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†

劉 澎†† 清木 康††† 益田隆司††††

データベースや知識ベースを対象とする問い合わせの処理においては，問い合わせを構成する演算群への計算機資源（プロセッサ資源およびバッファ資源）の割り当てが処理効率に大きな影響を与える．我々は，関係データベースへの問い合わせの処理方式として，ストリーム指向型関係演算処理方式を提案している．この処理方式の有効性を引き出すためには，計算機資源の割り当てが重要である．本論文では，ストリーム指向型関係演算処理方式の有効性を引き出すためのバッファ資源の最適な割り当てを決める新しい計算方法を提案する．この方法により，任意の構造の問い合わせに対して，最適なバッファ資源割り当てを必ず決定することが可能となる．この方法は，問い合わせ処理に要する計算回数，問い合わせ処理に要する I/O 処理回数，あるいは，両者を考慮した計算量を最小にするバッファ資源割り当てを決定する場合に用いることができるので，適用範囲が広い．

1. まえがき

データベースや知識ベースのような大量データを対象とする問い合わせの処理においては，問い合わせを構成する演算群への計算機資源の割り当てが処理効率に大きな影響を与える（文献 2)〜5)，11)〜14）等）．

我々は，関係データベースへの問い合わせの並列処理方式として，ストリーム指向型関係演算処理方式を提案し[7]，その方式を並列処理システム上で実現する場合の計算機資源（プロセッサ資源およびバッファ資源）の割り当てを決める方法[9]を提案している．ストリーム指向型関係演算処理方式は，リレーションに対する関係演算を，ストリームに対する関数計算に対応させ，要求駆動型制御による関数型計算（文献 1)，6)，15)，16)等参照）の枠組みの中で問い合わせの並列処理を実現する方式である．この方式は，問い合わせを構成する関係演算群の計算の進行を要求駆動により制御することにより，限られた計算機資源（プロセッサおよびバッファ資源）の中で問い合わせ処理を実現できる点を特徴とする．

関係データベースへの問い合わせを限られたバッファ資源の中で処理する方式として，Won Kim によって提案された nested-block-method[4] がある．nested-block-method は逐次処理を対象としたアルゴリズムであるのに対し，ストリーム指向型関係演算処理方式は，並列処理を指向している点，および，アルゴリズムだけでなく，並列処理の制御方法を含めたものである点で異なる．しかし，アルゴリズムに関しては，ストリーム指向型関係演算処理方式では，1台のプロセッサに対して，問い合わせを構成する複数の関係演算を割り当て，それらを擬似並列的に逐次処理する場合があり，その場合には，プロセッサ内の両者の処理アルゴリズムは，似かよったものとなる．両者のアルゴリズムに関する共通点は，プロセッサ内の処理において，問い合わせを構成する関係演算群に対して，限られたバッファ資源を分割して割り当てることにより，各関係演算を一部分ずつ交互に駆動し，一時に中間結果リレーションの完全な生成を行うことなく，限られたバッファ資源の中で問い合わせを処理できる点にある．

nested-block-method では，ストリーム指向型処理方式と同様に，各関係演算の演算対象リレーションに対するバッファ資源の割り当てが，処理効率に大きな影響を与える．文献 4）では，リレーションの I/O 処理回数を軽減するために，ヒューリスティック・アルゴリズムを用いた資源割り当て計算を行う方法が提案されている．しかしながら，そのアルゴリズムは，次の点により，必ずしも一般的ではない．

（1） 対象とする問い合わせの構造がチェイン型問い合わせに限られており，任意の構造の問い合わせを対象とできない．また，チェイン型問い合わせにおいて，結合演算間の実行順序を，演算対象リレーションの大きさの順に実行するように組み換えることができ

† A Computation Method for Buffer Resource Allocation in the Stream-oriented Processing Scheme for Relational Database Operations by PENG LIU (Doctoral Program in Engineering, University of Tsukuba), YASUSHI KIYOKI (Institute of Information Sciences and Electronics, University of Tsukuba) and TAKASHI MASUDA (Department of Information Science, Faculty of Science, University of Tokyo).

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

るという前提が必要である．

（2） 総 I/O 処理回数を求める式を設定し，その式を近似することにより，その式の値（I/O 処理回数）を軽減するバッファ資源割り当て計算を単純化しているが，その近似のために，特別の場合にしか I/O 処理回数の最小値を求めることができなくなっている．すなわち，計算により得られた資源割り当ては，必ずしも I/O 回数を最小とする割り当てとはならない．

（3） I/O 処理回数の軽減を対象として資源割り当てを行うので，関係演算の計算回数を軽減するためのバッファ量の割り当てに関しては考慮されない．

我々は，文献9）において，ストリーム指向型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要する計算回数を最小化するためのバッファ資源割り当て計算の方法を示した．この方法は，共有メモリ型並列処理マシン上での共有メモリの資源割り当て計算にも適用できる．この方法は，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 (selectivity factor) および各関係演算ノードへ割り当てる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，タプル数および選択率を与えて，さらに総バッファ資源量を一定とする制約条件のもとで，式の値を最小にする各関係演算ノードへのバッファ資源の割り当てを，拡大ラグランジュ関数法を用いて求める．

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

このバッファ資源割り当て計算の方法は，次の点が特徴である．

（1） 任意の構造の問い合わせを対象とすることができる．また，問い合わせ内の関係演算の実行順序を組み換える必要がない．

（2） 問い合わせ処理に要する総計算回数を最小にするバッファ資源割り当てを必ず決定することができる．

（3） この方法は，総計算回数を表す式の最小化を拡大ラグランジュ関数法を用いた数値計算により行うものであり，また，整数問題として総計算回数の最小値を解いていないために，バッファ資源配置の修正が必要となる．また，文献9）では，問い合わせ処理に要する計算回数の最小化だけを対象としていた．

本論文では，ストリーム指向型関係演算処理方式のバッファ資源割り当てを決定する新しい計算方法を提案する．この方法は，文献9）において示した方法と同様に，任意の問い合わせを対象とし，総計算回数を最小にするバッファ資源割り当てを任意の構造の問い合わせに対して必ず決定できる．さらに，この方法は，文献4）が扱った I/O 処理に関しても，任意の構造の問い合わせに対して，その処理回数を最小にするバッファ資源の割り当てを決定する場合に適用できる．さらに，この方法は，総計算回数と I/O 処理回数の両者を加えた総計算量を最小にするバッファ資源の割り当てを決定する場合にも用いることができるので，適用範囲が広い．さらに，本論文では，この方法により最適なバッファ資源の割り当てを決定するための計算のオーダに関する解析を行い，この方法の有効性を明らかにする．

2. バッファ資源割り当て

ここでは，ストリーム指向型関係演算処理方式における最適なバッファ資源割り当てを決定する計算方法について述べる．ストリーム指向型関係演算処理方式については，文献7），9）に詳しく述べられている．

