ストリーム指向型並列データベース処理を対象とした分散メモリ資源割り当て方式

村岡正則[†,☆] 佐藤聡[††] 清木康[†††]

データベースの多様な応用分野に適応可能な並列処理方式として，ストリーム指向型並列処理方式を提案している．この方式では，問い合わせを構成するデータベース演算群へのメモリ資源の割り当てが処理効率に大きな影響を与える．本論文では，共有メモリをもたない分散処理環境において，任意のデータベース演算から構成される問い合わせをストリーム指向型並列処理方式により，並列に処理する場合に，問い合わせを効率よく実行するためのメモリ資源割り当て方式を提案する．また，実際の問い合わせに適用した場合の割り当て結果，および，問い合わせの処理時間の予測値を示し，本論文で提案するメモリ資源割り当て方式の有効性を明らかにする．

A Distributed Memory Allocation Method for Stream-oriented Parallel Database Processing

MASANORI MURAOKA,[†,☆] AKIRA SATO[††] and YASUSHI KIYOKI[†††]

We have proposed a stream-oriented parallel processing scheme which is applied to various application areas of databases. In this scheme, it is important to perform optimization for memory resource allocation in query processing for arbitrary combination of various database operations. This paper presents an efficient computational method for obtaining optimal distributed memory resource allocation for a query consisting of an arbitrary set of database operations in distributed query processing environments. Furthermore, this paper shows several experimental results for computations of actual memory resource allocation and clarifies the effectiveness of the proposed method.

1. はじめに

データベースの問い合わせ処理において，メモリ資源は，ソースデータや中間結果の格納，および，二次記憶へのアクセス回数を軽減するためのI/Oバッファとして用いられる．メモリ資源の割り当ては，問い合わせの処理効率に重大な影響を与えることが広く認識されており，最適なメモリ資源割り当てのための方法が数多く提案されている[2),4),7),11),13)]．それらの方法は，関係データベースの問い合わせ処理を対象としている．

関係データベースシステムにおいて，問い合わせは，データベース演算の組み合わせによって構成される．使用されるデータベース演算は，固定的に定められた関係データベース演算である．したがって，従来のメモリ資源割り当て方式では，各関係データベース演算の性質に依存したメモリ資源割り当てを求めることにより，問い合わせの処理効率を向上させる方法を用いていた．しかし，近年，データベースの応用分野が拡大するにつれ，対象とする応用分野に適したデータ構造やデータベース演算を柔軟に定義する機能が，データベース・システムに求められている．それらの機能を提供するデータベース・システムでは，利用されるデータベース演算が固定化されていないため，データベース演算に依存したメモリ資源割り当て方法を確定することは困難である．したがって，任意のデータベース演算を対象としたメモリ資源割り当ての方法が必要となる．

データベース・システムの応用分野の多様化ととも

† 筑波大学 情報学類
College of Information Sciences, University of Tsukuba
†† 筑波大学 工学研究科
Doctoral Program in Engineering, University of Tsukuba
††† 筑波大学 電子・情報工学系
Institution of Information Sciences and Electronics, University of Tsukuba
☆ 現在，(株) 日立製作所 システム開発研究所
Presently with System Development Laboratory, Hitachi, Ltd.

に，データベース処理を行うハードウェア環境も大きく変化している．ハードウェア技術の進歩に伴い，マルチプロセッサ・システムおよび，ネットワークを用いた並列・分散処理環境が容易に利用できるようになっている．これらの環境を利用して，データベース処理を高速に行うための研究も活発である[1),3),5),6)]．これらの環境を利用した問い合わせ処理においても，メモリ資源割り当ては，その効率に大きな影響を与える．したがって，このような環境を用いた問い合わせ処理を効率よく行うためのメモリ資源割り当ての方法が必要となる．

本論文では，ストリーム指向型並列処理方式を対象とした分散メモリ資源割り当て方式を提案する．ストリーム指向型並列処理方式は，データベースの多様な応用分野および並列・分散処理環境に適応可能な並列処理方式として提案されているものである[6),12)]．

文献 7), 9), 10) において，ストリーム指向型並列処理方式を対象とした共有メモリ資源割り当て方式およびその拡張方式が提案されている．これらの方式は，単一のプロセッサ・システム，または，共有メモリをもつマルチプロセッサ・システム上において，ストリーム指向型並列処理方式による問い合わせ処理を行う場合に，その処理時間を最短にするメモリ資源割り当てを求めるための方式である．

本論文では，共有メモリ資源をもたない並列処理環境を対象としたメモリ資源割り当て方式を提案する．本方式の特徴は，問い合わせの実行効率に影響するすべての処理の要素，すなわち，入出力処理，演算処理，および，プロセス切換に要する時間を対象とした最適化を実現できる点にある．

2 章では，ストリーム指向型並列処理方式を概説する．3 章では，ストリーム指向型並列処理方式を対象とした分散メモリ資源の割り当て方式を提案する．4 章では，3 章で示す方式の有効性を評価するために行った実験の結果を示し，提案方式の有効性を明らかにする．

2. ストリーム指向型並列処理方式

ストリーム指向型並列処理方式とは，ストリームを入出力とする関数群から構成される式を要求駆動型評価方式を用いて並列に処理する方式である．この方式では，データベース本体，および，問い合わせの中間結果は，ストリームとして表現され，データベース演算は，ストリームを引数とする関数として定義される．問い合わせは，それらの関数を組み合わせることによって構成される．問い合わせの発行時には，各データベース演算は関数インスタンスとして生成され，それらの関数インスタンス群によって構成された問い合わせは，関数型計算の枠組の中で，要求駆動型評価によって並列に処理される．具体的には，次の 2 種類の並列性が抽出される[6),12)]．

(1) 関数引数の並列評価による並列性
(2) 引数の生成側の関数と消費側の関数間での並列性（ストリーム型並列性）

以下，中間結果のストリームを生成する関数インスタンスを生成者関数インスタンス，それらを消費する関数インスタンスを消費者関数インスタンスとよぶ．

この方式では，生成者関数インスタンスと消費者関数インスタンスの間において，ストリーム要素，および，デマンドが送受される．この方式では，1 デマンドに対し，1 グレイン分のストリーム要素群が生成される．1 グレイン分のストリーム要素を格納できる大きさのバッファ領域が，生成者と消費者関数インスタンスに設定される．また，ストリームによって接続される関数インスタンス群が異なるプロセッサに配置される場合，そのストリームには，ダブルバッファリング機構が必要となる．1 グレイン分のストリーム要素を格納する大きさのバッファ領域が 2 領域分設定される．これにより，ストリーム要素の生成と消費が同時に行われ，ストリーム型並列性を抽出することができる．

生成者関数インスタンスと消費者関数インスタンスの間にダブルバッファリング機構が設定されている場合における，消費者関数インスタンスのアルゴリズムを次に示す．

Step-1 消費対象のストリーム要素群が格納されているバッファ領域を消費対象領域とよぶ．消費対象領域に対するデマンドを先行的に発行する．

Step-2 消費対象領域に格納されている最初のストリーム要素を参照する時に，他のバッファ領域に対するデマンドを先行的に発行する．そして，消費対象領域のすべてがストリーム要素群によって満たされていないならば，実行を停止する．

Step-3 消費対象領域のすべてがストリーム要素群によって満たされた時点で実行を再開し，消費対象領域に格納されているストリーム要素群に対する処理を実行する．そして，消費対象領域に格納されているストリーム要素群に対する処理が完了すると，他のバッファ領域を消費対象領域として，**Step-2** に戻る．

