異機種分散環境におけるアイドル計算機を利用したODBの性能改善に関する考察

吉田　裕介†　　有次　正義††　　金森　吉成††

† 群馬産業技術センター　〒 379–2147 群馬県前橋市亀里町 884–1
†† 群馬大学工学部情報工学科　〒 376–8515 群馬県桐生市天神町 1–5–1
E-mail: †yosida@tec-lab.pref.gunma.jp, ††{aritsugi,kanamori}@cs.gunma-u.ac.jp

あらまし　我々は，これまでに分散環境におけるオブジェクトデータベースを対象にデータベース処理の性能改善について研究を行ってきた．永続オブジェクトのメソッドをサーバ／クライアント／アイドル計算機のいずれでも実行可能にした．これらの計算機の中から最小のレスポンスタイムを得られる計算機の組合せを決定する方法を提案し，実験により有効性を確認した．これまでの研究では，同種のアーキテクチャから成る分散環境で実験を行ってきた．本稿は，異機種分散環境において XML データに対する問合わせをアプリケーションとした実験を行った．その結果，我々の提案手法が異機種分散環境に適用できることを確認した．本稿では，異機種分散環境での実験結果について報告する．

キーワード　オブジェクトデータベース，異機種分散環境，アイドル計算機，メソッドマイグレーション，データマイグレーション

A consideration on performance improvement of ODBs in heterogeneous distributed environments with idle machines

Yusuke YOSHIDA†, Masayoshi ARITSUGI††, and Yoshinari KANAMORI††

† Gunma Industrial Technical Center,　Kamesato 884–1, Maebashi, Gunma 379–2147, Japan
†† Dept. of Computer Science, Gunma University　Tenjin-cho 1–5–1, Kiryu, Gunma 376–8515, Japan
E-mail: †yosida@tec-lab.pref.gunma.jp, ††{aritsugi,kanamori}@cs.gunma-u.ac.jp

Abstract　We have studied on performance improvement for distributed object databases. It enables us to execute methods of persistent objects at servers, clients or idle machines. We proposed the way of determining the combination of hosts which can give us the minimum response time, and we verified validity of the way by our experiments. We had made experiments with hosts which have homogeneous architectures. In our experiments of this paper, we used heterogeneous architectures and XML documents. The results show that our contribution can be applied to not only homogeneous but also heterogeneous distributed environments.

1. はじめに

コンピュータ技術の発展により，我々は企業，研究所などで多数の計算機を保有することが可能となった．これらの計算機は，通常の計算機処理に対して充分な処理能力と二次記憶装置を備えている．また，これらの計算機はネットワークに接続され，相互に通信可能である．

これらの計算機は常に処理をしているわけではなく，これらの計算機のプロセッサの多くはアイドル状態である [1]．今までに，我々の研究グループはデータベース処理をこれらのアイドル計算機に移動させ性能改善を行う手法を提案してきた [6]．これまでの研究では同種のアーキテクチャから成る計算機環境で実験を行ってきた．本稿では，異機種アーキテクチャから成る計算機環境での実験結果について報告し，我々の提案手法が異機種分散環境においても適用可能であることを示す．また，実験ではアプリケーションとして XML 文書の問合せを用いた．

本研究では図 1 に示すようなマルチサーバ・マルチクライアントの分散環境を考える．また，OLAP や XML データベースからのフィルタリング処理など，更新の処理を含まないアプリケーションを対象として，本研究で扱うデータベース処理は更新を含まないものとする．

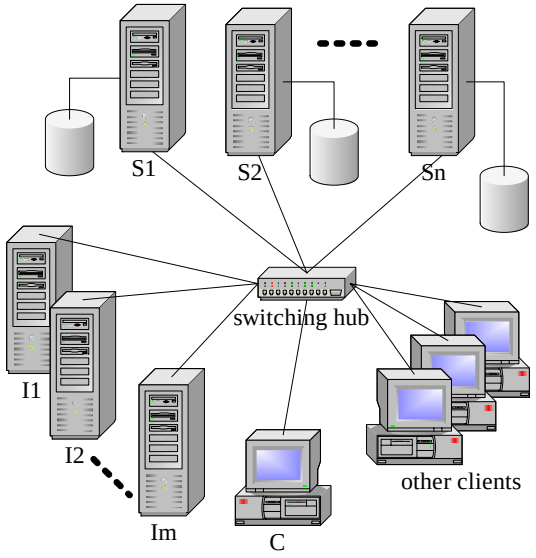

図 1　A distributed environment

2. アイドル計算機の有効利用

従来の ODB では，永続オブジェクトに対するメソッドの適用はクライアント計算機で行われてきた [2]．性能改善のために，メソッドの適用を行う計算機をサーバ計算機あるいはクライアント計算機から適応的に決定する研究が行われてきた [2], [3], [5]．我々はメソッド適用を実行する計算機の候補としてアイドル計算機を加えた [6]．つまり，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 方法を考えた．

