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当て方式†

劉 澎†† 清木 康††† 益田隆司†††*

関係データベースの基本演算である関係演算の並列処理方式として，我々はストリーム指向型関係演算処理方式を提案している．この方式を並列処理システム上で実現する場合の重要な課題は，計算機資源の割当て方法である．本論文では，データベースに関する統計情報が提供されていないために問い合わせ処理の開始前にバッファ資源の最適な割当てを行うことが困難である場合，および，バッファ資源量が実行時に動的に変化する場合において，ストリーム指向型関係演算処理方式の効果を引き出すための動的資源割当て方式を提案する．問い合わせを構成する各関係演算に割り当てられるバッファ資源量が実行時に動的に増加する場合に関係演算処理を継続して行うために，動的資源割当て方式においては，ストリーム指向型関係演算処理方式における関係演算処理アルゴリズムが拡張される．また，動的資源割当て方式を適用するために必要となる統計情報の予測，および，動的資源割当ての適用時点の決定を行うための指針を示す．そして，動的資源割当て方式を用いて実際の問い合わせ処理を行った実験の結果を示し，本方式の有効性について考察を行う．

1. まえがき

データベースの普及に伴い，データベース管理システムは計算機システムの中の最も重要な機能の一つになっている．データベース管理システムが扱うデータ量は増大し，計算機システムの中でデータベース処理の占める割合が大きくなってきており，また，その応用分野は多様化し，演繹データベース等の知識ベースへの拡張に関する研究も活発に行われている[5]．

このような背景の中で，多様な応用分野に柔軟に適応し，大量データを扱う処理を効率よく行う並列処理システムの実現が重要な課題となっている．

我々は，データベースや知識ベースのような大量データを扱う多様な応用分野に柔軟に適応できるストリーム指向型並列処理方式，および，それを実現する並列処理システムを提案している[8),9]．この並列処理方式は，要求駆動型制御による関数型計算の枠組みの中で，データベースや知識ベースを対象とする任意の基本演算群を並列に処理することを目指したものである．本方式では，大量データに対する基本演算処理は，ストリームに対する関数計算として要求駆動型制御により実現される．すでに，この並列処理システム上で実現される関係演算処理方式としてストリーム指向型関係演算処理方式を提案し[8]，また，その方式の有効性を引き出すための計算機資源割当て方法を提案している[10]．

並列処理システム上でストリーム指向型関係演算処理方式を実現する場合における重要な課題は，計算機資源（プロセッサ資源およびバッファ資源）の割当てである．文献 10)では，関係データベースの問い合わせ処理において，データベースの統計情報を利用することにより，限られた計算機資源の中でストリーム指向型処理方式の効果を最大限に引き出すための計算機資源の割当て方法を提案している．この方法は，問い合わせを実行する前にあらかじめ，データベースの統計情報を用いてプロセッサ資源およびバッファ資源の割当てを決めることを前提とした静的資源割当て方法である．

しかし，実際の問い合わせ処理においては，計算機資源の割当てに関して，次のような場合が考えられる．

（1） データベースの統計情報が提供されていない，あるいは，正確に提供されていない場合．

（2） マルチユーザあるいはマルチトランザクションを実現する環境のように，問い合わせ処理の過程で利用できる資源量が動的に変化する場合．

これらの場合において問い合わせ処理を行うとき，問い合わせのコンパイル時には最適な計算機資源の割当てを決定できない．このような場合においては，問い合わせ処理の実行時に各関係演算の実行状態をモニタすることにより統計情報を獲得し，それを用いて資源割当て計算を行った後，動的に計算機資源を割り当

† A Dynamic Resource Allocation Strategy in the Stream-Oriented Parallel Processing Scheme for Relational Database Operations by PENG LIU (Doctoral Program in Engineering, University of Tsukuba), YASUSHI KIYOKI and TAKASHI MASUDA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学工学研究科
††† 筑波大学電子・情報工学系
* 現在 東京大学理学部情報科学科

てることができれば，ストリーム指向型処理方式の効果を引き出すことが可能である．本論文では，文献 10)に示した資源割当て方法を実行時に動的に適用し，データベースに関する統計情報が提供されていない場合，あるいは，バッファ資源量が動的に変化する場合において，バッファ資源の最適な割当てを実現する動的資源割当て方式を提案する．この方式は，文献 10)に示した資源割当て計算を，問い合わせの実行の途中，統計情報およびバッファ資源量を把握した時点で動的に適用する．本論文では，ストリーム指向型関係演算処理における動的資源割当て方式について述べ，また，統計情報の予測方法を示す．そして，ここで提案する動的資源割当て方式の有効性を，実験結果により明らかにする．

以下，第2章では，すでに提案しているストリーム指向型関係演算並列処理方式[8]の概要を述べる．そして，動的資源割当て方式を提示し，この方式を実現する関係演算処理アルゴリズムを示す．

第3章では，統計情報の予測，および動的計算機資源割当ての適用時点の決定を行うための指針を示す．

第4章では，ここで提案する方式の有効性を確かめるために行った実験の結果を示す．

2. ストリーム指向型関係演算処理における資源割当て

ここでは，ストリーム指向型関係演算処理方式の概要を述べ，また，動的資源割当てを適用する処理方式について述べる．ストリーム指向型関係演算処理については文献 8)および 10)に詳しく述べている．

2.1 ストリーム指向型関係演算処理

ストリーム指向型関係演算処理方式では，関係演算は関数として要求駆動型評価による関数型計算[1),4),14)]の枠組みの中で次のような2種類の並列性を抽出しながら実行される．

（1） 関数引数の並列評価

関係データベースの問い合わせ処理においては，互いに独立な関係演算が並列に実行されることに対応する．この並列性は，複数の引数データの消費を行う関数が，それらの引数データを生成する各関数に対して，デマンドを同時に発行することによって抽出される．

（2） ストリーム型並列性

関係データベースの問い合わせ処理においては，リレーションを生成する関係演算とそれを消費する関係演算が並列に実行されることに対応する．

以下では，中間結果のタプル群を生成する関数（関係演算）を生産者ノード，それらを消費する関数（関係演算）を消費者ノードとする．

ストリーム指向型関係演算処理を逐次的に実現する場合には，消費者ノードは，入力バッファ内の1ページの処理を終えると，生産者ノードへデマンドを発行し，生産者ノードは，デマンドを受け取ると，出力バッファ内に演算結果の1ページを生成の完了するまで関係演算を実行し，その時点で実行を停止する．1デマンドに対してリレーション全体を生成せず，あらかじめ定められた1ページ分のタプル群を生成して停止するので，大量データを扱う関係演算処理を限られたバッファ資源の中で行うことが可能となる．

並列処理を行う場合には，生産者ノードと消費者ノードを異なるプロセッサに配置し，それらのノード間のバッファには，ダブルバッファリング機構を実現する．各バッファには2つの領域を用意する．消費者ノードは，一方の領域に格納されているタプル群に対して関係演算を実行する前に，他方の領域に次のページの格納を要求するために，生産者ノードへデマンドを先行的に発行する．生産者ノードは，デマンドを受け取ると次ページを生成するために関係演算の実行を開始する．消費者ノードは入力バッファ内のページのタプル群に対する処理を完了すると，その領域への次ページの格納を要求するために生産者側ノードへ再びデマンドを発行する．このようにして，ストリーム型並列処理が限られたバッファ資源の中で実現される．