生成者関数インスタンスは，デマンドを受けとると，そのデマンドに対応する 1 グレイン分のストリーム要素群を格納すべきバッファ領域がストリーム要素群に

よって満たされるまで，実行を続ける．

ストリーム指向型並列処理方式において，グレイン・サイズの設定は，関数インスタンスの実行に大きな影響を与える．ストリームのグレイン・サイズを全ストリーム要素数に設定した場合，そのストリームの消費者関数インスタンスは，生成者関数インスタンスが全要素を生成し，バッファ領域を満たすまで，待ち続けることになる．すなわち，このグレイン・サイズの設定では，ストリーム並列性が抽出されないことになる．したがって，並列性を抽出するためには，グレイン・サイズを全要素数より小さく設定する必要がある．一方，グレイン・サイズをより小さく設定すると，デマンド転送の回数，および，関数間のコンテキスト切替えの回数が増加し，その結果，転送のオーバヘッドが大きくなる．最適なメモリ資源割り当てを求めることは，最適なグレイン・サイズを決定することに対応する．提案方式では，グレイン・サイズを小さく設定することによる並列性の向上と，それを大きく設定することによって生じるオーバヘッドの軽減の両者を考慮した最適な各グレイン・サイズを，問い合わせの各バッファについて決定することができる．

また，ストリーム指向型並列処理方式では，各サイトに複数の関数インスタンスを割り当てることができる．問い合わせを構成する各関数インスタンスのサイト割り当てを行う場合，データベースを直接操作する関数や，問い合わせ結果を出力する関数は，割り当てに対して制約が存在する．関数の最適なサイト割り当てを決定する際には，この割り当てに関する制約を満たさなければならない．

メモリ資源割り当て，および，サイト割り当てを適切に行うことにより，ストリーム指向型並列処理方式による並列性の抽出を効果的に実現することができる．提案するメモリ資源割り当て方式は，サイト割り当てが行われた問い合わせに対して，そのサイト割り当てにおいて並列性を最も効果的に抽出できるメモリ割り当てを求めることができる．すなわち，このメモリ資源割り当て方式は，関数インスタンス群のサイト割り当て方式とは独立に適用することができる．

3. メモリ資源割り当て方式

本章では，木構造の問い合わせに対して，各関数インスタンス間に設定されるバッファへのメモリ割り当てを求める方式を提案する．木構造の問い合わせを構成する各関数インスタンスのサイト割り当ては，**図1**に示すように，チェイン型を構成するサイト割り当てとする．

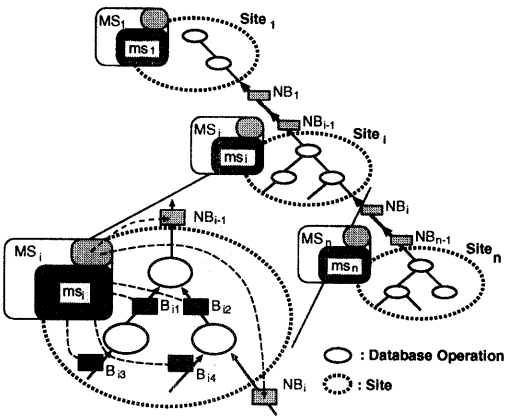


図1 分散処理環境におけるメモリ資源割り当て
Fig. 1 Distributed memory resource allocation in a distributed query processing environment.

本メモリ資源割り当て方式の目的は，分散処理環境上において，一つの問い合わせを並列に処理する場合，その問い合わせ処理の開始から終了までの時間（問い合わせの処理時間）を最短にすることである．

ストリーム指向型並列処理におけるメモリ資源割り当て方式として，次の2種類が考えられる．

(1) 静的メモリ資源割り当て方式
問い合わせが与えられたとき，あらかじめ獲得している統計情報をもとに，静的にメモリ資源を割り当てる方式．

(2) 動的メモリ資源割り当て方式
問い合わせ処理の実行時に，各関数インスタンスの実行状態をモニタリングすることにより，統計情報を獲得し，問い合わせ処理の実行途中において，メモリ資源割り当て計算を行った後，メモリ資源の動的割り当てを行う方式．

文献8)において，関係データベース処理を対象とした動的メモリ資源割り当てを行う場合，あらかじめ，静的メモリ資源割り当て方式を適用することにより，より効率的な問い合わせ処理が行われることが示されている．

本論文では，静的メモリ資源割り当て方式に着目して議論する．

分散処理環境上において，ストリーム指向型並列処理方式を用いて問い合わせを並列に処理する場合，問い合わせを構成する関数インスタンス群は，各サイトに分散して配置される．したがって，分散処理環境における問い合わせの処理時間は，あるサイトがそのサイトに配置された関数インスタンスを実行している時間（実行時間）と，そのサイトがアイドル状態になっている時間（アイドル時間）の合計である．各サイト

における実行時間とアイドル時間の合計は，同一となる．本論文において提案するメモリ資源割り当て方式は，各サイトにおける実行時間とアイドル時間の合計を最短にするための最適なバッファ・サイズを求める方式である．

3.1 メモリ資源割り当て問題

本方式では，問い合わせの処理時間が，バッファ・サイズをパラメータとする関数として表される．各サイトにおいて用意されるバッファには，ネットワークを介したストリーム要素の受渡しに用いられるバッファ（NB_{i-1}，NB_i）(以下，ネットワーク・バッファとよぶ）とそれ以外の m_i 個のバッファ（B_{i1}，B_{i2}，…，B_{im_i}）がある（図1）．ネットワーク・バッファには，ダブルバッファリング機構が実現される．本方式では，各サイト i におけるネットワーク・バッファ NB_{i-1}, NB_i 以外の各バッファ（$B_{i1}, B_{i2}, \ldots, B_{im_i}$）で利用されるメモリの総量（$ms_i$）(以下，メモリ・サイズとよぶ）が与えられたとき，それらの各バッファの大きさ（$bs_{i1}, bs_{i2}, \ldots, bs_{im_i}$）の最適値を示すバッファ・サイズ・テーブル（**図2**）が文献7) および10) に示した方式により求められているものとする．これをサイト内最適化とよぶ．したがって，問い合わせの処理時間は，各サイトにおいて利用されるメモリ・サイズ ms_i，および，各サイト間に用意されたバッファ NB_i のサイズ（nbs_i）を変数とする関数として表される（ここで，nbs_i は，1グレイン・サイズを表す変数とする）．

本方式が対象とするメモリ資源割り当ての問題は，次のように定義することができる．

目的関数

$$\min T_{QE}(ms_1, \ldots, ms_n, nbs_1, \ldots, nbs_{n-1})$$

制約条件

$$0 < ms_1 + nbs_1 \leq MS_1,$$
$$0 < nbs_{i-1} + ms_i + nbs_i \leq MS_i \quad (2 \leq i \leq n-1),$$
$$0 < nbs_{n-1} + ms_n \leq MS_n,$$

このとき，

ms_i	bs_{i1}	bs_{i2}	…	bs_{im_i}
1				
2				
⋮				
MS_i				

図2 バッファ・サイズ・テーブル
Fig. 2 Buffer size table.

