

ストリーム指向型関係データベースマシン・ アーキテクチャとその推論操作への応用

A Stream-oriented Relational Database Machine Architecture and
Its Application to Inference Operations

清木 康* 加藤和彦** 益田隆司*
Yasushi KIYOKI, Kazuhiko KATO and Takashi MASUDA

* 筑波大学 電子・情報工学系 ** 筑波大学 工学研究科

* Institute of Information Sciences and Electronics, University of Tsukuba
** Doctoral Program of Engineering, University of Tsukuba

1. はじめに

近年、関係データベース・システムの普及とともに、その応用分野が急速に広がってきている。我々は、関係データベース・システムの多様なアプリケーションおよび多種の問い合わせに柔軟に対応するための計算機構の検討を行っている[6],[10],[12]。

関係データベース・システムの基本演算である関係演算（関係データベース演算）の処理をアーキテクチャの側から支援するために、現在までに多くのデータベースマシンが提案されてきた[9]等。それらの中の多くは、マルチプロセッサ構成のアーキテクチャを指向し、個々の関係演算に内在する並列性の抽出による高速な関係演算処理の実現を目指している。

我々は、個々の関係演算に内在する並列性の抽出よりむしろ、問い合わせを構成する関係演算間に存在する並列性の抽出に主眼をおき、限られた計算機資源（メモリ資源およびプロセッサ資源）の中で問い合わせの並列処理を実現する方式を提案している[8],[12]。本稿では、その方式を実現するストリーム指向型関係データベースマシンのアーキテクチャを提示する。このアーキテクチャは、問い合わせを構成する関係演算群に対し、限られたメモリ資源および限られたプロセッサ資源を柔軟に適応させる機構を備えている。このアーキテクチャでは、要求駆動型制御による関数型計算の枠組みの中で関係演算の並列処理が行われる。

また、多様なアプリケーションに対処するために、このアーキテクチャは、論理型計算におけるファクト節に対する推論操作機構を備えている。関係データベースと論理型計算との統合は知識ベースシステムの実現に向けての有望な方式として注目されており、その基本操作をデータベースマシンがアーキテクチャレベルでサポートすることは重要であると考えられる。

2. 関係演算および推論操作の処理方式

2.1 関係演算の処理方式

関数型計算は、並列性の柔軟な抽出が容易であるこ

とが知られており[1]、その計算機構をデータベース処理に適用することは有効である。関数型計算の駆動方式として逐次型評価、データ駆動型評価、要求駆動型評価がある[14]。ここで提示する処理方式では、要求駆動型評価により次の2種類の並列性が引き出される。

(1) 関数引数の並列評価

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。図1の問い合わせにおいては、2つの選択演算は並列に実行可能である。

(2) 関数の適用側とその本体側、即ち、実引数データの生産者とその消費者間での並列評価（ストリーム型並列処理）

問い合わせ処理においては、リレーション（ソースリレーションあるいは中間リレーション）を生成する関数とそれを消費する関数が並列に実行されることに対応する。図1においては、結合演算と選択演算が並列に実行可能である。ここでは、中間結果のタプルを生成する関数（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）を消費者ノードとして参照する。

ストリーム型並列処理は次のようなアルゴリズムにより実現される。

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）は、1入力バッファおよび1出力バッファを持つ。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）は2入力バッファ及び1出力バッファを持つ。各バッファは生産者ノードと消費者ノードの2つの関数によって共有される。

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する。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ページを生成を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。各関係演算ノードは、1デマンドに対してリレーション全体を生成せず、リレーションの1ページ分のタプルを生成して停止する。したがって、各バッファは中間リレーション全体を格納する空間を必要としない。ここでは、各

バッファサイズをページサイズと呼ぶことにする。

ストリーム型の並列処理を行うために、生産者ノードと消費者ノードの間のバッファにはダブルバッファリング機構が実現される。図1の場合、選択演算ノードと結合演算ノードの間のバッファには、ダブルバッファリング機構が実現される。この場合、各バッファには2つの領域(Area-i, Area-j)が用意される。各領域はそれぞれ1ページ分のタプル群を格納する容量をもつ。消費者ノードは、一方の領域(ここではArea-iとする)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(Area-j)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。消費者ノードは入力バッファのArea-iに格納されているタプル群に対する処理を完了すると、Area-iへ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。そして、Area-jが先出したデマンドによってすでに埋められているならば、Area-j内のタプル群に対して関係演算の実行を開始する。このとき、生産者ノードはArea-iへ次ページを格納するために関係演算の実行を開始する。

結果として、生産者ノードと消費者ノード間でストリーム型並列処理が実現される。

2.2 推論操作

論理型計算と関係データベースの統合は、知識ベースシステムの実現に向けての有望なアプローチとして注目されている[2]。我々のアプローチでは、ホーン節の集合を関係データベースと統合化し、単一化(ユニフィケーション)に基づく推論操作を関数型計算の枠組みの中で実行する。論理型計算と関係データベースの統合には、次の3種類の方法が考えられる。

(1) ゴール節を関係演算列へ展開される高水準な問い合わせとして用いる方法である。この方法では、ゴール節の形式で与えられた問い合わせが関係演算列(

関係データベース演算列)へ変換され、それらの関係演算列が関係演算処理系によって実行される。

(2) 関係データベース内の各リレーションをファクト節の集合と見なす方法である。すなわち、関係データベースはファクト節の集合として、単一化に基づく推論操作の対象となる。関係データベースはファクト節の集合であるリレーションを集めたものと見なされる。このようなデータベースは、外延データベースと呼ばれる[2]。リレーションは、ファクト節を表現するために用いられ、また、ルール節は、内包データベース[2]として別の構造により表現される。