この計算方法では，問い合わせを構成する各関係演算ノード (Node-(1)～Node-(n): Node-(1) は問い合わせのルート・ノード) に限られたバッファ資源 (BS) を分割して最適に割り当てるために，問い合わせ処理に要する計算回数，I/O 処理回数，あるいは，両者を考慮した計算量を表す式を設定する．この場合，各関係演算ノードをルート・ノードとする部分問い合わせの計算量を，部分関数の展開式によって表現する．そして，バッファ割り当ての各候補値 t ($t=1, 2, \cdots, BS$) について，展開式を構成する各部分関数の値を最小とするバッファ資源の割り当てを Node-(n) から Node-(1) まで順々に計算していくことにより，最終的に問い合わせの計算量を最小とするバッファ資源割り当てを決定する．

2.1 問い合わせ処理における計算量

本節では，問い合わせ処理に要する計算回数，I/O 処理回数，あるいは，両者を考慮した計算量を最小にする最適なバッファ資源の割り当てを決定する問題を扱う準備として，まず，限られたバッファ資源内での

問い合わせ処理の計算回数，I/O 処理回数，および，両者を考慮した総計算量を表す式の設定を行う．

ここでは，**図1**に示すような結合演算群からなるチェイン型問い合わせ（結合演算群が1列に並んだ形の問い合わせ）を例として用いる．2.4 節において述べるように，以下で提示するバッファ割り当ての計算方法は，それを拡張することによって他の関係演算を含むチェイン型以外の任意の問い合わせに対しても同様に適用できる．

まず，ここで用いるパラメータを次のように定義する．

(以後，Join-(i) ノードは，図の中では Node-(i) に対応するものとする．)

jsf_i：Join-(i) ノードの結合演算選択率 (join selectivity factor).

((Join-(i) ノードが生成する中間リレーションのタプル数)＝jsf_i＊(Join-(i) のインナ・リレーションのタプル数)＊(Join-(i) のアウタ・リレーションのタプル数)．

BS_i：Join-(i) ノードのアウタ・リレーションの1ページを格納するためのバッファのサイズ

IBS_i：Join-(i) ノードのインナ・リレーションの1ページを格納するためのバッファのサイズ．

n：結合演算ノード数．

R_i：Join-(i) ノードのアウタ・リレーションのサイズ．

R：Join-(n) ノードのインナ・リレーションのサイズ．

IR_i：Join-(i) ノードのインナ・リレーションのサイズ．

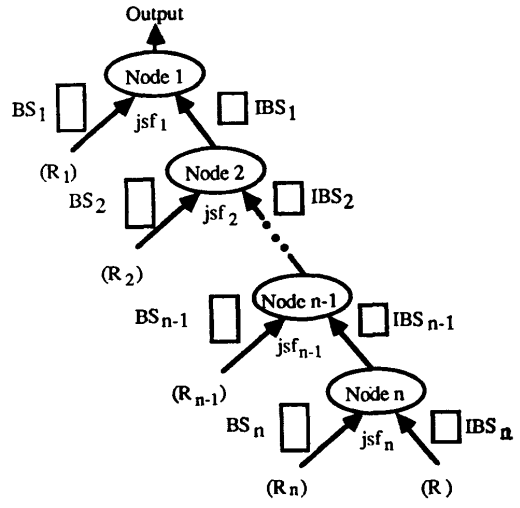

図1 チェイン型問い合わせ
Fig. 1 A chain query.

以下では，計算回数を最小化する問題，I/O 処理回数を最小化にする問題，および，両者を考慮した計算量を最小化する問題を同様に扱うために，リレーションのサイズの単位およびバッファ・サイズの単位を，次のように設定する．

① 問い合わせの実行に要するタプルの比較回数である総計算回数を表す式では，リレーションのサイズおよびバッファ・サイズの単位をタプル数とする．

② 問い合わせ処理における I/O 処理回数を表す式では，I/O 処理がブロック（1ブロック・サイズは固定長）を単位として行われるので，リレーションのサイズおよびバッファ・サイズの単位をブロックとする．

（1） 計算回数を最小にするための計算量

ここでは，問い合わせの実行に要するタプルの比較回数を計算回数とし，問い合わせの実行を完了するのに要する計算回数を表す計算式を示す．

計算回数を表す式においては，結合演算選択率があらかじめ予測されていることが前提となる．この予測の手法については文献 12)，17) 等で提案が行われている．また，ストリーム指向型関係演算処理における結合演算選択率の予測方法を，文献 10)において提案している．

ストリーム指向型関係演算処理方式では，アウタ・リレーション・バッファおよびインナ・リレーション・バッファ内のタプル間の比較アルゴリズムとして，nested-loop アルゴリズムあるいは sort-search アルゴリズムを対象とする．sort-search アルゴリズムでは，まず，アウタ・リレーション・バッファ内のタプル群を演算対象属性上でソートし，それらのタプル群に対して，インナ・リレーション・バッファ内の各タプルとバイナリ・サーチにより比較する．

ここでは，計算回数に関係のない各インナ・リレーション用バッファ（IBS_i）の割り当てはすでに終えているものとし，各ノードに割り当てるアウタ・リレーション用バッファの最適な割り当てを求める問題を考える．

各々の結合演算ノードの計算回数は，nested-loop および sort-search の場合に，それぞれ次のようになる．

nested-loop の場合：

Node 1： $R_1*(R_2*jsf_2)*\cdots*(R_n*jsf_n)*R$

Node 2： $\lceil R_1/BS_1\rceil*R_2*(R_3*jsf_3)*\cdots*$

$(R_n * jsf_n) * R$

⋮ ⋮

Node i: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{i-1}/BS_{i-1}\rceil * R_i$
$* (R_{i+1} * jsf_{i+1}) * \cdots * (R_n * jsf_n) * R$

⋮ ⋮

Node $n-1$: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{n-2}/BS_{n-2}\rceil * R_{n-1}$
$* (R_n * jsf_n) * R$

Node n: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{n-1}/BS_{n-1}\rceil * R_n * R$ (1)

sort-search の場合:

Node 1: $\lceil R_1/BS_1\rceil * (\log_2 BS_1 + jsf_1 * BS_1)$
$* (R_2 * jsf_2) * \cdots * (R_n * jsf_n) * R$

Node 2: $\lceil R_1/BS_1\rceil * \lceil R_2/BS_2\rceil * (\log_2 BS_2 + jsf_2$
$* BS_2) * (R_3 * jsf_3) * \cdots * (R_n * jsf_n)$
$* R$

⋮ ⋮

Node i: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{i-1}/BS_{i-1}\rceil$
$* \lceil R_i/BS_i\rceil * (\log_2 BS_i + jsf_i * BS_i)$
$* (R_{i+1} * jsf_{i+1}) * \cdots * (R_n * jsf_n) * R$

⋮ ⋮

Node $n-1$: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{n-2}/BS_{n-2}\rceil$
$* \lceil R_{n-1}/BS_{n-1}\rceil * (\log_2 BS_{n-1}$
$+ jsf_{n-1} * BS_{n-1}) * (R_n * jsf_n) * R$

Node n: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{n-1}/BS_{n-1}\rceil$
$* \lceil R_n/BS_n\rceil * (\log_2 BS_n + jsf_n * BS_n) * R$ (2)