$$ms_i = \sum_{j=1}^{m_i} bs_{ij} \quad (1 \leq i \leq n),$$

n： サイトの総数，
T_{QE}： 問い合わせ処理全体にかかる時間を表す関数，
ms_i： サイト i に割り当てるメモリ・サイズの変数，
MS_i： サイト i において利用可能なメモリ・サイズ，
nbs_i： サイト i とサイト $i+1$ の間のバッファ・サイズ（NB_i の大きさ），
bs_{ij}： サイト i のバッファ B_{ij} の大きさ，
m_i： サイト i の NB_i，NB_{i-1} 以外のバッファの個数．

この問題では，問い合わせの処理時間を表す関数 T_{QE} を設定する必要がある．その関数 T_{QE} の変数である ms_1，…，ms_n，nbs_1，…，nbs_{n-1} の取り得る値すべてについて，その関数値を計算することにより，最適なバッファ・サイズを求める方法が考えられる．しかし，その方法は，計算量が膨大となるため現実的ではない．本方式は，この目的関数を式 (1) に示すように複数個の式を組み合わせた形で与え，各式について順に，割り当てを求めていくことにより，最適なメモリ資源割り当てを求るものである．この方式により，最適なメモリ資源割り当てを求めるための計算量を減らすことが可能となる．

$$\begin{aligned} T_{QE} &= y_1, \\ y_1 &= F_1(X_1, y_2), \\ y_2 &= F_2(X_2, y_3), \\ &\vdots \\ y_i &= F_i(X_i, y_{i+1}), \\ &\vdots \\ y_n &= F_n(X_n). \end{aligned} \tag{1}$$

ここで，

$$\bigcup_{i=1}^{n} X_i = \{ms_1, \cdots, ms_n, nbs_1, \cdots, nbs_{n-1}\}.$$

3.2 目的関数の構成

問い合わせを処理する各サイトのプロセッサのタイムチャートは，**図3** のように表される．問い合わせの処理時間 T_{QE} は，ある1サイトにおいて，プロセッサが実行状態にある時間の合計 T_{SE} とプロセッサがアイドル状態にある時間の合計 T_{SI} との和として表すことができる．

$$T_{QE} = T_{SE_i} + T_{SI_i} \quad (1 \leq i \leq n), \tag{2}$$

T_{SE_i}： サイト i の実行時間，
T_{SI_i}： サイト i のアイドル時間，

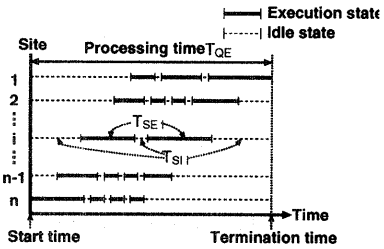


図3 問い合わせ処理のタイムチャート
Fig. 3 Time chart of query processing.

n： サイトの総数.

式(2)において，サイト i におけるアイドル時間 T_{SI_i} は，次の3要素に分解できる（式(3)）.

$$T_{SI_i} = T_{SIb_i} + T_{SIx_i} + T_{SIe_i} \quad (3)$$

T_{SIb_i}： 問い合わせ処理の開始時におけるアイドル時間．これは，問い合わせ処理の開始時において NB_i に最初のストリーム要素群が格納されるまでの間に，サイト i のプロセッサがアイドル状態となっている時間を表す.

T_{SIe_i}： 問い合わせ処理の終了時におけるアイドル時間．これは，サイト i に配置されているすべての関数インスタンスが終了してから，問い合わせ処理全体が終了するまでの時間を表す.

T_{SIx_i}： T_{SIb_i}, T_{SIe_i} 以外のアイドル時間

式(3)において，T_{SIb_i} と T_{SIe_i} は問い合わせの形とバッファ・サイズのみに依存して発生するアイドル時間である．したがって，これらを表す一般的な式を導き出すことは容易である．しかし，T_{SIx_i} は，問い合わせの中間結果が生成されるタイミングや各サイトのスケジューリングの方法などに依存して発生するアイドル時間であるため，それを表す一般的な式を導き出すことは困難である.

そこで，次の方法によって，T_{QE} を定式化する.

この定式化において，ボトルネック・サイトとは，そのサイトにおけるプロセッサの実行時間 T_{SE}，問い合わせ処理の開始時におけるアイドル時間 T_{SIb}，および，問い合わせの終了時におけるアイドル時間 T_{SIe} の合計時間が最も長いサイトとする．そして，問い合わせの処理時間 T_{QE} は，ボトルネック・サイトの T_{SE}, T_{SIb}，および，T_{SIe} の合計時間とする．したがって，サイト i における T_{SE_i}, T_{SIb_i}，および，T_{SIe_i} の合計を T_{QE_i} とおくと（式(4)），問い合わせの処理時間 T_{QE} は，T_{QE_i} $(1 \le i \le n)$ の最大値として表される（式(5)）.

$$T_{QE_i} = T_{SE_i} + T_{SIb_i} + T_{SIe_i} \ (1 \le i \le n), \quad (4)$$

$$T_{QE} \approx \max_{1 \le i \le n} T_{QE_i} = \max_{1 \le i \le n} \{T_{SE_i} + T_{SIb_i} + T_{SIe_i}\}. \quad (5)$$

式(5)における時間を表す変数 T_{SE}, T_{SIb}, T_{SIe} を，各サイトが利用するメモリ・サイズ ms_i $(1 \le i \le n)$，および，サイト間のバッファ・サイズ nbs_i $(1 \le i \le n-1)$ を変数とする関数として定義する．T_{SE_i} は，他のサイトのメモリ割り当てには依存しないため，ms_i, nbs_{i-1}, nbs_i を変数とする関数として定義される．また，T_{SIb_i} は，サイト i+1, ..., n の処理によって NB_i にストリーム要素群が格納されるまでのアイドル時間を表し，ms_i, ..., ms_n, nbs_i, ..., nbs_{n-1} を変数とする関数として定義される．T_{SIe_i} は，サイト i のデータベース演算が終了してからサイト 1, ..., $i-1$ の演算がすべて終了するまでの時間を表し，ms_1, ..., ms_{i-1}, nbs_1, ..., nbs_{i-1} を変数とする関数として定義される．したがって，問い合わせの処理時間を表す関数 T_{QE} は式(6)のように表される.

$$\begin{aligned}
&T_{QE}(ms_1, \ldots, ms_n, nbs_1, \ldots, nbs_{n-1}) \\
&= \max_{1 \le i \le n} T_{QE_i}(ms_1, \ldots, ms_n, nbs_1, \ldots, nbs_{n-1}) \\
&= \max\{ \\
&\quad T_{SE_1}(ms_1, nbs_1) \\
&\quad + T_{SIb_1}(ms_1, \ldots, ms_n, nbs_1, \ldots, nbs_{n-1}), \\
&\quad \vdots \\
&\quad T_{SE_i}(ms_i, nbs_{i-1}, nbs_i) \\
&\quad + T_{SIb_i}(ms_i, \ldots, ms_n, nbs_i, \ldots, nbs_{n-1}) \\
&\quad + T_{SIe_i}(ms_1, \ldots, ms_{i-1}, nbs_1, \ldots, nbs_{i-1}), \\
&\quad \vdots \\
&\quad T_{SE_n}(ms_n, nbs_{n-1}) \\
&\quad + T_{SIe_n}(ms_1, \ldots, ms_{n-1}, nbs_1, \ldots, nbs_{n-1}) \\
&\quad \}. \qquad (6)
\end{aligned}$$

ここで，サイト j+1 の処理が終了してからサイト j の処理が終了するまでの時間を T_{e_j} とする $(1 \le j \le i-1)$．T_{e_j} は，ms_j, nbs_{j-1}, nbs_j のみを変数とする関数として定義される．このとき，問い合わせ処理の終了時におけるアイドル時間 T_{SIe_i} は，式(7)のように表される.

