# データマイグレーションとメソッドマイグレーションの効率的な組合せ

吉田裕介[†] 有次正義[†] 金森吉成[†]

ネットワークでつながれた多数の高性能なワークステーションやパーソナルコンピュータから構成される計算機環境が一般的になっている．分散環境における永続オブジェクトへのメソッドの適用には大きく2つの手法が考えられる．すなわち，永続オブジェクトをメソッドのあるサイトへ移動させて適用するデータマイグレーションと，メソッドを永続オブジェクトの存在するサイトへ移動させて適用するメソッドマイグレーションである．マルチサーバ・マルチクライアントの分散環境では，これら2つの手法をうまく組み合わせることにより効率化を図ることが重要である．本稿では，データマイグレーションとメソッドマイグレーションの処理効率を解析し，それを基に最も効率の良い組合せを求める手法を提案する．また，実験から得られた組合せと比較して提案する手法の有効性を検証する．本稿で提案する方法により求めた組合せは，多くの場合正確に最も効率の良い組合せを求めることができることを示す．

# Efficient Combination of Data-migration and Method-migration

Yusuke Yoshida,[†] Masayoshi Aritsugi[†]
and Yoshinari Kanamori[†]

We have been able to make use of a number of powerful workstations and PCs which are connected to networks. In such an environment, there are two ways of applying methods to persistent objects: one is to move persistent objects to sites that hold methods, and the other is to move methods to sites that store persistent objects. They are called data-migration and method-migration, respectively. It is important to decide how we should combine the ways in order to obtain better performance when having multiple servers and multiple clients. In this paper, we analyze the costs of combinations of the two ways so that optimal combination can be calculated. Furthermore, we discuss experimental results in such a distributed environment.

## 1. はじめに

高性能なワークステーションやパーソナルコンピュータをネットワークで接続した分散環境が一般的になっている．分散環境に対応したオブジェクトデータベースシステムで，メソッドを永続オブジェクトに適用するには，大きく次の2つの手法が考えられる．

- 永続オブジェクトをメソッドのあるサイトへ移動させて適用する．
- メソッドを永続オブジェクトのあるサイトへ移動させて適用する．

本稿では，この2つの手法をそれぞれデータマイグレーション，メソッドマイグレーションによる処理と呼ぶことにする．

† 群馬大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Gunma University

従来の分散オブジェクトデータベースシステムは，主にデータマイグレーションによる処理手法を実装している[3),13),16)]．データベース分散処理の効率化のためには，データマイグレーションだけではなくメソッドマイグレーションによる処理を組み合わせることが有効であると報告されている[7),8)]．特に，マルチサーバ・マルチクライアントの環境では，データマイグレーションとメソッドマイグレーションの組合せが複数考えられる．サーバが複数あるときは，サーバごとに独立に処理できること，およびサーバにかかる負荷を考慮する必要がある．また，サーバが $n$ 台あるときは，組合せの総数は $2^n$ となるため，これらの組合せの中から最も効率の良い組合せを求めることが重要である．本稿では，レスポンスタイムを最小にする組合せを最も効率の良い組合せと呼ぶ．

メソッドマイグレーションを実現する方法を考えるとき，メソッドを実行するすべてのサイトに，メソッ

ド自身をあらかじめ分散配置する必要がある環境は，分散配置を必要としない環境と比べて，すべてのサイトのメソッドを同一に保つための管理の手間がかかる．Rodríguez-Martínez らは文献 17) で「データを扱うサイト数が増加した場合，メソッドを適切に分散配置する管理の手間が増加することから，あらかじめメソッドを分散配置することは非実用的である」と述べている．同様に，我々はメソッドをデータベースで管理し，処理の際にデータベースから動的にメソッドをロードして実行することにより，メソッドをあらかじめ分散配置する必要がない環境を実現している[1),2)]．

本稿では，データマイグレーションとメソッドマイグレーションの最も効率の良い組合せを求める方法について議論する．具体的には，レスポンスタイムを見積もるコストモデル，およびそのコストモデルを基に最も効率の良い組合せを決定するアルゴリズムを提案する．また，実際にマルチサーバ・マルチクライアントの分散環境ですべての組合せについて実験を行い，本稿で提案する方法を検証する．検証の結果から，多くの場合に正確に最も効率の良い組合せを求められることを示す．

複数の組合せの中から最も効率の良い組合せを選択することは，問合せ最適化を構成する一部である．オブジェクトデータベースの問合せ最適化に関する研究では，問合せやメソッドが内包する論理に基づく方法を用いることが多い[6),7)]．これに対して本稿では，問合せやメソッドの論理を分解せずに効率化を図る方法を提案する．たとえば，ストアドプロシージャを使った問合せについて考える．ストアドプロシージャの結果があらかじめ計算できる場合は，問合せを分解する従来の最適化手法を適用できると報告されている[8),12)]．そうでない場合は，問合せを分解せずに効率化を図る方法が必要となる．また，オブジェクトデータベースの場合は，メソッドの実行時間も考慮にいれた最適化が必要である[6)]．ただし，本稿では従来の最適化手法に代わるものを提案するのではなく，むしろ従来の手法と本稿の提案する手法とを組み合わせることにより，さらに効率化を実現できると考えている．

本稿の構成は，以下のとおりである．2 章で関連研究と比較し，3 章で実験に用いた分散データベースシステムの概要を述べる．次に，4 章でメソッドマイグレーションとデータマイグレーションのコストについて考察する．5 章で我々の行った実験を報告する．6 章で本稿のまとめを述べる．

## 2. 関連研究

クライアント・サーバにおけるデータベース処理の効率化は重要な課題である．Franklin らが指摘しているように，関係データベースの多くはメソッドマイグレーションによる処理を，オブジェクトデータベースの多くはデータマイグレーションによる処理をそれぞれサポートしてきたが，処理の効率化のためにそれらを組み合わせることはこれまであまり考えられていない[7)]．本稿では，1) メソッドマイグレーションによる手法を実装し，2) マルチサーバ・マルチクライアントの環境で，3) サーバサイトの負荷を考慮し，4) データマイグレーションとメソッドマイグレーションの組合せを用いて効率化を実現した．従来，1) あるいは 4) を実現した研究は存在するが[7),8),17)]，1)〜4) をすべて考えた研究は見られない．

Franklin らは，分散環境における処理対象のデータの移動（data shipping）と演算処理の移動（query shipping），およびそれらの組合せ（hybrid shipping）による最適化手法を提案している[7)]．Franklin らは，オブジェクトデータベースへのアクセスを Selection などの単項演算，および二項演算の Join に分解し，提案している 3 種類の手法を用いて生成した実行プランに対して，従来の問合せ最適化の技術を応用している．これに対して我々は，問合せやメソッドの論理を分解せずにデータマイグレーションとメソッドマイグレーションを組み合わせて効率化を図る手法について考察する．

Rodríguez-Martínez らは，クライアント・サーバのデータベースシステムでメソッドマイグレーションとデータマイグレーションのトレードオフについて議論している[17)]．Rodríguez-Martínez らは data shipping と code shipping を選択する方法を提案している．メソッドの実行結果のデータ量と元のデータ量との割合（Volume Reduction Factor）を指標として，data shipping/code shipping のどちらかを選択している．Rodríguez-Martínez らの研究と本稿は以下の点で異なる．想定している環境がシングルサーバ・シングルクライアントであることから，複数のクライアントによるサーバサイトの負荷について考慮されていない．メソッドマイグレーションでは，サーバサイトでメソッドの実行が行われるため，クライアントが増加したときのサーバサイトへの影響の解析が重要になるが，文献 17) では，これに関する考察は見られない．本稿では，クライアントの増加をサーバサイトの負荷の増加と見なし，クライアントの増加による影響を解

