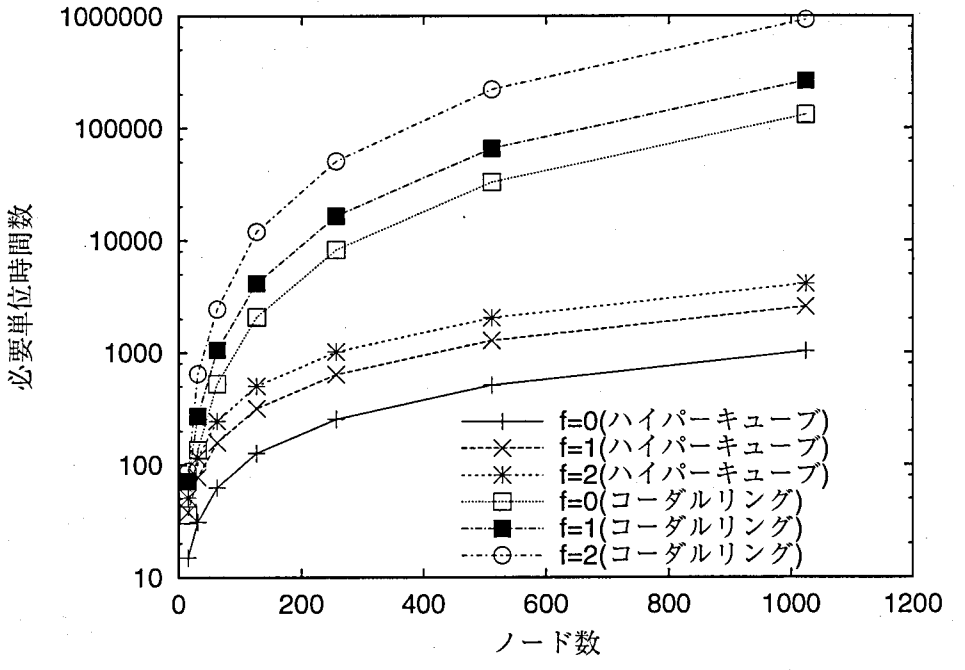


図 4.21: 全対全個別通信の場合のコーダルリングとハイパーキューブにおけるノード数と通信時間の関係

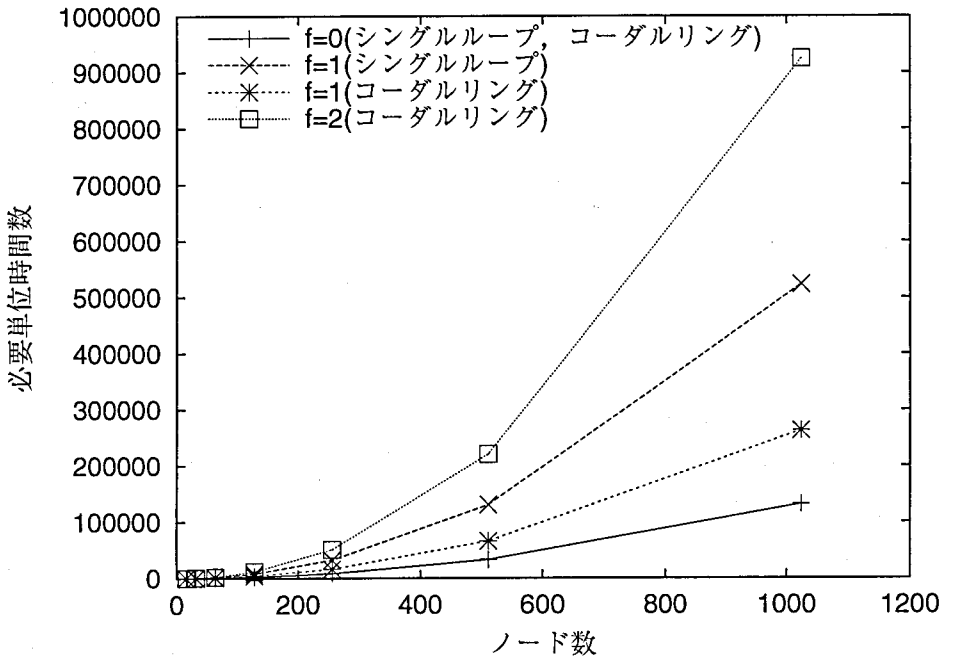


図 4.22: 全対全個別通信の場合のコーダルリングとシングルループにおけるノード数と通信時間の関係

1 個の故障を持つシングルループにおいて，全対全ブロードキャストが $O(n)$ の通信時間を必要とし，全対全個別通信が $O(n^2)$ の通信時間を必要とすることは明かである．コードを追加することで通信時間が改善できるはずである．図 4.22 はシングルループとコーダルリングを比較したものであるが，全対全個別通信において，故障がある場合にコードが有用であることが示されている．今後の課題として，提案するアルゴリズムの性能が改善可能かどうか調べることと，無故障のコーダルリングにおいて，より通信時間が短いアルゴリズムが存在するか否かを検討することが残されている．