2.2 資源割当て方式

文献 10)では，ストリーム指向型関係演算処理方式を実現する場合に，1プロセッサ内の限られたバッファ資源内で問い合わせ処理における計算回数を最小化するための資源割当て計算の方法，および，並列処理環境において，本方式の並列処理の効果を最大限に引き出すための資源割当ての条件を示した．

（1） 計算回数の最小化のための資源割当て計算

問い合わせ処理における計算回数の最小化のための資源割当て計算は，次の場合に用いられる．

① 1プロセッサ内で問い合わせ処理を行う場合，限られたバッファ資源の中で問い合わせを構成する関係演算ノード群を実行するとき，各関係演算ノードへ割り当てる最適なバッファ資源量を決定する．

② ストリーム指向型関係演算処理方式では，各プ

ロセッサに，1つの問い合わせの中の複数の関係演算ノードを割り当てることができる．そのとき，各プロセッサに割り当てられた部分問い合わせ (subquery) を構成する各関係演算ノードへ最適なバッファ資源量を割り当てる．

この資源割当て計算の方法は，次のようにまとめられる．

1) 問い合わせを構成する各関係演算を nested-loop あるいは sort-search アルゴリズムにより実行する場合，その問い合わせにおける総計算回数を求める式を，ベース・リレーションのタプル数，選択率およびバッファ資源量をパラメータとして設定する．

2) 1)で設定した式について，総バッファ資源量を一定とし，その制約条件のもとで，式の値を最小にする各関係演算ノードへのバッファの割当てを拡大ラグランジュ関数法を用いて求める．

3) バッファ割当ての修正アルゴリズムにより，各関係演算ノードへのバッファの最適な割当てを決定する．

この資源割当て計算を適用する場合には，次のような前提がある．

① 演算対象データに関する統計情報が存在する．

文献 10)に示したように，ストリーム指向型関係演算処理方式では，問い合わせ処理における計算回数は，演算対象とするベース・リレーション内のタプル数，結合演算選択率およびバッファ資源量によって決定される．

これらの中でベース・リレーションのタプル数は容易に獲得できるが，結合演算選択率については，正確な情報を得ることが難しい場合，あるいは，それが提供されていない場合がある．そのような場合には，問い合わせの実行前に（問い合わせのコンパイル時に）最適な資源割当てを決定できない．

② 総バッファ資源量が固定である．

総計算回数を求める式の値を最小にするバッファ割当ては，総バッファ資源量が固定であることを前提として計算される．しかし，マルチトランザクションを実現する環境では，問い合わせを処理するための総バッファ資源量が実行時に動的に変化する．

（2） 並列性の抽出

ストリーム指向型並列処理の効果を最大限に引き出すための資源割当ての条件は，次のようにまとめられる．

異なるプロセッサに割り当てられたパイプライン・ノード（チェイン型問い合わせにおいては，インナ・リレーションを生成するノード）のアウタ・リレーション用バッファには，アウタ・リレーション全体を格納するバッファ資源を割り当てるか，あるいは，パイプライン・ノード間での並列性抽出の条件[10]を満たすようにバッファ資源を割り当てる．nested-loop あるいは sort-search アルゴリズムを用いた場合，ストリーム型の並列性を抽出するためのバッファ資源割当ての条件は，関係演算の選択率によって決定される．

この場合にも，選択率が与えられていることが前提となっている．

文献 10)では，ストリーム型処理方式における資源割当てを問い合わせのコンパイル時に，静的に行うことを前提としていた．以下では，データベースの統計情報である選択率が与えられていないか，あるいは，正しく与えられていない場合，およびバッファ資源量が動的に変化する場合に対して柔軟に対応できるように，ストリーム指向型処理における資源割当て方式を拡張した動的資源割当て方式を示す．

2.3 動的資源割当て方式

動的資源割当てを行うためには，次のような拡張が必要となる．

（1） ストリーム指向型方式の2項関係演算処理アルゴリズムでは，アウタ・リレーション用バッファの資源量が実行前に決定されていることが前提となっている．動的な環境では，アウタ・リレーション用バッファ資源量が関係演算処理の途中で変わることになる．そのために，ストリーム指向型処理アルゴリズムの拡張が必要となる．

（2） 問い合わせ処理の途中で処理を中断し，資源割当て計算の方法を用いて，新しい資源割当て量を求めるための計算を行い，その結果に従って，バッファ資源の動的な再割当てを行う機構が必要となる．

問い合わせ処理における動的資源割当てに関しては，文献 2),3)等において，データの流量に関する情報をもとに，プロセッサ資源の動的割当てを行う方法が提案されている．

本論文では，ストリーム指向型関係演算処理方式の計算回数の軽減および並列性の抽出のための方式として，問い合わせを構成する各関係演算ノードのプロセッサ群への割当てが静的に行われた後に行われるバッファ資源の動的な割当て方式を提案する．

動的資源割当て方式では，問い合わせ処理の実行時に，各関係演算の実行状態を把握（モニタ）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で，資源割当て計算を行った後，バッファ資源の各関係演算ノードへの動的割当てを行う．

以下では，中間結果のタプル群を生成する関数（関係演算）を生産者ノード，それらを消費する関数（関係演算）を消費者ノードとする．静的資源割当て方式により関係演算を処理する場合には，生産者ノードと消費者ノードの間のバッファ・サイズは演算を実行する前に決定され，演算が完了するまで変化しない．動的資源割当て方式により処理を行う場合には，生産者ノードと消費者ノードの間のストリーム・データの伝達用バッファのサイズは，処理の途中で動的に変化することになる．すなわち，バッファ資源量の変化および並列処理の状態に従って，ストリーム・データの伝達用バッファの容量は，実行時に動的に変化する．ここでは，演算処理における各バッファは，実行時に増加する場合を考える．それは，次の理由による．

1) 統計情報が提供されていないか，あるいは正しく提供されていない場合，まず，各演算に少量バッファを割り当てて問い合わせ処理を進め，データベースの統計情報を予測した後，計算機資源の再割当てを行う方法が考えられる．

ストリーム指向型関係演算処理方式では，問い合わせを構成する各関係演算ノードに少量バッファを割り当てることにより，各関係演算を早い時点で起動することができる．したがって，各演算に対する統計情報，特に，選択率の測定を早い時点で実現できる．この場合には，バッファ資源の動的な割当てにより，各関係演算ノードへのバッファ資源の割当て量が実行時に増加することになる．

2) マルチトランザクションを実現する環境下で問い合わせ処理を行う際に，他のユーザの問い合わせ処理あるいはトランザクション処理の終了時に解放される資源を，実行中の問い合わせ処理に割り当てれば処理効率を高めることができる．この場合にも，実行時にバッファ資源の割当て量が増加することになる．

以下では，動的バッファ資源割当てを適用するストリーム指向型関係演算処理方式を提示する．

［1項関係演算］ 1項関係演算（演算対象リレーション数が1個の関係演算：選択演算等）処理では，動的資源割当ての適用により入力バッファ（サイズ：BS_ADD）が追加されると，追加されたバッファをもとのバッファ（サイズ：BS_OLD）に加えて，この新しいバッファ（サイズ：BS_OLD+BS_ADD）をストリーム・データの受渡しの新しい単位として，引き続いてストリーム指向型関係演算処理を行う．動的資源割当てが行われた後では，生産者ノードは，1デマンドに対して，(BS_OLD+BS_ADD)個のタプルのデータを生成することになる．消費者ノードは，BS_OLD+BS_ADD タプルのデータを単位（グラニュラリティ）として処理を行う．