(3) (2)と同様に、関係データベースはホーン節の集合として見なすが、ファクト節だけでなくルール節もリレーションのタプルとして表現する方法である[15]。推論操作はルール節およびファクト節を含むリレーションに対して実行される。この方法では、ルール節の集合も関係データベースの枠組みの中で管理されるので、ルール節の量が膨大になった場合に有効である[15]。

一般に、データベースのアプリケーションでは、ルール節の量に比べ、ファクト節のデータ量が膨大であると考えられるので、ファクト節の処理が、並列計算による実質的な効果を引き出す。そこで、我々のアプローチでは、(1)および(2)の方法をアーキテクチャレベルで支援する対象とする。

(1)の方法では、ゴール節として記述された問い合わせが関係演算列に変換され、2.1で示したように関数型計算の枠組みの中で並列処理される。

(2)の方法を実現するために、ゴール節とリレーション内のタプルとして表現されたファクト節の単一化を行う機構を関係演算処理系の中に実現する。ルール節に対するリダクションは、ルール操作系においてゴール節中の全リテラルがファクト節との単一化可能な段階まで続けられる。すなわち、ゴールとして表現された問い合わせは、ファクト節と単一化が可能なサブゴール群(ANDリテラル群)に変換される。同じ述語名をもつファクト節の集合は、同じ名前の1リレーションとして扱われ、各ファクト節はそのリレーション内のタプルとして表現される。サブゴールは、それらが同じ変数を共有していない限り、各々独立に処理される。その結果、AND並列性が引き出される。変数を共有しているサブゴール間では、変数/値の束縛環境がストリームとして引き渡される。文献[3]、[4]において提示されたように、我々のアプローチにおいても、各サブゴールは変数/値の束縛環境を入力し、新たな束縛環境を生ずるプロセスとして動作する。ゴール節内の各サブゴールを扱うプロセスは、束縛環境をストリームとして返す関数として実行される。関係演算と同様に、関係データベースに対する推論操作は関数計算の枠組みの中で要求駆動型制御のもとでストリーム型の並列処理により実行される。例えば、3リテラルから成るゴール節 " $\leftarrow L_1, L_2, L_3$ " は3つの関数ノードとして処理される。各関数ノードの束縛環境の出力ストリームは、他のノードの入力ストリームとして扱われる。関係演算処理と同様に、各サブゴ

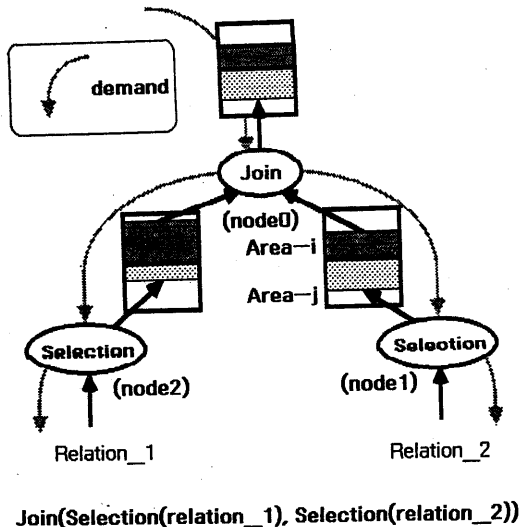


図1 要求駆動型評価による並列処理

ールの起動は、生成者ノードからの全ストリームデータを持つことなくページ単位に行われる。これによりストリーム指向型の並列処理がリテラル間（サブゴール間）で実現される。関数ノード間では、次のアルゴリズムにより要求駆動型制御のもとで並列処理が行われる。

(1) 1リテラルに対する推論操作に対応する関数ノードは、消費者ノードからデマンドを受け取ると、新たな束縛環境を1ページ分生成し、出力バッファへ格納するまで、(2)、(3)の操作を繰り返す。

(2) ダブルバッファリング機構を有する入力バッファの一方の領域に格納されている1ページ分の束縛環境をアクセスする。このとき、入力バッファの他方の領域へのページの格納を要求するデマンドを生産者ノードへ先出しする。結果として、ストリーム指向型並列処理がこのノードと生産者ノードの間で実現される

(3) (2)でアクセスしたページ内の束縛環境を用いて演算対象リレーションに表現されたファクト部に対する探索および単一化を実行し、新しい束縛環境を生成し、出力バッファへ格納する。出力バッファが1ページ分の新しい束縛環境で埋められると、このノードは処理を中断し、ストリームデータの消費者ノードからの次のデマンドを待つ。さもないと(2)、(3)を繰り返し実行する。操作した入力ページがストリームデータの最終ページの場合には、推論操作を終了する。