第5章　結論

今後，コンピュータシステムはより複雑，高度なものになり，インターネットを代表とするコンピュータネットワークはますます大規模なものになるであろう．そこで扱われる情報量は人間を溺れさせかねないほど膨大なものになってゆくのは間違いない．自律性，協調性，適応性を有するエージェントが人間の代理となって問題解決を行うマルチエージェントシステムは，人間にとって優しい高度情報化社会を実現させる鍵となる技術である．

本論文はマルチエージェントシステムの構築に対して，主として基礎技術の側面からアプローチを試みたものである．

第2章では，マルチモバイルエージェントシステム記述用言語Maglogを提案し，その有用性を簡単な例を通して示した．Maglogはエージェントの移動方式が強マイグレーションであること，ホストをまたいだユニフィケーションやバックトラックが可能であること，移動先のデータおよびプログラムを直接利用できるフィールドという概念を備えていること，エージェントの移動が往復で対になっていることから，エージェントの記述が容易かつ簡潔に行える言語となっている．本論文では実装の基本方針を与えたが，実働する処理系の作成は今後の課題である．

第3章は分散環境におけるマルチエージェントシステムを対象とし，データやプログラムをスーパバイザ・エージェントが集中管理している場合と各エージェントが分散して管理している場合について，データアクセスの効率特性を明らかにした．各エージェントが必要とするデータをどのエージェントが所有するかの確率が均等である場合においても，分散管理方式は集中管理方式と比べて遜色ない効率を持つが，各エージェントが必要とするデータを，そのエージェントに近いエージェントほど高い確率で所有する場合には，分散管理方式は集中管理方式より高い効率を持つことが分かった．また，分散管理方式において，キューを持つ各エージェントが，すべてのデータを等確率で検索する場合におけるデータアクセス効率の時刻変化を解析し，その特性を明確にした．これにより，コストパフォーマンスを考慮したキューの適当な長さはどの程度かということが分かった．さらに，実働しているWWWキャッシュシステムから抽出されたWWWアクセス記録を基に，データ検索効率のシミュレーション実験を行い，データの検索が頻繁に行われるほど分散管理方式が優位に立つことが確かめた．

第4章では，マルチエージェントシステムにおいて耐故障性のあるエージェント

間通信を実現することを目標として問題を一般化し，並列計算機のインターコネクションネットワークにおける耐故障性通信について論じた．まず，故障を含むハイパーキューブ上の全対全個別通信アルゴリズムを提案した．提案したアルゴリズムは多ポート通信モデルに基き，故障位置がすべてのノードにとって既知であることを仮定している．提案したアルゴリズムは全ノード数の半分以下の故障に対する耐故障性を持ち，その実行時間は故障ノード数によらず，ほぼ $5 \cdot 2^{n-1}$ であることを示した．これは，無故障ハイパーキューブにおける全体全個別通信アルゴリズムの最適実行時間の約2.5倍である．次に，コーダルリング上の全対全ブロードキャストおよび個別通信アルゴリズムを提案した．このアルゴリズムは，ノード間通信ができないような致命的な故障を除いて耐故障性を持つ．アルゴリズムの実行時間をシングルループのそれと比較することで，コーダルリングを特徴づけているコードが，故障ノードが存在する場合の通信において有効に使用されることを示した．

謝辞

本研究は神戸大学大学院自然科学研究科情報メディア科学専攻金田悠紀夫教授の長年に亙る懇切な御指導と御鞭撻のもとで行われたものであり，ここに深く感謝の意を表します．

本論文をまとめるにあたり，丁寧かつ適切な御助言を頂きました神戸大学大学院自然科学研究科情報メディア科学専攻瀧和男教授と神戸大学都市安全研究センター上原邦昭教授に感謝申し上げます．

鳥取大学工学部知能情報工学科増山博教授の熱心な御指導，御助力なしには本研究は遂行できませんでした．心より感謝致します．

遅々として進まない研究を忍耐強く見守り，叱咤激励してくださった，鳥取大学名誉教授でもあられる岡山理科大学工学部情報工学科小林康浩教授と故 井上倫夫先生 (当時鳥取大学工学部知能情報工学科助教授) に感謝申し上げます．

最後に，本研究を進めるにあたり，家庭を守り心身両面における支えとなってくれた妻紫乃と，元気な笑顔で力付けてくれた長男新樹に感謝する．

参考文献

[1] 中島秀之: なぜ協調なのか, コンピュータ科学, Vol. 1, No. 3, pp. 160–165 (1991).

[2] Minsky, M.: *The Society of Mind*, Simon & Schuster (1986). 邦訳: 安西 祐一郎訳, 心の社会, 産業図書 (1990).

[3] 所真理雄: マルチエージェントシステム研究の目指すもの, コンピュータソフトウェア, Vol. 12, No. 1, pp. 78–84 (1995).

[4] Hewitt, C.: The Challenge of Open Systems, *Byte*, Vol. 10, No. 4, pp. 223–242 (1085).

[5] 青山幹雄, 中所武司, 向山博: コンポーネントウェア, 共立出版 (1998).

[6] 畑井毅之, 川村尚生, 井上倫夫: コンポーネント指向エージェントアーキテクチャCODA, 第8回マルチ・エージェントと協調計算ワークショップ (1999).

[7] 西田豊明: ソフトウェアエージェント, 人工知能学会誌, Vol. 10, No. 5, pp. 44–51 (1995).