［2項関係演算］ 2項関係演算（演算対象リレーション数が2個の関係演算：結合演算等）処理では，2つの演算対象リレーションが依存関係をもつので，1項関係演算処理のように，追加されるバッファ資源ともとのバッファ資源を単純に合わせるだけでは問い合わせ処理を遂行できない．ここでは，結合演算を例として，動的バッファ資源割当てを実現するストリーム指向型関係演算処理方式を示す．

図1に示すように，結合演算処理では，アウタ・リレーション（R1）の各ページとインナ・リレーション（R2）の全ページの付き合わせを行うことにより演算が完了する．アウタ・リレーション用バッファ・サイズは，演算処理効率に次のような重要な影響を与える．

① 問い合わせ処理の計算量に対する影響

ストリーム指向型処理方式では，関係演算を限られたバッファ資源の中で行う場合，アウタ・リレーションの主体が入力バッファに入りきらないときには，インナ・リレーションの再生成が必要となる[7),10)]．たとえば，図1において，アウタ・リレーションのページ数が N（ページ，$N>1$），アウタ・リレーション用バッファの大きさを1（ページ）とした場合には，インナ・リレーションを N 回生成する必要がある．

② 問い合わせ処理の並列性に対する影響

生産者ノードと消費者ノード間でストリーム指向型並列性を最大限に引き出すためには，アウタ・リレー

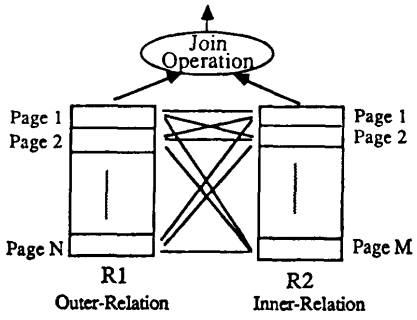


図1 結合演算処理
Fig. 1 Processing for the join operation

ション用バッファは，並列性抽出のための条件[10]を満たさなければならない．

インナ・リレーション用バッファ・サイズは，消費者ノードと生成者ノード間でのストリーム・データの転送回数，デマンドの発行回数に影響を与えるが，①と②に対する影響はない．

そこで，ここでは，インナ・リレーション用バッファ・サイズは固定し，アウタ・リレーション用バッファに対する動的資源割当てを考える．

問い合わせ処理の途中で動的資源割当てが適用され，アウタ・リレーション用バッファに新しいバッファ（サイズ：BS_ADD）が追加される時点で，アウタ・リレーション用バッファ内のタプル群とインナ・リレーションのタプル群との付き合わせ処理について次の2つの場合が考えられる．

1) アウタ・リレーション用バッファ内のタプル群とインナ・リレーション全体との付き合わせ処理がすでに終わっている場合．

2) アウタ・リレーション用バッファ内のデータがインナ・リレーションの1部分との付き合わせしか終わっていない場合である．

1)の場合には，動的に追加されたバッファをもとのバッファと単純に合わせて，そのバッファを新しいストリーム・データの受渡し単位として関係演算処理を行えばよい．

2)の場合には，もとのアウタ・リレーション用バッファ内のタプル群は，インナ・リレーションのすでに比較を終えた部分との付き合わせを行う必要はないが，動的に追加されたバッファ内のタプル群に対してはインナ・リレーションの全部と付き合わせ処理を行わなければならない．それを実現するために，次の3通りの方法が考えられる．

① 問い合わせを構成する2項関係演算群のアウタ・リレーション用バッファへの新しいバッファの追加は，各2項関係演算ごとに，インナ・リレーション全体との付き合わせ処理が完了した時点で行う．

② 動的資源割当てを適用する機構を簡単化するため，追加されたバッファをもとのバッファと単純に合わせてインナ・リレーションの全部との付き合わせ処理を行う．しかし，もとのバッファ（BS_OLD）内のデータとインナ・リレーションの1部分との付き合わせ処理が2回行われて，余分なオーバヘッドが引き起こされる．また，出力結果に対する重複の排除を行わないと，正確な出力データを得ることができない．

③ 動的資源割当てを行った時点の付き合わせ操作を行っているインナ・リレーションのタプルの位置を適用点として保持し，その適用点以降のインナ・リレーションのタプル群は，引続きもとのアウタ・リレーション用バッファ内のタプル群および新しく追加されたアウタ・リレーション用バッファ内のタプル群と付き合わせ処理する．もとのバッファ（BS_OLD）内のタプル群は，すでに適用点以前のインナ・リレーションのタプル群との付き合わせを終えているので，適用点以降のインナ・リレーションのタプル群だけと付き合わせを行うことが必要となる．一方，追加されたバッファ（BS_ADD）内のタプル群は，インナ・リレーションの全タプルとの付き合わせ処理を行わなければならないので，インナ・リレーションを再生成し，動的資源割当てを行った適用点まで，インナ・リレーションの全体との付き合わせ処理を行う．1項関係演算処理のように，アウタ・リレーション用のもとのバッファと追加されたバッファとを単純に合わせて新しいバッファを単位として処理するのではなく，インナ・リレーションの各タプルに対して，それぞれ，もとのバッファ内のタプル群および追加されたバッファ内のタプル群との付き合わせ処理を行うことが必要となる．このように，アウタ・リレーション用バッファの各部分（もとのバッファおよび追加されたバッファ群）は，インナ・リレーションの全体との付き合わせ処理が終わっているか否かを判断するために，動的資源割当ての適用点を保持することが必要となる．この方法の適用は，インナ・リレーションを生成する関係演算群の中に2項関係演算が含まれない場合，あるいは，2項関係演算が含まれていても，そのアウタ・リレーション用バッファに新たにバッファが追加されていない場合に限られる．③の場合は，動的資源割当てを実現する演算処理では，アウタ・リレーション用バッファはいくつかの部分バッファから構成されるので，ストリーム・データおよびデマンドの受渡しは，部分バッファを単位として行われることになり，デマンドの発行回数が多くなる．しかし，アウタ・リレーションに関しては，同じストリーム・データの生成に対しては，高々1回のデマンドが発行されるだけなので，追加されたバッファのサイズが適当であれば，デマンドの発行のためのオーバヘッドは大きくない．

③の方法により動的資源割当てを適用した場合の一般的な2項関係演算処理アルゴリズムを示す．

ここでは，アルゴリズムを明確化するためにスト

リーム・データおよびデマンドの転送の記述を省略して，ストリーム・データの受渡し用バッファ（アウタ・リレーション用バッファ）に対する操作に着目してアルゴリズムを示す．

動的資源割当てを実現する結合演算処理アルゴリズム

1）ここでは，適用時点の異なる r 回の動的バッファ資源割当て（静的資源割当てが1回として含まれている）が行われた場合を考える．アウタ・リレーションに対しては，アウタ・リレーション用バッファ（r 個の部分からなる）を介してインナ・リレーションとの間で付き合わせが行われる．ここでは，アウタ・リレーション用バッファの r 個の部分バッファのリストは，次のように表すものとする．

$BUF_1, BUF_2, \cdots, BUF_r$.