したがって，総計算回数は，

nested-loop の場合

$$TC = \sum_{i=0}^{n-1}\left\lceil \prod_{k=1}^{i}\left(\left\lceil \frac{R_k}{BS_k}\right\rceil\right) * R_{i+1} * \prod_{k=i+2}^{n}(jsf_k * R_k) * R\right\rceil \quad (3)$$

sort-search の場合:

$$TC = \sum_{i=1}^{n}\left\lceil \prod_{k=1}^{i-1}\left(\left\lceil \frac{R_k}{BS_k}\right\rceil\right) * \left\lceil \frac{R_i}{BS_i}\right\rceil * (\log_2 BS_i + jsf_i * BS_i) * \prod_{k=i+1}^{n}(jsf_k * R_k) * R\right\rceil \quad (4)$$

となる．

このように，計算回数を最小とするバッファ資源の割り当てを求める問題は，(3)式あるいは(4)式の総計算回数 TC を最小とするバッファ資源割り当てを求める問題になる．

(2) I/O 処理回数を最小にするための計算量

ここでは，問い合わせの実行を完了するのに要するリレーションの I/O 処理回数を表す計算式を設定する．

図1に示すように，ベース・リレーション R_i ($i=1,2,\cdots,n$) のブロックを格納するバッファのサイズを各々 BS_i ($i=1,2,\cdots,n$)，ベース・リレーション R を格納するバッファのサイズを IBS_n とする．ここで，各ベース・リレーションの I/O 処理回数は，次のように表される．

R_1: R_1

R_2: $\lceil R_1/BS_1\rceil * R_2$

⋮ ⋮

R_i: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{i-1}/BS_{i-1}\rceil * R_i$

⋮ ⋮

R_{n-1}: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{n-2}/BS_{n-2}\rceil * R_{n-1}$

R_n: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_{n-1}/BS_{n-1}\rceil * R_n$

R: $\lceil R_1/BS_1\rceil * \cdots * \lceil R_n/BS_n\rceil * R$ (5)

総 I/O 処理回数 TC は

$$TC = \sum_{i=0}^{n}\left\lceil \prod_{k=1}^{i}\left(\left\lceil \frac{R_k}{BS_k}\right\rceil\right) * R_{i+1}\right\rceil$$
$$R_{n+1} = R \quad (6)$$

となる．

そこで，I/O 処理回数を最小とするバッファ資源割り当てを求める問題は，(6)式を最小にするバッファ割り当てを求める問題になる．

この I/O 処理回数を表す(6)式は，文献4)において最適化の対象とされた式と同じ式である．

(3) I/O 処理回数と計算回数を最小にするための計算量

計算回数と I/O 処理回数の間の関係により，(1)で述べた(3)式あるいは(4)式に，(6)式を単位の整合を行った後に加えることにより，I/O 処理回数と計算回数の両方を考慮する総計算量を表す式が得られる．例えば，nested-loop アルゴリズムの場合には，計算回数と I/O 処理回数の両方を計算回数の単位に整合した後に加えることにより表される総計算量は，

$$TC = \sum_{i=1}^{n-1}\left\lceil \prod_{k=1}^{i}\left(\left\lceil \frac{R_k}{BS_k}\right\rceil\right) * \left(R_{i+1} * \prod_{k=i+2}^{n}(jsf_k * R_k) * R + R_{i+1}'\right)\right\rceil + \prod_{k=1}^{n}\left(\left\lceil \frac{R_k}{BS_k}\right\rceil\right) * R_{n+1}' \quad (7)$$

となる．

ここでは，R_{i+1}' は，(6)式におけるブロックを単位としたリレーション・サイズ R_{i+1} を，タプルを単位とした計算回数に整合した場合のリレーション・サ

イズを表す．計算回数と I/O 処理回数の両者を考慮したバッファ資源割り当てを求める問題は，(7)式を最小にするバッファ割り当てを求める問題になる．

2.2 バッファ資源割り当てのための計算式

本節では，2.1 節で提示した計算量を表す(3)，(4)，(6)，(7)式を最小にするバッファ資源割り当ての計算を行う方法を示す．この計算方法は，2.1 節で示した(1)計算回数の式，(2)I/O 処理回数の式，(3)計算処理と I/O 処理を加えた計算量を表す式の各式に対して同じように適用することができる．

(1) nested-loop の場合には，(3)式に示したように，BS_n は総計算回数と関係がないので，TC の最小値を求める計算から決定されない．理論的には，BS_n の最小値は1（タップルあるいはブロック）である．すなわち，BS_n は計算回数の最小化とは関係なく，物理的制約により決定されることになる．同様に，I/O 処理回数を最小にする資源割り当て問題を考慮する場合，バッファ IBS_n は，リレーションの I/O の処理回数と関係なく，物理的制約により決定されることになる．

(2) sort-search アルゴリズムの場合には，総計算量 TC において，ページ内の計算回数を表す項目 $(\log_2 BS_i + jsf_i * BS_i)$（インナ・リレーションの1タプルに対して，バイナリ・サーチによるアウタ・リレーション用バッファ内のタプル群との比較回数）の中にバッファ・サイズの値が入っているので，バッファ資源の最適な割り当てを行う計算が複雑になる．この項目は，ページ内の計算回数を表す項目であり，ほかの関係演算ノードの計算回数に影響を与えない．ここでは，sort-search の場合のバッファ資源の最適な割り当てを行う計算における複雑さを排除するために，ページ内の計算回数を表す項目の中の BS_i の値を定数として扱う．ここでは，その項目の中の BS_i $(i=1, 2, \cdots, n)$ は，定数 BS/n（平均値）とする．

このように，2.1 節の(3)，(4)，(6)，(7)式を考察するうえで，それらの計算量式を最小にするバッファ資源割り当て問題は，バッファ・サイズを変数とする次のような(8)式を最小化する問題となる．つまり，2.1 節で示した3種類の資源割り当て問題を同一の問題として扱うことができる．

$$TC = \sum_{i=0}^{n} \left\lceil \prod_{k=1}^{i} \left(\left\lceil \frac{R_k}{BS_k} \right\rceil \right) * a_i \right\rceil \qquad (8)$$

ここで，a_i $(i=0, 1, \cdots, n)$ は，扱う問題により決定される定数である．また，nested-loop の総計算回数に関しては BS_n は物理的制約によってすでに決定されているものとし，BS_1 から BS_{n-1} までのバッファ割り当てについて考えればよい．

(8)式の計算量を最小とするバッファ割り当て計算を，次の制約条件 1〜4 のもとで行う必要がある．

制約条件 1：総バッファ資源量 BS $(=BS_1+BS_2+\cdots+BS_n)$ が限られている．

制約条件 2：各関係演算ノードに割り当てられるバッファ・サイズは，ある定数 c_i より大きい．つまり，$BS_i \geqq c_i$ $(i=1, 2, \cdots, n)$ である．c_i は，物理的制約（プロセッサ間での通信速度，計算機資源量，システムの稼働状況等）により決定される値である．（例えば，ストリーム指向型関係演算処理においてバッファのサイズは，関係演算ノード間の並列性に影響を与える．バッファ・サイズが小さい場合には，デマンドの発行回数が増えるためにプログラム間の通信によるオーバヘッドが増大し，処理効率の低下が引き起こされる．バッファ・サイズの処理効率に対する影響については，すでに文献7)，9）に示している．）