$$T_{SIe_i}(ms_1, \ldots, ms_{i-1}, nbs_1, \ldots, nbs_{i-1}) = \sum_{j=1}^{i-1} T_{e_j}(ms_j, nbs_{j-1}, nbs_j) \ (2 \le i \le n). \quad (7)$$

式(7)を式(6)に代入し，さらに $\sum_{j=1}^{i-1} T_{e_j}$ を展開する．max関数の第二引数以降のすべての引数，すなわち T_{QE_i} $(2 < i < n)$ は，T_{e_1} を含む．したがって，max関数は，以下のように変形することができる．

$$\max_{1\le i\le n} T_{QE_i} = \max\{T_{QE_1}, T_{e_1} + \max_{2\le i\le n}\{T_{QE_i} - T_{e_1}\}\}$$

この式において，内側のmax関数の引数を展開すると，第2引数以降のすべての引数は T_{e_2} を含む．すなわち，内側のmax関数に対して同様な変形を行うことができる．このようにmax関数に対する変形は，再帰的に適用することができる．その結果，T_{QE} は，式(8)のように表される．

$$\begin{aligned}
&T_{QE}(ms_1,\ldots,ms_n,nbs_1,\ldots,nbs_{i-1})\\
&=\max_{1\le i\le n} T_{QE_i}(ms_1,\ldots,ms_n,nbs_1,\ldots,nbs_{n-1})\\
&=\max\{\\
&\quad T_{SE_1}(ms_1,nbs_1)\\
&\quad +T_{SIb_1}(ms_1,\ldots,ms_n,nbs_1,\ldots,nbs_{n-1}),\\
&\quad T_{e_1}(ms_1,nbs_1)\\
&\quad +\max\{\\
&\quad T_{SE_2}(ms_2,nbs_1,nbs_2)\\
&\quad +T_{SIb_2}(ms_2,\ldots,ms_n,nbs_2,\ldots,nbs_{n-1}),\\
&\quad T_{e_2}(ms_2,nbs_1,nbs_2)\\
&\quad \vdots\\
&\quad +\max\{\\
&\quad T_{SE_i}(ms_i,nbs_{i-1},nbs_i)\\
&\quad +T_{SIb_i}(ms_i,\ldots,ms_n,nbs_i,\ldots,nbs_{n-1}),\\
&\quad T_{e_i}(ms_i,nbs_{i-1},nbs_i)\\
&\quad \vdots\\
&\quad +\max\{\\
&\quad T_{SE_{n-1}}(ms_{n-1},nbs_{n-1})\\
&\quad +T_{SIb_{n-1}}(ms_{n-1},ms_n,nbs_{n-1}),\\
&\quad T_{e_{n-1}}(ms_{n-1},nbs_{n-1})\\
&\quad +T_{SE_n}(ms_n,nbs_{n-1})\\
&\}\}\ldots\}
\end{aligned} \tag{8}$$

式(8)では，1メモリ・サイズ変数，および，ネットワーク間のバッファ・サイズ変数が，T_{SIb} のパラメータとして複数の部分関数に含まれている．したがって，T_{QE} を最小にするような解を求めるためには，すべてのメモリ・サイズ変数およびバッファ・サイズ変数の候補値の組み合わせを列挙していかなければならない．しかし，サイト割り当てがチェイン型の場合，問い合わせ処理の開始時におけるアイドル時間 T_{SIb_i} を変動させる大きな要因は，そのサイトのバッファ割り当て（すなわち，ms_i），一つ下位のサイトとの間のネットワーク・バッファの大きさ（nbs_i），および，一つ下位のサイトのバッファ割り当て（ms_{i+1}）である．それ以外のパラメータの変動による T_{SIb_i} の変動量は，それら3要因の変動による変動量に比べて無視できるほど小さい．したがって，本論文においては，T_{SIb_i} は，ms_i，nbs_i，ms_{i+1} を変数とする関数として定義する．これより，問い合わせの処理時間 T_{QE} は，式(9)のように複数の部分式の組み合わせとして表現される．式(9)において y_i は，サイト i を根としたときの部分問い合わせの処理時間を示している（図4）．

$$\begin{aligned}
T_{QE} &= y_1\\
y_1 &= F_1(ms_1,ms_2,nbs_1,y_2)\\
&= \max\{\\
&\quad T_{SE_1}(ms_1,nbs_1)\\
&\quad +T_{SIb_1}(ms_1,ms_2,nbs_1)\ ,\\
&\quad T_{e_1}(ms_1,nbs_1)+y_2\\
&\quad \}\\
y_2 &= F_2(ms_2,ms_3,nbs_1,nbs_2,y_3)\\
&= \max\{\\
&\quad T_{SE_2}(ms_2,nbs_1,nbs_2)\\
&\quad +T_{SIb_2}(ms_2,ms_3,nbs_2)\ ,\\
&\quad T_{e_2}(ms_2,nbs_1,nbs_2)+y_3\\
&\quad \}\\
&\vdots\\
y_i &= F_i(ms_i,ms_{i+1},nbs_{i-1},nbs_i,y_{i+1})\\
&= \max\{\\
&\quad T_{SE_i}(ms_i,nbs_{i-1},nbs_i)\\
&\quad +T_{SIb_i}(ms_i,ms_{i+1},nbs_i)\ ,\\
&\quad T_{e_i}(ms_i,nbs_{i-1},nbs_i)+y_{i+1}\\
&\quad \}
\end{aligned}$$

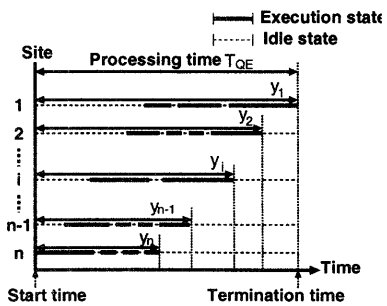


図4 部分問い合わせの処理時間
Fig. 4 The chart of processing time of sub-queries.

$$
\begin{aligned}
y_{i+1} &= F_{i+1}(ms_{i+1}, ms_{i+2}, nbs_i, nbs_{i+1}, y_{i+2}) \\
&= \max\{ \\
&\quad T_{SE_{i+1}}(ms_{i+1}, nbs_i, nbs_{i+1}) \\
&\quad + T_{SIb_{i+1}}(ms_{i+1}, ms_{i+2}, nbs_{i+1}) , \\
&\quad T_{e_{i+1}}(ms_{i+1}, nbs_i, nbs_{i+1}) + y_{i+2} \\
&\} \\
&\vdots \\
y_n &= F_n(ms_n, nbs_{n-1}) \\
&= T_{SE_n}(ms_n, nbs_{n-1}) \qquad (9)
\end{aligned}
$$

式 (9) を構成する T_{SE}, T_{SIb}, T_e の詳細を付録に添付する.

3.3 アルゴリズム

本節では, 式 (9) で表される T_{QE} を最小にする $ms_1, \ldots, ms_n, nbs_1, \ldots, nbs_{n-1}$ の値を効率良く求めるためのアルゴリズムについて述べる.

式 (9) において, ms_{i+1}, nbs_i は, F_{i+1} および F_i のパラメータとして設定されている. したがって, y_{i+1} の計算時に ms_{i+1}, nbs_i の値を決定することができない.