[8] 石田享: エージェントを考える, 人工知能学会誌, Vol. 10, No. 5, pp. 3–7 (1995).

[9] Franklin, S. and Graesser, A.: Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents, *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, published as Intelligent Agents III*, Springer-Verlag, pp. 21–35 (1996).

[10] 本位田真一, 飯島正, 大須賀昭彦: エージェント技術, 共立出版 (1999).

[11] 新谷虎松, 大囿忠親, 福田直樹: モバイルエージェントの動向—マルチエージェントシステムのためのモビリティの利用—, 人工知能学会誌, Vol. 16, No. 4, pp. 488–493 (2001).

[12] Lange, D. B. and Oshima, M.: Seven good reasons for mobile agents, *Communications of the ACM*, Vol. 42, No. 3, pp. 88–89 (1999).

[13] 佐藤一郎: モバイルエージェントの動向, 人工知能学会誌, Vol. 14, No. 4, pp. 22–29 (1999).

[14] Schoder, D. and Eymann, T.: The real challenges of mobile agents, *Communications of the ACM*, Vol. 43, No. 6, pp. 111–112 (2000).

[15] Fuggetta, A., Picco, G. P. and Vigna, G.: Understanding Code Mobility, *IEEE Transactions of Software Engineering*, Vol. 24, No. 5, pp. 342–361 (1998).

[16] Suri, N., Bradshaw, J. M., Breddy, M. R., Groth, P. T., Hill, G. A., Jeffers, R., Mitrovich, T. S., Pouliot, B. R. and Smith, D. S.: NOMADS: toward a strong and safe mobile agent system, *Proceedings of the fourth international conference on Autonomous agents*, pp. 163–164 (2000).

[17] Osuga, A., Nagai, Y., Irie, Y., Hattori, M. and Honiden, S.: Plangent: An Approach to Making Mobile Agents Intelligent, *IEEE Internaet Computing*, Vol. 1, No. 4, pp. 50–57 (1997).

[18] Tarau, P.: Inference and Computation Mobility with Jinni, *The Logic Programming Paradigm: a 25 Year Perspective* (Apt, K., Marek, V. and Truszczynski, M.(eds.)), Springer, pp. 33–48 (1999).

[19] Fukuta, N., Ito, T. and Shintani, T.: MiLog: A Mobile Agent Framework for Implementing Intelligent Information Agents with Logic Programming, *Proceedings of the 1st Pacific Rim International Workshop on Intelligent Information Agents*, pp. 113–123 (2000).

[20] 渡辺慎哉, 原田康徳, 三谷和史, 宮本衛市: 場とイベントによる並列計算モデル — Kamui 88, コンピュータソフトウェア, Vol. 6, No. 1, pp. 41–55 (1989).

[21] 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, pp. 1071–1079 (1990).

[22] 中島秀之, 久野巧, 木下佳樹, 半田剣一, 松原仁, 石川裕, 車谷浩一, 大澤一郎: 協調アーキテクチャ関連研究の現状, 調査報告 221, 電子技術総合研究所 (1991).

[23] Kumeno, F., Ohsuga, A. and Honiden, S.: Flage: A Programming Language for Adaptive Software, *IEICE Transactions of Information & System*, Vol. E81–D, No. 12, pp. 1394–1403 (1998).

[24] Ueda, K. and Chikayama, T.: Design of the Kernel Language for the Parallel Inferece Machine, *The Computer Journal*, Vol. 33, No. 6, pp. 494–500 (1990).

[25] Dijkstra, E. W.: Go To Statement Considered Harmful, *Communications of the ACM*, Vol. 11, No. 3, pp. 147–148 (1968).

[26] 川村尚生, 斎藤善徳, 金田悠紀夫: エージェント間の共有知識を実現したPrologに基づく協調処理言語, 情報処理学会研究報告, 92-PRG-8, pp. 59–65 (1992).

[27] 川村尚生, 斎藤善徳, 金田悠紀夫: エージェント間で知識を共有する協調処理モデル, 神戸大学自然科学研究科紀要, No. 11-B, pp. 121–128 (1993).

[28] 浦田泰裕, 齋田明生, 田村直之, 金田悠紀夫, 川村尚生: 分散環境下におけるマルチエージェントシステム記述用言語, 情報処理学会研究報告, 95-PRO-2, pp. 153–159 (1995).

[29] Kawamura, T., Saito, Y. and Kaneda, Y.: Multi-agent Programming Language based on Distributed Multi-workstation Systems, *Proceedings of IEEE Region 10's Ninth Annual International Conference* (Chan, T. K.(ed.)), Vol. 1, pp. 61–66 (1994).

[30] Kaneda, Y., Tamura, N., Urata, Y., Saita, A. and Kawamura, T.: Multi-agent Programming System on a Workstation Network, *Proceedings of IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, pp. 315–318 (1995).

[31] 坂本浩二, 金田悠紀夫, 川村尚生: モバイルエージェントシステム記述用言語とその実装方式について, 第8回マルチ・エージェントと協調計算ワークショップ (1999).

[32] 川村尚生, 半場寛之, 金田悠紀夫: 適応型モバイルエージェントシステムMaglog, ソフトウェアエージェントとその応用特集ワークショップ (SAA2000) 講演論文集, pp. 117–124 (2000).