制約条件 3：$1 \leqq BS_i \leqq R_i$ $(i=1, 2, \cdots, n)$．各バッファには少なくとも1タプル（I/O 処理回数の最小化を対象とする場合，1ブロック）を格納する容量が必要である．また，リレーションの全タプルが入り切る場合には，それ以上のバッファ量を与える必要がない．

(8)式はガウス記号が付いている整数問題である．ここで，次に示す制御条件4を加えることによりガウス記号をはずし，次の(9)式を最小にするバッファ資源割り当てを求める計算を行う．

$$TC = \sum_{i=0}^{n} \left\lceil \prod_{k=1}^{i} \left(\frac{R_k}{BS_k} \right) * a_i \right\rceil \qquad (9)$$

制約条件 4：$BS_i \in \{\lceil R_i/m \rceil \mid m=1, 2, \cdots, R_i\}$ $(i=1, 2, \cdots, n)$

(8)式では，$\lceil R_i/BS_i \rceil$ $(i=1, 2, \cdots, n)$ が整数であるので，$\lceil R_i/BS_i \rceil$ を1減らすだけのバッファ・サイズの増加を行うことができなければ，計算回数あるいは I/O 処理回数は軽減されない．したがって，制約条件4を加えることにより，(8)式を，(9)式に変換することができる．

このように，最適資源割り当て問題は，制約条件 1〜4 のもとで(9)式を最小とする非線形整数プログラミングの問題 (non-linear integer programming problem) として扱うことができる．

2.3 バッファ資源割り当て計算のアルゴリズム

ここでは，バッファ資源割り当て計算のアルゴリズ

ムを示す.

まず,(9)式を次のように展開して表す.

$$\begin{aligned} TC=f_1=a_0 &+\frac{R_1}{BS_1}*f_2\\ f_2=a_1 &+\frac{R_2}{BS_2}*f_3\\ f_3=a_2 &+\frac{R_3}{BS_3}*f_4\\ &\vdots\\ f_{n-1}=a_{n-2} &+\frac{R_{n-1}}{BS_{n-1}}*f_n\\ f_n=a_{n-1} &+\frac{R_n}{BS_n}*a_n \end{aligned} \tag{10}$$

(10)式からわかるように,各部分関数 f_k ($k=n-1, \cdots, 1$) は,部分関数 f_{k+1} および変数 BS_k の関数であり,部分関数 f_n は変数 BS_n の関数として表現できる.つまり,$f_k(f_{k+1}, BS_k)$ ($k=n-1, n-2, \cdots, 1$), f_n (BS_n) である.ここで,制約条件1および3により,t タプル ($t=1, 2, \cdots, BS$) のバッファ資源がある場合の部分関数 f_k の最小値 $F_k(t)$ は,次のように表すことができる.

$$\begin{aligned} F_k(t)=\min\{&f_k(BS_k, BS_{k+1}, \cdots, BS_n) \mid\\ &\sum_{j=k}^{n} BS_j=t,\ BS_j: \text{integer},\ BS_j>0\}\\ &(k=n, n-1, \cdots, 1)\\ &(t=1, 2, \cdots, BS) \end{aligned} \tag{11}$$

最終的な目標は,総計算量の最小値 $F_1(BS)$ の場合の各バッファ資源量 BS_i ($i=1, 2, \cdots, n$) の値を求めることである.ここで,(11)式を計算するために,次のように漸化式(12)式を設定する.

$$\begin{aligned} &F_n(t)=f_n(t)\\ &F_k(t)=\min_{1\leq r<t} f_k(F_{k+1}(t-r), r)\\ &\quad(k=n-1, \cdots, 1)\\ &\quad(t=1, 2, \cdots, BS) \end{aligned} \tag{12}$$

この式は次のように計算される.まず,リーフ・ノード Node-(n) から Node-($k+1$) のバッファ BS_i ($i=n, \cdots, k+1$) に割り当てる総バッファ・サイズ $\left(\sum_{i=k+1}^{n} BS_i\right)$ の各候補値 ($1, 2, \cdots, BS$) に対して,部分関数 f_{k+1} の最小値 F_{k+1} を求めておく.次に,Node-(n) から Node-(k) のバッファ BS_i ($i=n, \cdots, k$) に割り当てる総バッファ・サイズ $\left(\sum_{i=k}^{n} BS_i\right)$ の各候補値 t ($t=1, 2, \cdots, BS$) に対して,F_{k+1} を用いて,f_k の最小値 $F_k(t)$ を求める.このように,$F_k(t)$ は,Node-(i) ($i=k+1, \cdots, n$) までに割り当てられるバッファ資源量の各候補値 t ($1\sim BS$) ごとに求められている最小値 $F_{k+1}(t)$ の結果を用いることにより求めることができる.この手続きを F_{n-1} から F_1 まで繰り返すことにより $F_1(BS)$ を求めることができる.

(12)式において,BS_n から BS_k までに割り当てる総バッファ資源量の候補値 t ($t=1, 2, \cdots, BS$) の中で,BS_k に r ($1\leqq r<t$) を割り当てるとすれば,BS_n から BS_{k+1} のバッファには残りのバッファ資源量 $(t-r)$ が割り当てられることになる.$\sum_{i=k+1}^{n} BS_i$ ($=t-r: 1\leqq(t-r)<t$) における部分関数 f_{k+1} の最小値は,$F_{k+1}(t-r)$ としてすでに求められているので,部分関数 f_k は,$F_{k+1}(t-r)$ と r をパラメータとする関数 $f_k(F_{k+1}(t-r), r)$ として表すことができる.つまり,$BS_k=r$ の場合,関数 $f_k(F_{k+1}(t-r), r)$ を計算することができる.その最小値 $F_k(t)$ の値は,r のとりうる値 ($r=1, 2, \cdots, t$) 各々に対して,$f_k(F_{k+1}(t-r), r)$ を計算し,その中の最小値をみつけることにより得られる.

(10)式より明らかなように,f_k の値は f_{k+1} が最小値のとき最小になる.$BS_k=r$ の場合,$t-r$ のバッファ資源割り当てによる部分関数 f_{k+1} が最小値 $F_{k+1}(t-r)$ をとるので,部分関数 f_k の最小値 $F_k(t)$ は,$f_k(F_{k+1}(t-r), r)$ となる.すなわち,(10)式における f_k のパラメータ f_{k+1} は,その最小値 $F_{k+1}(t-r)$ に固定すればよく,ほかの可能性を考慮する必要はない.t の値は,次の部分関数 f_{k-1} の最小値 F_{k-1} を求めるための準備として,t がとりうるすべての値 ($t=1, 2, \cdots, BS$) に対して計算を行わなければならない.以上のようにして,漸化式(12)式により,BS_n から BS_k までに割り当てる総バッファ資源量の各候補値 t ($=1, 2, \cdots, BS$) について,各々,最小値 $F_k(t)$ を求めることができる.

したがって,漸化式(12)式を $F_1(t)$ まで計算することにより,総計算量 TC を最小とするバッファ資源割り当てが必ず得られることになる.

