分散環境におけるアイドル計算機を利用したオブジェクトデータベース処理の効率化

吉田裕介[†] 有次正義[†] 金森吉成[†]

これまで我々は，マルチサーバ・マルチクライアントの分散環境におけるオブジェクトデータベース処理の効率化手法を提案し，その有効性を示してきた．それは，サーバから永続オブジェクトを移動しクライアントで実行するデータマイグレーションと，クライアントからメソッドを移動しサーバで実行するメソッドマイグレーションを組み合わせる手法である．ネットワークにつながれた計算機は，つねに処理をしているわけではなく，アイドル状態でいることが多い．そこで本稿では，永続オブジェクトおよびそのメソッドをアイドル計算機に移動し，アイドル計算機でメソッドを実行し，その結果のみをクライアントで受け取る方法を加え，これら3方法の組合せによる効率化手法を提案する．この提案手法はデータマイグレーション・メソッドマイグレーションのみの組合せと比較して，さらなる効率化が可能である．また，実験によって本稿の提案手法の有効性を示す．

Increasing the Efficiency of Object Database Processing with Idle Computers in Distributed Environments

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

We have investigated an efficient processing technique of object databases which combines data-migration and method-migration in distributed environments consisting of multiple servers and multiple clients. In this paper, we enhance it by combining another primitive processing using idle machines in a network with the two ways. Because not all of the machines connected by a network operate continuously, it would be practical to use such machines for efficiency. The processing way is to move both persistent objects and the method to idle machines, to apply the method to the objects, and to send the results to the client site. We also present some experimental results for showing the effectiveness of our proposal.

1. はじめに

今までに，我々はマルチサーバ・マルチクライアント環境でのオブジェクトデータベース処理の効率化手法を提案した[4),5),15)~17)]．そこでは，サーバから永続オブジェクトを移動しクライアントで実行するデータマイグレーションと，クライアントからメソッドを移動しサーバで実行するメソッドマイグレーションを組み合わせ，それらの組合せから最小のレスポンスタイムを与える組合せを選択することによって処理の効率化を実現している．

本稿では，ネットワーク上に存在するアイドル計算機を使った処理の効率化について考察する．分散環境において，ネットワークにつながれた計算機は，つねに処理をしているわけではなく，アイドル状態でいることが多い[6)]．これらアイドル計算機をうまく利用することで，処理の効率化が可能である．

本稿ではアイドル計算機をオブジェクトデータベース処理に上手く利用する効率化手法を提案する．本稿で提案する効率化手法は，今までの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にもう1つの方法を加え，これら3方法の組合せからレスポンスタイムを最小にする組合せを求める．具体的には，永続オブジェクトおよびそのメソッドを，アイドル計算機に移動し，アイドル計算機でメソッドを実行し，結果をクライアントで受け取る方法を考える．本稿で提案する3方法の組合せによる効率化手法は，従来の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みの組合せを用いる効率化手法と比較し，さらなる効率化を達成できる．特に，サーバの負荷が高い場合に，本稿の提案手法はより効果的である．

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

これまでに，データマイグレーションとメソッドマイグレーションの組合せによる効率化の研究が報告されている[8),9),13),17)]．これらの研究は，分散データベース処理を，サーバとクライアントのどちらで実行することが効率的であるかを議論している．また，Hara らはデータベース移動に基づく処理について議論している[11)]．しかし，本稿のようにアイドル計算機を利用し，データマイグレーションとメソッドマイグレーションと組み合わせたオブジェクトデータベースの効率的な処理に関する議論は見られない．

アイドル計算機を有効に利用し，処理の効率化を図る研究は多くある[1),3)]．これらは主に，アイドル計算機をうまく使ったデータ並列処理と，その負荷分散について議論している．一方本稿では，アイドル計算機にデータだけではなくそれに適用するメソッドも移動させ，そこで処理をして結果を得る方法を考え，それを従来の処理と組み合わせることによるオブジェクトデータベース処理の効率化を議論する．

本稿の構成は，次のとおりである．2 章で従来の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による効率化を説明する．3 章でアイドル計算機を用いた効率化を提案する．4 章で実験について報告し，考察を述べる．5 章でまとめを述べる．

2. 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

ここでは，従来の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について，本稿の議論に必要な概要を述べる．詳細は文献 17) を参照されたい．

我々は**図 1** に示すような分散環境における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方法として，次の二つの方法を議論した．また，これらの組合せによる効率化手法について提案した[17)]．

- 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．

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．

図 1 に示すような，n 台のサーバが存在する，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は 2^n 通り考えられる．サーバが複数あるときは，効率の良い組合せの決定に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

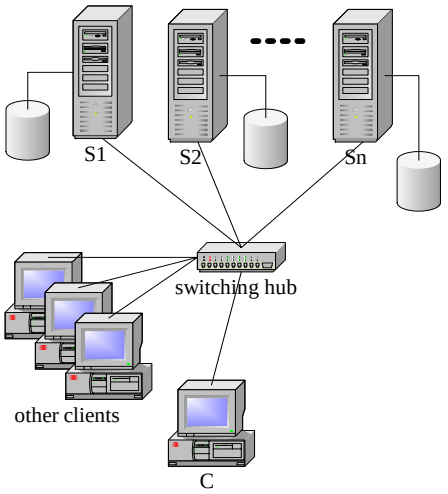

図 1 分散環境におけるオブジェクトデータベース
Fig. 1 Object databases in a distributed environment.