（1） 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．

（2） 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．

（3） 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．

我々は 1, 2 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と呼んでいる [5]．

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，1 に示すような分散環境では，メソッドの適用をどの計算機で実行するかを示す組合せの総数は $(m+2)^n$ である．これら組合せの中から最小のレスポンスタイムを得られる組合せを決定することで性能改善を行う．次節では，組合せを決定するためのコストモデルについて説明する．

3. コストモデル

本節では，レスポンスタイムを見積るためのコストモデルを説明する．図 1 において，$S_1, \cdots, S_n$ 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 $S_i (1 \leqq i \leqq n)$ 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．各サイト間のネットワークバンド幅を NW(pages/sec)，サイト X のディスク I/O 速度を DW_X(pages/sec) で表す．また，サイト X が，1 秒間にオブジェクトの直列化，直列化復元できるサイズをそれぞれ $SER_X, DESER_X$ と表す．メソッドの実行結果として返されるデータの割合を f で表す．したがって，実行結果のサイズは $f \times D_X$ と表せる．サイト X が 1 秒間に処理できるデータサイズを PT_X と表す．

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．

$$T_M = M \times \frac{1}{DW_{MR}} + M \times \frac{1}{NW} \tag{1}$$

マルチクライアントの環境の場合，クライアントが複数あることによる，サーバ計算機の負荷をモデル化する必要がある．サーバ計算機が，CPU およびディスク I/O の処理能力を使用している割合を負荷率と呼ぶ．CPU, ディスク I/O の負荷率をそれぞれ ρ_{cpu}, ρ_{disk} と表すと，サーバ計算機の実際のメソッド実行能力，ディスク I/O の能力，直列化・直列化復元能力を表すパラメータはそれぞれ式 (2), (3), (4), (5) に示す $PT'_{S_i}, DW'_{S_i}, SER'_{S_i}, DESER'_{S_i}$ と表せる．