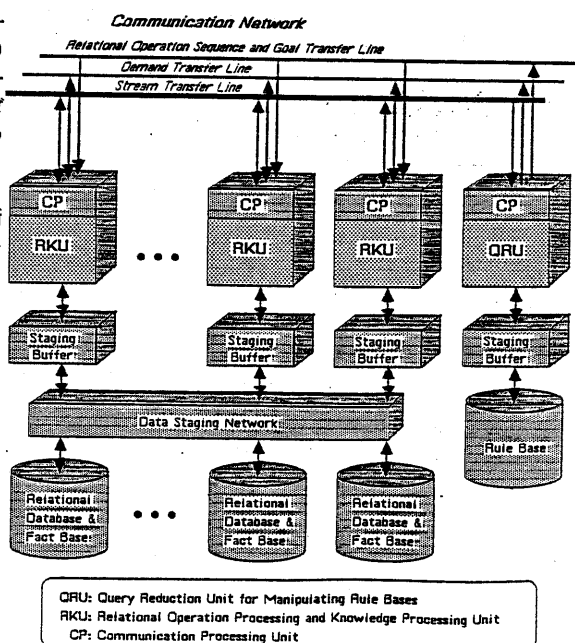


図2 アーキテクチャ

3. アーキテクチャ

ストリーム指向型関係データベースマシンのアーキテクチャを図2に示す。各プロセッサ(RKUまたはQRU)は逐次型の汎用プロセッサであり、それらは通信ネットワークによって結合されている。データベース内のリレーション群は、ディスク装置に分散して格納される。ディスク装置はステージングバッファ(ディスクキャッシュ)を介してプロセッサに結合されている。

システムは次のコンポーネントにより構成されている。

3.1 QRU

関係データベースへの問い合わせを関係演算列あるいはゴール節へ変換し、それぞれに対応する各関数をRKU群のいずれか1つに割り当てる。QRUでは、Rule Baseに格納されているルール群を用いて問い合わせの変換および最適化を行い、また、並列性を最大限に引き出すために、各関数のRKU群への最適割り当てが行われる。

3.2 RKU

各RKUは関数として定義された関係演算および推論操作を実行する。

各RKUには、1つあるいは複数の関数が配置される。同じRKUに配置された複数の関数間では2.で示した要求駆動型評価による関数計算方式が疑似並列処理により実現される。図3に示すように、RKUに

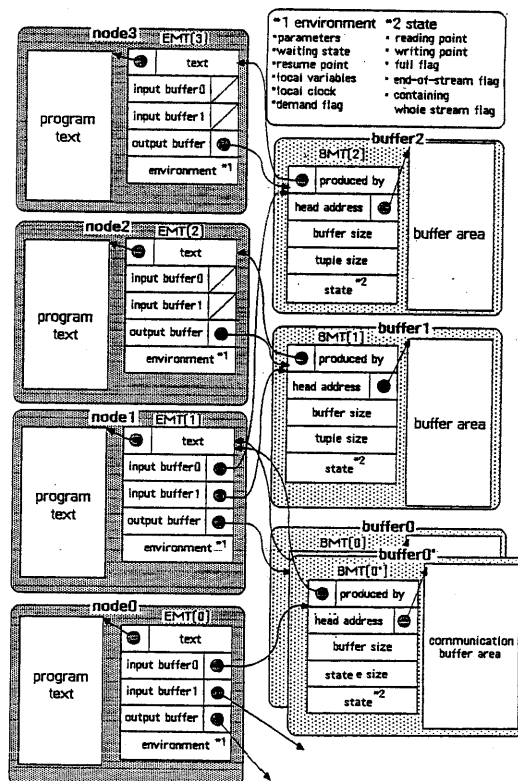


図3 関係演算プロセス

において、各関数ノードは関数本体のプログラムテキストおよび実行状態管理テーブル(EMT)をもつ。EMTは、関数の実行環境を保持し、またその関数ノードの入力バッファおよび出力バッファを指定する。また、消費者ノードから発行されるデマンドもこのテーブル内に示される。各バッファ内のバッファ管理テーブル(BMT)は、バッファの状態を示すために用いられる。バッファ内の現在の読みだし点、書き込み点、およびバッファがストリームデータで埋められていることを示すフラグが用意されている。1 R K U内では各関数の実行順序は関係演算プロセス内の1スケジューラによって制御される。スケジューラは実行可能な関数を見つけ、それらの中の1つに制御を与える。スケジューリング方式として、現在はラウンドロビン方式を採用している。

3. 2. 1 関係演算プロセス

関係演算プロセス内では、表1に示す基本プリミティブがアークテクチャレベルで実現される。これらの基本プリミティブの詳細については文献[6],[10]に詳しい。これらの基本プリミティブは、要求駆動型制御による関数計算の並列処理を実現するために設定したものである。

各関数は図4に示すような3状態(初期状態(Initial State)、実行状態(Executing State)、停止状態(Suspended State))を遷移する。初期状態では、この関数ノードは、このノードが消費するストリームデータを生成する関数ノードへデマンドを発行し、停止状態へ移行する。実行状態では、関数ノードは次の条件の1つが成立すると停止状態へ移行する。

- (i) "put-wait": 出力バッファへ1ページ分のストリームデータの格納を終えたとき。
- (i i) "get-wait": 入力バッファに格納されているページに対する関数計算の実行が完了したとき。