析している．

Goel らは，クライアント・サーバのデータベースシステムにおいて，データマイグレーションとメソッドマイグレーションの効率について考察している[8]．Goel らは，サーバとクライアント各 1 台を用いて，3 つのネットワーク環境（LAN，MAN，WAN），および異なるオブジェクトのサイズで実験を行っている．その実験結果から，オブジェクトのサイズが大きい，すなわちデータ転送量が大きい場合，およびネットワークのバンド幅が小さい場合はメソッドマイグレーションが効率が良いと結論づけている．本稿で述べる実験でも，データ転送量について同様の結論を得ている．また，本稿のコストモデルもネットワークバンド幅が小さい環境ではメソッドマイグレーションが優位であることを示している．本研究との主な違いは，我々はマルチサーバ・マルチクライアントの分散環境を対象にしている点と，その環境で考えられるデータマイグレーションとメソッドマイグレーションの組合せの中から最も効率の良い組合せを求めるコストモデルとアルゴリズムを提案している点である．

原らは，データベース移動に基づく分散データベース処理について議論している[10),11)]．原らは，ATM 等の帯域幅の大きなネットワーク環境では転送データ量の削減よりも帯域幅の有効利用が重要であるとし，従来のメッセージのやりとりによるデータベース処理に加え，データベース移動による処理を提案している．また，通信量の見積りとトランザクションのアクセスパターンを考慮し，メッセージのやりとりによる処理とデータベース移動による処理を適応的に選択する方法を提案している．しかし，原らが想定しているトランザクション発生サイトは 1 つであり，我々が考えている複数のクライアントが存在する環境とは異なる．

我々はこれまでにメソッドマイグレーションとデータマイグレーションの効率的な組合せを求めるコストモデルとアルゴリズムを提案してきた[21)]．文献 21) と本稿は以下の点で異なる．文献 21) では M/M/1 待ち行列を用いてサーバの負荷を表現した．本稿では，サーバの CPU，ディスク I/O の処理能力が使用されている割合でサーバの負荷を表現した．文献 21) の実験では，`java.lang.Thread.sleep()` を使って，Thread を指定した時間だけ停止させ，負荷による影響を表現した．本稿の実験では，文献 19) で述べられている方法を参考に，サーバで実際に CPU，ディスク I/O を使用するプログラムを走行させ，サーバの負荷による影響を表現した．これにより，文献 21) に比べて，より実際の環境を考慮している．

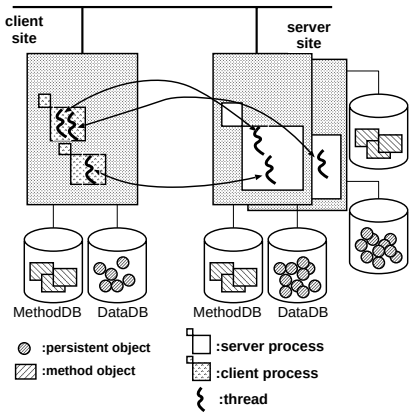


**図 1** 分散メソッドのプロセス構造
Fig. 1 Process structure for distributed methods.

## 3. 分散データベースシステムの環境

1 章で述べたように，多くのクライアント・サーバアーキテクチャのオブジェクトデータベースシステムはデータマイグレーションを用いる．ここでは，オブジェクトデータベースシステムでメソッドマイグレーションを実現する方法として，我々が研究・開発している分散メソッドについて簡単に述べる[1),2),23)]．

我々は，Java と ObjectStore[13),14)] を用いた環境で分散メソッドの実装を行ってきた．メソッドを実行するためには，メソッドが処理されるサイトで動くプロセスがメソッドを動的にロードする必要がある．この動的ローディングは，Java の ClassLoader の機能を使って実現している[2)]．本稿で報告している実験はこのシステムを用いている．

**図 1** に我々が実装した分散メソッドのプロセス構造を示す．分散メソッドのサーバプロセスとクライアントプロセスはマルチスレッドプロセスである．データマイグレーションとメソッドマイグレーションが同一プロセス内の別々のスレッドで実行される場合，各スレッドは互いに並行に処理される．一方，同一プロセス内のデータマイグレーション，およびメソッドマイグレーションを処理する複数のスレッドは直列に処理される．これはスレッドでの処理中に，メソッドの実行が直列に処理されるためである．分散メソッドのサーバプロセスはクライアントからのリクエストを受け取るとスレッドを生成し，可能な限りリクエストを並行に処理するように構成されている．クライアントプロセスもサーバプロセスと同様にリクエストを処理するために複数のスレッドを生成する．

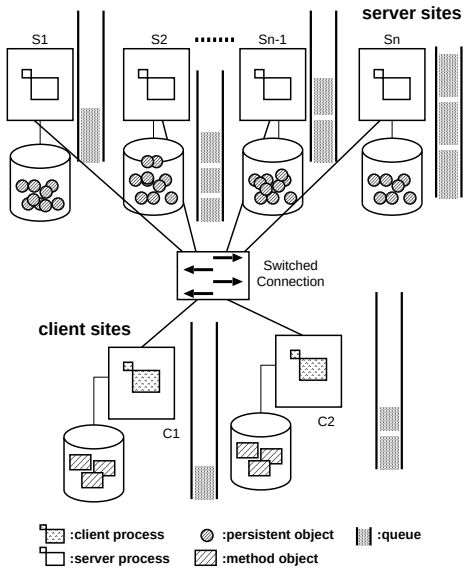


**図 2** 分散データベース環境の例
Fig. 2 An example of distributed database environments.

## 4. 効率的な組合せ

本章では，データマイグレーションとメソッドマイグレーションを組み合わせたときのレスポンスタイムを見積もるコストモデルを提案する．また，そのコストモデルを用いて，複数の組合せの中から最も効率の良い組合せを求めるアルゴリズムについて述べる．

本稿では，マルチサーバ・マルチクライアントの分散環境を考える（**図 2**）．図 2 の各サイトはスイッチング装置で接続されており，サーバとディスクドライブは単独のチャネルで接続される．また，サーバおよびクライアントはシングル CPU の計算機であるとする．

マルチサーバ・マルチクライアントの環境において，あるサーバサイトのデータに対して，データマイグレーションを用いて処理するか，あるいはメソッドマイグレーションを用いて処理するかという選択を，クライアントサイトで行うとする．$n$ 台のサーバサイトに対する処理では，メソッドマイグレーションとデータマイグレーションの組合せは $2^n$ 通り存在する．これらの組合せの中から最も効率の良い組合せを選択する．ただし本稿では，適用するメソッドは更新の処理を含まないものとする．

### 4.1 コストモデル

データベースが異なるサイトで分散して管理されているとき，クライアントプロセスがデータベースに対して処理を要求すると，それはそれらのサイトに対して処理を要求することを意味する．たとえば，従業員データベースを部門ごとに構築している場合に，全従業員に対して処理をする場合などが考えられる．また一般に，サーバは複数のクライアントが共有するデータを管理する．この場合，サーバは複数のクライアントプロセスからの処理要求を受け取り，処理できることが必要である．