がある．

2^n 通りの組合せから最も効率の良い組合せを決定するために，レスポンスタイムを見積もるコストモデルを構築した[17)]．以下では，そのコストモデルについて述べる．

サーバサイト $S_i(1 \leq i \leq n)$ はサイズが D_{S_i}(pages) のデータを管理している．適用するメソッドのサイズを M(pages) とする．S_i 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．

メソッドを実行するサイトにメソッド自身を移動し，プロセスにロードする時間は式 (1) で表される．ここで，DW_{MR} はメソッドが存在するサイトのディスク I/O 速度を表すものとする．メソッドを実行するサイトとメソッドが存在するサイトが同一なら，第 2 項は 0 になる．

$$T_M = M \times \frac{1}{DW_{MR}} + M \times \frac{1}{NW} \qquad (1)$$

はじめに，サーバサイトが 1 台（S_i），クライアントサイト（C）が 1 台の環境について説明する．

メソッドを C で実行した場合，レスポンスタイム T_C は式 (2) で表される．

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

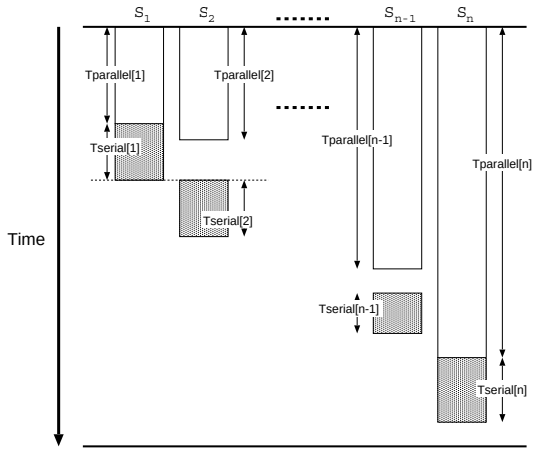

図 2 並列処理のコストモデル
Fig. 2 The parallel processing cost model.

メソッドを S_i で実行した場合，レスポンスタイム T_{S_i} は式 (3) で表される．ここで，f はサーバからメソッドの返り値としてクライアントに返されるデータサイズの割合とし，$0.0 \leq f \leq 1.0$ とする．

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

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．マルチサーバ・マルチクライアントの環境ではサーバの負荷を考慮する必要がある．文献 14) 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 $\rho_{cpu}, \rho_{disk} (0 \leq \rho_{cpu} \leq 1.0,\ 0 \leq \rho_{disk} \leq 1.0)$ と表現する．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．サーバの負荷を考慮した場合，1 秒あたりに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) で表される．

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

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

図 2 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 $T_{serial}[i]$ と表す．図 2 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．

サーバサイト S_i に対する $T_{parallel}[i], T_{serial}[i]$ は，式 (1)，(2)，(3)，(4)，(5) を基にして，それぞれ次のように表せる．

$$T_{parallel}[i] = \begin{cases} T_M + D_{S_i} \times \dfrac{1}{DW'_{S_i}} & (C\text{で実行する場合}) \\ T_M + D_{S_i} \times \left(\dfrac{1}{DW'_{S_i}} + \dfrac{1}{PT'_{S_i}} \right) & (S_i\text{で実行する場合}) \end{cases} \quad (6)$$

$$T_{serial}[i] = \begin{cases} D_{S_i} \times \left(\dfrac{1}{NW} + \dfrac{1}{PT_C} \right) & (C\text{で実行する場合}) \\ D_{S_i} \times f \times \dfrac{1}{NW} & (S_i\text{で実行する場合}) \end{cases} \quad (7)$$

レスポンスタイム T は，式 (8) に示す $Response\ Time$ から求めることができる．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 $\{S_k | k \in \boldsymbol{N}\}$，データマイグレーション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で処理するサイトの集合を $\{S_i | i \in \boldsymbol{MM}\}$ で，それぞれ表すことにする．

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

また，最も効率の良い組合せは，式 (9) に示す $EfficientCombination$ から求めることができる．

$$(\boldsymbol{DM}, \boldsymbol{MM}) = EfficientCombination(\boldsymbol{N}) \quad (9)$$

ただし，式 (8)，(9) の詳細はここでは省略する．

3. アイドル計算機を利用した効率化

3.1 アイドル計算機の利用

ネットワークにつながれた計算機は，つねに処理をしているわけではなく，アイドル状態でいることが多い[6]．そこで，永続オブジェクトのメソッドを実行するサイトの候補として，アイドル計算機を加えることを考える．したがって，本稿では，永続オブジェクトにメソッドを適用する方法として次の 3 方法の組合せを考える．

- 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．
- 永続オブジェクトおよびそのメソッドを，ともにアイドル計算機に移動させて適用する．