文献 [5], [6] において，直列化および直列化復元は表現されていなかった．文献 [5], [6] では，これらの処理はデータ転送のパ

```
function ResponseTimeWithIdle(M)
begin
    var S : an empty list;
    for j ← 1 to m do ID_j ← φ;
    foreach k ∈ M do begin
        if k = C then begin
            t_p ← T_M + D_{S_i} × (1/DW'_{S_i} + 1/SER'_{S_i});
            t_s ← D_{S_i} × (1/NW + 1/PT_C + 1/DSER_C)
            append the pair (t_p, t_s) into S;
        end
        else if k ∈ {S_1, S_2, ···} begin
            t_p ← T_M + D_{S_i} × (1/DW'_{S_i} + 1/PT'_{S_i} + f/SER'_{S_i});
            t_s ← f × D_{S_i} × (1/NW + 1/DSER_C);
            append the pair (t_p, t_s) into S;
        end
        else if k = I_j then
            ID_j ← ID_j ∪ {k};
        end
    end
    for j ← 1 to m do begin
        if ID_j ≠ φ then begin
            {ResponseTime in [5] is used
             where I_j is treated as C }
            t_p ← ResponseTime(ID_j, φ)
                  + Σ_{k∈ID_J} f × D_{S_k} × 1/SER_{I_j}
            t_s ← Σ_{k∈ID_j} f × D_{S_k} × (1/NW + 1/DSER_C);
            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 2　Algorithm for the response time.

ラメータ NW に含まれていた．今回，直列化および直列化復元をパラメータとして追加した理由は，扱うデータが XML 文書の DOM 木オブジェクトの場合，直列化および直列化復元の処理時間がデータ転送に占める割合が大きくなったためである．

$$PT'_{S_i} = (1-\rho_{cpu}) \times PT_{S_i} \quad (2)$$

$$DW'_{S_i} = (1-\rho_{disk}) \times DW_{S_i} \quad (3)$$

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

$$DESER'_{S_i} = (1-\rho_{cpu}) \times DESER_{S_i} \quad (5)$$

式 (6), (7) は，それぞれ n 台のサーバ計算機のデータにメソッドを適用する処理のうち並列に処理できる部分，逐次で処理する部分のコスト式を表す．式 (6) 中の $ResponseTime(X,Y)$ は，X, Y をそれぞれデータマイグレーションで処理するサーバ計算機の集合，メソッドマイグレーションで処理するサーバ計算機の集合とした場合のレスポンスタイムを計算する関数である [5].

データマイグレーションで処理する場合，並列に処理する部分の時間（式 (6)）は，メソッドを移動する時間，サーバ計算機でのディスク I/O の処理時間，クライアントにデータを転送するための直列化の処理時間の合計となる．逐次で処理する部分の時間（式 (7)）は，サーバからのデータを受信する処理時間，受信したデータを直列化復元するための処理時間，クライアント計算機でメソッドを適用する処理時間の合計となる．

メソッドマイグレーションで処理する場合，並列に処理する部分の時間（式 (6)）は，メソッドを移動する時間，サーバ計算機でのディスク I/O の処理時間，サーバ計算機でメソッドを適用する処理時間，クライアントにメソッド処理結果を転送するための直列化の処理時間の合計となる．逐次で処理する部分の時間は，メソッドの実行結果を受信する処理時間，実行結果を直列化復元する処理時間の合計である．

アイドル計算機で実行する場合，並列に処理する部分の時間（式 (6) は，アイドル計算機をクライアント計算機とみなしてデータマイグレーションで処理する時間とメソッドの実行結果を当該サーバの台数分を直列化する処理時間の合計となる．逐次で処理する部分の時間は，実行結果を受信する時間，直列化復元する処理時間を当該サーバの分合計した時間である．

$$T_{parallel}[i] = \begin{cases} T_M + D_{S_i} \times (\frac{1}{DW'_{S_i}} + \frac{1}{SER'_{S_i}}) \\ \qquad (data\ migration) \\ T_M + D_{S_i} \times (\frac{1}{DW'_{S_i}} + \frac{1}{PT'_{S_i}} + \frac{f}{SER'_{S_i}}) \\ \qquad (method\ migration) \\ ResponseTime(ID_j, \phi) \\ \quad + \sum_{k \in ID_J} f \times D_{S_k} \times \frac{1}{SER_{I_j}} \\ \qquad (processing\ on\ I_j) \end{cases} \quad (6)$$