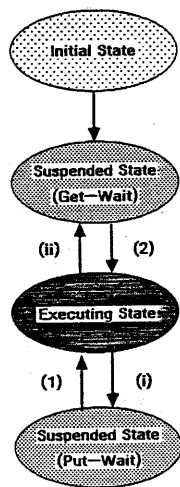


図4 関数ノードの状態遷移

表1 基本プリミティブ

プリミティブ	機 能	
	パラメータ	指定内容
channel	入力ストリームまたは出力ストリームを受け渡すためのチャンネルを設定し、その識別子をreturn valueとして返す。	
	type	チャンネルを介してやり取りをされるストリームの要素のデータタイプを指定する。
	granularity	1回のデマンドにより送られるデータ量を指定する。
	parameter_passing_method	ストリームが再参照されたときに再計算を行う(call-by-name)か、cachingを行う(call-by-need)かを指定する。
new	関数のインスタンスを生成し、その識別子をreturn valueとして返す。	
	f	関数名
	parameters	関数fの入力引数(入力ストリームおよびその他の引数)
	pid	関数インスタンスが配置されるプロセッサの識別子
	cid	関数fが生成したストリームデータが出力されるチャンネル
pre-demand	ストリームの生産者側の関数へデマンドを伝達する。	
	cid	入力チャンネルを指定する。cidで示される入力チャンネルにストリームを出力する関数へデマンドが伝達される。
get1	入力チャンネルのバッファからストリームの1要素を取り出す。バッファが空のときは、そのチャンネルに出力を行う関数へデマンドを伝達し、そのチャンネルのバッファがデータで満たされるまで待つ。ストリームの各要素は、一度バッファから取り出されると削除される。	
	cid	入力チャンネルを指定する。
get2	cidで示された入力チャンネルの二つのバッファ領域の一方が空のときは、そこへストリームデータを格納するためのデマンドを発行する。それから、他方のバッファ領域を埋めているストリームデータを取り出しを行う。データを取り出している方のバッファが空となったときには、デマンドを発行したのち、他方のバッファからストリームデータの取り出しを開始する。バッファにストリームデータが到着していない場合には、その到着を待つ。ストリームの各要素は、一度バッファから取り出されると排除される。	
	cid	入力チャンネルを指定する。
put1	dで示されたストリームデータの1要素(その関数インスタンスのreturn valueの一部)を出力チャンネルのバッファに関数のreturn valueとして書き出す。1回のデマンドに対してあらかじめ指定された粒度のデータを生成し終えると、次のデマンドを待つ。	
	d	出力されるストリームデータの1要素。
put2	機能的にはput1と同じであるが、出力チャンネルのバッファにダブルバッファリング機構が実現されている場合に用いられる。	
	d	出力されるストリームデータの1要素。
mark_end_of_stream	ストリームデータの終端にストリームの生成の終了を示す識別子として"EOS"を書き込む。	
check_end_of_stream	cidで示されたストリームの終端("EOS")を検出した場合にTRUE、そうでない場合にFALSEをreturn valueとして返す。	
	cid	入力チャンネルを指定する。

(get1およびput1は、ストリームの消費者側の関数と生産者側の関数が同じプロセッサに配置された場合に、get2およびput2は、それらの2関数が異なるプロセッサに配置された場合に用いられる。)

“put-wait”によって停止状態へ移行した場合には、関数ノードはEMT内に“put-wait”フラグをセットし、また、そのノードの実行再開点および実行環境を保持しておく。また、そのノードの出力バッファのBMT内に“full flag”をセットする。もしも、生成し終えたページの消費者ノードが他のR KUに配置されている場合には、そのページの転送を通信プロセスへ要求する。その後、制御権をスケジューラへ返す。“get-wait”によって停止状態へ移行した場合には、ノードはEMT内に“get-wait”フラグをセットし、そのノードの実行再開点および実行環境を保持しておく。それから、入力バッファへの次ページの格納を生産者ノードに要求するためにデマンドを発行する。もしも、ストリームデータの生産者ノードが他のR KUに配置されている場合には、デマンドの送出を通信プロセスへ要求する。その後、ノードは制御権をスケジューラへ返す。

停止状態では、関数ノードは次の条件の1つが成立すると実行状態へ移行する。

- (1) ノードが“put-wait”によって停止した場合、消費者ノードからのデマンドが到着し、さらに、スケジューラから制御権を得たとき。
- (2) ノードが“get-wait”によって停止した場合、次の演算対象ページが入力バッファへ格納され、さらに、スケジューラから制御権を得たとき。

3.2.2 通信プロセス

通信プロセスは、各R KUおよびQ RUに配置され、R KU間あるいはR KUとQ RU間でのストリームデータ、デマンドおよび関数の割り当て情報の転送を実現する。通信プロセスの動作は次のとおりである。

- (1) 別のプロセッサに割り当てられた消費者ノードから発行されたデマンドが通信プロセスに到着すると、通信プロセスは関係演算プロセスへの割り込み処理により、生産者ノードの実行状態管理テーブルにデマンドの到着を示す。
- (2) 別のプロセッサに割り当てられた生産者ノードからのストリームデータが通信プロセスに到着すると、通信プロセスは関係演算プロセスへの割り込み処理により、消費者ノードの入力バッファへそのストリームデータを格納する。
- (3) 自R KU内のノードから他のR KU内のノードへのデマンドの転送が要求され、また、通信ラインへのアクセス権が獲得されると、通信プロセスはバケットによって他のR KUへデマンドの転送を行う。
- (4) 自R KU内のノードから、他のR KU内のノードの入力バッファへのストリームデータの転送が要求され、また通信ラインのアクセス権が獲得されると、通信プロセスはバケットにより他のR KUへストリームデータの転送を行う。

3.3 通信ネットワーク

ストリームデータおよびデマンドは、Q RUおよびR KU群を結合する通信ネットワークを介して転送される。また、関数ノードの割り当てのための情報および

関数ノード間を結ぶチャネルの設定のための情報が、Q RUとR KU間で転送される。

ストリームデータおよびデマンドの転送は各R KU内の通信プロセスによって制御される。

3.4 データ・ステージング・ネットワーク

問い合わせの処理対象となったソース・リレーション群はデータステージング・ネットワークを介してディスク装置からステージング・バッファへ転送される。各ソース・リレーションは、R KUに結合されたステージング・バッファのいずれか1つに格納される。