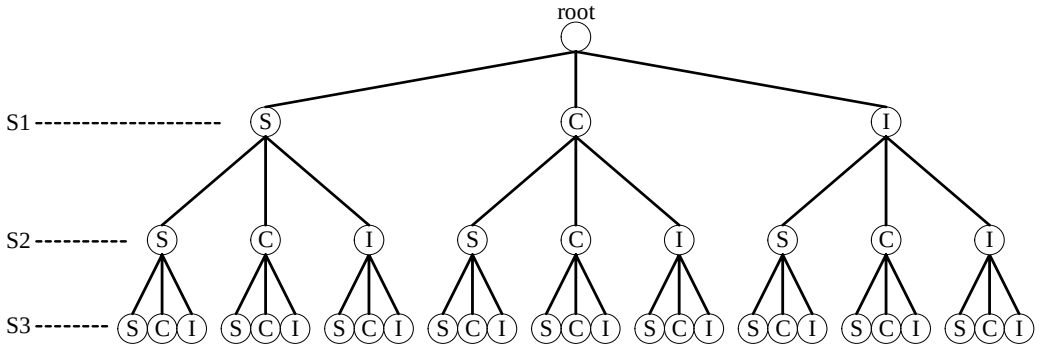

図 3 サーバ 3 台，アイドル計算機 1 台の場合のすべての組合せ
Fig. 3 All combinations when three servers and one idle machine exist.

アイドル計算機を有効に利用することで，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みの組合せと比較して，さらなる効率化が可能である．さらに，本稿の提案手法は使われずにいる計算機の資源を有効に用いる方法であるから，新たなハードウェアの投資が不要である．

サーバが n 台，アイドル計算機が m 台の環境では，サーバ S_i のデータに対してメソッドを実行するサイトは $S_i, C, I_1, \cdots, I_m$ の $(m+2)$ 通りあり，これがサーバの台数分あるから，組合せの総数は $(m+2)^n$ である．これら組合せの中から最も効率の良い組合せを決定する．たとえば，サーバ S_1，S_2，S_3，クライアント C，アイドル計算機 I からなる分散環境の場合，組合せの総数は $(1+2)^3 = 27$ 通りある．**図 3** にこの分散環境におけるすべての組合せを示す．図 3 では，深さ 1 にあるノードから葉ノードまでの経路にあるノードの列で組合せを表している．深さ 1，2，3 にある頂点は，それぞれ S_1，S_2，S_3 のデータに対するメソッドを実行するサイトを示す．たとえば，$S-C-I$ の経路は，S_1，S_2，S_3 が持つデータに対する処理を，それぞれ S_1，C，I で実行することを意味する．

3.2 コストモデル

図 4 にアイドル計算機を加えた分散環境を示す．S_1, $\cdots$, S_n はサーバ，C はクライアント，$I_1, \cdots, I_m$ はアイドル計算機を表す．

サーバサイト，クライアントサイト，アイドル計算機の間のネットワークバンド幅を NW(pages/sec)，I_j が 1 秒あたりに処理できるサイズを PT_{I_j} で表す．その他のパラメータは，2 章と同じものとする．ただし，本稿では対象とするメソッドは更新の処理を含まないものとする．

サーバ S_i，クライアント C，アイドル計算機 I_j が各 1 台ずつの環境では，メソッドをアイドル計算機で実行する場合のレスポンスタイム T_{I_j} は式 (10) で表

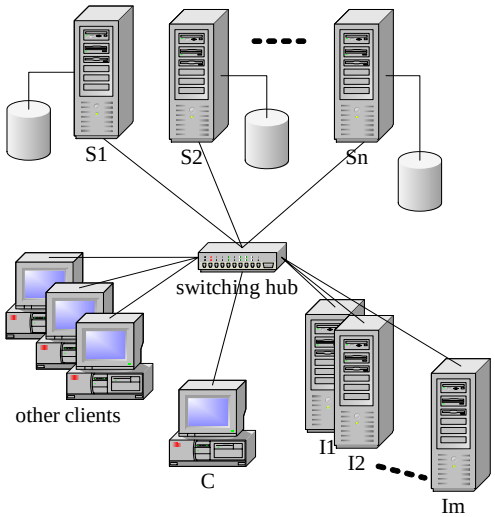

図 4 アイドル計算機を加えた分散環境
Fig. 4 A distributed environment with idle computers.

される．

$$T_{I_j} = T_M + D_{S_i} \times \left(\frac{1}{DW_{S_i}} + \frac{1}{NW} + \frac{1}{PT_{I_j}} \right) + f \times D_{S_i} \times \frac{1}{NW} \quad (10)$$

第 1 項は，C が M(pages) のメソッドをディスクから読み出し I_j へ転送する時間である．第 2 項は，D_{S_i}(pages) のデータを S_i がディスクから読み込み，I_j へ転送し，I_j がメソッドを実行する時間である．第 3 項は，その実行結果を転送する時間である．ここで，2 章と同様に，f を用いてメソッドの実行結果のサイズを $f \times D_{S_i}$ と表す．

次に，マルチサーバ・マルチクライアントの分散環境を考える．アイドル計算機でメソッドを実行する場合，$T_{parallel}[i]$ は，アイドル計算機をクライアントと見なし，$f = 1.0$ で式 (8) を適用した結果に相当する．

$T_{serial}[i]$ はアイドル計算機からの実行結果をクライアントへ転送する時間である．このとき，1 台のアイドル計算機で複数のサーバのデータに対する処理を実行することを考慮する必要がある．また，本稿では各アイドル計算機の処理能力は既知であり，順序付けられているものとする．