サーバサイト $S_i$ $(1 \leq i \leq n)$ は処理対象となるデータベースを管理しており，処理対象となるデータサイズは $D_{S_i}$ (pages) とする．適用するメソッドのサイズを $M$ (pages) とする．サーバサイトとクライアントサイト $C$ の間のネットワークバンド幅を $NW$ (pages/sec)，$S_i$ のディスク I/O 速度を $DW_{S_i}$ (pages/sec)，$C$ のディスク I/O 速度を $DW_C$ (pages/sec)，$C$ が 1 秒あたりに処理できるデータのサイズを $PT_C$ (pages/sec)，$S_i$ が 1 秒あたりに処理できるデータのサイズを $PT_{S_i}$ (pages/sec) で表す．本稿では，図 2 に示すように，各サイトはスイッチング装置で接続されており，文献 20) の設定と同様に，サイト間の通信が他のサイト間の通信と競合することはないものと仮定する．本稿ではディスク I/O は読込みを対象としているため，$DW_{S_i}$，$DW_C$ はリードアクセスを表すものとする．

以下では，メソッドを実行するサイトにメソッドを移動し，プロセスにロードする時間を $T_M$ で表す．式 (1) 中の $DW$ はメソッドが存在するサイトのディスク I/O 速度を表す．第 2 項はメソッドの実行が行われるサイトにメソッドを移動するためのネットワークの転送時間であり，メソッドが存在するサイトとメソッドを実行するサイトが同一の場合は 0 になる．

$$T_M = M \times \frac{1}{DW} + M \times \frac{1}{NW} \quad (1)$$

はじめに，サーバサイトが 1 台 ($S_1$)，クライアントサイトが 1 台の環境における処理を考える．$C$ が $S_1$ のデータをデータマイグレーションで処理するときのレスポンスタイム $T_C$ は，以下のようになる．

$$T_C = T_M + D_{S_1} \times \left(\frac{1}{DW_{S_1}} + \frac{1}{NW} + \frac{1}{PT_C}\right) \quad (2)$$

クライアントサイトでメソッドが管理されているとき，メソッドが実行されるサイトとメソッドが存在するサイトは同一であるため，第 1 項は $C$ が $M$ (pages) のメソッドをディスクから読み出す時間に相当する．第 2 項は $S_1$ が $D_{S_1}$ (pages) のデータをディスクから読み込み，ネットワークで転送し，$C$ がメソッドを実行する時間の合計である．

$C$ が $S_1$ のデータをメソッドマイグレーションで処理するときのレスポンスタイム $T_{S_1}$ は，以下のよう

になる．

$$T_{S_1} = T_M + D_{S_1} \times (\frac{1}{DW_{S_1}} + \frac{1}{PT_{S_1}} + \frac{f}{NW}) \quad (3)$$

クライアントサイトでメソッドが管理されているとき，メソッドが実行されるサイトとメソッドが存在するサイトは異なる．したがって，第 1 項は $C$ が $M$ (pages) のメソッドをディスクから読み出しサーバへ転送する時間になる．第 2 項は $D_{S_1}$ (pages) のデータを $S_1$ がディスクから読み込み，メソッドを実行し，その実行結果を転送する時間である．ここで，$f$ $(0 \leq f \leq 1)$ を用いて，メソッドの実行結果のサイズを $fD_{S_1}$ と表す．

次に，これらを基にマルチサーバ・マルチクライアントの環境における処理を考える．複数のクライアントプロセスがメソッドマイグレーションで処理を要求することによって，サーバサイトではメソッドマイグレーションの処理要求の競合が起きる．クライアントサイトでは複数のサーバサイトのデータをデータマイグレーションで処理する際に，データマイグレーション処理要求の競合が起きる．一方，サーバサイト，クライアントサイトのそれぞれではデータマイグレーションとメソッドマイグレーションは並行に処理されるので，メソッドマイグレーションとデータマイグレーションの競合は起こらない．したがって図 2 に示すようなマルチサーバ・マルチクライアントの環境では，サーバサイト，クライアントサイトで起こるこれら 2 つの競合を考慮する必要がある．

サーバサイトで処理する場合，サーバの負荷による影響を考慮する必要がある．文献 19) では，1 秒間に CPU，ディスク I/O を使用する時間を用いて負荷を表している．これらの値は OS のカーネルが管理する統計情報から求めることができる．本稿は文献 19) と同様にサーバの CPU，ディスク I/O が使用される割合を，それぞれ $\rho_{cpu}$, $\rho_{disk}$（$0 \leq \rho_{cpu}$, $\rho_{disk} \leq 1.0$）と表現する．$\rho_{cpu}$, $\rho_{disk}$ を使ってサーバの負荷を表現することにより，他のクライアントの処理要求によるサーバサイトへの影響はもちろん，その他の様々な処理による負荷をすべてモデル化することができる．

サーバの負荷を考慮した場合，1 秒あたりに CPU が処理できるデータサイズ，ディスク I/O 速度はそれぞれ式 (4)，(5) で表せる．

$$PT'_{S_i} = (1 - \rho_{cpu}) \times PT_{S_i} \quad (4)$$

$$DW'_{S_i} = (1 - \rho_{disk}) \times DW_{S_i} \quad (5)$$

データマイグレーションで処理する場合は，データ受信とメソッドの実行，およびメソッドマイグレーションで処理する場合のメソッドの実行結果の受信から，クライアントサイトで競合が生じる．**図 3** に競合の様子を示す．図 3 で，サーバサイト $S_i$ に対する処理のうちでサーバサイト間で並列に処理される時間を $T_{parallel}[i]$，クライアントサイトで直列に処理される時間を $T_{serial}[i]$ とそれぞれ表す．図 3 では，$T_{parallel}[i]$ の値でソートしたサイトの集合を $S_1, \cdots, S_n$ とすることで，一般性を失うことなく $T_{parallel}[1] \leq \cdots \leq T_{parallel}[n]$ とする．

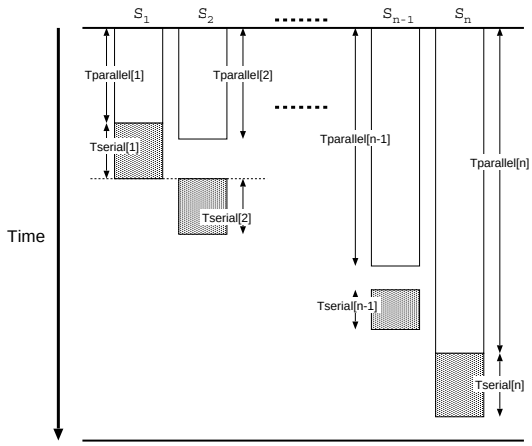


**図 3** 並列処理のコストモデル
Fig. 3 The parallel processing cost model.

データマイグレーションで処理する場合，$T_{parallel}[i]$ は式 (1) で表される時間とサーバサイトでのディスク I/O の処理時間であり，$T_{serial}[i]$ はネットワーク転送の時間とクライアントでのメソッド実行の時間である．また，メソッドマイグレーションの場合，$T_{parallel}[i]$ は式 (1) で表される時間とサーバサイトでの他のクライアントの競合による待ち時間，およびデータのディスク I/O，メソッドの実行による処理時間であり，$T_{serial}[i]$ はメソッドの実行結果の転送の時間を表す．