4. 並列性

4.1 ストリーム型並列性

提案した処理方式の特徴の1つは、ストリーム型並列処理を実現する機構である。ここでは、ストリーム型並列処理における並列性を引き出すための条件について検討する。図5に示すような2個の結合演算ノードからなる問い合わせ（の一部）を考える。この場合、Join-2ノードが2つの入力バッファ内に格納されている2ページに対して結合演算を行っている間に、Join-1ノードがJoin-2のインナリレーションの次ページの生成を完了できれば、Join-2ノードにおいて実行の中断は生じない。すなわち、Join-2ノードにおける演算実行の中断は、Join-1によって生成される次ページの生成が遅れることによって引き起こされる。

ここでは、Join-2ノードの実行を中断させないための条件を設定する。ここで用いるパラメータを次のように定義する。

jsf1: Join-1ノードの結合率 (join selectivity factor)

((Join-1ノードが生成する中間リレーションのタプル数) = jsf1 * (Join-1のインナリレーションのタプル数) * (Join-1のアウトリレーションのタプル数))

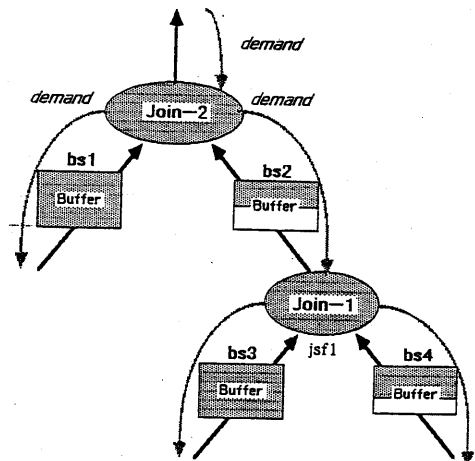


図5 ストリーム型並列処理

js1: Join-2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(タプル数)
 bs2: Join-2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(タプル数)
 bs3: Join-1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(タプル数)
 bs4: Join-1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(タプル数)
 k1: bs1内のタプル群の結合演算対象属性値の種類の数
 k3: bs3内のタプル群の結合演算対象属性値の種類の数

Join-1, Join-2ノードでは各々、結合演算属性上でソートされたアウタリレーションのページに対してバイナリサーチ・アルゴリズムによって結合演算を行う。すでにソートされたアウタリレーションのページとインナリレーション内のタプル群との比較演算回数は次の通りである。

$$bs2 * (\log_2 bs1 + bs1/k1) \quad (1)$$

Join-2ノードのインナリレーションの1ページを生成するためのJoin-1ノードにおける比較演算回数は次の通りである。

$$bs2 * (\log_2 bs3 + bs3/k3) / (jsf1 * bs3) \quad (2)$$

ここで、演算結果のタプル群の出力バッファへの書き込み時間を無視すると、Join-2ノードが中断なく結合演算を行うための条件は次のようになる。

$$(1) \geq (2), \text{すなわち,} \\ (\log_2 bs1 + bs1/k1) \geq (\log_2 bs3 + bs3/k3) / (jsf1 * bs3)$$

異なるプロセッサに配置された全ての2ノード(生産者ノードと消費者ノード)間でこの条件が成立する場合にストリーム型並列処理の効果が最大限に引き出されることになる。

4.2 通信処理オーバーヘッドの排除

問い合わせを並列処理する場合の通信処理オーバーヘッドを排除するための条件を示す。

通信処理は排他的に行われる。すなわち、2台のプロセッサ間でネットワークを介して通信が行われている間には、他のプロセッサはネットワークを使用することはできない。また、デマンドの転送とストリームデータの転送も同時には行えない。ここでは図6に示すように、3台のプロセッサによって問い合わせを実行する場合を考える。Triは演算結果の1ページを生成するのに要するプロセッサiの実行時間、Tdiは1デマンドの転送時間、Tsiは1ページの転送時間とする。この場合、通信オーバーヘッドによる中断なく各プロセッサが実行を継続するためには、ボトルネックとなっているプロセッサが他プロセッサへ転送するための1ペ

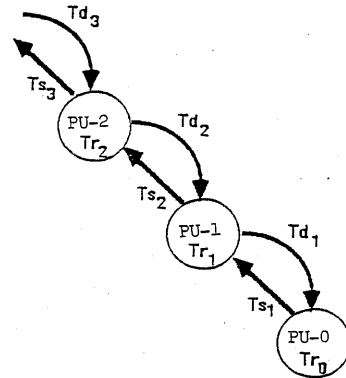


図6 プロセッサ間通信処理

ージを生成している間に、消費者ノードとプロセッサ-2, プロセッサ-2とプロセッサ-1, プロセッサ-1とプロセッサ-0の間で、3回のデマンド転送と、3回のページ転送が完了することが必要である[5]。これは、ストリーム型の並列性が、ボトルネックとなっているプロセッサによって制限されていることによる。したがって、ボトルネックとなっているプロセッサの実行が、通信処理によって中断されないことが重要となる。

一般に、ストリーム型の並列処理において、通信処理によって並列性を損なわないための条件は次のようになる。

$$\sum_{i=1}^n Tdi + \sum_{i=1}^n Tsi < \max_{i=1,n} Tri$$

5. 実験

ストリーム指向型データベースマシンの性能を評価する環境を実現するために、Sun-2ワークステーション[13]上に関係演算処理系を開発した。この処理系は、3.で示した構成のソフトウェアによって実現されており、2.で述べた処理方式をシミュレーションによって評価できる環境を備えている。この処理系は、R KUにおける関係演算プロセスに対応しており、実際に、表1に示した基本プリミティブを使用することによって関係演算処理を行う。問い合わせの実行時間は、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時間およびシステムコール処理時間の実測値である。我々は、プロセッサ間での並列処理の効果を評価するために、この関係演算処理系上に並列処理のシミュレーション環境を実現した。