本アルゴリズムは, ms_i, nbs_{i-1} のある候補値について, y_i の値を最小とする ms_{i+1}, nbs_i の値を決定する. この決定を, ms_i, nbs_{i-1} のすべての候補値について行う. これらを, $y_{n-1}, \ldots, y_1$ に対して順に適用することにより, T_{QE} を最小とする $ms_1, \ldots, ms_n, nbs_1, \ldots, nbs_{n-1}$ の値を求める.

本アルゴリズムでは, 問い合わせの処理時間を表す各式ごとに, メモリ割り当ての候補値を表すテーブル (以下, "割り当て候補テーブル" とよぶ) を, y_n に対応する $table_n$ から順に y_1 に対応する $table_1$ まで生成する (図 5). この割り当て候補テーブル $table_i$ は, ms_i および nbs_{i-1} の候補値, そのときの y_i の最小値, および, y_i の最小値を与える ms_{i+1}, nbs_i の値の組から構成される. ここで, ms_i の各候補値に対する最適な $bs_{i1}, \ldots, bs_{im_i}$ の値を表すバッファ・サイズ・テーブル (図 2) は, 文献 7), 10) に示した方式により, 各サイトごとに求められているものとする.

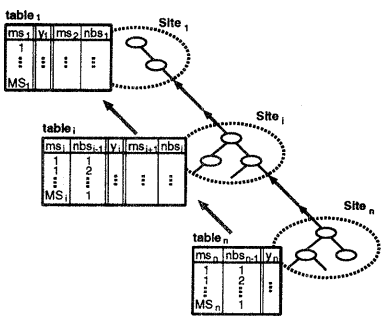


図 5 割り当て候補テーブルの生成

Fig. 5 The creation process of the table for representing memory allocation canditates.

以下に, メモリ資源割り当ての問題を解くアルゴリズムを示す.

Step-1: ms_n および nbs_{n-1} のすべての候補値について y_n を計算し, 割り当て候補テーブル ($table_n$) を生成する (図 5).

Step-2: 問い合わせ全体の処理時間を表す式 y_1 についての割り当て候補テーブル $table_1$ が求められるまで, $i = n-1, \ldots, 1$ の順に **Step-3** を繰り返す.

Step-3: ms_i, nbs_{i-1} のある候補値に対して, y_i の値を最小とする ms_{i+1}, nbs_i の値を決定し, ms_{i+1}, nbs_i, y_i を $table_i$ に書き込む (図 5). これを, ms_i, nbs_{i-1} のすべての候補値に対して行う.

Step-4: 最終的に求められた $table_1$ において, y_1 の値が最小となる ms_1, ms_2, nbs_1 の値を獲得し (これが ms_1, ms_2, nbs_1 の最適値である), このときの ms_2, nbs_1 の値と $table_2$ から ms_3, nbs_2 の値を獲得する (これが ms_3, nbs_2 の最適値である). 同様に, $table_3, \ldots, table_n$ により, $ms_4, \ldots, ms_n, nbs_3, \ldots, nbs_{n-1}$ の値を獲得する (これらが $ms_4, \ldots, ms_n, nbs_3, \ldots, nbs_{n-1}$ の最適値である). また, あらかじめ作成されているバッファ・サイズ・テーブル (図 2) において, 最適な ms_i の値に対応する各バッファ・サイズ $bs_{i1}, \ldots, bs_{im_i}$ の値を, 最適なバッファ・サイズとして獲得する.

3.4 計算の複雑さ

本アルゴリズムの **Step-3** において, 割り当て候補テーブル $table_i$ を生成するために, 二つの変数 ms_i, nbs_{i-1} のすべての候補値の組み合わせについて, 割り当て候補テーブル $table_{i+1}$ の各値とのつき合わせを行う. よって, 割り当て候補テーブル $table_i$ のテーブルを生成するための計算量は, 式 (10) で表される.

$$\frac{MS_i(MS_i+1)}{2} \times \frac{MS_{i+1}(MS_{i+1}+1)}{2} \qquad (10)$$

したがって, すべてのサイトが利用するメモリ・サイズを求めるための計算量は, 各サイトに対応する式の計算量の総量となる (式 (11)).

$$\sum_{i=1}^{n-1} \frac{MS_i(MS_i+1)MS_{i+1}(MS_{i+1}+1)}{4} + \frac{MS_n(MS_n+1)}{2} \qquad (11)$$

ここで，各サイトにおける利用可能なメモリ・サイズの最大値がすべて等しい大きさ（MS）であるとすると，計算量は式(12)のように表される．

$$\frac{n-1}{4}\left(MS^4+MS^3+MS^2\right)+\frac{MS^2+MS}{2} \tag{12}$$

また，ms および nbs の候補値の間隔（以後，ステップ・サイズとよぶ）を STP とすると，計算量は式(13)のように表される．

$$\frac{n-1}{4}\left\{\left(\frac{MS}{STP}\right)^4+\left(\frac{MS}{STP}\right)^3+\left(\frac{MS}{STP}\right)^2\right\}+\frac{1}{2}\left\{\left(\frac{MS}{STP}\right)^2+\frac{MS}{STP}\right\} \tag{13}$$

したがって，メモリ資源割り当てを求めるための計算量のオーダは，

$$O\left(n\times\frac{MS^4}{STP^4}\right)$$

となる．

4. 実　　験

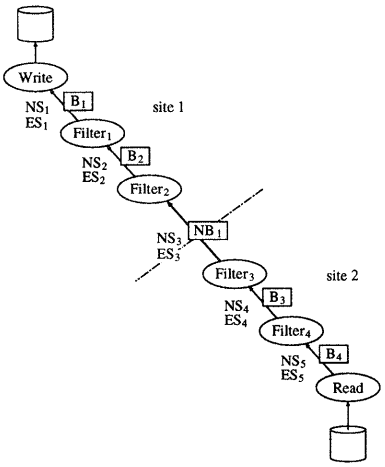
4.1 問い合わせ

本実験では，四つのフィルタ演算からなる4種類の問い合わせ（$query_1$〜$query_4$，**図6**(a)），および，三つのマージ（ストリーム・マージ）演算からなる四つの問い合わせ（$query_5$〜$query_8$，図6(b)）について，本論文において提案したメモリ資源割り当て計算を行い，その割り当てを適用した場合の問い合わせの処理時間の予測値と利用メモリ・サイズについて評価を行った．これらの問い合わせは，図6に示すような構造をもつ．

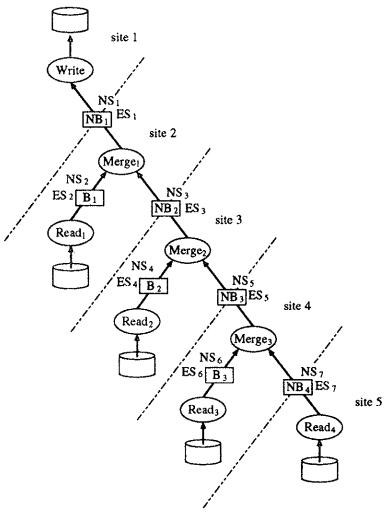
実験において用いたフィルタ演算，および，マージ演算は，任意のデータベース演算を疑似的に表す演算とした．フィルタ演算は，入力ストリームを一つもつ演算であり，マージ演算は，入力ストリームを二つもつ演算である．これら2種類の演算は，入力ストリームの各ストリーム要素に対して処理を行い，その演算の返り値となるストリーム要素群を生成する．これら2種類の演算の処理時間を表す式において，次の3パラメータを設定した．