以上のことから，アイドル計算機を考慮にいれた $T_{parallel}$，T_{serial} は，式 (6)，(7) を拡張してそれぞれ式 (11)，(12) のように表せる．ここで，I_j でメソッドを実行するサイトの集合を $\{S_i | i \in \boldsymbol{ID}_j\}$ で表す．1 つのアイドル計算機で複数のサイトのデータを処理する場合，これらを 1 つにまとめ，サイトの添字番号を付け直す必要がある．たとえば，(I_1, C, I_1, S_4) という組合せの場合，$T_{parallel}[1]$ と $T_{parallel}[3]$ で 2 度 I_1 での処理時間を計算してしまう．したがって，$T_{parallel}[1]$ で I_1 に関する計算を 1 つにまとめ添字番号を付け直し，式 (11) で，$T_{parallel}[1]$ は I_1 での処理，$T_{parallel}[2]$ は C での処理，$T_{parallel}[3]$ は S_4 での処理を表す．

$$T_{parallel}[i] = \begin{cases} T_M + D_{S_i} \times \dfrac{1}{DW'_{S_i}} & (C\text{で実行する場合}) \\ T_M + D_{S_i} \times \left(\dfrac{1}{DW'_{S_i}} + \dfrac{1}{PT'_{S_i}} \right) & (S_i\text{で実行する場合}) \\ ResponseTime(\boldsymbol{ID}_j, \phi) & (I_j\text{で実行する場合.} \\ & f = 1.0, I_j\text{をクライアントと} \\ & \text{見なして，式 (8) を用いる.}) \end{cases} \tag{11}$$

$$T_{serial}[i] = \begin{cases} D_{S_i} \times \left(\dfrac{1}{NW} + \dfrac{1}{PT_C} \right) & (C\text{で実行する場合}) \\ f \times D_{S_i} \times \dfrac{1}{NW} & (S_i\text{で実行する場合}) \\ \displaystyle\sum_{k \in \boldsymbol{ID}_j} (f \times D_{S_k}) \times \dfrac{1}{NW} & (I_j\text{で実行する場合}) \end{cases} \tag{12}$$