次に,制約条件 2, 3, 4 を利用してこの方法の計算効率をさらに改良する.

漸化式(12)式を計算するとき,t に対して1から BS までのすべての可能値を候補値とし,また,$F_k(t)$ を計算するために r に対して1から t までのすべての可能値を対象としている.しかし,r と t の値に対して次のように制約を加えることができる.

① r はバッファ BS_k に割り当てる資源量であるので,制約条件4を満足しなければならない.つま

り，$\lceil R_k/BS_k \rceil$ の値を 1 減らすためのバッファ資源量の値を r の候補値とするだけでよい．

② $t-r$ は，Node-(n) から Node-($k+1$) のバッファ BS_n から BS_{k+1} に割り当てるバッファ資源量であり，それらの各バッファ BS_i $(i=n,\cdots,k+1)$ に割り当てるバッファ資源量の値は制約条件 4 を満足しなければならないので，必ずしも 1 から BS までのすべての候補値について F_{k+1} を計算しておく必要はない．つまり，Node-(n) から Node-($k+1$) の中の少なくとも 1 つのノードの $\lceil R_i/BS_i \rceil$ の値を 1 減らすことができるバッファ資源量の値を候補値とするだけでよい．

③ 漸化式(12)式では，$1 \sim BS$ のすべての整数値を候補値として計算を行っている．しかし，候補値は，制約条件 2 および制約条件 3 の範囲に限ることができる．

④ $F_1(t)$ の候補値 t は，問い合わせ処理における総計算量を最小とするために割り当てるバッファ資源量であるので，その候補値 t は BS に設定するだけでよい．

(12)式の計算において以上の候補値に関する制約を加えることにより，次のバッファ資源割り当てアルゴリズムが得られる．

バッファ資源割り当て計算アルゴリズム：

（1） 次の式により $F_n(t)$ を計算する．

$$F_n(t)=f_n(t)$$
$$\left(\left(t=\left\lceil\frac{R_n}{p_n}\right\rceil,\ \left\lceil\frac{R_n}{p_n-1}\right\rceil,\ \left\lceil\frac{R_n}{p_n-2}\right\rceil,\cdots,\left\lceil\frac{R_n}{q_n}\right\rceil\right)\middle|\ \left\lceil\frac{R_n}{p_n-i}\right\rceil\neq\left\lceil\frac{R_n}{p_n-(i-1)}\right\rceil\ (i=1,2,\cdots,p_n-q_n)\right)$$
$$\left(p_n=\left\lfloor\frac{R_n}{c_n}\right\rfloor\right)$$
$$\left(q_n=\left\lceil\frac{R_n}{\min\{R_n,\ BS\}}\right\rceil\right)$$

ここで，$F_n(t)$ の候補値 t を $(t_1,t_2,\cdots)$ とする．$k=n-1$.

（2） バッファ BS_k に割り当てるバッファ量を r とし，次の式を計算する．

$$F_k(r+s)=f_k(F_{k+1}(s),r)$$
$$\left(\left(r=\left\lceil\frac{R_k}{p_k}\right\rceil,\ \left\lceil\frac{R_k}{p_k-1}\right\rceil,\ \left\lceil\frac{R_k}{p_k-2}\right\rceil,\cdots,\left\lceil\frac{R_k}{q_k}\right\rceil\right)\middle|\ \left\lceil\frac{R_k}{p_k-i}\right\rceil\neq\left\lceil\frac{R_k}{p_k-(i-1)}\right\rceil\ (i=1,2,\cdots,p_k-q_k)\right)$$
$$\left(p_k=\left\lfloor\frac{R_k}{c_k}\right\rfloor\right)$$
$$\left(q_k=\left\lceil\frac{R_k}{\min\{R_k,\ BS\}}\right\rceil\right)$$
$$(s=t_1,t_2,\cdots)$$
$$(r+s<BS)$$

ここでは，$r'+t_u=r''+t_v=p$ $(r'\neq r'',\ r',\ r''$ は r の中の一つの要素) のとき，$F_k(r'+t_u)$ と $F_k(r''+t_v)$ の中から

$$F_k(p)=\min\{F_k(r'+t_u),\ F_k(r''+t_v)\}$$

を $F_k(t)$ の候補とする．

これにより，候補値 t $(t=t_1,t_2,\cdots)$ とする $F_k(t)$ が得られる．

（3） $k=2$ ならば，（4）へいく．さもなければ，$k=k-1$ とし，（2）へいく．

（4） バッファ BS_1 に割り当てるバッファ資源量 r に対して次の式によって $F_1(BS)$ を計算する．

$$F_1(BS)=\min_r f_1(F_2(BS-r),\ r)$$
$$\left(\left(r=\left\lceil\frac{R_1}{p_1}\right\rceil,\ \left\lceil\frac{R_1}{p_1-1}\right\rceil,\ \left\lceil\frac{R_1}{p_1-2}\right\rceil,\cdots,\left\lceil\frac{R_1}{q_1}\right\rceil\right)\middle|\ \left\lceil\frac{R_1}{p_1-i}\right\rceil\neq\left\lceil\frac{R_1}{p_1-(i-1)}\right\rceil\ (i=1,2,\cdots,p_1-q_1)\right)$$
$$\left(p_1=\left\lfloor\frac{R_1}{c_1}\right\rfloor\right)$$
$$\left(q_1=\left\lceil\frac{R_1}{\min\{R_1,\ BS\}}\right\rceil\right)$$

ここで，$f_1(F_2(BS-r),r)$ のパラメータ $F_2(BS-r)$ が必ずしもすでに計算されていないという問題が生じる．この問題については，次のように解決する．

$F_2(t_1),F_2(t_2),\cdots,F_2(t_m)(t_i<t_{i+1},i=1,2,\cdots,m-1)$ がすでに計算されているとし，$f_1(F_2(BS-r),\ r)$ を計算するとき，

a） $BS-r<t_1$ ならば，その f_1 を計算しない．

b） $BS-r\geqq t_m$ ならば，$f_1(F_2(BS-r),\ r)$ を $f_1(F_2(t_m),\ r)$ とする．

c） $t_i\leqq BS-r<t_{i+1}$ $(i=1,2,\cdots,m-1)$ ならば，$F_2(t)$ は t に対して単調関数であるので，$f_1(F_2(BS-r),\ r)$ を $f_1(F_2(t_i),\ r)$ とする．

以上に述べたアルゴリズムにより，問い合わせ処理における総計算量を最小とするバッファ資源割り当てを必ず決定することができる．

2.4 任意の構造の問い合わせ処理のためのバッファ資源割り当ての計算アルゴリズム

本節では，より一般的な問い合わせ処理におけるバッファ資源割り当てのために，2.3 節のアルゴリズムを拡張する．ここでは，計算回数を最小にする場合

について述べるが，I/O 処理回数あるいは計算回数と I/O 回数の両者を考慮した計算量を最小にする場合についても同様に適用することができる．

ここでは，関係データベースの問い合わせの一般型である木構造の問い合わせを考える．チェイン型問い合わせと違って，木構造の問い合わせでは，各関係演算のアウタ・リレーションは，必ずしもベース・リレーションではなく，部分問い合わせ (subquery) によって生成される中間リレーションの場合がある．したがって，アウタ・リレーションを生成する部分問い合わせの計算量を総計算量に含めなければならない．そこで，アウタ・リレーションを生成する部分問い合わせの計算量も含めて，2.1 節で提示した総計算回数と同様に，木構造の問い合わせ処理における総計算回数を求める式を設定する．このとき，2.2 節で述べた制約条件 1〜4 はそのまま適用できる．したがって，チェイン型問い合わせ処理におけるバッファ資源割り当て計算と同様に，木構造問い合わせの処理におけるバッファ資源割り当てにおいては，制約条件 1〜4 のもとで総計算量を最小とするバッファ割り当てを求めることになる．