各々の部分バッファに対応して，動的適用点（インナ・リレーションにおける位置）が保持される．動的適用点のリストは次のように記述する．

$DAP_1, DAP_2, \cdots, DAP_r$.

また，インナ・リレーション内の現在付き合わせ処理を行っているタプルの位置を CP とする．

2）インナ・リレーションの1タプルに対して r 個の部分バッファからなるアウタ・リレーション用バッファ内のタプル群と各々付き合わせ処理を行う（この付き合わせでは，アウタ・リレーション用バッファの各部分バッファ内のタプル群がソートされているならば，バイナリ・サーチ・アルゴリズムが用いられ，また，ソートされていないならば，nested-loop アルゴリズムが用いられる）．インナ・リレーションの残りタプルが存在しないならば，インナ・リレーションの生産者ノードに対し，インナ・リレーションの再生成（再計算）要求[10]を発行する．

3）$CP=DAP_i$ $(i=1, 2, \cdots, r)$ ならば，つまり，BUF_i のタプル群がすでにインナ・リレーション全体との付き合わせ処理を完了しているならば，アウタ・リレーションを生成する生産者ノードから新しいタプル群を部分バッファ BUF_i に格納し，インナ・リレーションの次の1タプルを受け取って，2)に行く．もしアウタ・リレーションの残りタプルが存在しないならば，部分バッファ BUF_i を解放し，BUF_i をバッファ・リストから，DAP_i を適用点リストから削除して5)に行く．

4）$CP \neq DAP_i$ $(i=1, 2, \cdots, r)$ ならば，つまり，BUF_i 内のタプル群がインナ・リレーションの全体との付き合わせ処理を完了していない場合には，インナ・リレーションの次の1タプルを受け取って，2)に行く．

5）アウタ・リレーション用バッファの部分バッファのリストに要素が存在しない場合には，演算処理が終わる．さもなければ，インナ・リレーションの次の1タプルを受け取って，2)に行く．

ここでは，例を用いて動的資源割当てを適用した場合の結合演算処理の動きを説明する．

図2に示す例では，最初，アウタ・リレーション用バッファ BUF_1 を用いて結合演算処理が開始され，問い合わせ処理途中の DAP_2 時点（動的適用点）で動的資源割当てが行われ，バッファ BUF_2 がアウタ・リレーション用バッファに追加され，そして，さらに，問い合わせ処理途中の DAP_3 時点で動的資源割当てが行われて，バッファ BUF_3 が追加される．結合演算処理の動きは，図2に示すように，インナ・リレーションの x の部分の各タプル群は BUF_1 内のタプル群と，y の部分の各タプル群は BUF_1 および BUF_2 のタプル群と，また，z の部分のタプル群は BUF_1, BUF_2 および BUF_3 内のタプル群との間で付き合わせられる．それらの付き合わせが終了した時点で，バッファ BUF_1 内のタプル群はインナ・リレーションの全タプルとの付き合わせ処理を完了する．そのとき，もしアウタ・リレーションのストリーム・データが終端を迎えていないならば，新しいストリーム・データを BUF_1 に読み込む．もし BUF_1 に格納すべきタプル群が存在しないとき（アウタ・リレーション・ストリームの終端がすでに他のバッファ内に格納され

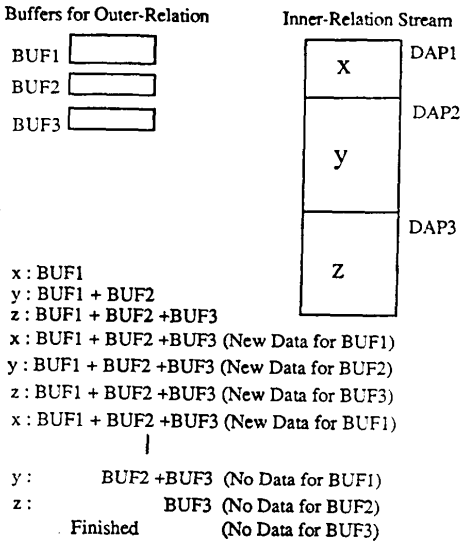


図2 動的資源割当て方式における関係演算処理
Fig. 2 Relational operation processing in the dynamic resource allocation strategy.

ているとき）には，バッファ BUF_1 は解放される．そのとき，DAP_1 は動的適用点リストから消去される．同様に，BUF_2 および BUF_3 に格納すべきタプル群が存在しなくなったとき，バッファ BUF_2 およびバッファ BUF_3 が解放され，関係演算処理を完了する．

3. 動的資源割当て方式の実現

3.1 動的資源割当ての処理系の構成

動的資源割当ての処理系の概観を図3に示す．この処理系は，6つのモジュール m1～m6 によって構成されている．以下に，各々のモジュールの役割について述べる．

モジュール m1: Query Transformation

問い合わせをコンパイルし，関係演算 (Join, Selection, Projection 等) をノードとする関係演算木に変換する．

モジュール m2: Resource Allocation

問い合わせ処理において参照されるデータベースの統計情報（リレーション・サイズ，選択率），あるいは，問い合わせの1部分の処理により得られた情報，および計算機資源に関する情報を利用して，各関係演算ノードに対して計算機資源の割当てを行う．つまり，各関係演算ノードのプロセッサへの配置，および各関係演算ノードに割り当てるバッファ資源量を決定する．

モジュール m3: Query Processing

動的資源割当て方式を適用するストリーム指向型関係演算処理方式により，関係演算間に内在する並列性を引き出しながら問い合わせ処理を行う．関係演算の処理が完了した後，計算機資源を解放する．

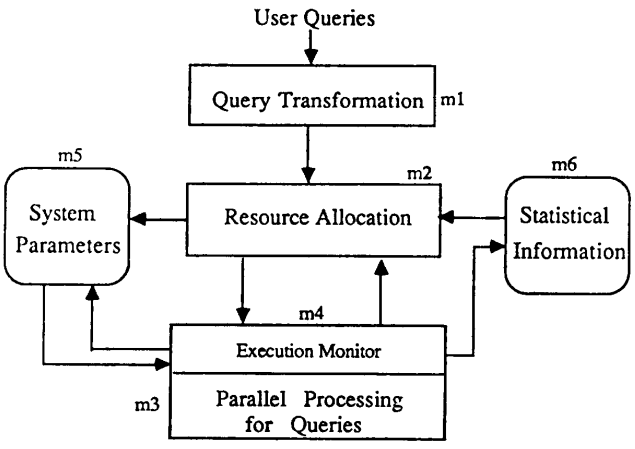


図3 動的資源割当ての処理系
Fig. 3 An overview of the dynamic resource allocation system.