- 1入力ストリーム要素あたりの処理時間
- 出力ストリーム要素のデータ長
- 入力ストリーム要素の総数に対する出力ストリーム要素の総数の割合（選択率）

ここで，1入力ストリーム要素あたりの処理時間が短い演算は，関係データベース演算の選択演算のように，簡単な処理を行う演算に対応する．逆に，1入力ストリーム要素あたりの処理時間が長い演算は，オブジェクト指向型データベースにおいて識別子参照が幾度も行われる演算のように，複雑な処理を行う演算に対応する．実験では，各演算の3パラメータをそれぞれ変化させた問い合わせを対象として行った．中間ストリーム要素数およびストリーム要素サイズを**表1**のように設定した．また，ハードウェア・パラメータを**表2**のように設定した．これらの値は，実際の分散処理環境上に実現したストリーム指向型並列処理システムにおいて実測することにより求めたものである．この分散処理環境は，5台の SparcStation1 ELC を Ethernet によって結合したハードウェアによって実



(a) $query_1$〜$query_4$



(b) $query_5$〜$query_8$

図6 問い合わせの構造
Fig. 6 Query structures.

表1 問い合わせの中間ストリーム要素数およびサイズ
Table 1 The number and size of intermediate stream elements in query processing.

(a) $query_1$〜$query_4$

	$query_1$	$query_2$	$query_3$	$query_4$
NS_1	8192	512	8192	8192
ES_1	64	64	16	256
NS_2	8192	1024	8192	8192
ES_2	64	64	32	128
NS_3	8192	2048	8192	8192
ES_3	64	64	64	64
NS_4	8192	4096	8192	8192
ES_4	64	64	128	32
NS_5	8192	8192	8192	8192
ES_5	64	64	256	16

(b) $query_5$〜$query_7$

	$query_5$ $query_8$	$query_6$	$query_7$
NS_1	8192	8192	1024
ES_1	64	64	64
NS_2	8192	4096	2048
ES_2	64	64	64
NS_3	8192	4096	2048
ES_3	64	64	64
NS_4	8192	2048	4096
ES_4	64	64	64
NS_5	8192	2048	4096
ES_5	64	64	64
NS_6	8192	1024	8192
ES_6	64	64	64
NS_7	8192	1024	8192
ES_7	64	64	64

NS[stream element]：流れるストリーム要素の総数
ES[byte/stream element]：1ストリーム要素の大きさ

現されている[12]．

$query_1$，$query_5$，および，$query_8$ は，中間ストリーム要素数，および，ストリーム要素サイズがすべて同じとなる場合である．

$query_2$ は，各フィルタ演算が入力ストリーム要素数の1/2倍の数のストリーム要素を生成する場合である．また，ストリーム要素サイズは一定である．

$query_3$ は，各フィルタ演算が入力ストリーム要素サイズの1/2倍のサイズのストリーム要素を，また，$query_4$ は2倍のサイズのストリーム要素を生成する場合である．

$query_6$ は，各マージ演算が各入力ストリーム要素数の合計と等しい数のストリーム要素を生成する場合である．

$query_7$ は，各マージ演算が各入力ストリーム要素数の合計の1/4倍の数のストリーム要素を生成する場合である．

表2 メモリ資源割り当て計算に用いたハードウェア・パラメータ
Table 2 The hardware parameters for memory allocation algorithm.

フィルタ演算の内部処理に要する時間	100 [μsec/stream-element]
$query5$〜$query7$におけるマージ演算の内部処理に要する時間	100 [μsec/stream-element]
$query8$ におけるマージ演算（$merge_1$）の内部処理に要する時間	90 [μsec/stream-element]
$query8$ におけるマージ演算（$merge_2$）の内部処理に要する時間	100 [μsec/stream-element]
$query8$ におけるマージ演算（$merge_3$）の内部処理に要する時間	110 [μsec/stream-element]
出力演算の内部処理に要する時間	12 [μsec/stream-element]
入力演算の内部処理に要する時間	24 [μsec/stream-element]
ストリーム要素の受け渡しに要する時間	16 [μsec/stream-element]
コンテキスト切替えに要する時間	154 [μsec]

$query_8$ は，他の $query$ と異なり，問い合わせを構成するマージ演算ごとに，1ストリーム要素あたりの内部処理に要する時間が異なる問い合わせである．

4.2 実験結果

実験結果を**表3**に示す．「提案方式（optinal allocation）」は，本論文において提案したメモリ資源割り当て方式を適用した場合である．また，「均等方式（equally partitioned allocation）」は，提案方式の比較対象として最適化を行わない方式を適用した場合である．

提案方式において用いる利用可能メモリ・サイズ（MS）は，各バッファへのメモリ資源割り当て候補をすべて選択可能となる値を設定した．すなわち，MS の値は，与えられた問い合わせの各ストリームにおいて，全ストリーム要素を一度に格納できるバッファを確保できるメモリ・サイズの合計値に設定した．

均等方式とは，利用可能メモリをすべてのバッファに均等に割り当てた方式である．表3において，均等方式に用いた利用可能メモリ・サイズは，提案方式に用いた利用可能メモリ・サイズの4分の1の大きさに設定した．

4.2.1 問い合わせの処理時間

図7は，$query_1$ について，各サイトの利用可能メモリ・サイズの合計値による問い合わせの処理時間の変化を示している．

図7のグラフより，利用可能メモリ・サイズのす

表3 メモリ資源割り当て計算の実験結果
Table 3 Experimental results of allocation algorithm.

(a) $query_1$〜$query_4$

	$query_1$		$query_2$		$query_3$		$query_4$	
	均等方式	提案方式	均等方式	提案方式	均等方式	提案方式	均等方式	提案方式
MS_1 [byte]	393216	1572864	233472	933888	921600	3686400	921600	3686400
MS_2 [byte]	393216	1572864	233472	933888	921600	3686400	921600	3686400
ms_1 [byte]	262144	40960	155648	6144	614400	12288	614400	188416
ms_2 [byte]	262144	106496	155648	786432	614400	292864	614400	6144
bs_1 [stream element]	2048	320	1216	32	19200	256	1200	456
bs_2 [stream element]	2048	320	1216	64	9600	256	2400	560
bs_3 [stream element]	2048	832	1216	4096	2400	1024	9600	128
bs_4 [stream element]	2048	832	1216	8192	1200	632	19200	128
nbs_1 [stream element]	2048	256	1216	128	4800	256	4800	128
処理時間 [sec]	3.22	2.72	2.18	1.92	4.89	3.14	4.89	3.04

(b) $query_5$〜$query_8$

	$query_5$		$query_6$		$query_7$		$query_8$	
	均等方式	提案方式	均等方式	提案方式	均等方式	提案方式	均等方式	提案方式
MS_1 [byte]	131072	524288	131072	524288	16384	65536	131072	524288
MS_2 [byte]	393216	1572864	262144	1048576	65536	262144	393216	1572864
MS_3 [byte]	393216	1572864	131072	524288	131072	524288	393216	1572864
MS_4 [byte]	393216	1572864	65536	262144	262144	1048576	393216	1572864
MS_5 [byte]	131072	524288	16384	65536	131072	524288	131072	524288
ms_2 [byte]	131072	53248	65536	65536	16384	1024	131072	2048
ms_3 [byte]	131072	40960	32768	4096	32768	1024	131072	8192
ms_4 [byte]	131072	8192	16384	4096	65536	65536	131072	88064
bs_1 [stream element]	2048	832	1024	1024	512	16	2048	32
bs_2 [stream element]	2048	640	512	64	1024	16	2048	128
bs_3 [stream element]	2048	128	256	64	2048	1024	2048	1376
nbs_1 [stream element]	2048	224	2048	256	256	16	2048	96
nbs_2 [stream element]	2048	224	1024	128	512	16	2048	96
nbs_3 [stream element]	2048	128	516	128	1024	128	2048	256
nbs_4 [stream element]	2048	256	256	128	2048	256	2048	416
処理時間 [sec]	4.07	2.83	1.76	1.56	1.79	1.33	4.03	2.93