木構造問い合わせの処理では，**図 2** に示すように，各関係演算ノード（結合演算ノード）Node-(i) において，アウタ・リレーション R_i は部分問い合わせ outer-subquery-(i) によって生成され，インナ・リレーション IR_i は部分問い合わせ inner-subquery-(i) によって生成される．ここで，部分問い合わせ outer-subquery-(i) の計算回数を $f_{\text{outer-}i}$，部分問い合わせ inner-subquery-(i) の計算回数を $f_{\text{inner-}i}$ とすると，Node-(i) をルートとする部分問い合わせの総計算回数 f_i は次のようになる．

nested-loop アルゴリズムを採用する場合：

$$f_i = R_i * IR_i + f_{\text{outer-}i} + \left\lceil \frac{R_i}{BS_i} \right\rceil * f_{\text{inner-}i} \quad (13)$$

sort-search アルゴリズムを採用する場合：

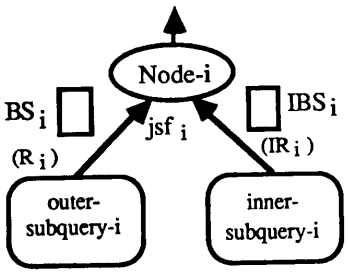

図 2 木構造問い合わせ
Fig. 2 A tree type query.

$$f_i = \left\lceil \frac{R_i}{BS_i} \right\rceil * \left[\log_2 \left(\frac{BS}{n} \right) + jsf_i * \left(\frac{BS}{n} \right) \right] * IR_i + f_{\text{outer-}i} + \left\lceil \frac{R_i}{BS_i} \right\rceil * f_{\text{inner-}i} \quad (14)$$

部分問い合わせ outer-subquery-(i) の中のルート・ノードが Node-(p) であるとき，$f_{\text{outer-}i} = f_p$ となり，部分問い合わせ inner-subquery-(i) の中のルート・ノードが Node-(q) であるとき，$f_{\text{inner-}i} = f_q$ となる．もしそのルート・ノードがベース・リレーションであれば，$f_{\text{outer-}i}$ あるいは $f_{\text{inner-}i}$ は 0 となる．

このように，問い合わせ中の各関係演算ノード Node-(i) ($i = 1, 2, \cdots, n$) に対して，そのノードをルートとする部分問い合わせの計算量 f_i を求める (13)式あるいは (14)式を各ノードについて設定していく．Node-(1) が問い合わせ木のルート関係演算ノードであるので，f_1 は，問い合わせ処理での総計算量を表す式となる．

このとき，2.3 節のアルゴリズムと同様に，バッファ資源量の候補値 t ($t = 1, 2, \cdots, BS$) タプルのバッファ資源を与えた場合の部分問い合わせの計算量を表す関数 f_k ($k = n, n-1, \cdots, 1$) の最小値を $F_k(t)$ とすると，$F_1(BS)$ が総計算量の最小値となる．

部分関数 f_k ($k = n, n-1, \cdots, 1$) の最小値を表す関数 F_k を計算するために，f_k を 2 つ関数の和として考える．

$$f_k = f_k' + f_k'' \quad (15)$$

f_k', f_k'' はそれぞれ次のように表すことができる．

Nested-loop アルゴリズムを採用する場合：

$$f_k' = f_{\text{outer-}k}$$

$$f_k'' = R_k * IR_k + \left\lceil \frac{R_k}{BS_k} \right\rceil * f_{\text{inner-}k}$$

sort-search アルゴリズムを採用する場合：

$$f_k' = f_{\text{outer-}k}$$

$$f_k'' = \left\lceil \frac{R_k}{BS_k} \right\rceil * \left[\log_2 \left(\frac{BS}{n} \right) + jsf_k * \left(\frac{BS}{n} \right) \right] * IR_k + \left\lceil \frac{R_k}{BS_k} \right\rceil * f_{\text{inner-}k}$$

f_k' と f_k'' は，バッファのサイズに関して，互いに異なる変数を対象とする関数である．各候補値 t ($t = 1, 2, \cdots, BS$) について，$f_{\text{outer-}k}$ および $f_{\text{inner-}k}$ の各最小値が各々 $F_{\text{outer-}k}(t)$ および $F_{\text{inner-}k}(t)$ である場合，Node-(k) をルート・ノードとする部分問い合わせに割り当てられる総バッファ資源量の各候補値 t ($t = 1, 2, \cdots, BS$) について，部分問い合わせの計算量を求め

る関数 f_k の最小値 $F_k(t)$ は，次の2ステップにより求めることができる．

ステップ1：制約条件 1〜4 のもとで f_k'' の最小値 $F_k''(t)$ を Node-(k) をルート・ノードとする部分問い合わせに割り当てられる総バッファ資源量の各候補値 t ($t=1,2,\cdots,BS$) について求める．この最小値 $F_k''(t)$ は，2.3 節で述べた方法と同様の方法により計算する．

ステップ2：f_k の最小値 $F_k(t)$ を，次の計算式で求める．

$$F_k(t)=\min_{1\le r<t}\{F_{\text{outer-}k}(t-r)+F_k''(r)\} \quad (16)$$

アウタ・リレーションがベース・リレーションである場合には，その演算ノードをルートとする部分問い合わせ処理の計算回数の最小値 $F_k(t)$ は，ステップ1だけにより計算できる．

ステップ2では，限られたバッファ資源の中でバッファ・サイズの各候補値 t ($t=1,2,\cdots,BS$) に対してそれぞれ各部分関数の計算量 f_k' と f_k'' を最小とするバッファ資源割り当て計算を行い，そして，その2つの計算量関数の和で表現される計算量関数を最小とするバッファ資源割り当てを Node-(k) をルート・ノードとする部分問い合わせに対して割り当てられる総バッファ量の各候補値 t ($t=1,2,\cdots,BS$) について求めることにより，2.3 節と同じ方法で最適なバッファ資源の割り当てを決定できる．このとき，2つの計算量関数 f_k' および f_k'' は，バッファ・サイズの増加に対して単調減少関数であるので，一方の部分関数のバッファ・サイズ変数に r タプルを与えると，残りのバッファの全部を他方の部分関数のバッファ・サイズ変数に与えれば，計算量を最小にすることになる．したがって，ステップ2において $f_{\text{outer-}k}$ には $t-r$ を割り当てればよいことになる．