モジュール m4: Execution Monitor

問い合わせ処理の並列性が十分に抽出されているか，問い合わせ処理へ動的資源割当てを適用する必要があるかを判断するために，問い合わせ処理の実行状態をモニタリングし，ベース・リレーションあるいは中間リレーションのサイズ，演算選択率等の統計情報の測定を行う．

モジュール m5: System Parameters

システム・パラメータを把握するモジュールである．システム・パラメータには，システムのバッファ・サイズ，プロセッサの台数が含まれている．単一ユーザ，単一問い合わせあるいは単一トランザクション処理を行う場合，これらのパラメータは一定であるが，マルチ問い合わせあるいはマルチトランザクションの場合には，これらのパラメータは，システムの実行時に動的に変化する．

モジュール m6: Statistical Information

データベースに関する統計情報を格納する．ここには，リレーションのサイズ，演算選択率，およびリレーションが配置されているサイト名が保持される．

3.2 統計情報の予測方法および動的資源割当ての適用時点

データベースに関する統計情報，特に，結合演算選択率の予測は，データベース処理の効率を高めるために重要である[13),15)]．ストリーム指向型処理方式では，問い合わせを構成する各関係演算ノードは，少量の演算対象データが揃った時点で起動可能となるので，まず，少量のバッファ領域を各関係演算ノードに配置して各関係演算を起動し，そこで，結合演算選択率を予測することが可能である．

2.2 節で述べたように，アウタ・リレーション用バッファのサイズは，関係演算間での処理の並列性および計算量に影響を与える．各関係演算ノードを起動するためには，アウタ・リレーションの1バッファ分のタプル群の生成を待つ必要があるので，アウタ・リレーション用バッファのサイズを大きく設定すると，各関係演算の実行を開始するまでの時間が長くなり，結果として，選択率を把握するまでの時間が長くなってしまう．そこで，選択率を予測するために，アウタ・リレーション用バッファのサイズを小さく設定することが必要である．

結合演算選択率の予測は次のようにして行うこ

とができる．インナ・リレーションの各タプルとアウタ・リレーション用バッファ内のタプル群の間の付き合わせアルゴリズムとして，nested-loop を用いた場合，演算結果の1タプルを生成するのに，平均して

1/jsf

(jsf：結合演算選択率，(中間リレーションのタプル数)＝jsf＊(アウタ・リレーションのタプル数)＊(インナ・リレーションのタプル数)) 回の比較操作が行われる．したがって，演算結果として生成されたタプル数と比較操作回数の関係から，結合演算選択率を予測することができる．

また，文献 10)において示したように，sort-search アルゴリズムを用いた場合には，演算結果の1タプルを生成するのに，平均して

$(\log_2 BS_i + jsf_i * BS_i)/(jsf_i * BS_i)$

(BS_i：結合演算ノード (i) のアウタ・リレーション用バッファのサイズ（単位はタプル数），バッファ内のタプル群は，すでにソートされているものとする．jsf_i：結合演算ノード (i) の結合演算選択率) 回の比較操作が行われる．したがって，演算結果として生成されたタプル数と比較操作回数の関係から，この式を用いて結合演算選択率を予測することができる．

動的資源割当て方式においては，動的資源割当ての適用時点の選択が重要である．

3.1 節で示した Execution Monitor は，次のような評価式により動的資源割当ての適用時点を決定する．

文献 10)では，ベース・リレーションのサイズおよび結合演算選択率を利用して，問い合わせ処理における総計算量を求める計算式を提示している．本方式では，その計算式を用いて動的資源割当ての適用前後の総計算量の計算を行い，下の条件を満たす場合に，動的資源割当てを行う．

$$TC_b - TC_a - D > c \quad (c \geqq 0)$$

TC_b：動的資源割当てを適用することなく，引続き処理を継続する場合，処理が完了するまでの総計算量

TC_a：動的資源割当てを適用して，処理が完了するまでの総計算量

D：新しい資源割当てを行うために要する時間

ここで，c は TC_b と TC_a の差が十分大きくなったかどうかを判断するための値であり，問い合わせの性質，システムの稼働状況等に応じて設定される．

4. 実　　験

文献 10)では，ストリーム指向型関係演算処理系[8]を用いて行った実験結果を示し，静的資源割当て方式の有効性を明らかにした．ここでは，その処理系を用い，さらに動的資源割当て方式を実現する処理系を用いて，動的資源割当てを適用する場合の問い合わせ処理の効率を評価するために実験を行った．実験環境は次のとおりである．

（1） Sun-2 ワークステーション[12]（プロセッサ：MC 68010, OS：UNIX 4.2 BSD) 上に，2.3 節で述べた動的資源割当てを適用するストリーム指向型関係演算処理方式により実際に問い合わせ処理を行い，各関係演算の実行時間を実測した．

（2） 複数台のプロセッサによる並列処理の効果を測定する際，プロセッサ間のデータ転送（ストリーム・データおよびデマンドの送出）時間だけは計算式によって求め，関係演算の処理時間は(1)の処理系を用いて実測した．データ転送の速度については，プロセッサ間データ転送速度を 16 msec/2 k-byte packet に設定した[8),11)]．

（3） 問い合わせの実行時間については，Sun-2 ワークステーションによってモニタされる実際の CPU 時間およびディスク・アクセス処理等のシステム・コール処理の実行時間を実測した．

（4） 資源割当てに要する時間は，実際に資源割当て計算を行うプログラムを実行することによって測定し，問い合わせ処理時間内に組み込んだ．

4.1 問い合わせ

動的資源割当ての効果と静的資源割当ての効果を比較するために，文献 10)において用いた**図 4** に示すような構造をもつ3つの結合演算からなる4種類の問い合わせを対象として実験を行った．

各問い合わせにおけるベース・リレーションのタプル数，結合演算選択率は，各問い合わせにおいて**表 1** のように設定した．結合演算選択率については，それが問い合わせ処理を行う前に提供されている場合と，提供されていない場合について実験を行った．

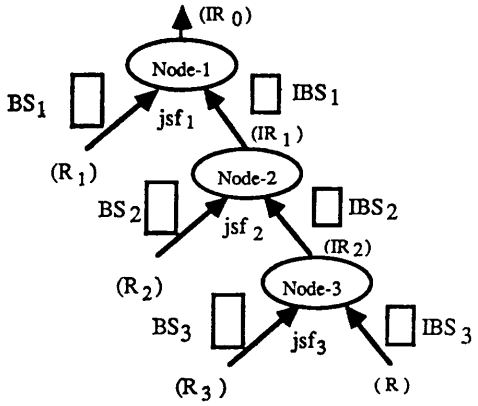
ベース・リレーションのタプル数は，各問い合わせにおいて各々同じ値（1024 タプル）に設定する．各ベース・リレーションのタプル長は 64 バイトとし，各タプルは結合演算対象属性として2つの整数型属性（各4バイト）を含む．それらの整数型属性内の値は一様分布の乱数を用いた．

結合演算においてインナ・リレーションの1ページとアウタ・リレーションの1ページを比較するアルゴリズムとしては，sort-search アルゴリズムを用いた．

4.2 実験結果

実験結果を表2に示す．Query1, Query2, Query3, Query4 の case 1 は，全結合演算ノードのアウタ・リレーション用バッファの総和（タプル数：BS）が BS＝1800（タプル）のとき，静的資源割当ておよび動的資源割当ての適用を行わず，各結合演算ノードのアウタ・リレーション用バッファに均等にバッファ領域を割り当てて問い合わせ処理を行った場合である．

Query 1～4 の case 2 は，表1に示す統計情報を用いてバッファ資源割当て計算[10]（拡大ラグランジュ関数法によりバッファ割当ての値を求め，修正アルゴリズムによってその値を修正する計算）の結果に従って静的にバッファ資源を割り当てた後，問い合わせ処理を行った場合である．



BSi(i=1,2,3) : Outer-Relation Buffer Size(the number of tuples)
IBSi(i=1,2,3) : Inner-Relation Buffer Size
Ri(i=1,2,3) : Outer-Relation Size
IRi(i=0,1,2) : Inner-Relation (Intermediate Relation) Size
R : Base-Relation (Inner-Relation) Size
jsfi(i=1,2,3) : Join Selectivity Factor

図 4 評価用問い合わせの構造
Fig. 4 A structure of evaluated queries.