以上のことから，サーバサイト $S_i$ に対する $T_{parallel}[i]$，$T_{serial}[i]$ は，式 (2)，(3)，(4)，(5) を基にして，それぞれ次のように表せる．ここで，整数集合 $\boldsymbol{N} = \{1, 2, \cdots, n\}$, $\boldsymbol{DM} \subseteq \boldsymbol{N}$, $\boldsymbol{MM} \subseteq \boldsymbol{N}$ $(\boldsymbol{DM} \cap \boldsymbol{MM} = \phi \ and \ \boldsymbol{DM} \cup \boldsymbol{MM} = \boldsymbol{N})$ を用いて，サーバサイト全体を集合 $\{S_k | k \in \boldsymbol{N}\}$ で表し，データマイグレーション方式で処理するサイトの集合を $\{S_i | i \in \boldsymbol{DM}\}$，メソッドマイグレーション方式で処理するサイトの集合を $\{S_i | i \in \boldsymbol{MM}\}$ で表すことにする．

$$T_{parallel}[i] = \begin{cases} T_M + D_{S_i} \times \dfrac{1}{DW'_{S_i}} & (i \in \boldsymbol{DM}) \\ T_M + D_{S_i} \times (\dfrac{1}{DW'_{S_i}} + \dfrac{1}{PT'_{S_i}}) & (i \in \boldsymbol{MM}) \end{cases} \quad (6)$$

$$T_{serial}[i] = \begin{cases} D_{S_i} \times (\dfrac{1}{NW} + \dfrac{1}{PT_C}) & (i \in \boldsymbol{DM}) \\ D_{S_i} \times f \times \dfrac{1}{NW} & (i \in \boldsymbol{MM}) \end{cases} \quad (7)$$

式 (6)，(7) を使って，クライアントプロセスのレスポンスタイム $T$ を求めるアルゴリズムを**図 4** に示す．図 4 で，$t_p$，$t_s$ が式 (6)，(7) の $T_{parallel}[i]$，$T_{serial}[i]$ にそれぞれ対応する．アルゴリズム *ResponseTime* を用いてレスポンスタイム $T$ を求めることができる（式 (8)）．

$$T = ResponseTime(\boldsymbol{DM}, \boldsymbol{MM}) \quad (8)$$

### 4.2 効率化の戦略

最も効率の良い組合せを求める問題は，メソッドマイグレーションの並列処理による効率化と，メソッドマイグレーションで生じる競合をデータマイグレーションで避けることによる効率化のトレードオフを決定する問題としてとらえることができる．通常，サーバサイトはクライアントサイトと比較して高性能である場合が多い．また，複数のサイトに存在するデータに対する処理要求をメソッドマイグレーションで実行した場合，これらの処理要求を並列に実行できる．一方，複数のデータに対する処理要求をデータマイグレーションで実行した場合，これらの処理要求はクライアントサイトで直列に処理される．したがって，クライアントのレスポンスタイムを最小にするには，できるだけメソッドマイグレーションによって並列処理することが効率化につながる．ただし，メソッドマイグレーションで処理するときには他のクライアントによる競合が起こる場合があるため，この競合を考慮する必要がある．

これらを考慮した，効率の良い組合せを求めるアルゴリズムを**図 5** に示す．図 5 のアルゴリズムは，はじめに，$S_1, \cdots, S_n$ に対してすべてデータマイグレーションで処理すると仮定する．このときクライアントのレスポンスタイム $T$ は，$\boldsymbol{DM}=\boldsymbol{N}$，$\boldsymbol{MM}=\phi$ として式 (8) を使って表せる．$\boldsymbol{DM}$ 中のいくつかのサイトに対する処理をメソッドマイグレーションによる処理に変更することによって全体の処理効率を良くする．メソッドマイグレーションに変更するサイトを選択するときは，$T_{parallel}[i]$ が小さいサイトを優先して候補にする．

```
function ResponseTime(DM, MM)
begin
    var S: an empty list;
    foreach i ∈ DM ∪ MM do
    begin
        if i ∈ DM then begin
            t_p ← T_M + D_{S_i} × 1/DW'_{S_i};
            t_s ← D_{S_i} × (1/NW + 1/PT_C)
        end
        else begin
            t_p ← T_M
                + D_{S_i} × (1/DW'_{S_i} + 1/PT'_{S_i});
            t_s ← D_{S_i} × f × 1/NW;
        end
        append the pair (t_p, t_s) into S
    end
    sort S by t_p;
    T ← 0;
    foreach (t_p, t_s) in S do
        if T > t_p then
            T ← T + t_s
        else
            T ← t_p + t_s;
    end
    return T
end ;
```

**図 4** レスポンスタイム $T$ を求めるアルゴリズム
Fig. 4 The algorithm for the response time $T$.

## 5. 実　　験

4.1 節で述べたコストモデルと，4.2 節で述べた効率の良い組合せを求めるアルゴリズムを検証するため分散処理実験を行った．

### 5.1 実験環境

実験には，100 Mbps のイーサネットで接続された 4 台のワークステーションを用いた（**表 1**）．サイト $S_i$ $(i = 1, 2, \text{or } 3)$ をサーバサイト，サイト $C$ をクライアントサイトとして用いた．サーバサイトとして同種のワークステーションを用いた．クライアントサイトはこれらのサーバサイトよりも性能の低いものを選

```
function EfficientCombination(N)
begin
    {T_parallel[i] ≤ T_parallel[j](i < j; i, j ∈ N)}
    DM←N; MM←φ;
    T' ←ResponseTime(DM, MM);
    T ←ResponseTime(DM−{1}, MM∪ {1});
    k ←2;
    while T < T'and k ≤ n do
    begin
        DM←DM−{k};
        MM←MM∪{k};
        T'←T;
        T ←ResponseTime(DM, MM);
        k ←k + 1
    end
    if T ≥ T' then
        DM←DM∪{k};
        MM←MM−{k};
    end return (DM, MM)
end ;
```

**図 5** 効率の良い組合せを求めるアルゴリズム
Fig. 5 The algorithm for efficient combination.

**表 1** 実験に用いた計算機の構成
Table 1 Testbed configuration.

| Site | $S_1$, $S_2$, $S_3$ | $C$ |
|---|---|---|
| Type | Sun Ultra30 | Sun Ultra1 |
| CPU | UltraSparcII | UltraSparcI |
| Clock | 248 MHz | 167 MHz |
| Memory | 128 MB | 128 MB |
| Disk | SUN4.2 G | SUN2.1 G |
| Page size | 8,192 (bytes) | |
| OS | Solaris2.6 | |
| Java 処理系 | JDK 1.1.6(native thread) | |
| DBMS | ObjectStore R5.1 & Java Interface R3.0 | |

択した．各サイトでは Solaris2.6 が動作している．実験中，これらの計算機は実験の処理だけを動作させた．

実験のため，我々は `name`，`age`，`salary`，`x` と 2 Kbytes のビットマップ画像の `image` の属性を持つクラス `Person` のオブジェクト集合を用いた．この `Person` オブジェクトの属性の数は，OO1 ベンチマーク[5]の `Part` の持つ属性のうち，リスト構造である `to` と `from` を除いた数である．属性 `age` は，0 から 99 の範囲のランダムに生成された整数で，各サーバサイトでの値は一様に分布している．各サーバサイトでは，5,000 個の `Person` オブジェクトを要素に持つ集合を生成し，約 10 MB のデータベースを構築した．また，属性 `salary` は `age` から導出した値を用いた．実験では，`Person` オブジェクトは生成順に集合オブジェクトに挿入され，インデックスなどは考えていない．