式 (11)，(12) を使って，レスポンスタイム T を求めるアルゴリズムを**図 5** に示す．図 5 で，t_p，t_s が式 (11)，(12) の $T_{parallel}[i]$，$T_{serial}[i]$ にそれぞれ対応する．$ResponseTimeWithIdle$ を用いてレスポンスタイム T を求めることができる（式 (13)）．$\boldsymbol{M}$ は長さ n のベクトルで，サーバ S_i のデータに対して，メソッドを実行するサイトを表す文字を i 番目の要素として持つ．たとえば，図 4 の例において，$\boldsymbol{M} = (S_1, I_1, C)$ のとき，S_1，S_2，S_3 のデータに対するメソッドは，それぞれ S_1，I_1，C で実行される．

```
function ResponseTimeWithIdle(M)
begin
    var S : an empty list;
    foreach k ∈ M do begin
        if k = C 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 if k ∈ {S_1, S_2, ···} then 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
        else if k = I_j then
            ID_j ←ID_j ∪ {k};
        end
        append the pair (t_p, t_s) into S;
    end
    for j ←1 to m do begin
        if ID_j ≠ φ then begin
            {procedure (8) is used
             where f=1.0 and I_j is treated as C }
            t_p ←ResponseTime(ID_j, φ);
            t_s ←Σ_{k∈ID_j} (f × D_{S_k}) × 1/NW;
            append the pair (t_p, t_s) into S;
        end
    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 5 レスポンスタイム T を求めるアルゴリズム
Fig. 5 The algorithm for the response time T.

$$T = ResponseTimeWithIdle(\boldsymbol{M}) \quad (13)$$

3.3 効率の良い組合せを決定する手順

3.2 節のコストモデルを基に，最も効率の良い組合せを以下のような手順によって求めることができる．以下の手順では，サーバ n 台，アイドル計算機 m 台の場合，すべての組合せを図 3 に示すような高さ n の完全 $(m+2)$ 分木で表現する．図 3 はサーバ 3 台，アイドル計算機 1 台の場合に相当する．

(1) 解の候補集合を全組合せに初期化する．

(2) $(\boldsymbol{DM}, \boldsymbol{MM}) \leftarrow EfficientCombination(\{1, \cdots, n\})$ から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[17]．$T \leftarrow ResponseTime(\boldsymbol{DM}, \boldsymbol{MM})$ とする．

(3) $\boldsymbol{M} \leftarrow (x_1, \cdots, x_n)$ とする．ただし，要素 x_i は $x_i = S_i(i \in \boldsymbol{MM})$ *or* $C(i \in \boldsymbol{DM})$ なる値である．$\boldsymbol{DM}$，$\boldsymbol{MM}$は (2) で求めたものである．

(4) $\boldsymbol{M}$以外の I を含まない組合せを候補集合から除外する．

(5) 候補集合の要素数が 1 になるまで，以下の処理を繰り返す．候補集合の要素数が 1 であれば，(11) へ行く．

(6) root ノードから深さ優先でノードを巡回する．

(7) 現在のノードまでの経路が表す組合せを M' とし，$T' \leftarrow ResponseTimeWithIdle(M')$ を計算する☆．

(8) $T < T'$ ならば，現在のノードを頂点とする部分木を含む組合せを候補集合から除外し，(5) に戻る．

(9) $T > T'$ かつ現在ノードが葉ノードならば，$\boldsymbol{M}$ ←（現在のノードまでの経路が表す組合せ），$T \leftarrow T'$ として，(5) へ戻る．

(10) $T > T'$ かつ，現在のノードが葉ノードでないならば，現在のノードを頂点とする部分木を含む組合せを候補集合から除外し，深さ優先で次のノードを決定し (7) へ戻る．

(11) 求める組合せは M である．

本稿の提案手法は，企業や大学の研究室等の比較的小規模な環境を想定している．4 章で述べる実験で用いた環境では，この手順を使って効率の良い組合せを決定するのに約 0.012 秒を要した．つまり，実際のディスク I/O やネットワーク転送に要する時間と比較して，このオーバヘッドは小さく，我々が想定する環境では十分実用的である．しかしサーバが数百台のような大規模な環境では，組合せ決定に要する時間は大きくなる．これについては，以下のような対応が可能である．たとえば，効率の良い組合せをあらかじめ計算しておくことで，リクエストごとに計算をする必要をなくすことが可能である．このとき，すべてのパラメータの組とその計算結果を保存するのではなく，サーバの負荷やパラメータをモニタリングして出現頻度の高いパラメータの組のみをあらかじめ計算しておく等の工夫も考えられる．これらにより，本稿の提案手法を効果的に用いることができる．

☆ M' は長さ n 以下であるが，$ResponseTimeWithIdle(M')$ は，M' が表すサイトの組合せでメソッドを実行したときのレスポンスタイムである．

4. 実　　験

3 章のコストモデルを検証するために，実験を行った．

4.1 実験環境