[33] 川村尚生, 半場寛之, 金田悠紀夫: マルチモバイルエージェントシステム記述用言語Maglog, 情報処理学会研究報告 (2001).

[34] Aridor, Y. and Oshima, M.: Infrastructure for Mobile Agents: Requirements and Design, *Proceedings of the Second International Workshop on Mobile Agents*, Vol. 1477, Springer-Verlag, pp. 38–49 (1998).

[35] Warren, D.H.D: An abstract Prolog instruction set, Technical Note 309, SRI International (1983).

[36] Ait-Kaci, H.: *Warren's Abstract Machine*, MIT Press (1991).

[37] Banbara, M. and Tamura, N.: Translating a linear logic programming language onto Java, *Proceedings of ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, pp. 19–39 (1999).

[38] 番原睦則, 姜京順, 田村直之: 線形論理型言語の Java 言語による処理系の設計と実装, 情報処理学会論文誌, Vol. 40, No. SIG 10(PRO 5), pp. 1–16 (1999).

[39] Lange, D. B. and Oshima, M.: *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley (1998).

[40] IBM Tokyo Research Laboratory: Aglets Software Development Kit. http://www.trl.ibm.co.jp/aglets/.

[41] Wong, D., Paciorek, N., Walsh, T. and Dicelie, J.: Concordia: An Infrastructure for Collaborating Mobile Agents, *Proceedings of the First International Workshop on Mobile Agents*, Vol. 1219, Springer-Verlag, pp. 86–97 (1997).

[42] Mitsubishi Electric Information Technology Center: Concordia. http://www.metica.com/HSL/Projects/Concordia/.

[43] Satoh, I.: MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System, *Proceedings of IEEE International Conference on Distributed Computing Systems*, IEEE Press, pp. 161–168 (2000).

[44] ObjectSpace Inc.: Voyager. http://www.objectspace.com/voyager/prodVoyager.asp.

[45] 川村隆治, 田原康之, 長谷川哲夫, 大須賀昭彦, 本位田真一: Bee-gent: 移動型仲介エージェントによる既存システムの柔軟な活用を目的としたマルチエージェントフレームワーク, 電子情報通信学会論文誌 D-I, Vol. J82-D-I, No. 9, pp. 1165–1180 (1999).

[46] Winikoff, M.: W-Prolog. http://www.cs.mu.oz.au/ winikoff/wp/.

[47] IF Comuter Corp.: MINERVA. http://www.ifcomputer.com/MINERVA/.

[48] Demoen, B. and Tarau, P.: jProlog. http://www.cs.kuleuven.ac.be/ bmd/PrologInJava/.

[49] Information-technology Promotion Agency, Japan: Flage Home Page. http://www.ipa.go.jp/NEWSOFT/public/Flage/.

[50] Tarau, P.: Jinni: Intelligent Mobile Agent Programming at the Intersection of Java and Prolog (1999). http://www.cs.unt.edu/ tarau/research/JinniPapers/paam99.html.

[51] BinNet Corporation: Jinni. http://www.binnetcorp.com/Jinni/.

[52] Fukuta, N., Ito, T., Ozono, T. and Shintani, T.: A Framework for Cooperative Mobile Agents and Its Case-Study on *BiddingBot*, *Proceedings of the JSAI 2001 International Workshop on Agent-based Approaches in Economic and Social Complex Systems*, pp. 91–98 (2001).

[53] 本位田真一, 飯島正, 大須賀昭彦: エージェント技術, 共立出版, pp. 15–16 (1999).

[54] Perkins, C. E. and Bhagwat, P.: Highly Dynamic Destination-Sequenced Distance-Vector Routing(DSDV), *SIGCOMM'94* (1994).

[55] 高木康志, 堀口進, 川添良幸, 重井芳治: タスク型マルチプロセッサのシステム性能評価, 電子情報通信学会論文誌, Vol. J72-D-1, No. 5, pp. 357–366 (1989).

[56] Mohammed, F. A., Ayad, N. M. A., Esmat, H., Omar, A. A. and Ghonaimy, M. A. R.: Performance Analysis of Access Media Protocols in Local Networks, *Proceedings of the Second IEEE Symposium on Computers & Communications*, pp. 249–258 (1997).

[57] El-Hadidi, M. T. and Hegazi, N. H.: Performance Analysis of the Kerberos Protocol in a Distributed Environment, *Proceedings of the Second IEEE Symposium on Computers & Communications*, pp. 235–248 (1997).

[58] 川村尚生, 増山博: 階層型ネットワーク上での分散データベースの性能に関する一考察, 情報処理学会論文誌, Vol. 40, No. 9, pp. 3620–3623 (1999).

[59] Kawamura, T. and Masuyama, H.: Performance Analysis of a Distributed Database in Multi-Agent Architecture, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, pp. 167–171 (1999). Boston, Massachusetts, USA.

[60] Kawamura, T., Inoue, M. and Masuyama, H.: Optimal Distribution of Database in Hierarchical Networks, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems* (Guizani, M. and Shen, X.(eds.)), Vol. 2, pp. 536–543 (2000). Las Vegas, Nevada, USA.