生成した集合オブジェクトは，ObjectSpace Inc. が提供する集合クラス[15]を用いて実装した．この集合クラスは内部で要素を単方向リストで管理し，リストを先頭から走査し要素を取り出す．実験で用いたメソッドは，ある年齢以下の `Person` オブジェクトの持つ `salary` の平均値を求める．このメソッドはサイト $C$ で開発・実装し，サイト $S_1$，$S_2$ および $S_3$ で管理されている永続オブジェクト集合に対し適用した．Rodríguez-Martínez らが述べているように，メソッドをすべてのサイトにあらかじめ分散配置し管理することは非実用的である[17]．また，Gray らが主張するようにデータベースの管理の手間の削減は重要である[9]．メソッドをデータベースに分散配置することによって管理の手間が増大するという点から，実験ではメソッドのあるサイトをクライアントサイトとした．

他プロセスによるサーバの CPU 処理およびディスク I/O の影響を明らかにするために，実際に CPU 処理およびディスク I/O を生じるプロセスをサーバサイトで生成し，その処理を行いながら実験を行った[19]．このプロセスは文献 19) と同様に，CPU（ディスク I/O）の利用を $\rho_{cpu}(\rho_{disk})$ 秒間，その後 $1-\rho_{cpu}(1-\rho_{disk})$ 秒間 WAIT することを無限回繰り返す．したがって，このプロセスは CPU（ディスク I/O）処理能力の $\rho_{cpu}(\rho_{disk})$ を使用する処理となる．

具体的にこのプロセスは次のように実装されている．CPU 処理に関しては次のとおり．1) 変数 loop に 1 を代入し，初期化する．変数 loop は WAIT 処理と CPU 処理を切り替えるときのフラグとして用いる．2) SIGALRM シグナルのハンドラを設定する．シグナルハンドラは変数 loop に 0 を代入する関数である．3) $\rho_{cpu}$ 秒間有効なタイマを設定する．4) 変数 loop が非 0 である間，特定の変数の加算を繰り返す．5) タイマにより 3) のループを抜けた後，SIGALRM シグナルのハンドラを再び設定する．シグナルハンドラは変数 loop に 1 を代入する関数である．6) $(1-\rho_{cpu})$ 秒間有効なタイマを設定する．7) シグナルを受信するまで休眠し，5) のタイマにより覚醒する．8) 2)〜7) を無限に繰り返す．ディスク I/O に関しては，次の点を除いて CPU の場合とまったく同じである：上記説明の $\rho_{cpu}$ を $\rho_{disk}$ に置き変え，4) において 8192 バイトのバッファをファイルへ書き込む操作を繰り返し実行する．

シグナルハンドラの設定には `signal()` 関数，タイ

**表 2** 各パターンの $\rho_1, \rho_2, \rho_3$ の組合せ
Table 2 The combination of $\rho_1$, $\rho_2$and $\rho_3$ for each pattern.

| パターン | $\rho_1$ | $\rho_2$ | $\rho_3$ |
|---|---|---|---|
| $L$ (Low load) | 0.2 | 0.2 | 0.2 |
| $I$ (Intermediate load) | 0.2 | 0.5 | 0.8 |
| $H$ (High load) | 0.8 | 0.8 | 0.8 |

マの設定には `setitimer()` システムコール，プロセスの休眠は `pause()` システムコール，ファイルの書き込みは `write()`, `lseek()` システムコールを用いている[18]．本稿の実験では $\rho_{cpu}$ と $\rho_{disk}$ を設定し，他のクライアントの要求処理による影響の大きさを調整している．本稿では $\rho_{cpu}$ と $\rho_{disk}$ は同じ値としている☆．以下，サイト $S_i$ の $\rho_{cpu}, \rho_{disk}$ を $\rho_i$ で表す．

複数のクライアントサイトで様々なアプリケーションプログラムが実行されることを考えると，各アプリケーションプログラムのアクセスパターンは一定であるとは限らない．したがって，本稿ではサーバキャッシュの効果は考えない．さらに今回の実験では，実験に用いたプログラムはデータベース全体を走査する．この場合，ObjectStore がクライアントキャッシュの管理に用いる LRU アルゴリズムでは，クライアントキャッシュが有効に使われることはない．一般にページサーバアーキテクチャのサーバキャッシュの効果は，トランザクションのアクセスパターンとクラスタリングが大きく影響する[4),20)]が，本稿では以上の理由により，キャッシュの効果は考慮しない．

### 5.2 実験結果と考察

実験では $\rho_1, \rho_2, \rho_3$ の組合せについて 3 つのパターンを用いた．それぞれのパターンで返送されるデータの割合を変化させた場合のレスポンスタイムを計測した．**表 2** に各パターンの $\rho_1, \rho_2, \rho_3$ の組合せを示す．$\rho_i$は 0.0〜1.0 であるから中程度の$\rho_i$を 0.5 とした．低負荷な状況を$\rho_i$を 0.2，高負荷な状況を$\rho_i$を 0.8 とした．パターン $L$ はすべてのサーバサイトにほとんど負荷が存在しない環境である．このパターンはメソッドマイグレーションが効率が良いことを確認するために用いた．パターン $H$ はすべてのサーバに高い負荷が存在する環境である．パターン $H$ はメソッドマイグレーションよりもデータマイグレーションが効率が良いことを示すために用いた．パターン $I$ は，各サーバに異なる負荷が存在する環境である．このパターンはデータマイグレーションとメソッドマイグレーションが混在する組合せが効率が良いことを示すために用いた．

☆ $\rho_{cpu}$，$\rho_{disk}$ が異なる場合も本稿と同様の結果を得ている．

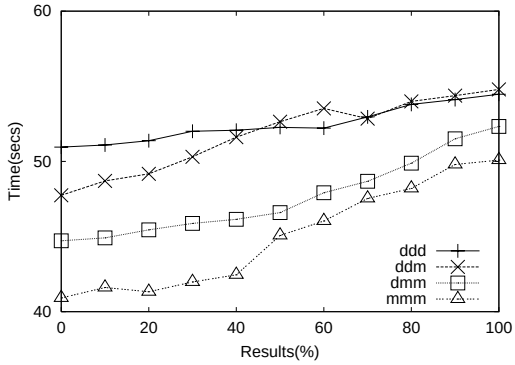


**図 6** パターン $L$ の実験結果
Fig. 6 Experimental results: $L$.

**表 3** に，コストモデルに用いたパラメータの値を示す．これらは実験に用いたデータベースに対するプログラムにより実測した値である．本稿のコストモデル評価を簡単にするため，これらの値を固定して用いた．

実験結果は，組合せを長さ 3 の文字列を使って表す．すなわち，サーバサイト $S_i$ $(i = 1, 2, or\ 3)$ のデータを，データマイグレーションで処理するときは $i$ 番目の文字を `d`，メソッドマイグレーションで処理するときは $i$ 番目の文字を `m` を使って表す．たとえば，`mdm` は $S_1$ に対してメソッドマイグレーション，$S_2$ に対してデータマイグレーション，$S_3$ に対してメソッドマイグレーションで処理する組合せを表す．