5.1 問い合わせ

関係演算処理系を評価する場合に生じる問題は、評価用の問い合わせの選択である。ここでは、複数の結合演算からなる問い合わせの処理時間を評価する。結合演算は関係演算の中で最も重要でかつ長い処理時間を要する演算として知られている。問い合わせの中に

は、選択演算や射影演算が含まれている場合がより一般的であるが、ここでは結合演算間での並列処理の効果を明確にするために、結合演算だけからなる問い合わせを対象とする。3つの結合演算からなる4種類の問い合わせを用いる。それらの問い合わせは図7に示すような構造を持つ。問い合わせを構成する各結合演算ノードは別々のプロセッサに配置されているものとする。

ソースリレーションのタプル数、結合演算結合率 (join selectivity factor) および各バッファサイズは、各問い合わせにおいて表2のように設定する。

ストリーム指向型アプローチでは、結合演算結合率およびバッファサイズが変わらなければ、通信トラフィックの頻度、1回のデータ転送量、およびページ間の演算時間は変化しない。本実験の目的は並列性の測定にあるので、ソースリレーションのタプル数の変化による問い合わせ処理時間の変化に関する実験結果の詳細については、ここでは提示しない。

ソースリレーションのタプル数は、各々同じ値に設定し、また、結合演算結合率は、表2に示すように各問い合わせにおいて異なる値に設定する。各リレーションのタプル長 (ソースリレーションおよび中間リレーション) 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。また、それらの整数型属性内の値は一様分布している。

結合演算においてインナリレーションの1ページとアウトタリレーションの1ページを比較する場合には、

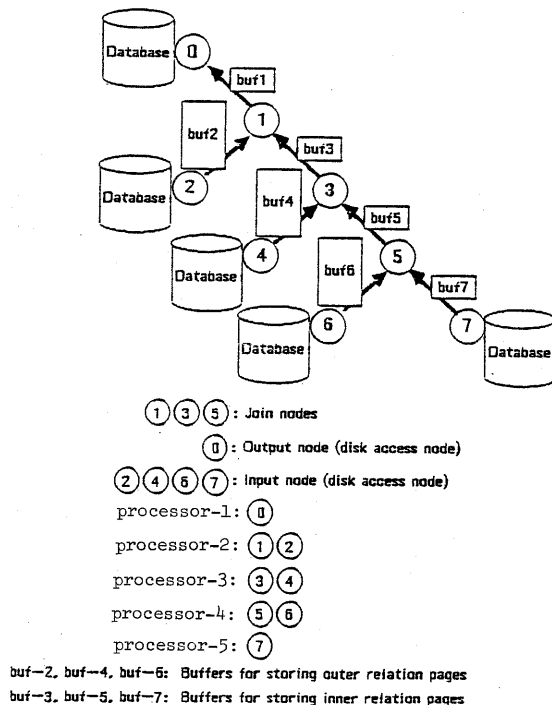


図7 問い合わせの構成

4. 1で述べたように、アウトタリレーションのページがあらかじめ結合演算属性上でソートされ、インナリレーションの各タプルをアウトタリレーションのページ内のタプル群に対してバイナリサーチするアルゴリズムを用いる。

5.2 実験結果

提案したストリーム指向型並列処理方式の特徴を明らかにするために、データ駆動型評価による並列処理方式と対比させて考察を行う。ここでは、その方式をデータ駆動型アプローチ (data-driven approach) と呼ぶ。データ駆動型アプローチでは、関係演算のストリーム型並列処理がデータ駆動型評価の枠組みの中で実現される。データ駆動型アプローチに関しても、やはり、5.1節に示した実験環境で実測値を測定した。データ駆動型アプローチでは、デマンドは用いられない。従って、生産者ノードにおける関係演算の実行は、出力バッファの容量 (消費者ノードの入力バッファの容量) に関係なく継続される。したがって、データ駆動型のアプローチでは、各ノードのバッファは、問い合わせ実行中に生成される中間結果のページ群を格納するのに十分な容量を持つものとする。すなわち、ネットワークが利用可能であるならば、生産者ノードは、消費者ノードの実行状況に関係なく、1ページを生成すると直ちに消費者側ノードへそのページを転送することになる。

図8 (a) および図8 (b) はそれぞれ、データ駆動型アプローチと我々の提案するストリーム型並列処理アプローチによる問い合わせ-1 (Query-1) の処理の様子を示している。

ここでは、通信速度は 16 (milliseconds/2Kbytes-packet) に設定した。

データ駆動型アプローチと我々のアプローチを比較すると、両者とも同等の並列性を引き出している。

図9 (a) および (b) はそれぞれ、データ駆動型アプローチとストリーム型並列処理アプローチによる問い合わせ-2 (Query-2) の処理の様子を示している。

表2 パラメータの設定

node #	outer-relation-size (tuples)	inner-relation-size (tuples)	join selectivity factors (intermediate relation size (tuples))			
			Query-1	Query-2	Query-3	Query-4
5	1000	1000	1/1000 (1000)	2/1000 (2000)	0.5/1000 (500)	1/1000 (1000)
3	1000 #1 2000 #2	----	1/1000 (1000)	2/1000 (4000)	0.5/1000 (250)	1/2000 (1000)
1	1000	----	1/1000 (1000)	2/1000 (8000)	0.5/1000 (125)	1/1000 (1000)

#1: Query-1, Query-2, Query-3

#2: Query-4

tuple size: 64 (bytes)

buffer size (buf-1, 3, 5, 7 for inner relations): 8 (Kbytes)

buffer size (buf-2, 4, 6 for outer relations): The same size as the outer relation size.