表 1 評価用問い合わせにおけるベース・リレーション・サイズおよび結合演算結合率
Table 1 Base-relation sizes and join-selectivity-factors in the evaluated queries.

		Query 1	Query 2	Query 3	Query 4
R1, R2, R3, R		1024	1024	1024	1024
Node 3	jsf 3	1/1024	1/512	1/512	1/2048
	IR 3	1024	1024	1024	1024
Node 2	jsf 2	1/1024	1/1024	1/512	1/2048
	IR 2	1024	2048	2048	512
Node 1	jsf 1	1/1024	1/1024	1/512	1/2048
	IR 1	1024	2048	4096	256
output relation	IR 0	1024	2048	8192	128

(R 1-R 3, IR 0-IR 3, R: the number of tuples)

表 2 実験結果
Table 2 Experimental results.

	Query 1			Query 2			Query 3			Query 4		
	case 1	case 2	case 3	case 1	case 2	case 3	case 1	case 2	case 3	case 1	case 2	case 3
BS 1	600	1024	[100] (1024)	600	1024	[100] (1024)	600	1024	[100] (1024)	600	1024	[100] (1024)
BS 2	600	512	[100] (512)	600	512	[100] (512)	600	512	[100] (512)	600	512	[100] (512)
BS 3	600	264	[100] (264)	600	264	[100] (264)	600	264	[100] (264)	600	264	[100] (264)
SP (sec)	82.7	52.3	56.4	105.6	65.5	69.1	141.9	99.9	104.2	68.5	46.7	49.7
PP (sec)	47.5	27.3	29.2	56.1	29.7	32.6	61.6	38.8	40.1	43.2	27.5	29.2
SP/PP	1.74	1.92	1.86	1.88	2.21	2.12	2.30	2.57	2.60	1.59	1.70	1.70

$BS = \sum_{i=1}^{3} BS_i$

BS 1～BS 3: Outer-Relation Buffer Size (number of tuples)
IBS 1～IBS 3: Inner-Relation Buffer Size (=100 tuples)
[ ]: Initial Buffer Size for Outer-Relation Buffer (number of tuples)
(Buffer Size before Dynamic Resource Allocation)
( ): Buffer Size after Dynamic Resource Allocation
SP: Execution Time in Sequential Processing Case
PP: Execution Time in Parallel Processing Case

Query 1-4 の case 3 は，問い合わせ処理を行う前に選択率に関する統計情報が提供されていない場合である．すなわち，静的資源割当てを行うことができない場合であり，まず3.2節で提示した手法で統計情報を予測し，動的資源割当てを適用して，2.3 節で提示した動的資源割当て方式によりバッファ資源を割り当てた後，問い合わせ処理を行った場合である．初期のバッファ割当てとして，ここでは，統計情報を予測するために，それぞれのアウタ・リレーション用バッファ BS 1～BS 3 を，インナ・リレーション用バッファ (＝100 タプル) と同じ量に設定した．ここでは，資源割当ての計算を行うための時間は，逐次処理の場合に問い合わせの実行時間に含まれている．

各 case の逐次実行および並列実行の場合の実行時間および逐次実行時間と並列実行時間の比を表2に示す．逐次実行の場合は，各結合演算が1台のプロセッサにより擬似並列に実行される．また，並列実行の場合は，各関係演算のアウタ・リレーションの各ページのアクセスおよび各結合演算の実行は，異なる3台のプロセッサにより実行され，また，Node-3 のインナ・リレーションRの各ページのアクセスは，他の1台のプロセッサにより実行される．

各問い合わせの各 case において，case 1 は，問い合わせ処理の効率が最も低い場合であり，また，case 2 は，データベースに関する統計情報を用いた静的最適化を行った後問い合わせを処理する理想的な場合である．case 3 は，問い合わせを処理する前に計算機資源を最適に割り当てることができないので，動的資源割当て方式を適用する場合である．表2に示すように，逐次処理の場合には，各問い合わせの case 1 と case 3 の処理時間の比はそれぞれ，82.7：56.4，105.6：69.1，141.9：104.2，68.5：49.7 であり，並列処理の場合には，それぞれ，47.5：29.2，56.1：32.6，61.6：40.1，43.2：29.2 である．また，各問い合わせの case 2 と case 3 の逐次実行および並列実行の場合の比較において，case 3 は，理想的な場合である case 2 に近い処理時間になっている．3.2 節で述べた統計情報の予測時間は，Query 1 の case 3 の場合に 2.4 秒と短く，理想的な場合に近い実行時間で問い合わせ処理を行うことができることを示している．また，資源割当て計算に要するオーバヘッドは文献 10)において示したように関係演算処理時間と比較してほとんど無視できる．

図5～図8は，Query 1 および Query 3 の処理を例として，動的資源割当てを適用する際のバッファ資源量の増加が問い合わせ処理の実行時間に与える影響を表している．それらは，逐次処理環境および並列処理環境において動的資源割当てを適用する際，各関係演算に均等にバッファ資源を割り当てた場合，および，資源割当て計算による最適バッファ割当てを行う場合の実行時間を示している．

前述したように，動的資源割当ての適用は，適用時点および問い合わせ処理に利用できるバッファ資源量に依存する．ここでは，それぞれ問い合わせ処理において，問い合わせ処理に利用できるバッファ資源量が動的に追加される場合については，その動的適用時点を次のように4通り設定した．

問い合わせ Query 1：問い合わせ処理の1タプル目 ($t=1$)，100 タプル目 ($t=100$)，500 タプル目 ($t=500$)，800 タプル目 ($t=800$) の結果が得られた時点．

問い合わせ Query 3：問い合わせ処理の1タプル目 ($t=1$)，100 タプル目 ($t=100$)，1000 タプル目 ($t=1000$)，5000 タプル目 ($t=5000$) の結果が得られた時点．

また，Query 1，Query 3 の各適用時点について，各々次のような実験を行った．

実験①：均等的にバッファ資源を静的に割り当てて処理を行う．また，動的資源割当てを適用しない．(EPS)

実験②：データベースに関する統計情報を用い，文献 10)で提示した方法で最適にバッファ資源割当てを行って処理するが，動的資源割当てを適用しない．(OPS)

実験③：動的資源割当てを適用するが，バッファ資源を均等に割り当てる．(EPD)

実験④：動的資源割当てを適用し，また，2.3 節で提示した方法で最適にバッファ資源割当てを行う (OPD)．これは本稿での提案方式である．ただし，動的資源割当ての適用時点では正確な統計情報が得られているものと仮定する．一般的には，より多くの結果が得られた時点においてより正確な選択率を把握できると考えられるが，ここでは，簡単のために，このような仮定を行った．

各問い合わせ処理では，1024 タプルのバッファ資源で問い合わせ処理を開始し，実行途中で利用できるバッファ資源量が増加する．図5～図8の横軸は，その増加分を表している．図5～図8において，実線は最適バッファ割当てを行った場合，点線は均等割当て