[61] 川村尚生, 井上倫夫, 増山博: 階層型ネットワーク上での分散データベースの性能評価, 日本応用数理学会論文誌, Vol. 11, No. 1, pp. 15–26 (2001).

[62] Shim, J., Scheuermann, P. and Vingralek, R.: Proxy Cache Design: Algorithms, Implementation and Performance, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 549–562 (1999).

[63] Wessels, D. and Claffy, K.: Internet Cache Protocol(ICP) version 2, Internet Draft(RFC2186) (1997).

[64] Smith, R. G.: Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers*, Vol. C-29, No. 12, pp. 1104–1113 (1981).

[65] Erman, L. D., Hayes-Roth, F., Resser, V. and Reddy, D.: Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty, *ACM Computing Surveys*, Vol. 21, No. 3, pp. 323–357 (1989).

[66] Cohen, P. R., Cheyer, A., Wang, M. and Baeg, S. C.: An Open Agent Architecture, *Proceedings of the Spring Symposium on Software Agents*, pp. 1–8 (1994).

[67] Carriero, N. and Gelernter, D.: Linda in Context, *CACM*, Vol. 32, No. 4, pp. 444–458 (1989).

[68] Carriero, N. and Gelernter, D.: How to write Parallel Programs – A Guide to the Perplexed, *ACM Computing Surveys*, Vol. 21, No. 3, pp. 323–357 (1991).

[69] Nassimi, D. and Sahni, S.: An Optimal Routing Algorithm for Mesh-connected Computers, *Journal of the Association for Computing Machinery*, Vol. 27, No. 1, pp. 6–29 (1980).

[70] Saad, Y. and Schultz, M. H.: Topological Properties of Hypercubes, *IEEE Transactions on Computer*, Vol. 37, No. 6, pp. 867–872 (1988).

[71] Tyszer, J.: A Multiple Fault-Tolerant Processor Network Architecture for Pipeline Computing, *IEEE Transactions on Computer*, Vol. 37, No. 11, pp. 1414–1418 (1988).

[72] Masuyama, H., Ichimori, T. and Ishizuka, O.: The Optimum Design Method of Reliable Networks, *IEICE Transactions*, Vol. E-71, No. 12, pp. 1723–1281 (1988).

[73] Varuma, A. and Raghavendra, C. S.: Fault-Tolerant Routing in Multistage Interconnection Networks, *IEEE Transactions on Computer*, Vol. 38, No. 3, pp. 385–393 (1989).

[74] Masuyama, H., Kawamura, T. and Masuyama, E.: An All-to-All Personalized Communication Algorithm for Faulty Hypercubes, *Proceedings of Internaitional Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pp. 89–96 (2001). Orlando, Florida, USA.

[75] Kawamura, T., Masuyama, H. and Masuyama, E.: All-to-All Personalized Communication in Hypercubes with Link Faults, *Proceedings of International Conference on Software, Telecommunications and Computer Networks*, Vol. 2, pp. 791–798 (2001). Split-Dubrovnik, Croatia, Ancona-Bari, Italy.

[76] Masuyama, H., Kawamura, T. and Miyoshi, T.: All-to-All Broadcasting and Personalized Communication Algorithms in Chordal Ring Networks, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems* (Gonzalez, T.(ed.)), pp. 234–241 (2001). Anaheim, California, USA.

[77] Ramanathan, P. and Shin, K.: Reliable Broadcast in Hypercube Multicomputers, *IEEE Transactions on Computer*, Vol. 37, No. 12, pp. 1654–1657 (1988).

[78] Raghavendra, C. S. and Sridhar, M. A.: Broadcasting Algorithms in Faulty SIMD Hypercubes, *Proceedings of Fourth IEEE Symposium on Parallel and Distributed Processing*, pp. 4–11 (1992).

[79] Bertsekas, D. P., Ozveren, C., Stamoulis, G. D., Tseng, P. and Tsitsiklis, J. N.: Optimal Communication Algorithms for Hypercubes, *Journal of Parallel and Distributed Computing*, Vol. 11, pp. 263–275 (1991).

[80] Chen, M. S. and Shin, K.: Adaptive Fault-Tolerant Routing in Hypercube Multicomputers, *IEEE Transactions on Computer*, Vol. 39, No. 12, pp. 1406–1416 (1990).

[81] Fraigniaud, P.: Asymptotically Optimal Broadcasting and Gossiping in Faulty Hypercube Multicomputers, *IEEE Transactions on Computer*, Vol. 41, No. 11, pp. 1410–1419 (1992).

[82] Johnsson, S. L. and Ho, C. T.: Optimum Broadcasting and Personalized Communication in Hypercubes, *IEEE Transactions on Computers*, Vol. 38, No. 9, pp. 1249–1268 (1989).

[83] Duzett, B. and Buck, R.: An Overview of the nCUBE3 Supercomputer, *Proceedings of Fourth Symposium on Frontiers of Massively Parallel Computation*, pp. 458–464 (1992).

[84] Gaughan, P. T. and Yalamanchili, S.: Adaptive Routing Protocols for Hypercube Interconnection Networks, *Computer*, Vol. 26, No. 5, pp. 12–23 (1993).