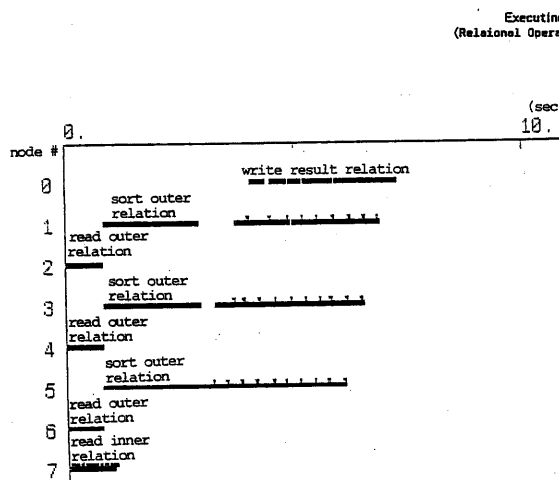


図 8(a) データ駆動型アプローチの
タイムチャート(Query-1)

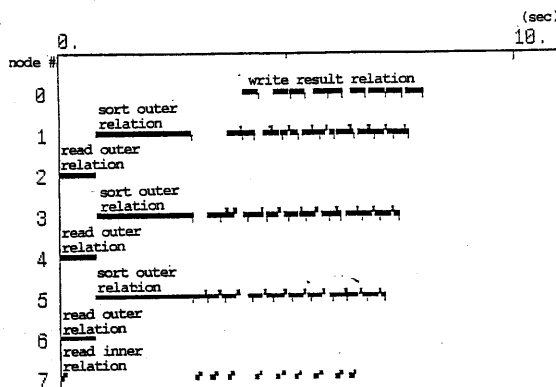


図 8(b) ストリーム型アプローチの
タイムチャート(Query-1)

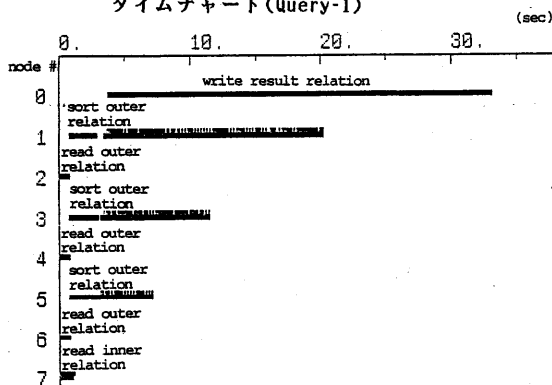


図 9(a) データ駆動型アプローチの
タイムチャート(Query-2)

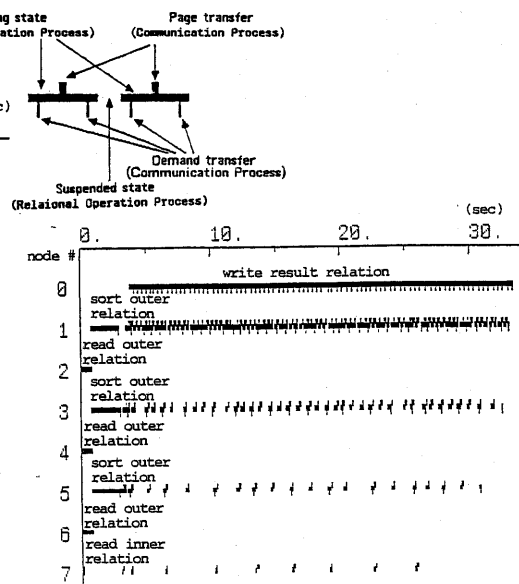


図 9(b) ストリーム型アプローチの
タイムチャート(Query-2)

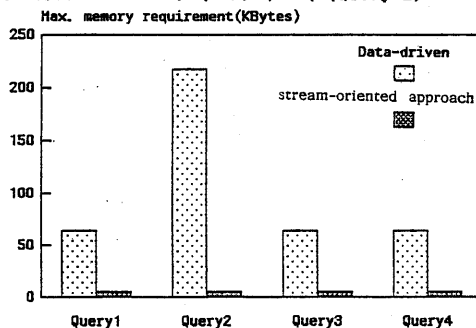


Fig. 11. 図 10 メモリ使用量

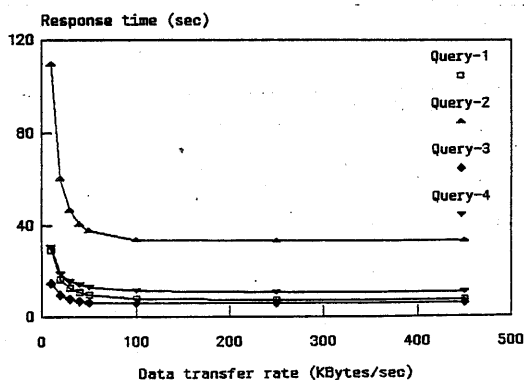


図 11 データ転送速度と応答時間

問い合わせ-2では、問い合わせ中のnode-1(①)がボトルネックノードとなる。これは、問い合わせ-2の中の結合演算ノードがソースリレーションよりも大きな中間リレーションを生成することによる。図9(b)は、デマンド転送のオーバーヘッドが問い合わせ処理効率を劣下させないことを示している。さらに、図9(a)と比較すると、通信処理が一時に集中せず一様に分散しているので、データ転送時の衝突(collision)によるオーバーヘッドが軽減される。