$$T_{serial}[i] = \begin{cases} D_{S_i} \times (\frac{1}{NW} + \frac{1}{PT_C} + \frac{1}{DSER_C}) \\ \qquad (data\ migration) \\ f \times D_{S_i} \times (\frac{1}{NW} + \frac{1}{DSER_C}) \\ \qquad (method\ migration) \\ \sum_{k \in ID_j} f \times D_{S_k} \times (\frac{1}{NW} + \frac{1}{DSER_C}) \\ \qquad (processing\ on\ I_j) \end{cases} \quad (7)$$

式 (6), (7) を用いて，レスポンスタイムを計算するアルゴリズム $ResponseTimeWithIdle()$ を図 2 に示す．図 2 において引数の M は，図 2 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．$\boldsymbol{M}$ 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 1 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．

2. 節で述べたように，サーバ n 台，アイドル計算機 m 台ある環境では，メソッドの適用をどの計算機で実行するかを示す組合わせの総数は $(m+2)^n$ 通り存在する．

図 2 のアルゴリズムを基に，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，全ての組合せを図 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 [5]．$T \leftarrow ResponseTime(\boldsymbol{DM}, \boldsymbol{MM})$ 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(注1)．

（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，$\boldsymbol{M} \leftarrow$ (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

(注1)：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．

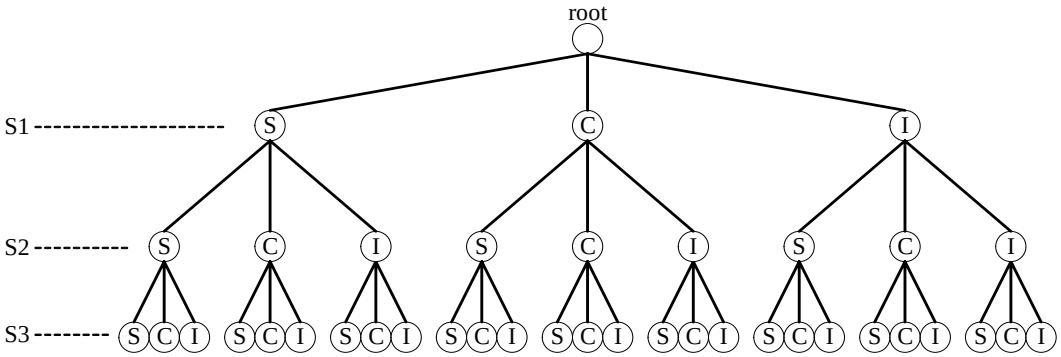

図 3　サーバ 3 台，アイドル計算機 1 台の場合の組合せを表現する木

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．

4. 実　　験

前節で述べたコストモデルを検証するために，実験を行った．本節では，実験と検証結果について述べる．

4.1 実験環境

実験で用いた環境を表 1 に示す．S_1, S_2, S_3 をサーバ計算機，C をクライアント計算機，I をアイドル計算機として用いた．サーバ計算機は，同種のアーキテクチャのワークステーションを用いた．クライアント計算機はサーバ計算機よりも性能が低いワークステーションとした．アイドル計算機は，サーバ計算機・クライアント計算機とは異なるアーキテクチャを使用している．パラメータ PT_{S_i}, PT_C, PT_I の値はメソッド毎に異なるが，本稿の実験ではあらかじめ実測した値を用いている．表 2 に実測したパラメータを示す．

実験では XML データに対する問合わせをアプリケーションとして用いた．XML 文書から特定の条件を満たすタグを選択し，新たな XML 文書として返す処理を実装した．利用した XML 文書は Xmark プロジェクト [4] が提供するベンチマーク用 XML 文書生成プログラムから生成した XML 文書を用いた．

実験で用いたメソッドは，データベースから XML 文書を取りだし，解析を行い DOM 木を構築する．構築した DOM 木に対して特定の条件を満たすタグを探し出す．DOM 木のオブジェクト表現をネットワークで転送するためには DOM 木のオブジェクト表現を直列化する．

しかしながら，Java 処理系の標準直列化アルゴリズムでは DOM 木を直列化することは困難である．一般的に DOM 木は多数のノードから構成されるため，DOM 木のオブジェクト表現は非常に多数のオブジェクトを持つ大きなオブジェクトグラフになる．Java 処理系が標準で用いるオブジェクトの直列化アルゴリズムの実装は，関数の再帰呼び出しを使ってオブジェクトの参照を探索し直列化を行う．したがって，Java 処理系の標準直列化アルゴリズムで DOM 木のオブジェクト表現を直列化を行うと，Java 処理系のスタック領域を使い果たしてしまう．

この問題を解決するために，本稿の実験システムでは，DOM 木のオブジェクト表現を直列化するために XML パーザを用いている．DOM 木オブジェクトを属性とするラッパクラスに，DOM 木を XML 文書に変換する `writeObject()` メソッドを実装することで，DOM 木の直列化にはこれらのメソッドが使用される．同様に，直列化復元のために，XML 文書を DOM 木に変換する `readObject()` を実装している．

4.2 実験結果と考察

表 3 に実験結果を示す．表 3 は，実験から求めた効率の良い組合せ，本稿の提案手法で述べた方法で求めた組合せ，および両組合せが異なった場合の誤差率を示している．誤差率とは，効率の良い組合せを間違えたときの損失を表す指標であり，次の式で定義される．

$$\text{誤差率} = \frac{X - Y}{Y} \tag{8}$$

X は，アルゴリズムで求めた組合せで実行した場合の実際にかかった時間であり，Y は実際に効率の良い組合せで実行した場合の時間である．

実験では，サーバ計算機の負荷について 3 通りの環境を想定した．3 台のサーバ計算機が全て負荷の高い状況を「高負荷環境」とし，S_1, S_2, S_3 の負荷率をそれぞれ $0.8, 0.8, 0.8$，サーバ計算機の負荷が異なる状況を「負荷混在環境」とし，S_1, S_2, S_3 の負荷率をそれぞれ $0.2, 0.5, 0.8$，負荷のない環境を「無負荷環境」とし，S_1, S_2, S_3 の負荷率をそれぞれ $0.0, 0.0, 0.0$ とした．CPU の負荷率とディスク I/O の負荷率は，同一のサーバ計算機では同じ値とした．