3. バッファ資源割り当ての計算例

ここでは，**図3**に示すような4つの結合演算からなる問い合わせ処理における各関係演算ノードのアウタ・リレーション用バッファに対し，2.3 節で提示したバッファ資源割り当て計算方式によりバッファ資源を割り当てる場合を示す．問い合わせ処理アルゴリズムを nested-loop アルゴリズムを採用する．2.2 節で述べたように，インナ・リレーション用バッファ IBS_i ($i=1,2,3,4$) およびバッファ BS_4 が計算回数と関係ないので，ここでは，それらのバッファ・サイズはすでに決定されているものとし，総バッファ資源量

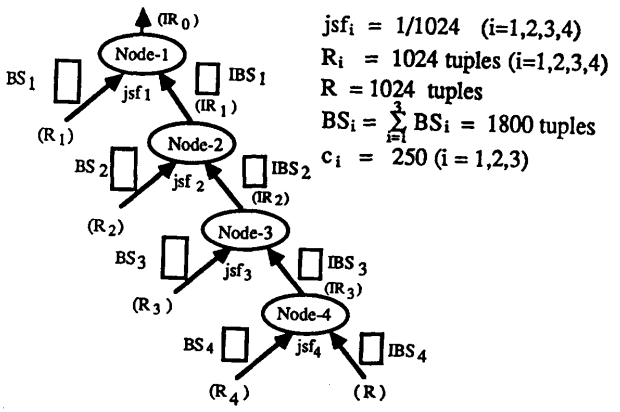

図3 問い合わせ例
Fig. 3 An example query.

BS (=1800 タプル) において，バッファ BS_k ($k=1,2,3$) の割り当てを考える．

図3に示す問い合わせの総計算回数を求める式 TC は，2.1 節の(3)式に，各演算対象のベース・リレーションのタプル数および各結合演算選択率を代入することにより得られる．

$$TC=\sum_{i=0}^{3}\left[\prod_{k=1}^{i}\left(\left\lceil\frac{1024}{BS_k}\right\rceil\right)*1024*1024\right] \quad (17)$$

総計算回数 TC を最小とするバッファ資源割り当て計算を，2.2 節で述べた制約条件 1〜4 のもとで求めることになる．

(17)式を展開する．

表1 バッファ資源割り当て計算の過程
Table 1 Processes of computations for the optimal buffer resource allocation.

t	$F_3(t)$ (×1024²)	t	r	$F_2(t)$ (×1024²)	t	r	$F_1(t)$ (×1024²)
256	5*	512	256	21	1800	256	17
342	4	598	256	17		342	16
512	3		342	16		512	13
1024	2	684	342	13		1024	12*
		768	256	13			
			512	11*			
		854	342	10			
			512	9			
		1024	512	7			
		1280	251	9			
			1024	6			
		1366	342	7			
			1024	5			
		1536	512	5			
			1024	4			

t: candidate buffer size allocated to BS_k to BS_n
r: candidate buffer size allocated to BS_k
$F_k(t)$: minimal number of total computations of Node-(k) to Node-(n)

$$TC=f_1=1024*1024+\left\lceil\frac{1024}{BS_1}\right\rceil*f_2$$
$$f_2=1024*1024+\left\lceil\frac{1024}{BS_2}\right\rceil*f_3$$
$$f_3=1024*1024+\left\lceil\frac{1024}{BS_3}\right\rceil*1024*1024 \tag{18}$$

次に，2.3 節で提示したバッファ資源割り当て計算アルゴリズムを用いて，バッファ資源割り当て計算を行う．**表1**は，各部分関数 f_k $(k=1,2,3)$ の最小値 $F_k(t)$ の計算過程を示している．アンダーラインがついているものは，バッファ BS_k の候補値を r とするとき部分関数 f_k の最小値 $F_k(t)$ を表している．総計算量 TC $(=f_1)$ を最小とするバッファ資源割り当ては表1の中の＊で示されている．したがって，総計算量 TC を最小とするバッファ割り当ては，$BS_1=1024$, $BS_2=512$, $BS_3=264$ となる．

4. アルゴリズムの解析

2.3 節では，2.2 節で述べたバッファ資源割り当て計算問題を解決するアルゴリズムを提示した．本章では，そのアルゴリズムによる計算のオーダに関する解析を行う．

以下では，枚挙法，文献4）の方法，および，我々の提案方法において解析する．

［枚挙法］ 枚挙法では，すべての候補値に対して総計算回数 TC を計算し，その計算結果の中から総計算回数を最小とする割り当てを最適な割り当てとするので，関数 TC の計算回数は

$$\prod_{i=1}^{n}(\min\{BS, R_i\})\geq K^n$$
$$K=\min\{BS, \min_{1\leq i\leq n}\{R_i\}\} \tag{19}$$

となる．

［文献4）の方法］ 文献4）の方法では，総I/O処理回数の最後の項目を最小とするバッファ割り当てを求めるため，均等割り当て値を初期値として増分探索木を展開し，その木の最初の r レベルで総回数を最小とするバッファ割り当て値を最適割り当て値とする．そこで，最初の r レベルの展開回数（最後の項目の計算回数）は，

$$\sum_{i=0}^{r}(2n-2)^i=\frac{(2n-2)^{r+1}-1}{2n-3}\geq(2n-2)^r \tag{20}$$

となる．

［提案方法］ ここでは，提案する方法における計算オーダを解析する．

① 漸化式(12)式を解くための f_k $(k=n, n-1, \cdots, 1)$ の計算回数は，次のとおりである．

$$n*\sum_{p=1}^{BS}p=n*\frac{BS*(BS+1)}{2} \tag{21}$$

となる．したがって，漸化式(12)式を計算するためのオーダは，$O(n*BS^2)$ となる．

② 2.2 節で述べたように，各 BS_i $(i=n, n-1, \cdots, 1)$ の候補値として，制約条件4を満足する BS_i $(i=n, n-1, \cdots, 1)$だけを対象とすればよい．BS_i $(i=n, n-1, \cdots, 1)$ の候補値の個数（BS_i のドメイン集合において異なる要素の個数）は，$2*\lceil\sqrt{R_i}\rceil$ $(i=n, n-1, \cdots, 1)$ であるので，2.3 節のバッファ資源割り当て計算アルゴリズムの計算オーダは，

$$\overline{O}(n*BS*\sqrt{BS}) \tag{22}$$

となる．

③ さらに，制約条件2および条件3を考慮して，BS_i $(i=n, n-1, \cdots, 1)$ の候補値の数 d_i は，

$$d_i=\begin{cases}p_i-q_i+1 & c_i\geq\sqrt{R_i}\\ 2\sqrt{R_i}-c_i-q_i+2 & c_i<\sqrt{R_i}\end{cases}$$
$$p_i=\left\lfloor\frac{R_i}{c_i}\right\rfloor$$
$$q_i=\left\lceil\frac{R_i}{\min\{R_i, BS\}}\right\rceil$$
$$i=n, n-1, \cdots, 1 \tag{23}$$