[85] Wu, J. and Fernandez, E. B.: Reliable Broadcasting in Faulty Hypercube Computers, *Proceedings of 11th Symposium on Reliable Distributed Systems*, pp. 122–129 (1992).

[86] Tien, S.-B. and Raghavendra, C. S.: Algorithms and Bounds for Shortest Paths and Diameter in Faulty Hypercubes, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 6, pp. 713–718 (1993).

[87] Park, S. and Bose, B.: All to All Broadcasting in Faulty Hypercubes, *IEEE Transactions on Computers*, Vol. 46, No. 7, pp. 749–755 (1997).

[88] Pradhan, D. K. and Reddy, S. M.: A Fault-Tolerant Communication Architecture for Distributed Systems, *IEEE Transactions on Computer*, Vol. C-31, No. 10, pp. 863–870 (1982).

[89] Arden, B. W. and Lee, H.: Analysis of Chordal Ring Network, *IEEE Transactions on Computer*, Vol. C-30, No. 4, pp. 291–295 (1981).

[90] Feng, C., Bhuyan, L. N. and Lombardi, F.: An Adaptive System-Level Diagnosis Approach for Hypercube Multiprocessors, *Proceedings of the fifth IEEE symposium on Parallel and Distributed Processing*, pp. 460–467 (1993).

[91] Masuyama, H. and Yoden, S.: An Adaptive System-Level Diagnosis for Chordal Ring Networks, *Proceedings of 5th IEEE International Conference on Computer Communications and Networks*, pp. 48–54 (1996).

[92] Kobayashi, H. and Koga, Y.: Sorting on a Chordal-Ring-Connected Parallel Computer, *IECE Transactions*, Vol. J68-D, No. 3, pp. 253–260 (1985).

[93] Masuyama, H.: Algorithms to Realize on Arbitrary BPC Permutation in Chordal Ring Networks and Mesh Connected Networks, *IEICE Transactions on Information and Systems*, Vol. E77-d, No. 10, pp. 1118–1129 (1994).

[94] Masuyama, H. and Masuyama, E.: Algorithms to Realize on Arbitrary BPC Permutation in Chordal Ring Networks with Failures, *International Journal of Reliability, Quality and Safety Engineering*, Vol. 1, No. 2, pp. 310–314 (1994).

[95] Scott, D. S.: Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies, pp. 398–403 (1991).

[96] Petrini, F.: Total-Exchange on Wormhole k-ary n-cubes with Adaptive Routing, *Proceedings of the First Merged IEEE International Parallel Processing Symposium & Symposium on Parallel and Distributed Processing*, pp. 267–271 (1998).

[97] Thakur, R. and Choudhary, A.: All-to-All Communication on Meshes with Wormhole Routing, pp. 561–565 (1994).

[98] Suh, Y. J. and Yalmanchili, S.: All-to-All Communication with Minimum Start-up Costs in 2D/3D Tori and Meshes, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 5, pp. 442–45i (1998).

[99] Yang, Y. and Wang, J.: Efficient All-to-All Broadcast in All-Port Mesh and Torus Networks, pp. 290–299 (1999).

[100] Tseng, Y.-C. and Gupta, S.: All-to-All Personalized Communication in a Wormhole-Routed Torus, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 498–505 (1996).

[101] Tseng, Y.-C., Lin, T.-H., Gupta, S. and Panda, D. K.: Bandwidth-Optimal Complete Exchange on Wormhole Routed 2D/3D Torus Network: A Diagonal-Propagation Approach, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 4, pp. 380–396 (1997).

[102] Suh, Y. J. and Shin, K. G.: Efficient All-to-All Personalized Exchange in Multidimensional Torus Networks, pp. 468–475 (1998).

[103] Yang, Y. and Wang, J.: All-to-All Personalized Exchange in Banyan Networks, pp. 78–86 (1999).

[104] Masuyama, H. and Miki, T.: Reorganization of Node Connection in Chordal-Ring Networks with Variation of the Number of Nodes, *IEICE Transactions*, Vol. J-72-A, No. 5, pp. 795–807 (1989).

付 録A　Maglogのプログラム例

A.1　スケジューリングのプログラム

```
% エージェントのプログラム
% schedulingという名前のファイルに格納されている
main(Day, Hour) :-
  Member = [person(fieldA, hostA),
            person(fieldB, hostB),
            person(fieldC, hostC)],
  search(Day, Hour, Member),
  book(Day, Hour, meeting_1, Member),!.

search(Day, Hour, []) :-!.
search(Day, Hour, [person(Field, Host)|Rest]) :-
  (free(Day, Hour) within 300 in Field import [free/2],
   search(Day, Hour, Rest)) on Host.

book(Day, Hour, App, []) :-!.
book(Day, Hour, App, [person(Field, Host)|Rest]) :-
  ((retract(free(Day, Hour)),
    assert(schedule(Day, Hour, App)) within 300
   ) in Field import [free/2, schedule/3],
   book(Day, Hour, App, Rest)) on Host.

% HostAのfieldAフィールドのプログラム

:- dynamic free/2, schedule/3.

free(1,9).
free(1,10).
free(1,11).
```

```
free(1,18).
free(2,11).
free(2,14).

% HostBのfieldBフィールドのプログラム

:- dynamic free/2, schedule/3.

free(1,10).
free(1,11).
free(1,14).
free(2,9).
free(2,14).
free(2,15).
free(2,16).

% HostCのfieldCフィールドのプログラム

:- dynamic free/2, schedule/3.

free(1,11).
free(2,10).
free(2,14).
free(2,16).
```