高負荷環境では，$f = 0.2, 0.3, 0.5, 0.8$ のとき正しい組合せを求めることができた．また，組合せを間違えた場合の誤差率の平均は 0.0127 となった．この誤差率は，充分小さく実用的であると考えられる．返すデータの割合 f が 0.2 〜 0.6 のときはサーバ計算機で実行した方が効率が良くなっている．永続オブジェクトをネットワークで転送するために，オブジェクトの直列化が必要である．アイドル計算機で実行する場合にはデータ全体を直列化する処理が高コストになるため，アイドル計算機で実行するよりもサーバ計算機で実行する組合せが効率が良い．

負荷混在環境では，$f = 0.2, 0.3, 0.4, 0.7$ の場合で正しい組合せを求めることができた．ただし，組合せを間違えた場合の誤差率は，高負荷環境の誤差率と比較して大きくなった．f が小

Site	S_1, S_2, S_3	C	I
Type	Sun Ultra30	Sun Ultra1	PC
CPU	UltraSparcII	UltraSparcI	PentiumIII
Clock	248MHz	167MHz	800MHz
Memory	128MB	128MB	128MB
Disk	SUN4.2G	SUN2.1G	Quantum ATALS_9_SCA
Page size	8192(bytes)		
OS	Solaris2.6	Solaris2.6	Solaris8(x86)
Java 処理系	J2SE1.4.2		
DBMS	Berkeley DB 4.0.14		

表 1　configuration

$(i = 1, 2, 3)$	S_1, S_2, S_3	C	I
D_{S_i} $(pages)$	463(≃3.62MB)		
NW $(pages/sec)$	437.14(≃3.42MB)	437.14(≃3.42MB)	437.14(≃3.42MB)
DW_{S_i}, DW_C $(pages/sec)$	135.420(≃1083KB)	95.839(≃766KB)	369.248(≃2.884MB)
PT_{S_i}, PT_C, PT_I $(pages/sec)$	163.778(≃1310KB)	118.688(≃950KB)	381.227(≃2.97MB)
SER_{S_i}, SER_C $(pages/sec)$	129.510(≃1036KB)	88.885(≃711KB)	452.458(≃ 3.53MB)
$DSER_{S_i}, DSER_C(pages/sec)$	188.672(≃1509KB)	132.286(≃1058KB)	686.740(≃5.36MB)

表 2　Parameter settings

	f	0.2	0.3	0.4	0.5	0.6	0.7	0.8
高負荷環境	実験	SSS	SSS	SSS	ISS	SSS	CSS	ISC
	計算	SSS	SSS	ISS	ISS	ISS	ISC	ISC
	誤差率	-	-	0.016	-	0.011	0.011	-
負荷混在環境	実験	IIS	IIS	IIS	CIS	SSS	CCI	CSC
	計算	IIS	IIS	IIS	CSI	CSI	CCI	CCI
	誤差率	-	-	-	0.099	0.084	-	0.026
無負荷環境	実験	SSS	SSS	SSS	SSS	SSS	SSS	SSS
	計算	SSS	SSS	SSS	SSS	SSS	SSS	SSS
	誤差率	-	-	-	-	-	-	-

表 3　results

さなときは，負荷の低い S_1, S_2 に対する処理はアイドル計算機で実行した方が効率が良くなる．これは，負荷の低い S_1, S_2 では，永続オブジェクトの直列化のオーバーヘッドを高性能なアイドル計算機で実行する性能改善が上回ったため，アイドル計算機で実行した方が効率が良くなった．

無負荷環境では，全ての場合において正しい組合せを求めることができた．

全体としては，21 通りの実験のうち 15 通りで正しい効率の良い組合せを求めることができた．これは全体の 76.2%に相当する．また，全体の平均誤差率は 0.041 であった．文献 [5] では，正しい組合せを求められた割合は約 78.2%，平均誤差率は 0.03 であったから，異機種分散環境においても同等の有効性を確認できた．

5. ま と め

いままでに，分散オブジェクトデータベースの性能改善のためにアイドル計算機を利用する方法を提案してきた．その提案手法を異機種分散環境，および従来より実際の環境に即したアプリケーションに対して適用できることを確認した．

文　献

[1] M. Aritsugi, H. Fukatsu, and Y. Kanamori, “Several Partitioning Strategies for parallel image convolution in a network of heterogenceous worksations,” Parallel Computing, Vol. 27, No. 3, pp.269–293, 2001.

[2] M.J. Franklin, B.T. Jónsson, and D. Kossmann, “Performance tradeoffs for client-server query processing,” Proc. ACM SIGMOD Conf., pp.149–160, June 1996.

[3] M. Rodríguez-Martínez and N. Roussopoulos, “MOCHA: A self-extensible database middleware system for distributed data sources,” Proc. ACM SIGMOD Conf., pp. 213–224, May 2000.

[4] XMark - An XML Benchmark Project, http://www.xml-benchmark.org/

[5] 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).

[6] 吉田 裕介， 有次 正義， 金森 吉成，“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,” 情報処理学会：データベース, Vol. 44, No. SIG3(TOD17), pp.22–32, (2003).