実験には，100 Mbps のイーサネットで接続された 5 台のワークステーションを用いた（**表 1**）．サイト $S_i(i = 1, 2, or\ 3)$ をサーバサイト，サイト C をクライアントサイト，サイト I をアイドル計算機として用いた．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およびアイドル計算機はサーバサイトよりも性能の低いものを選択した．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．PT_C，PT_{S_i}，PT_{I_j} の値は問合せごとに異なるが，本稿の実験ではあらかじめ実測した値を用いている．これらのパラメータを問合せごとに決定するための機構としては，現在以下を含むいくつかの手法を検討中である．1) 一度メソッドを実行した場合，そのメソッドに関する処理効率を表すパラメータを，メソッドの種類やデータ形式等とともにデータベースに保存しておく．その情報をうまく使うことで，ある程度正しい値を見積もることが可能である．2) プログラミング言語の分野で，Java のバイトコードを解析する研究が行われている[2),12)]．それらと同様に，バイトコードを解析し，

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, I
Type	Sun Ultra30	Sun Ultra1
CPU	UltraSparcII	UltraSparcI
Clock	248 MHz	167 MHz
Memory	128 MB	128 MB
Disk	SUN4.2G	SUN2.1G
Page size	8,192(bytes)	
OS	Solaris2.6	
Java 処理系	J2SE1.3.1	
DBMS	Berkeley DB 4.0.14	

プリミティブな命令列と適用するデータセットの性質を解析することにより，パラメータを適切に設定する．ただし，これらは検討中であり今後の課題である．

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 PersonSet 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[7]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，2,000 個の Person オブジェクトを要素に持つ集合を生成し，約 7.88 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る．実験ではインデックスは特に考えていないが，パラメータ D_{S_i} をインデックスを使った場合にディスクから読み出されるデータサイズとすることで，インデックスを用いた場合でも本稿の手法を適用可能である．

実験では，サーバの ρ_{cpu}，ρ_{disk} を同じ値に設定した．以後，サーバ S_i の ρ_{cpu}，ρ_{disk} をともに ρ_i で表すことにする．ρ_1，ρ_2，ρ_3 の組合せについて 3 つのパターンを用いた．各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 ρ_1，ρ_2，ρ_3 の組合せを示す．ρ_i は 0.0〜1.0 であるから中程度の ρ_i を 0.5 とした．低負荷な状況を ρ_i を 0.2，高負荷な状況を ρ_i を 0.8 とした．

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．

実験に用いた PersonSet オブジェクトは，Java 処理系に付属の java.util.HashSet クラスを用いて実装した．PersonSet オブジェクトはイテレータを用いて，集合中の要素を取り出す．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 の平均値を求めるメソッドである．実験では，メソッドはクライアントサイト C に存在するものとする．

4.2 実験結果と考察

図 6 に実験結果のグラフを示す．以降では，組合せを長さ 3 の文字列を使って表す．この文字列の i 番目の文字は，S_i のデータに対してどのサイトでメソッドを実行するかを示す．i 番目の “S” はサーバ (S_i) で実行，“C” はクライアントで実行，“I” はアイドル計算機で実行することを表している．たとえば，文字列 “SIC” は，S_1 のデータを S_1 で，S_2 のデータを I で，S_3 のデータを C で処理する組合せを表している．図 3 に示すように各パターンで 27 通りの組合せが存在し，図 6 では，すべての組合せが示されているので，グラフ上で各組合せが判別しにくくなっている．したがって，f が 20〜80%の変化の間に，最も効率の良い組合せになることがない組合せを省略したグラフを，パターン L，I，H について，それぞれ**図 7**，**図 8**，**図 9** に示す．ただし，図 7 に関しては，f の値にかかわらず SSS が最も効率の良い組合せなので，SSS 以外の効率の良い組合せを参考のために，グラフ上に示している．また，図 7，図 8，図 9 から最も効率の良い組合せを**表 4** に，コストモデルから求めた効率の良い組合せを**表 5** に示す．表 4，表 5 からパターン I の f が 30%の場合以外は，コストモデルから正しい組合せを求めることができることが分かる．

はじめに，パターン L については，メソッドの返り値の割合 f が 20〜80%のすべての場合において，SSS が最も効率が良い組合せである．これは，サーバの負荷が低いときは，各サーバによる並列処理の効果および，メソッドの実行結果のみをクライアントに転送することによる転送コストの削減による効果が現

表 2 各パターンの ρ_1, ρ_2, ρ_3 の組合せ
Table 2 The combination of ρ_1, ρ_2, and ρ_3 for each pattern.

パターン	ρ_1	ρ_2	ρ_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

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	I
D_{S_i} $(pages)$	1009(≃7.88 MB)	-	-
NW $(pages/sec)$	155.38(≃1243 KB)	155.38(≃1243 KB)	155.38(≃1243 KB)
DW_{S_i}, DW_C $(pages/sec)$	179.84(≃1439 KB)	143.37(≃1147 KB)	-
$PT_{S_i}, PT_C, PT_I (pages/sec)$	532.5(≃4.16 MB)	368.2(≃2.88 MB)	369.0(≃2.88 MB)

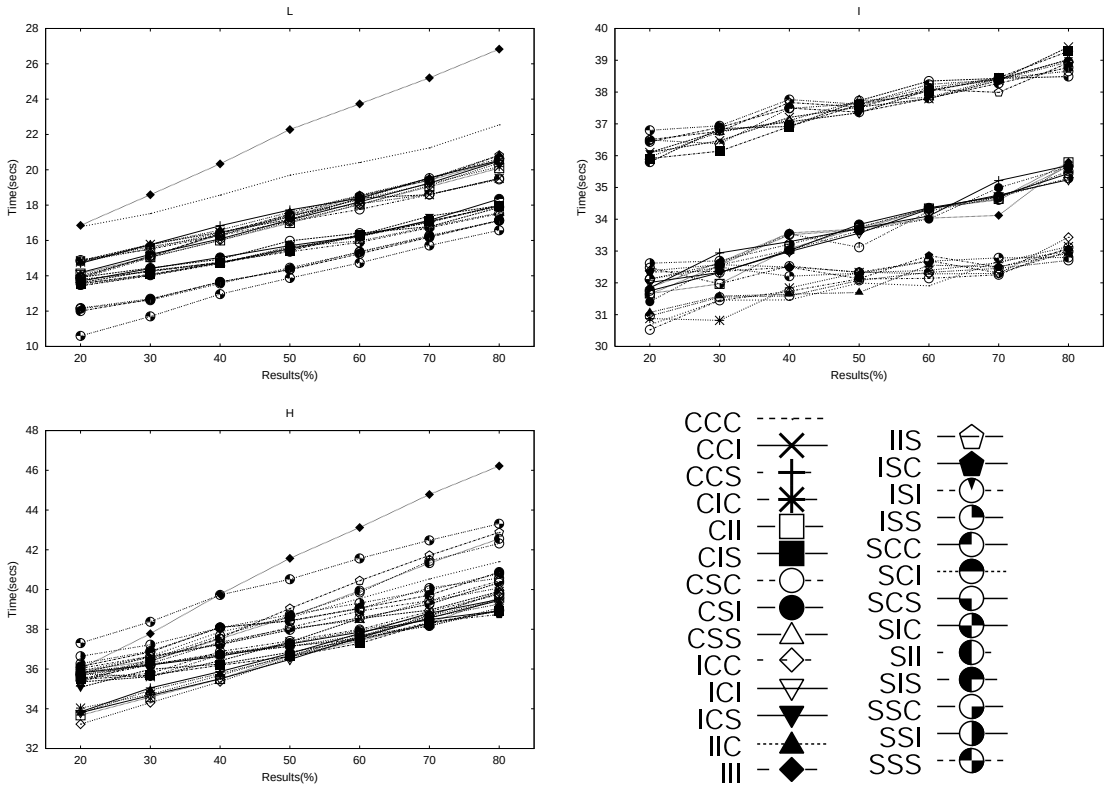