A.2 買物のプログラム

```
% 親エージェントのプログラム
main:-
  create_children([hostA, hostB, hostC], Agents),
  collect_prices(Agents, Prices),
  buy(Prices), !.

create_children([], []):- !.
create_children([H|Hrest], [Id|Arest]):-
  create(Id, child, main(parent, Id, H, computer)),
```

```
  create_children(Hrest, Arest).

collect_prices([], []):- !.
collect_prices([A|Arest], [A/P|Prest]):-
  recv(A, P) within 300,
  collect_prices(Arest, Prest).

buy(Prices):-
  find_floor_price(Prices, Buyer),
  order(Prices, Buyer).

find_floor_price([Buyer], Buyer):- !.
find_floor_price([A,B|Rest], Buyer):-
  find_floor_price([B|Rest], X),
  min(A, X, Buyer).

min(A/P, B/Q, A/P):- P =< Q, !.
min(A/P, B/Q, B/Q).

order([], _) :-!.
order([Buyer/Price|Prest], Buyer/Price):-
  send(Buyer, buy) within 300, !,
  order(Prest, Buyer/Price).
order([A/_|Prest], Buyer/Price):-
  send(A, notbuy) within 300, !,
  order(Prest, Buyer/Price).

% 子エージェントのプログラム
main(Parent, Id, Host, Goods):-
  (get_price(Goods, Price),
   send(Parent, Price, 300),
   recv(Parent, Direction) within 300,
   Direction = notbuy;
   retract(stock(Goods, N) within 300),
   N1 is N-1,
   assert(stock(Goods, N1)),
```

```
  ) in market import [get_price/2,stock/2] on Host.

% HostA の market フィールドのプログラム
:- public get_price/2.
:- dynamic stock/2.

get_price(Goods, Price) :-
  price(Goods, Price).

price(computer, 1000).
stock(computer, 10).

% HostB の market フィールドのプログラム
:- public get_price/2.
:- dynamic stock/2.

get_price(Goods, Price) :-
  price(Goods, Price).

price(computer, 500).
stock(computer, 5).

% HostC の market フィールドのプログラム
:- public get_price/2.
:- dynamic stock/2.

get_price(Goods, Price) :-
  price(Goods, Price).

price(computer, 800).
stock(computer, 20).
```

研究業績

学術論文 (査読つき)

1. 川村尚生, 斎藤善徳, 金田悠紀夫: エージェント間で知識を共有する協調処理モデル, 神戸大学自然科学研究科紀要, No. 11-B, pp. 121–128 (1993).

2. 井上倫夫, 介中敦子, 川村尚生, 小林康浩, 中島健二: 事象関連脳電位 (ERP) を用いて ALS 患者とコミュニケーション, コンピュータ & エデュケーション, Vol. 2, pp. 79–84 (1997).

3. 川村尚生, 増山博: 階層型ネットワーク上での分散データベースの性能に関する一考察, 情報処理学会論文誌, Vol. 40, No. 9, pp. 3620–3623 (1999).

4. 川村尚生, 井上倫夫, 増山博: 階層型ネットワーク上での分散データベースの性能評価, 日本応用数理学会論文誌, Vol. 11, No. 1, pp. 15–26 (2001).

国際会議 (査読つき)

1. Kawamura, T., Saito, Y. and Kaneda, Y.: Multi-agent Programming Language based on Distributed Multi-workstation Systems, *Proceedings of IEEE Region 10's Ninth Annual International Conference* (Chan, T. K.(ed.)), Vol. 1, pp. 61–66 (1994).

2. Kaneda, Y., Tamura, N., Urata, Y., Saita, A. and Kawamura, T.: Multi-agent Programming System on a Workstation Network, *Proceedings of IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, pp. 315–318 (1995).

3. Kawamura, T. and Masuyama, H.: Performance Analysis of a Distributed Database in Multi-Agent Architecture, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, pp. 167–171 (1999). Boston, Massachusetts, USA.

4. Kawamura, T., Inoue, M. and Masuyama, H.: Optimal Distribution of Database in Hierarchical Networks, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems* (Guizani, M. and Shen, X.(eds.)), Vol. 2, pp. 536–543 (2000). Las Vegas, Nevada, USA.

5. Masuyama, H., Kawamura, T. and Masuyama, E.: An All-to-All Personalized Communication Algorithm for Faulty Hypercubes, *Proceedings of Internaitional Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pp. 89–96 (2001). Orlando, Florida, USA.

6. Masuyama, H., Kawamura, T. and Miyoshi, T.: All-to-All Broadcasting and Personalized Communication Algorithms in Chordal Ring Networks, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems* (Gonzalez, T.(ed.)), pp. 234–241 (2001). Anaheim, California, USA.

7. Kawamura, T., Masuyama, H. and Masuyama, E.: All-to-All Personalized Communication in Hypercubes with Link Faults, *Proceedings of International Conference on Software, Telecommunications and Computer Networks*, Vol. 2, pp. 791–798 (2001). Split-Dubrovnik, Croatia, Ancona-Bari, Italy.