**図 6** にパターン $L$ の実験結果を示す．パターン $L$ は，各サーバサイトの条件がまったく同じになるため，`ddm`, `dmd`, `mdd` の組合せと `dmm`, `mdm`, `mmd` の組合せに，結果の違いはない．したがって，`ddd`, `ddm`, `dmm`, `mmm` の 4 通りの組合せのみを示す．グラフの縦軸はクライアントプロセスのレスポンスタイムを表し，横軸は返送されるデータの割合を示す．パターン $L$ では，各サーバサイトに競合するクライアントが少ない．この場合，サーバサイトがクライアントサイトよりも処理能力が高い環境では，すべての処理をメソッドマイグレーションで実行する組合せが最も効率が良いと考えられる．実験結果が示すとおり，`mmm` の組合せが返送されるデータの割合の変化に関係なく最も効率の良いことが分かる．`ddd` は，他の組合せと比較して結果のサイズの増加に対してレスポンスタイムの増加はわずかであることが分かる．一方，`mmm` は，結果のサイズの増加に対して最もレスポンスタイムの増加が大きい．返送されるデータの割合が 100% $(f = 1.0)$ のときに `ddd` と `mmm` に差が生じるのは，クライアントサイトとサーバサイトの処理能力の差が原因であると考えら

**表 3** パラメータ設定値
Table 3 Parameter settings.

| パラメータ ($i = 1, 2, 3$) | 値 | |
|---|---|---|
| | $S_1, S_2, S_3$ | $C$ |
| $D_{S_i}$ (pages) | 1283(≃10 MB) | |
| $NW$ (pages/sec) | 273.6(≃2189 KB) | 273.6(≃2189 KB) |
| $DW_{S_i}, DW_C$ (pages/sec) | 222.2(≃1820 KB) | 222.2(≃1820 KB) |
| $PT_{S_i}, PT_C$ (pages/sec) | 928.0(≃7.3 MB) | 520.0(≃4.1 MB) |

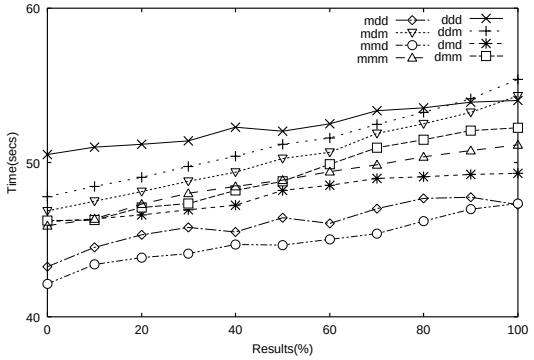


**図 7** パターン $I$の実験結果
Fig. 7 Experimental results: $I$.

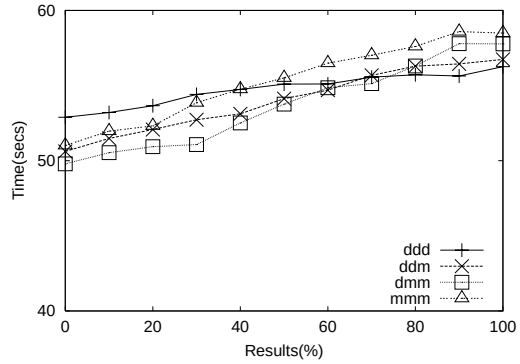


**図 8** パターン $H$の実験結果
Fig. 8 Experimental results: $H$.

れる．

**図 7** にパターン $I$ の実験結果を示す．ddm, dmm, mdm, mmm の 4 つの組合せは，最も負荷の高い $S_3$ に対してメソッドマイグレーションで処理を行うため，$S_3$ に対する処理がボトルネックとなってこれらの組合せは最も効率の良い組合せとはなりえない．一方，mmd, mdd の組合せでは，$S_3$ の高負荷の影響を避けることができる．ddd はパターン $L$ の場合とほぼ同様のレスポンスタイムを示す．これはデータマイグレーションは，高負荷なサーバの影響が少ないことを示している．

**図 8** にパターン $H$の実験結果を示す．パターン $L$ と同様に，サーバサイトの条件が同じであるため ddd, ddm, dmm, mmm の 4 通りの組合せのみを示す．結果のサイズが大きな場合はデータマイグレーションが効率の良い手法である．結果のサイズが小さな場合，結果のサイズが大きな場合と比較してサーバサイトの負荷がレスポンスタイムに及ぼす影響が小さくなるために，サーバサイトが高負荷な場合でもメソッドマイグレーションによる処理の方が効率が良くなる．

**表 4** コストモデルから求めた効率の良い組合せ
Table 4 Efficient combinations obtained with the cost-model.

| $100 \times f(\%)$ | 0 | 10 | 20 | 30 | 40 | |
|---|---|---|---|---|---|---|
| $L$ | mmm | mmm | mmm | mmm | mmm | |
| $I$ | mmd | mmd | mmd | mmd | mmd | |
| $H$ | mmm | dmm | dmm | dmm | dmm | |
| $100 \times f(\%)$ | 50 | 60 | 70 | 80 | 90 | 100 |
| $L$ | mmm | mmm | mmm | mmm | mmm | mmm |
| $I$ | mmd | mmd | mmd | mmd | mmd | mmd |
| $H$ | dmm | dmm | ddm | ddm | ddd | ddd |

**表 5** 各パターンの $\rho_1$，$\rho_2$，$\rho_3$ の組合せ
Table 5 The combination of $(\rho_1, \rho_2, \rho_3)$ for each pattern.

| パターン | $S_1$ | $S_2$ | $S_3$ |
|---|---|---|---|
| $R2$ | 0.2 | [0.2, 0.8] | [0.2, 0.8] |
| $R5$ | 0.5 | [0.2, 0.8] | [0.2, 0.8] |
| $R8$ | 0.8 | [0.2, 0.8] | [0.2, 0.8] |

**表 4** に 4.1 節で説明したコストモデルを使って求めた最も効率の良い組合せを示す．ここで誤差率を次のように定義する．実験結果から得られた効率の良い組合せのレスポンスタイムとコストモデルから得られた効率の良い組合せが実際に要したレスポンスタイムの差を，実験結果から得られた効率の良い組合せのレスポンスタイムで割った値を誤差率とする．たとえば，パターン $H$の $f = 0.8$ のとき，実験結果から得られた効率の良い組合せは ddd で，コストモデルから得られた効率の良い組合せは ddm であった．このとき，(ddm のレスポンスタイム) − (ddd のレスポンスタイム) から 0.5756 を得る．このときの誤差率は，0.5756 を ddd のレスポンスタイム 55.718 で割って 0.0103 となる．最も効率の良い組合せの判定を誤る場合は，パ

| $\rho_2\backslash\rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.3 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.4 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.5 | mmm | mmm | mmm | mmd | mmm | mmm | mdd |
| 0.6 | mmm | mmm | mmm | mdm | mdm | mdd | mdd |
| 0.7 | mmm | mmm | mmm | mmd | mdm | mdd | mdd |
| 0.8 | mdm | mdm | mdm | mdm | mdm | mdm | mdd |

(a) 実験から得られた組合せ

| $\rho_2\backslash\rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.3 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.4 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.5 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.6 | mmm | mmm | mmm | mmd | mmd | mmd | mmd |
| 0.7 | mmm | mmm | mdm | mmd | mdd | mdd | mdd |
| 0.8 | mdm | mdm | mdm | mdm | mdd | mdd | mdd |