図 6 パターン L（高負荷），I（中間），H（低負荷）の結果
Fig. 6 The results of patterns: $L(LowLoad)$, $I(IntermediateLoad)$ and $H(HighLoad)$.

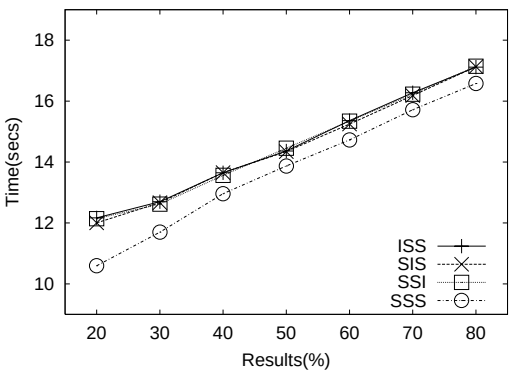

図 7 パターン L（低負荷）の実験結果
Fig. 7 The results of pattern: $L(LowLoad)$.

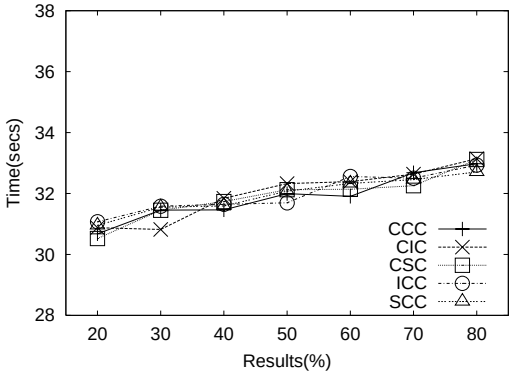

図 8 パターン I（中間）の実験結果
Fig. 8 The results of pattern: $I(IntermediateLoad)$.

れている．パターン L では，f の増加に対してレスポンスタイムの増加が一定の割合であることが分かる．これは，レスポンスタイムに占めるサーバの負荷によるオーバヘッドが小さいためである．表 4，表 5 から，パターン L では f が 20〜80%のすべての場合で正しく効率の良い組合せを決定している．また，図 7 から，27 通りの組合せの中でも，サーバでメソッドを実行する組合せである ISS，SIS，SSI 等が効率が良いことが分かる．

次に，パターン I について考察する．パターン I では，負荷の高い S_3 に対して，C でメソッドを実行するのが効率が良いという結果が出ている．これは，S_3 の負荷を避けることができるからである．パターン I では f の増加に対してレスポンスタイムの増加が小さい．これは，組合せに C が多く含まれる場合，実行結果の転送コスト削減効果が現れないことが原因

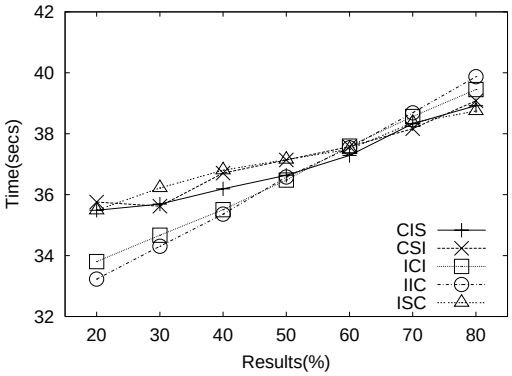

図 9 パターン H（高負荷）の実験結果
Fig. 9 The results of pattern: $H(HighLoad)$.

表 4 実験から求めた効率の良い組合せ
Table 4 Efficient combinations obtained with experiments.

$100 \times f(\%)$	20	30	40	50
L(低負荷)	SSS	SSS	SSS	SSS
I(中間)	CSC	CIC	CCC	ICC
H(高負荷)	IIC	IIC	IIC	ICI
$100 \times f(\%)$	60	70	80	
L(低負荷)	SSS	SSS	SSS	
I(中間)	CCC	CSC	SCC	
H(高負荷)	CIS	CSI	ISC	

表 5 コストモデルから求めた効率の良い組合せ
Table 5 Efficient combinations obtained with the cost-model.

$100 \times f(\%)$	20	30	40	50
L(低負荷)	SSS	SSS	SSS	SSS
I(中間)	CSC	CSC	CCC	ICC
H(高負荷)	IIC	IIC	IIC	ICI
$100 \times f(\%)$	60	70	80	
L(低負荷)	SSS	SSS	SSS	
I(中間)	CCC	CSC	SCC	
H(高負荷)	CIS	CSI	ISC	

である．f の値にかかわらず，C でメソッドを処理し，負荷の高い S_3 でのメソッドの実行を避けることによって効率良く処理できることが分かる．パターン I では f が 30%のときに，コストモデルによる見積りを誤る．実験で効率の良い組合せである CIC とコストモデルで効率の良い組合せである CSC の差は，わずかであるので実用上問題ないと考えられる．

最後に，パターン H の場合については，図 9 から，f が 20〜50% までは IIC，ICI が効率が良く，60〜80%では CIS，CSI，ISC が効率が良いことが分かる．f が低いときは，結果の転送コスト削減によってアイドル計算機に永続オブジェクトおよびメソッドを転送するオーバヘッドを犠牲にしても，なおサーバの負荷を避けるほうが効率が良いことを表している．一方，f が高い場合は，結果の転送コストが大きくなり，アイドル計算機に永続オブジェクトおよびメソッドを転送するコストが無視できないため，IIC，ICI の組合せは最も効率の良い組合せではなくなる．逆に，CIS，CSI，ISC の組合せのように，クライアント/サーバ/アイドル計算機で，メソッドを並列に実行することの利点を享受できる組合せが効率が良くなる．

表 4 から，パターン I の 50%，パターン H の 20〜80%の場合に，I を含む組合せが効率が良いことが分かる．これらの組合せは，今までの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みの組合せにはなかったものである．つまり，アイドル計算機をメソッドを実行するサイトとして導入することによって，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みの組合せと比較してさらなる効率化が実現できた場合がある．アイドル計算機を含む組合せが効率が良くなる場合のレスポンスタイムを**表 6** に示す．特に，サーバの負荷が高く，かつメソッドの実行結果の割合 f が低い環境では，効率化の割合が高い傾向がある．表 6 において，提案手法による組合せとは，本稿で提

表 6 アイドル計算機がある場合とない場合の比較
Table 6 The comparison between combinations with and without idle machines.

パターン	f	提案手法による組合せ	タイム (sec)	アイドル計算機がない場合の組合せ	タイム (sec)
I(低負荷)	50	ICC	31.70	CCC	31.99
H(高負荷)	20	IIC	33.23	CCS	35.35
H(高負荷)	30	IIC	34.31	CCS	35.62
H(高負荷)	40	IIC	35.36	CCS	36.42
H(高負荷)	50	ICI	36.48	SCC	37.30
H(高負荷)	60	CIS	37.29	SCC	37.94
H(高負荷)	70	CSI	38.17	SCC	38.70
H(高負荷)	80	ISC	38.75	CSC	39.76

案した手法から求めた効率の良い組合せである．アイドル計算機がない場合の組合せとは，本稿の実験において，アイドル計算機を含まない組合せの中で最も効率の良い組合せである．すなわち，従来までのデータマイグレーション・メソッドマイグレーションのみからなる組合せの中で最も効率の良い組合せと，本稿のアイドル計算機を加えた場合の効率の良い組合せを比較している．