MS： 利用可能メモリ・サイズ
ms： 利用メモリ・サイズ
((b) におけるサイト 1 およびサイト 5 においては，ネットワーク・バッファ以外のバッファがないため，$ms = 0$ となる.)
bs： バッファ・サイズ
nbs： ネットワークを介したバッファ・サイズ

べての設定において，問い合わせの処理時間は，均等方式の場合に比べて提案方式の場合の方が短い．提案方式を適用した場合，利用可能メモリ・サイズが大きくなるにしたがって問い合わせの処理時間が短くなってきているが，利用可能メモリ・サイズがある大きさ（230 Kbyte）以上では，処理時間が一定となっている．この結果から，提案方式を適用することにより，メモリ供給による問い合わせの処理時間の短縮を最大限に行うことができることが判る．また，処理時間が一定となりはじめるメモリ資源量（230 Kbyte）以下のとき，すなわち，メモリ資源が不足している状況においても，提案方式を適用することにより，問い合わせの処理時間を短縮することができる．

表3より，提案方式を適用した場合におけるメモリ利用量は，均等方式の場合に比べて小さい．また，提案方式を適用した場合における問い合わせの処理時間は，均等方式を適用した場合に比べて短い．これは，メモリ資源が過分に提供されている場合に，均等方式では過分なメモリ資源を排除することができないのに対し，提案方式では，過分なメモリ資源を最適化の計算によって排除できることによる．均等方式では，バッファ・サイズが最適値よりも大きいため，ストリーム型並列性が引き出されず，処理効率が悪化していることを表している．

$query_1$ について，本方式を適用した場合の利用可能メモリ・サイズと利用メモリ・サイズの関係を，**図8**

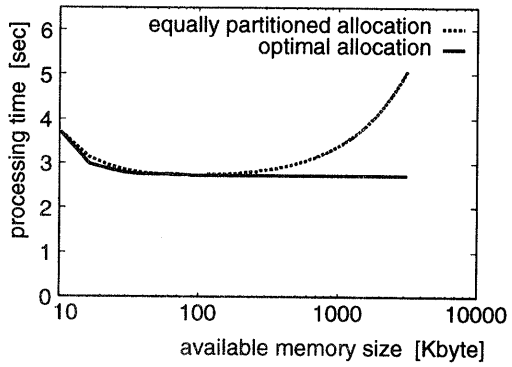


図7 利用可能メモリ・サイズと問い合わせ処理時間の関係 ($query_1$)
Fig. 7 Relationship between query processing time and available memory size ($query_1$).

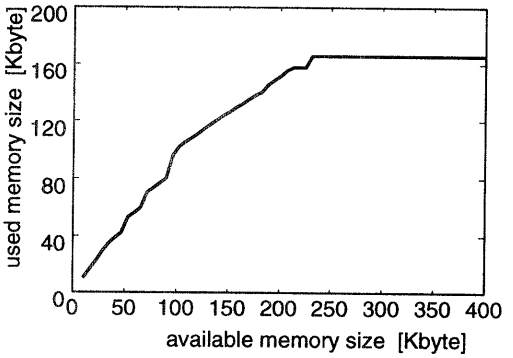


図8 利用可能メモリ・サイズと利用メモリ・サイズ ($query_1$)
Fig. 8 Relationship between the used memory size and the available memory size ($query_1$).

のグラフに示す．このグラフと図7のグラフから，本方式を適用することにより，過剰なメモリ供給により問い合わせの処理時間が長くなってしまう現象が引き起こされないことが判る．

提案方式では，サイト内最適化，および，本論文で提案しているサイト間の関係に依存する最適化の両最適化において，各々独立に，過分なメモリ資源の排除を行うことができる．提案方式では，3.1節で述べたように，サイト内最適化は，同じサイトに割り当てられている関数群から構成される部分問い合わせを対象とする．サイト内最適化により，サイトの内部に配置される各バッファへの最適なメモリ資源割り当てが決定される．サイト内最適化の計算には，文献7)，10)において示されている方法を用いている．この方法は，単独の計算機（シングルプロセッサの計算機，および，共有メモリ型マルチプロセッサの計算機）の環境を対象としている．提案方式では，このサイト内最適化によるサイト内の処理を考慮した過分なメモリ資源の排除に加えて，サイト間の処理の依存関係により発生する過分なメモリ資源を排除することが可能となる．これは，提案方式において，問い合わせを構成する各部分問い合わせにおいて使用する最適なメモリ資源量を，サイト間の処理の依存関係を考慮して求めることにより実現される．

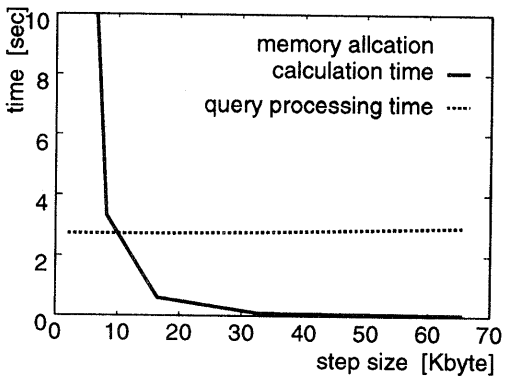


図9 問い合わせの処理時間と割り当て計算時間の関係 ($query_1$)
Fig. 9 Relationship of query processing time and memory allocation calculation time.

4.2.2 割り当て計算時間

3.4節の式(13)より，メモリ資源割り当てを求めるための計算量は，メモリ・サイズのステップ・サイズと利用可能メモリ・サイズに大きく影響されている．

ステップ・サイズが小さすぎる場合，割り当て計算時間が非常に長くなる．したがって，ステップ・サイズを問い合わせの処理時間に影響しない範囲で大きく取ることにより，割り当て計算時間を減少させることが必要である．**図9**は，$query_1$ について，ステップ・サイズによる問い合わせの処理時間および割り当て計算時間の変化を示している．このグラフより，ステップ・サイズが大きくなると，きめ細かいメモリ資源割り当ての最適化を行えなくなるので，問い合わせの処理時間が長くなっている．しかし，処理時間の変化に対し，割り当て計算時間は急激に短くなっている．したがって，ステップ・サイズの設定においては，その値を比較的大きく設定することにより，問い合わせの処理時間に対して，ほとんど無視できる時間でメモリ資源割り当てを求めることが可能となることがわかる．

このようなステップ・サイズの値は，ストリーム要素の流量と各演算の1ストリーム当たりの処理時間から求めることができると考えられる．その設定方法については，今後，さらに検討を行っていく予定である．

5. おわりに

本論文では，ストリーム指向型並列処理方式を，共有メモリ資源をもたない分散処理環境において実現す

る場合を対象とした分散メモリ資源の最適割り当て方式を提案した．このメモリ資源割り当て方式を適用することにより，分散問い合わせ処理において各サイト間の並列性を最大限に抽出することができ，かつ，問い合わせの実行時間を最短化することができる．また，本論文では，メモリ資源割り当てのための計算時間を短縮する方法を示した．さらに，本方式を実際に計算機上に実現し，それを用いて行ったメモリ資源割り当ての実験により，問い合わせの処理効率を向上できることを示した．