(b) コストモデルから得られた組合せ

| $\rho_2\backslash\rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | – | – | – | – | – | – | – |
| 0.3 | – | – | – | – | – | – | – |
| 0.4 | – | – | – | – | – | – | – |
| 0.5 | – | – | – | 0.02 | – | – | 0.00 |
| 0.6 | – | – | – | 0.01 | 0.03 | 0.04 | – |
| 0.7 | – | – | – | 0.03 | 0.04 | – | – |
| 0.8 | – | – | – | – | 0.01 | 0.02 | – |

(c) 誤差率

**図 9** パターン $R2$ の実験結果とコストモデルによる組合せ
Fig. 9 Experimental results: $R2$.

| $\rho_2\backslash\rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.3 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.4 | mmm | mmm | mmm | mmm | mmm | mmm | mmd |
| 0.5 | mmm | mmm | mmm | mmm | mmm | mdd | mdd |
| 0.6 | mmm | mmm | mmm | mdd | mdd | mdd | mdd |
| 0.7 | mmm | mmm | mmm | mdd | mdd | mdd | mdd |
| 0.8 | mdm | mdm | mdm | mdd | mdd | mdd | mdd |

(b) コストモデルから得られた組合せ

| $\rho_2\backslash\rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | – | – | – | – | – | – | – |
| 0.3 | – | – | – | – | – | – | – |
| 0.4 | – | – | – | – | – | – | – |
| 0.5 | – | – | – | 0.02 | – | 0.05 | 0.03 |
| 0.6 | – | – | – | 0.01 | 0.03 | 0.03 | 0.02 |
| 0.7 | – | – | 0.02 | 0.02 | – | – | – |
| 0.8 | – | – | – | 0.04 | – | – | – |

(c) 誤差率

**図 10** パターン $R5$ の実験結果とコストモデルによる組合せ
Fig. 10 Experimental results: $R5$.

ターン $I$ の横軸が 100%のとき，パターン $H$の横軸が 0%，60〜80%のときである．判定を誤る場合でも誤差率は 0.002〜0.018 であり，その平均は 0.0073 であり実用上は問題ないと考えられる．

次に，返送されるデータの割合を固定し $\rho_1$, $\rho_2$, $\rho_3$ の組合せを変化させた場合の効率の良い組合せを考える．$\rho_1$ を 0.2, 0.5, 0.8 の 3 パターンとし，それぞれに対して，$\rho_2$, $\rho_3$ を 0.2〜0.8 まで 0.1 刻みで変化させ，合計 147 (=3×7×7) 通りの $\rho_1$, $\rho_2$, $\rho_3$ の組合せについて実験を行った．**表 5** に各パターン $R2$, $R5$, $R8$ の組合せを示す．**図 9**，**図 10**，**図 11** にそれぞれ $\rho_1$ が 0.2, 0.5, 0.8 のときの実験結果を示す．図 9，図 10，図 11 の各図では，実験結果より得られた効率の良い組合せを (a)，提案するコストモデルから得られた効率の良い組合せを (b)，誤差率を (c) に示す．各図 (c) で，“–” は実験結果から得られた効率の良い組合せとコストモデルから得られた効率の良い組合せが一致する場合を表す．

図 9，図 10，図 11 から，$\rho_1$, $\rho_2$, $\rho_3$ の負荷の組合せの総数 147 通りのうち 115 通りの組合せで正しく効

| $\rho_2 \backslash \rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | dmm | dmm | dmm | dmm | dmm | dmd | dmd |
| 0.3 | dmm | dmm | dmm | dmm | dmm | dmd | dmd |
| 0.4 | dmm | dmm | dmm | dmm | dmm | dmd | dmd |
| 0.5 | dmm | dmm | dmm | ddm | ddm | dmd | dmd |
| 0.6 | dmm | dmm | dmm | ddm | ddm | dmd | dmd |
| 0.7 | dmm | dmm | dmm | ddm | ddm | dmd | ddd |
| 0.8 | ddm | ddm | ddm | ddm | ddm | ddd | ddd |

(a) 実験から得られた組合せ

| $\rho_2 \backslash \rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | dmm | dmm | dmm | dmm | dmm | dmd | dmd |
| 0.3 | dmm | dmm | dmm | dmm | dmm | dmd | dmd |
| 0.4 | dmm | dmm | dmm | dmm | dmm | dmd | dmd |
| 0.5 | dmm | dmm | dmm | dmm | dmm | ddd | ddd |
| 0.6 | dmm | dmm | dmm | ddm | ddm | ddd | ddd |
| 0.7 | dmm | ddm | ddm | ddd | ddd | ddd | ddd |
| 0.8 | ddm | ddm | ddm | ddd | ddd | ddd | ddd |

(b) コストモデルから得られた組合せ

| $\rho_2 \backslash \rho_3$ | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 |
|---|---|---|---|---|---|---|---|
| 0.2 | – | – | – | – | – | – | – |
| 0.3 | – | – | – | – | – | – | – |
| 0.4 | – | – | – | – | – | – | – |
| 0.5 | – | – | – | 0.04 | 0.04 | 0.04 | 0.05 |
| 0.6 | – | – | – | – | – | 0.06 | 0.02 |
| 0.7 | – | 0.00 | 0.03 | 0.02 | 0.03 | 0.06 | – |
| 0.8 | – | – | – | 0.03 | 0.07 | – | – |

(c) 誤差率

**図 11** パターン $R8$ の実験結果とコストモデルによる組合せ
Fig. 11 Experimental results: $R8$.

率の良い組合せを求めることができたことが分かる．これは $(\rho_1, \rho_2, \rho_3)$ の全組合せの約 78.2%に相当する．また，効率の良い組合せの判定を誤る場合は 32 通りある．誤差率の平均は 0.03 であり，**図 12** の誤差率の度数分布から，27 通りの組合せが誤差率 0.04 以下であることが分かる．つまり，本稿のアルゴリズムを用いた場合は全体の約 96.6%は誤差率が 0.04 以下であり，多くの場合でごく低い誤差率であるといえる．実験結果から，以下の結論を得ることができた．1) 誤差率の平均は 0.03 である．2) 全体の約 78.2%は最も効率の良い組合せを求められる．3) 判定を誤った場合でも誤差率は低く，全体の約 96.6%の組合せで誤差率は 0.04 以下である．以上のことから，本稿で提案する方法は多くの場合で有効であり，実用上問題ないといえる．32 通りの誤りの原因として，ディスク I/O，ネットワーク処理，CPU 処理の並列処理を厳密に表現できていないなどが考えられる．たとえば，図 9 の $(\rho_2, \rho_3) = (0.5, 0.5), (0.5, 0.8), (0.6, 0.5), (0.6, 0.6), (0.6, 0.7)$ の箇所などは，実際にはデータマイグレーションで処理した方が効率的なときでも，コストモデルはメソッドマイグレーションで処理するほうが効率的であると誤っている．これら並列処理などを，より忠実にモデル化することは今後の課題である．