となる．

制約条件2および4は，資源割り当て計算の効率の向上のために非常に重要である．**図4**には，関係演算ノード Node-(i) のアウタ・リレーション用バッファ・サイズとそのインナ・リレーションを生成する生産者ノードの計算回数の関係を示している．2.3 節で述べたように，Node-(i) をルート・ノードとする部分問い合わせのバッファ資源割り当てを考える場合，インナ・リレーションを生成する生産者ノードの計算回数を1回減らすことができるだけのバッファ資源量をバッファ BS_i の候補値とすればよい．図4からわかるように，BS_i $(i=n, n-1, \cdots, 1)$ に対する候補値は，バッファ・サイズが大きい範囲では少ない．たとえば，3章で述べた例において，各関係演算ノードのアウタ・リレーション用バッファ・サイズについて $BS_i\geqq250$（タプル）という制約条件が加えられると，(23)式により，BS_i $(i=1, 2, 3)$ の候補値は 1024 個ではなく，4個だけになる．これは，図4より明らかである．

そこで，2.3 節のバッファ資源割り当て計算アル

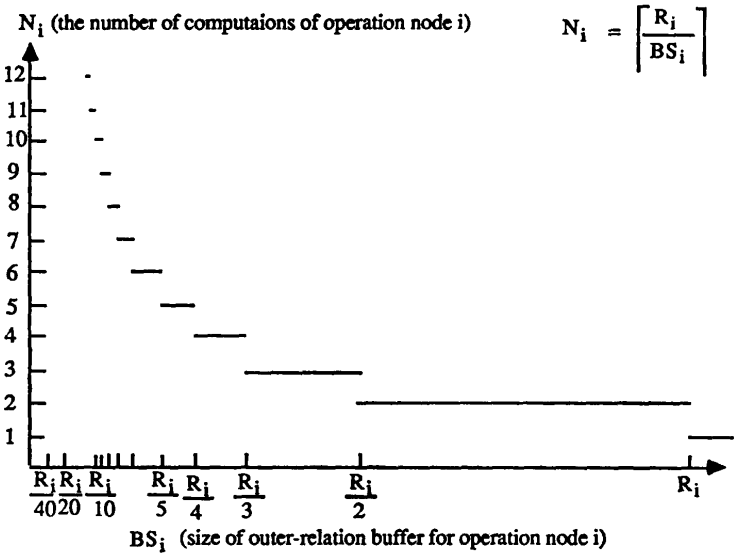

図 4 バッファ・サイズと計算回数の関係
Fig. 4 Relationship of buffer size and the number of computations.

ゴリズムを計算するために，部分関数 $f_k(k=n, n-1, \cdots, 1)$ の計算回数は，次のようになる．

$$d_n+K_nd_{n-1}+K_{n-1}d_{n-2}+\cdots+K_3d_2+d_1$$
$$K_i<BS \quad (i=n, n-1, \cdots, 3) \qquad (24)$$

ここで，K_i $(i=n, n-1, \cdots, 3)$ は，部分関数 f_i の最小値 $F_i(t)$ の候補値 t の個数である．前述したように，ある程度のバッファ資源量がある場合に，K_i $(i=n, n-1, \cdots, 3)$ は BS よりはるかに小さくなる．また，(23)式からわかるように，各関係演算ノードのアウタ・リレーション用バッファ BS_i $(i=n, n-1, \cdots, 1)$ に $\sqrt{R_i}$ タプル以上のバッファ資源量を与えることができれば，d_i $(i=n, n-1, \cdots, 1)$ は，$\sqrt{R_i}$ より小さくなるので，2.3 節で提示したバッファ資源割り当て計算アルゴリズムの計算オーダを(22)式のオーダより大幅に軽減することができる．

次に，一般的な問い合わせ処理におけるバッファ資源割り当てに対する 2.4 節の拡張アルゴリズムの解析を行う．

2.4 節で述べた拡張アルゴリズムのステップ 1 では，f_k'' の計算回数が $BS*(BS+1)/2$ であり，ステップ 2 では，たし算の回数が同じく $BS*(BS+1)/2$ である．したがって，計算アルゴリズムの中での部分関数 f_k $(k=1, 2, \cdots, n)$ の総計算回数は，(21)式で表される．

制約条件を加える場合の 2.4 節のアルゴリズムの解析は，前述したチェイン型問い合わせ処理におけるバッファ資源割り当て計算アルゴリズムの解析と同様に行うことができる．このように，一般的な問い合わせ処理におけるバッファ資源割り当てアルゴリズムの計算オーダは，チェイン型問い合わせの場合と同じである．

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により問い合わせ処理を行う場合に，最適なバッファ資源の割り当てを決定する新しい計算方法を提案した．この計算方法により，任意の構造の問い合わせに対して，最適なバッファ資源割り当てを必ず決定できることを示した．また，本論文では，計算回数を最小にするバッファ資源割り当てだけでなく，I/O 処理回数を最小にするバッファ資源割り当て，および，計算回数と I/O 処理回数の両者を考慮した計算量を最小にするバッファ資源割り当てにこの方法を適用できることを示し，この方法の適用範囲が広いことを明らかにした．

また，この方法によって最適なバッファ資源割り当てを決定するための計算のオーダに関する解析を行い，この方法の有効性を明らかにした．

今後は，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として現在開発を行っているストリーム指向型並列処理システム SMASH[7),8)] 上の関係演算処理系の中に，本論文で提案したバッファ資源割り当ての計算方法を実現する予定である．また，問い合わせを並列に処理する場合の関係演算ノード群のプロセッサ群への割り当て方法の設計が今後の課題である．

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. Database Syst.*, Vol.

6, No. 2, pp. 227-256 (1981).
4) Kim, W.: A New Way to Compute Product and Join of Relations, *Proc. of the ACM-SIGMOD Conf.*, pp. 179-187 (1980).
5) Kitsuregawa, M., Nakano, M., Harada, L. and Takagi, M.: Functional Disk System for Relational Database, *Proc. of the 3rd Int. Conf. on Data Engineering*, pp. 88-95 (1987).
6) 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).
7) 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).
8) Kiyoki, Y., Kato, K., Yamaguchi, N. and Masuda, T.: A Stream-oriented Approach to Parallel Processing for Deductive Databases, *Proc. 5th Int. Workshop on Database Machines*, pp. 102-115 (1987).
9) 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).
10) 劉, 清木, 益田：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).
11) 物井, 森田, 伊藤, 岩田, 酒井, 柴山：マルチポートページメモリを用いた知識ベースマシンの並列制御方式と処理性能, 情報処理学会論文誌, Vol. 29, No. 5, pp. 513-521 (1988).
12) 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).
13) Stonebraker, M.: Operating System Support for Database Management, *CACM*, Vol. 24, No. 7, pp. 412-418 (1981).
14) Tanaka, Y.: Adaptive Segmentation Schemes for Large Relational Database Machines, in *Advanced Database Machine Architecture*, pp. 293-318, Springer-Verlag (1983).
15) 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 (1982).
16) Vegdahl, S. R.: A Survey of Proposed Architectures for the Execution of Functional Languages, *IEEE Trans. Comput.*, Vol. C-33, No. 12, pp. 1050-1071 (1984).
17) Yu, C. T. and Chang, C. C.: Distributed Query Processing, *ACM Comput. Surv.*, Vol. 16, No. 4, pp. 399-433 (1984).

(昭和63年1月29日受付)
(昭和63年6月24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.