図10は、データ駆動型アプローチと我々のアプローチにおいて、各問い合わせ処理におけるボトルネックノードのインナリレーションのページ群を格納するためのメモリ使用量を示している。要求駆動型評価に基づく我々のアプローチでは、問い合わせ処理が限られたメモリ内で進められる。一方、データ駆動型アプローチでは、中間ページ群を格納するために、より多くのメモリ空間が必要となる。これは、高い結合演算結合率をもつ結合演算ノードが、その演算結果の中間ページ群の消費者ノードに対し、消費者ノードの処理能力以上に中間ページ群を供給し続けることによって起こる。結果として、消費者ノードの入力バッファには、多くの中間ページが保持されることになる。このような場合、限られたメモリ資源の中で問い合わせ処理を遂行する我々のアプローチの有効性が顕著となる。

我々のアプローチにおいて、プロセッサ間のデータ通信速度が問い合わせ処理の応答時間に与える影響を検討するために、異なるデータ通信速度における応答時間を測定した。図11に、各問い合わせ処理の応答時間とデータ通信速度の関係を示す。応答時間は、120(Kbytes/sec)以上では影響を受けていない。一般に、10Mbpsの通信速度を実現するネットワークにおいては、120(Kbytes/sec)の実効通信速度を実現することは容易である。図11は、ここで行ったような問い合わせ処理においては、データ通信速度によって並列性が低下しないことを示している。しかしより複雑な問い合わせをより多くのプロセッサによって処理する場合には、データ通信が並列性を低下させる原因となるものと考えられる。

6. おわりに

本稿では、データベースシステムの多様なアプリケーションおよび多様な問い合わせに柔軟に対応するストリーム指向型関係データベースマシンのアーキテクチャを提示した。また、このマシンが要求駆動型制御のもとで大量データに対する関数計算を行うための基本プリミティブを示し、さらに、それらの基本プリミティブを用いて実現されるR K Uの関係演算処理系の構成を示した。

現在、我々は複数台のワークステーションをLAN(Local Area Network)により結合した環境で、提案アーキテクチャの擬似的な実現を試みている。この試作システムは、LAN上でのデータベースサーバとしての役割を果たすものである[11]。現在、R K Uにおける推論操作系および通信プロセスの設計、および、Q

R Uにおける問い合わせ最適化機構、R K U群への関数の割り当て方式の設計を行っている。

参考文献

- [1] D. P. Friedman and D. S. Wise, "Aspects of applicative programming for parallel processing," IEEE Trans. Comput., vol. C-27, pp. 289-296, Apr. 1978.
- [2] H. Gallaire, J. Minker and J. M. Nicolas, "Logic and databases: a deductive approach," ACM Computing Surveys, vol. 16, no. 2, Jun 1984.
- [3] Z. Halim and I. Watson, "An or-parallel data-driven model for logic programs," in Proc. Int. Workshop High-level Computer Architecture, pp. 1.26-1.36, May 1984.
- [4] R. Hasegawa and M. Amamiya, "Parallel execution of logic programs based on dataflow concept," in Proc. 1984 Int. Conf. Fifth Generation Computer Systems, pp. 507-516, 1984.
- [5] 加藤, 清木, 益田, "要求駆動型制御による関係演算パイプライン処理の実現法" 情報処理学会第32回全国大会論文集, Mar. 1986.
- [6] 加藤, 清木, 益田, "並行プログラムのストリーム型並列処理方式," 情報処理学会計算機アーキテクチャ研究会62-2, Jul. 1986.
- [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] Y. Kiyoki, R. Hasegawa and M. Amamiya, "A stream-oriented parallel processing scheme for relational database operations," in Proc. 1986 Int. Conf. Parallel Processing, 1986.
- [9] Y. Kiyoki, M. Isoda, K. Kojima, K. Tanaka, A. Minematsu and H. Aiso, "Performance analysis for parallel processing schemes of relational operations and a relational database machine architecture with optimal scheme selection mechanism," in Proc. 3rd Int. Conf. Distributed Computing Systems, Oct. 1982.
- [10] Y. Kiyoki, K. Kato, T. Masuda, "A relational database machine based on functional programming concepts," in Proc. ACM-IEEE Computer Society Fall Joint Computer Conf., Nov. 1986.
- [11] Y. Kiyoki, K. Kato, T. Masuda, "A stream-oriented approach to distributed query processing in a local area network," Research Report, Institute of Information Sciences and Electronics, Univ. of Tsukuba, Mar. 1986.
- [12] 清木, 加藤, 益田, "要求駆動型制御による関係演算パイプライン処理方式の実現法," 情報処理学会アドバンスドデータベースシンポジウム予稿集, pp. 51-58, 1985.

[13] "Programmers Reference Manual for the Sun Work-station", Sun Micro Systems, Inc., 1982.

[14] P. C. Treleaven, D. R. Brownbridge and R.P. Hopkins, "Data-driven and demand-driven computer architecture," ACM Computing Surveys, vol.14, no. 1, Mar. 1982.

[15] H. Yokota and H. Itoh, "A model and an architecture for a relational knowledge base," Proc 13th Int. Symp. Computer Architecture, pp. 2-9, June 1986.