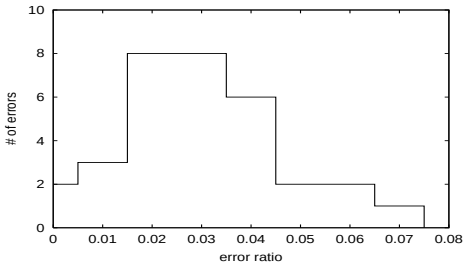


**図 12** 誤差率の度数分布
Fig. 12 The histogram of error ratio.

## 6. ま と め

計算機環境の発達により，ネットワークでつながれた比較的高性能の計算機を使うことが一般的になってきた．マルチサーバ・マルチクライアントの分散環境では，データマイグレーションとメソッドマイグレーションをうまく組み合わせ，レスポンスタイムを最小にする組合せを求めることができる．本稿では，データマイグレーションとメソッドマイグレーションの効率の良い組合せを求める手法について議論した．効率の良い組合せを求めるためにマルチサーバ・マルチクライアントの分散環境を対象に分散処理のコストモデルとアルゴリズムを構築した．また，我々が開発したプロトタイプシステムを用いた実験を行い，提案したコストモデルとアルゴリズムの有用性を示した．実験結果から，誤差率の平均は 0.03 となった．全体の約 78.2%で正確に効率の良い組合せを求めることができた．また，全体の約 96.6%は誤差率 0.04 以下に収まることが分かった．

本稿では，対象としたメソッドは更新処理を含まないものであった．今後の課題として，更新処理を含む場合にも適用できるコストモデルの改良が考えられる．

また，我々のシステムではサーバサイト，クライアントサイト以外の第三者的なサイトでメソッドを実行できる．第三者的なサイトとしてアイドル計算機を有効利用すればさらなる効率化を達成できる[22]．ディスクI/O，ネットワーク処理，CPU 処理の並列処理を厳密に表現することも今後の課題である．また，単項演算，二項演算を基本とした従来の問合せ最適化手法と本稿で提案する手法を組み合わせた効率化手法も今後の課題である．

**謝辞** 著者らと議論していただいた中村知久君（現在，日本電気株式会社）に深謝します．

## 参 考 文 献


1) 有次正義，吉田裕介，中村知久，金森吉成：分散メソッドの提案，情報処理学会研究会報告，pp.189–196, 情報処理学会 DBS 研 (1998).
2) Aritsugi, M., Yoshida, Y., Nakamura, T. and Kanamori, Y.: Method Migration in Object Database Environments, *Proc. 4th World Multiconference on Systemics, Cybernetics and Informatics* (*SCI2000*), Vol.VIII, pp.125–130 (2000).
3) Carey, M.J., DeWitt, D.J., Franklin, M.J., Hall, N.E., McAuliffe, M.L., Naughton, J.F., Schuh, D.T., Solomon, M.H., Tan, C.K., Tsatalos, O.G., White, S.J. and Zwilling, M.J.: Shoring Up Persistent Applications, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp.383–394 (1994).
4) Castro, M., Adya, A., Liskov, B. and Myers, A.C.: HAC: Hybrid Adaptive Caching for Distributed Storage Systems, *Proc. ACM Symposium on Operating System Principles*, ACM (1997).
5) Cattel, R.G.G. and Skeen, J.: Object Operations Benchmarks, *ACM Trans. Database Syst.*, Vol.17, No.1, pp.1–31 (1992).
6) Özsu, M. and Blakeley, J.: *Modern Database Systems: The Object Model, Interoperability, and Beyond*, chapter 8, pp.146–174, ACM Press/Addison-Wesley (1995).
7) Franklin, M.J., Jónsson, B.T. and Kossman, D.: Performance Tradeoffs for Client-Server Query Processing, *Proc. ACM SIGMOD Conf.*, pp.149–160, ACM (1996).
8) Goel, S., Bhargava, B. and Jiang, Y.H.: Supporting Method Migration in a Distributed Object Database System: A Performance Study, *Proc. 29th Annual Hawaii Intl. Conf. on System Sciences*, IEEE (1996).
9) Gray, J. and Shenoy, P.: Rules of Thum in Data Engineering, *Proc. 16th Intl. Conf. on Data Engineering*, pp.3–10 (2000).
10) Hara, T., Harumoto, K., Tsukamoto, M. and Nishio, S.: DB-MAN:A Distributed Database System Based on Database Migration in ATM Networks, *Proc. 14th Intl. Conf. on Data Engineering*, pp.522–531, IEEE (1998).
11) 原 隆浩，春本 要，塚本昌彦，西尾章治郎：データベース移動に基づく分散データベースシステムとその並行処理制御機構，電子情報通信学会論文誌，Vol.J81-D-I, No.6, pp.819–827 (1998).
12) Jhingran, A.: A Performance Study of Query Optimization Algorithms on a Database System Supporting Procedures, *Proc. 14th Intl. Conf. on Very Large Data Bases*, pp.88–99 (1988).
13) Lamb, G., Landis, G., Orestein, J. and Weinreb, D.: The ObjectStore database system, *Comm. ACM*, Vol.34, No.10, pp.51–63 (1991).
14) Object Design, Inc.: ObjectStore Java API Users Guide (1997).
15) ObjectSpace Inc.: JGL Version 3.1 User Guide (1997).
16) Ontologic Inc.: Ontos Object Database Version 3.0 Developer's Guide (1994).
17) Rodríguez-Martínez, M. and Roussopoulos, N.: MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources, *Proc. ACM SIGMOD Conf.*, pp.213–224 (2000).
18) Sun Microsystems, Inc.：システムインタフェース (1997).
19) 谷口秀夫：入出力性能の制御によりプログラム実行速度を調整する制御法の実装方式による比較評価，電子情報通信学会論文誌，Vol.J84-D-I, No.9, pp.1362–1371 (2001).
20) Voruganti, K., Ozsu, M.T. and Unrau, R.C.: An Adaptive Hybrid Server Architecture for Client Caching, Object DBMSs, *Proc. 25th Intl. Conf. on Very Large Data Bases*, pp.150–161 (1999).
21) Yoshida, Y., Aritsugi, M. and Kanamori, Y.: Performance Evaluation of Combining Data Migration and Method Migration in Object Database Environements, *Proc. 13th Australasian Database Conference* (*ADC2002*), pp.207–214 (2002).
22) 吉田裕介，有次正義，金森吉成：アイドル計算機を用いた分散 ODB 処理の効率化の一検討，第 13 回データ工学ワークショップ（DEWS2002），電子情報通信学会データ工学専門委員会 (2002). http://www.ieice.org/iss/de/DEWS/
23) 吉田裕介，中村知久，有次正義，金森吉成：分散環境でのデータ移動とメソッド移動，第 9 回デー

タ工学ワークショップ（DEWS98），電子情報通信学会データ工学専門委員会 (1997).

(平成 14 年 3 月 20 日受付)
(平成 14 年 7 月 3 日採録)

（担当編集委員　石川 博）

**吉田　裕介**（正会員）

1973 年生．1996 年群馬大学工学部情報工学科卒業．1998 年同大学大学院工学研究科博士前期課程情報工学専攻修了．現在，同大学院工学研究科博士後期課程電子情報工学専攻在学中．電子情報通信学会会員．

**有次　正義**（正会員）

群馬大学工学部情報工学科助教授．1991 年九州大学工学部情報工学科卒業．1996 年同大学大学院博士後期課程修了．博士（工学）．同年群馬大学工学部情報工学科助手．2000 年より現職．データベースシステム，分散並列データ処理等に興味を持つ．電子情報通信学会，IEEE-CS，ACM 等各会員．

**金森　吉成**（正会員）

1942 年生．1969 年東北大学大学院工学研究科博士課程電子工学専攻修了．工学博士．東北大学電気通信研究所助手，同助教授，仙台電波高専教授を経て，1990 年群馬大学工学部情報工学科教授．オブジェクト指向データベース，マルチメディアデータベースに関する研究に従事．ACM，IEEE-CS，電子情報通信学会各会員．