国内会議 (査読つき)

1. 本元隆義, 川村尚生: 移動エージェント保護のためのモジュール別暗号化, 第8回マルチ・エージェントと協調計算ワークショップ (1999).

2. 畑井毅之, 川村尚生, 井上倫夫: コンポーネント指向エージェントアーキテクチャCODA, 第8回マルチ・エージェントと協調計算ワークショップ (1999).

3. 坂本浩二, 金田悠紀夫, 川村尚生: モバイルエージェントシステム記述用言語とその実装方式について, 第8回マルチ・エージェントと協調計算ワークショップ (1999).

4. 川村尚生, 半場寛之, 金田悠紀夫: 適応型モバイルエージェントシステムMaglog, ソフトウェアエージェントとその応用特集ワークショップ (SAA2000) 講演論文集, pp. 117–124 (2000).

学術報告

1. 川村尚生, 斎藤善徳, 金田悠紀夫: エージェント間の共有知識を実現したPrologに基づく協調処理言語, 情報処理学会研究報告, 92-PRG-8, pp. 59–65 (1992).
2. 浦田泰裕, 齋田明生, 田村直之, 金田悠紀夫, 川村尚生: 分散環境下におけるマルチエージェントシステム記述用言語, 情報処理学会研究報告, 95-PRO-2, pp. 153–159 (1995).
3. 古城明宏, 川村尚生, 井上倫夫, 小林康浩, 川上孝志, 中島健二: ハール・ウェーブレット変換による事象関連脳電位 (ERP) の処理, 情報処理学会研究報告, HI-62-1, Vol. 95, No. 88, pp. 1–8 (1995).
4. 荒川修, 川村尚生, 井上倫夫, 小林康浩: 共有メモリ型並列計算機アーキテクチャWOSMに関する考察, 情報処理学会研究報告, 96-ARC-116, Vol. 96, No. 13, pp. 13–18 (1996).
5. 古石憲男, 河口秀敏, 川村尚生, 井上倫夫, 小林康浩: フレーム単位でデータ転送を行う並列画像処理システムの設計, 情報処理学会研究報告, 96-ARC-116, Vol. 96, No. 13, pp. 37–42 (1996).
6. 介中敦子, 川村尚生, 井上倫夫, 小林康浩: 事象関連脳電位 (ERP) を用いたコミュニケーションのための被選択項目の提示, 情報処理学会研究報告, 96-HI-68, Vol. 96, No. 85, pp. 17–24 (1996).
7. 加納尚之, 川村尚生, 井上倫夫, 中島健二, 介中敦子: 事象関連脳電位 (ERP) を用いたコミュニケーションエイド, 電子情報通信学会技術報告, HCS97-15, Vol. 15, No. 9, pp. 57–60 (1997).
8. Inoue, M., Kawamura, T., Taniguti, H., Takasugi, A. and Nakashima, K.: An Aid to Communicate for ALS Patients Based on Event Related Brain Potentials, *Proceedings of Rehabilitation Engineering Society of Japan*, pp. 343–348 (1998).
9. 川村尚生, 半場寛之, 金田悠紀夫: マルチモバイルエージェントシステム記述用言語Maglog, 情報処理学会研究報告 (2001).

学術講演

1. 本元隆義, 川村尚生, 井上倫夫: モーバイルエージェントとオントロジーによる仲介, 第7回マルチ・エージェントと協調計算ワークショップ (1998).

2. 高杉章, 川村尚生, 井上倫夫, 増山博: マルチエージェントシステムにおける情報検索のための分散情報管理方式の検討, 第7回マルチ・エージェントと協調計算ワークショップ (1998).

3. 高杉章, 川村尚生, 井上倫夫: マルチエージェントモデルによるWWWのための分散キャッシュシステム, 第7回マルチ・エージェントと協調計算ワークショップ (1998).

4. 高杉章, 川村尚生, 井上倫夫: 推論機構と知識ベースから動的に構成されるエージェントモデルの提案, 電気・情報関連学会中国支部第50回連合大会講演論文集, pp. 315–316 (1999).

5. 本元隆義, 川村尚生, 井上倫夫: 移動エージェント保護のための部分的暗号化, 電気・情報関連学会中国支部第50回連合大会講演論文集, pp. 318–319 (1999).

6. 菊澤隆司, 川村尚生, 井上倫夫: 聴覚刺激によるP300を用いた項目選択, 電気・情報関連学会中国支部第51回連合大会講演論文集, p. 477 (2000).

7. 藤原賢志, 川村尚生, 井上倫夫, 山田強: 虹彩中心推定による注視点の検出, 電気・情報関連学会中国支部第51回連合大会講演論文集, p. 478 (2000).

8. 守谷有司, 川村尚生, 加納尚之, 山田強: Hough変換による注視点検出方法, 電気・情報関連学会中国支部第52回連合大会講演論文集, pp. 438–439 (2001).

9. 谷口勝則, 川村尚生, 加納尚之: 視覚刺激と聴覚刺激を同時に与えることによるP300を用いた項目選択, 電気・情報関連学会中国支部第52回連合大会講演論文集, p. 440 (2001).