ただし表 6 から分かるように，今回用いた実験環境では，組合せにアイドル計算機を利用した場合の効率化の向上は数%でしかない．これは，実験で用いたアイドル計算機の性能が高くないためである．しかしながら，今後期待されるネットワークの広帯域化，CPU の性能向上等を考慮すると，アイドル計算機でメソッドを実行する方法のオーバヘッドは低下し，本稿の提案手法はより効果的になるといえる．

5. ま と め

従来の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に加え，アイドル計算機を用いる処理を考え，その 3 方法の組合せによるマルチサーバ・マルチクライアント環境でのオブジェクトデータベース処理の効率化手法を提案した．また，実機による実験を行い，データマイグレーション・メソッドマイグレーションのみからなる組合せに比べ，アイドル計算機を利用することにより効率化が実現できることを明らかにした．本稿の提案手法は，今後ネットワークの広帯域化，CPU の性能向上等によりさらに効果的なものとなる．

本稿で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，本稿の成果は，LAN 等の信頼性の高いネットワーク環境を前提としているが，文献 10) 等にあるような，移動端末等で構成されるアドホックネットワーク環境等への適用も今後の課題である．

謝辞 本稿を改善するにあたり，多くの有用なコメントをいただいた査読者の方々に感謝いたします．

参 考 文 献

1) Acharya, A., Edjlali, G. and Saltz, J.: The Utility of Exploiting Idle Workstations for Parallel Computation, *Proc. ACM SIGMETRICS Conf.* (1997).
2) The Apache Jakarta Project: BCEL—Byte Code Engineering Library. http://jakarta.apache.org/bcel/
3) Aritsugi, M., Fukatsu, H. and Kanamori, Y.: Several partitioning strategies for parallel image convolution in a network of heterogeneous workstations, *Parallel Computing*, Vol.27, No.3, pp.269–293 (2001).
4) 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，情報処理学会 DBS 研究会，pp.189–196 (1998).
5) 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).
6) Cap, C.H. and Strumpen, V.: Efficient parallel computing in distributed workstation environments, *Parallel Computing*, Vol.19, No.11, pp.1221–1234 (1993).
7) Cattel, R. and Skeen, J.: Object Operations Benchmarks, *ACM Trans. Database Syst.*, Vol.17, No.1, pp.1–31 (1992).
8) 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).
9) Goel, S., Bhargava, B. and Jiang, Y.: Supporting Method Migration in a Distributed Object Database System: A Performance Study, *Proc. 29th Annual Hawaii Intl. Conf. on System Sciences*, IEEE (1996).
10) Hara, T.: Effective Replica Allocation in Ad Hoc Networks for Improving Data Accessibility, *Proc. IEEE INFOCOM 2001*, Vol.3, pp.1568–1576, IEEE (2001).
11) 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).
12) Ogawa, H., Shimura, K., Matsuoka, S., Maruyama, F., Sohda, Y. and Kimura, Y.: OpenJIT: An Open-Ended, Reflective JIT Compiler Framework for Java, *Proc. ECOOP2000*, Vol.LNCS 1850, pp.362–387 (2000).
13) 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).
14) 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).

15) Yoshida, Y., Aritsugi, M. and Kanamori, Y.: Performance Evaluation of Combining Data Migration and Method Migration in Object Database Environments, *Proc. 13th Australasian Database Conference* (*ADC2002*), pp.207–214 (2002).
16) 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).
17) 吉田裕介，有次正義，金森吉成：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わせ，情報処理学会論文誌：データベース，Vol.43, No.SIG9(TOD15), pp.68–80 (2002).

(平成 14 年 10 月 1 日受付)
(平成 14 年 12 月 17 日採録)

（担当編集委員　掛下 哲郎）

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，日本データベース学会各会員．

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，日本データベース学会等各会員．

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．