今後は，様々な応用分野を対象としたストリーム処理における本方式の効果について，検討を行っていく予定である．

参考文献

1) DeWitt, D. and Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems, *Comm. ACM*, Vol.35, No.6, pp. 85–98 (1992).
2) Effelsberg, W. and Haerder, T.: Principles of Database Buffer Management, *ACM Trans. Database Syst.*, Vol.9, No.4, pp.560–595 (1984).
3) Hirano, Y., Satoh, T., Inoue, U. and Teranaka, K.: Load Balancing Algorithms for Parallel Database Processing on Shared Memory Multiprocessors, *Proc. 1st IEEE Int. Conf. on Parallel and Distributed Information Systems*, pp.210–217 (1991).
4) Ibaraki, T. and Katoh, N.: *Resource Allocation Problems*, MIT Press (1988).
5) Kitsuregawa, M. and Ogawa, Y.: Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method for Data Skew in the Super Database Compputer (SDC), *Proc. 16th Int. Conf. on Very Large Data Bases*, pp.210–221 (1990).
6) Kiyoki, Y., Kurosawa, T., Kato, K. and Masuda, T.: The Software Architecture of a Parallel Processing System for Advanced Database Applications, *Proc. 7th IEEE Int. Conf. on Data Engineering*, pp.220–229 (1991).
7) 劉澎，清木　康，益田隆司：ストリーム指向型関係演算処理におけるバッファ資源割り当ての計算方式，情報処理学会論文誌，Vol.29, No.8, pp.770–781 (1988).
8) 劉澎，清木　康，益田隆司：関係演算のストリーム指向型並列処理における動的資源割り当て方式，情報処理学会論文誌，Vol.29, No.7, pp.656–668 (1988).
9) Liu, P., Kiyoki, Y. and Masuda, T.: Efficient Algorithms for Resource Allocation in Distributed and Parallel Query Processing Environments, *Proc. 9th IEEE Int. Conf. on Distributed Computing Systems*, pp.316–323 (1989).
10) 大西　元，清木　康：関数型並列データベース・システム SMASH における資源割り当て方式の拡張，情報処理学会アドバンストデータベースシステムシンポジウム論文集，pp.43–51 (1990).
11) Sacco, G. M. and Schkolnick, M.: Buffer Management in Relational Database Systems, *ACM Trans. Database Syst.*, Vol.11, No.4, pp.473–498 (1986).
12) 佐藤　聡，清木　康：関数型計算に基づくデータベースシステムの並列処理実行系の実現方式，情報処理学会データベースシステム研究会研究報告 92-DBS-89, pp.151–160 (1992).
13) Wolf, J. L., Iyer, B. R., Pattipati, K. R. and Turek, J.: Optimal Buffer Partitioning for the Nested Block Join Algorithm, *Proc. 7th IEEE Int. Conf. on Data Engineering*, pp.510–519 (1991).

付　録

式 (9) を構成する T_{SE}，T_{SIb}，T_e の詳細を以下にしめす．

$$T_{SE_i}(ms_i, nbs_{i-1}, nbs_i) = E_i(ms_i) + f_{nput}(nbs_{i-1}, NS_{i-1}) + f_{nget}(nbs_i, NS_i)$$

$$T_{SIb_i}(ms_i, ms_{i+1}, nbs_i) = \max\{first_{i+1}(ms_{i+1}, nbs_i) + f_{nput}(nbs_i, nbs_i) - sub1_i(ms_i), 0\}$$

$$T_{e_i}(ms_i, nbs_{i-1}, nbs_i) = end_i(ms_i, nbs_i) + f_{nget}(nbs_i, nbs_i)$$

$E_i(ms_i)$：　サイト i のプロセッサが，サイズ ms_i のメモリ資源を用いて，このサイトに割り当てられている部分問い合わせを実行する時間．

$first_{i+1}(ms_{i+1}, nbs_i)$：　サイト $i+1$ のプロセッサが，サイズ ms_{i+1} のメモリ資源を用いて，最初のストリーム要素群（nbs_i 個分）を分生成するのに要する時間．

$sub1_i(ms_i)$：　サイト i のパイプライン以外の関数インスタンスが，サイズ ms_i のメモリ資源を用いて最初のストリーム要素群を生成するのに要する時間，これは，E_i の一部分である．

$end_i(ms_i, nbs_i)$：　サイト i のプロセッサが，サイズ ms_i のメモリ資源を用いて，バッファ NB_i を介

して受けとった最後のストリーム要素群（nbs_i 分）を消費するのに要する時間．これは，E_i の一部分である．

$f_{nput}(nbs, NS)$： バッファ・サイズが nbs に設定されているネットワーク・バッファを介して，ストリーム要素数 NS 分のストリームを送信するのに要する時間．

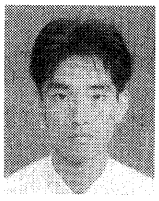
$f_{nget}(nbs, NS)$： バッファ・サイズが nbs に設定されているネットワーク・バッファを介して，ストリーム要素数 NS 分のストリームを受信するのに要する時間．

$NSproduce_i(nbs_i)$： サイト i において，一つ下位のサイトとの間のネットワーク・バッファ NB_i から nbs_i 個のストリーム要素群を消費した時に，一つ上位のサイトとの間のネットワーク・バッファ NB_{i-1} に生成するストリーム要素数．

NS_i： ネットワーク・バッファ NB_i を介して転送されるストリーム要素の総数．

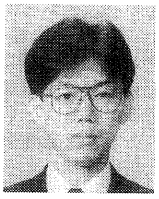
これらの実行時間を見積もる関数群のうち，E_i，$first_{i+1}$，$sub1_i$，end_i は，問い合わせを構成する各関数インスタンスの実行時間を表す関数から構成される．関数インスタンスの実行時間を表す関数は，その関数インスタンスが消費するストリーム量をパラメータとし，接続されるチャネルのバッファ・サイズを変数とする．消費するストリーム量などは，統計情報などから求めることができる．また，その関数インスタンスに設定されているバッファのバッファ・サイズは，そのサイトで利用するメモリ・サイズ ms から一意に定まるように，すでに割り当てが済んでいる．

(平成6年9月27日受付)
(平成7年9月 6 日採録)



村岡 正則（正会員）

1971年生．1990年（株）日立製作所入社．1993年より1年間筑波大学第三学群情報学類にて，データベース・システムに関する研究を行う．現在，同社システム開発研究所にて，オペレーティング・システムに関する研究・開発に従事．



佐藤 聡（学生会員）

1967年生．1991年筑波大学第三学群情報学類卒業．現在同大学大学院博士課程工学研究科電子・情報工学専攻に在学中．データベース・システム，並列処理，分散処理，マルチメディア処理に興味をもつ．日本ソフトウェア科学会学生会員．

清木 康（正会員）

1956年生．1978年慶應義塾大学工学部電気工学科卒業．1983年同大学院工学研究科博士課程修了．工学博士．同年，日本電信電話公社武蔵野電気通信研究所入所．1984年より，筑波大学電子・情報工学系に勤務，現在同学系助教授．データベース・システム，計算機アーキテクチャ，関数型プログラミングの研究に従事．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE各会員．