を行った場合の実験結果を表している．

図5は，問い合わせQuery 1の逐次処理の実行時間とバッファ増加分の関係を示している．実験①(EPS)および実験② (OPS) の場合には，動的資源割当てを適用しないので，バッファ容量の動的変化が実行時間に影響を与えていない．

実験③(EPD)および実験④(OPD)の場合には，動的資源割当てを適用しており，実行時間が短縮されている．特に，実験④の場合には，動的資源割当て方式において，計算量を最小とするためのバッファ割当てを行っているので，最も効率が良い．また，バッファ量の増加分が512(タプル) までの実行時間の短縮が顕著となっている．すなわち，総バッファ資源量が少なく制限される場合には，動的資源割当ての効果が大きいことがわかる．バッファ容量の増加分が2048タプル以上である場合には，全アウタ・リレーションの全タプルがバッファに入りきるので，それ以上バッファ資源の割当てを増やしても実行時間は変化しない．

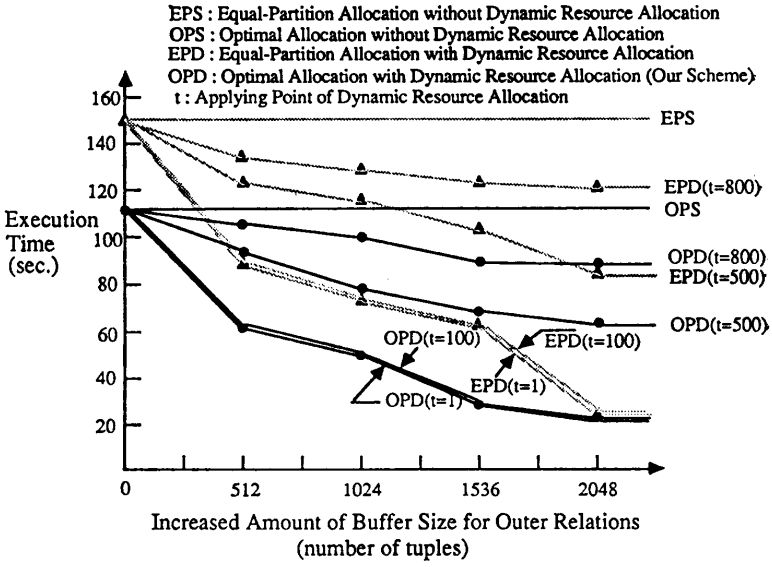


図5 バッファ資源の増加量と実行時間の関係 (Query 1: 逐次処理の場合)
Fig. 5 Relationship of execution time and increased amount of buffer size (Query 1: sequential processing case).

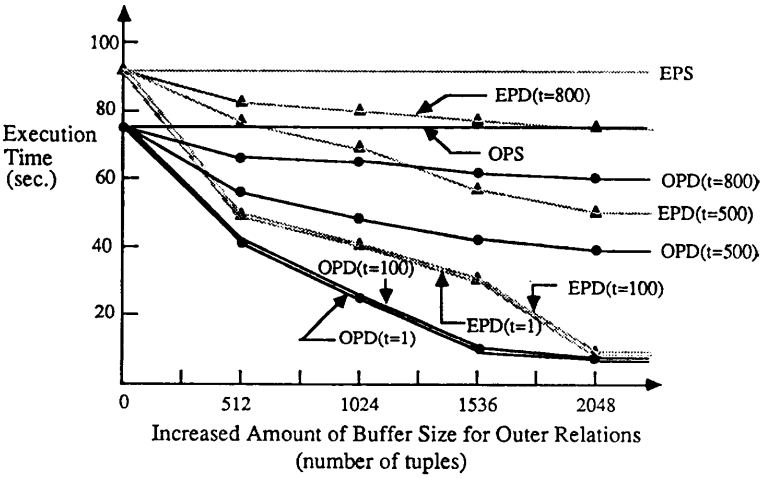


図6 バッファ資源の増加量と実行時間の関係 (Query 1: 並列処理の場合)
Fig. 6 Relationship of execution time and increased amount of buffer size (Query 1: parallel processing).

図6は，並列処理の実行時間とバッファ・サイズの増分の関係を示している．並列処理の場合には，逐次処理の場合と同様に，動的バッファ資源割当ての適用により計算回数を減らすことができる．その上，問い合わせ処理に利用可能なバッファ容量の増加は，関係演算間での処理の並列性にも影響を与える．したがって，図5の逐次処理の場合と比較において，並列処理の場合は，より多いバッファ量 (1536タプルまでのバッファ量) の増加において，実行時間の短縮が顕著である．

Query 3の実験結果 (図7と図8) については，Query 1と同様，動的資源割当ての適用および最適資源割当て計算により実行時間が短縮されている．ただし，Query 1の実験結果と比べて，512タプルまでのバッファ量の増加において，並列処理の実行時間の短縮が顕著であるが，それ以上の増加に対しては効果が小さくなっている．これは，バッファ増加分が512タプル以内である場合 (関係演算間での並列処理を最大限に引き出していない場合) には，バッファ量の増加が計算回数の軽減および並列性の抽出の両方に有効であり，それ以上のバッファ量の追加は，計算回数の軽減だけに影響を与えているからである．

動的資源割当ての4つの適用時点に対する実験結果から，動的資源割当ての適用時点が早いほど，その効

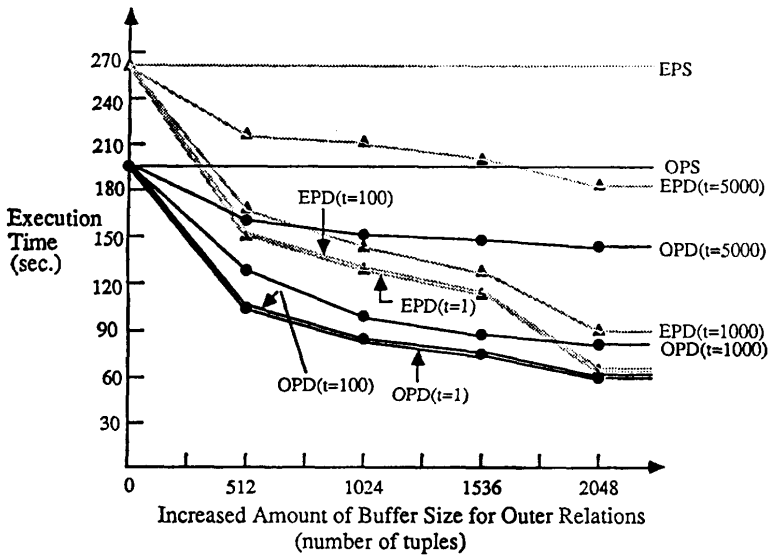


図 7 バッファ資源の増加量と実行時間の関係 (Query 3: 逐次処理の場合)
Fig. 7 Relationship of execution time and increased amount of buffer size (Query 3: sequential processing case).

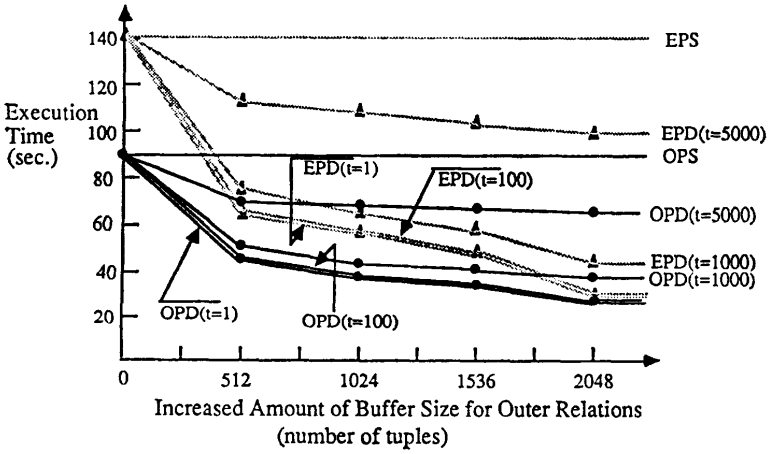


図 8 バッファ資源の増加量と実行時間の関係 (Query 3: 並列処理の場合)
Fig. 8 Relationship of execution time and increased amount of buffer size (Query 3: parallel processing case).

果が大きいことがわかる．1タプル目の結果および100タプル目の結果が得られるまでの実行時間はほとんど差異がないので，動的資源割当てによる問い合わせの実行時間は両者の場合にはほとんど同じである．

並列処理において，動的資源割当ての適用が遅い場合（たとえば，Query 1 の処理に対して 800 タプル目の結果が得られた時点，あるいは，Query 3 の処理に対して 5000 タプル目の結果が得られた時点で，適用する場合）には，512 タプルまでのバッファの増加の効果が大きいが，それ以上の場合には，効果は小さい．これは，計算量の軽減および並列性の抽出のために，ある程度のバッファ資源がある場合，資源量の増加による実行時間への影響が小さくなることを示している．

5. むすび

本論文では，ストリーム指向型関係演算処理方式を並列処理システム上で実現する場合の動的資源割当て方式を提案した．まず，データベースに関する統計情報が提供されていない場合，および，問い合わせ処理の途中で利用できるバッファ資源量が動的に変化する場合において問い合わせ処理を継続して行うために，ストリーム指向型関係演算処理のアルゴリズムを拡張した．また，動的資源割当て方式を適用するための統計情報の予測，および，動的資源割当ての適用時点の決定を行うための指針を示した．

最後に，実際に問い合わせ処理の実験を行い，本論文で提案した動的資源割当て方式の有効性を明らかにした．

本論文では，関係データベースへの問い合わせのように，問い合わせのコンパイル時に演算の組合せを決定できる場合のバッファの動的資源割当てについて述べた．しかし，演繹データベースへの再帰的問い合わせのように，問い合わせの実行過程で演算を動的に呼び出して実行する必要がある場合には，問い合わせのコンパイル時に演算の組合せを必ずしも決定できないので，資源割当てはより複雑となる．その問題の解決は，今後の課題となろう．

我々は，現在，データベースおよび知識ベースを対象とした並列処理システム SMASH の構築を進めており[6),9)]，本論文で述べた方式は，この並列処理システムに組み込まれる予定である．

謝辞 本研究にあたり，有意義な討論をしていただいた筑波大学工学研究科加藤和彦氏に感謝いたします．

参考文献

1) Amamiya, M. and Hasegawa, R.: Dataflow Computing and Eager and Lazy Evaluations, *New Generation Computing*, Vol. 2, No. 2, pp. 105-129 (1984).
2) Amstrong, W. W. and Mohamed, A. S.: A Mixed-Flow Query Processing Strategy for a Multiprocessor Database Machine, *Proc. 6 th Int. Conf. Distributed Computing Systems*, pp. 292-299 (1985).
3) Boral, H. and DeWitt, D. J.: Processor Allocation Strategies for Multiprocessor Database Machines, *ACM Trans. on Database Systems*, Vol. 6, No. 2, pp. 227-256 (1981).
4) Friedman, D. P. and Wise, D. S.: Aspects of Applicative Programming for Parallel Processing, *IEEE Trans. Comput.*, Vol. C-27, pp. 289-296 (Apr. 1978).
5) Gallaire, H., Minker, J. and Nicolas, J. M.: Logic and Databases: a Deductive Approach, *ACM Comput. Surv.*, Vol. 16, No. 2, pp. 153-185 (Jun. 1984).
6) 加藤, 清木, 益田：データベースおよび知識データベースを対象とした並列処理システム：SMASH, 信学技報, Vol. 87, No. 245, pp. 25-32 (1987).
7) 清木, 長谷川, 雨宮：先行・遅延評価機構を用いた関係演算処理方式, 情報処理学会論文誌, Vol. 26, No. 4, pp. 685-695 (1985).
8) Kiyoki, Y., Kato, K. and Masuda, T.: A Relational Database Machine Based on Functional Programming Concepts, *Proc. ACM-IEEE Computer Society Fall Joint Computer Conf.*, pp. 969-978 (Nov. 1986).
9) Kiyoki, Y., Kato, K., Yamaguchi, N. and Masuda, T.: A Stream-Oriented Approach to Parallel Processing for Deductive Databases, *Proc. 5 th Int. Workshop on Database Machines*, pp. 102-115 (1987).
10) 清木, 劉, 益田：関係演算のストリーム指向型並列処理における資源割り当て方式, 情報処理学会論文誌, Vol. 28, No. 11, pp. 1177-1192 (1987).
11) Lu, H. and Carey, M. J.: Some Experimental Results on Distributed Join Algorithms in a Local Network, *Proc. 11th Int. Conf. on VLDB*, pp. 292-304 (1985).
12) Programmers Reference Manual for the Sun Work-station, Sun Micro Systems, Inc. (1982).
13) Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. and Price, T. G.: Access Path Selection in a Relational Database System, *Proc. of the ACM-SIGMOD Conf.*, pp. 23-34 (1979).
14) Treleaven, P. C., Brownbridge, D. R. and Hopkins, R. P.: Data-Driven and Demand-Driven Computer Architecture, *ACM Comput. Surv.*, Vol. 14, No. 1, pp. 93-144 (Mar. 1982).
15) Yu, C. T. and Chang, C. C.: Distributed Query Processing, *ACM Comput. Surv.*, Vol. 16, No. 4, pp. 399-433 (Dec. 1984).

(昭和63年1月7日受付)
(昭和63年4月14日採録)

劉　澎（正会員）

1961年生. 1982年中国南京工科大学電子計算機科学学科卒業. 1986年筑波大学大学院修士課程理工学研究科修了. 現在同大学院博士課程工学研究科に在学中. データベースシステム, 並列処理, 関数型プログラミング言語に興味をもつ. ACM, IEEE 各会員.

清木　康（正会員）

昭和31年生. 昭和53年慶応義塾大学工学部電気工学科卒業. 昭和58年慶応義塾大学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. 昭和59年より筑波大学電子・情報工学系に勤務, 現在同学系講師. 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. 電子情報通信学会, ACM 各会員.

益田　隆司（正会員）

昭和14年生. 昭和38年東京大学工学部応用物理学科卒業. 昭和40年同大学院修士課程修了. 同年(株)日立製作所入社. 中央研究所, システム開発研究所に勤務. 昭和52年筑波大学電子・情報工学系講師, 昭和53年助教授, 昭和59年教授. 昭和63年3月から東京大学理学部情報科学科教授. 工学博士. オペレーティング・システムの研究開発を経て, その方式論, 計算機システムの性能評価の研究に従事.