Vol.2016-OS-137 No.1
2016/5/30

演算子間データ配送方式の検討

川島 英之† 建部 修見†

DBMS 内部で多様な分析処理を行うスキームは in-DB 分析と呼ばれる．In-DB 分析においては分析処理が複数の演算子により構成される．膨大量のデータを扱う場合には，それらの演算子を流れるデータ量が膨大になるため，演算子間のデータ配送方式が性能上の問題となり得る．最も基本的なデータ配送方式はタプルを 1 つずつ移動する方式であるが，移動するタプル数を増やすことにより性能を高められる可能性が存在する．その可能性を現実的なシステムにおいて調査するために，本研究ではプロトタイプ SQL 処理系を設計・実装する．同処理系においてタプル配送数を変動させる実験を，結合演算を対象にして行った結果，配送数をある程度まで引き上げることにより性能上の利点が顕れることを実験的に示す．

1. はじめに

様々な分析的データ処理が DBMS において行われている．RDBMS における古典的な分析的ワークロードとしては TPC-H ベンチマーク [9] が知られている．一方，新しいワークロードとしてはデータマイニング・機械学習などがある．リレーショナルデータベースシステムにおいてこのような処理を行う基盤として MADLib [7]や Hivemall [8] 等が研究開発されている. MADLib は PostgreSQL, Hivemall は Hive 上でこれらの高度な分析タスクを実行する. これらのワークロードが実行されるとき，その処理内容は論理的には演算木により表現される．演算木は演算子をノード，演算子間を繋ぐデータ配送機構をエッジとする非循環有向グラフ（Directed Acyclic Graph, DAG）である．

分析的データ処理を構成する演算木におけるデータ配送の特徴を眺めると，そこには既存のデータ処理系では注目されてこなかった二つの特徴が存在することに気づく．第一の特徴は，分析的な演算子は多量の結果を出力する可能性がある点である．例えば演算子が複合イベント処理 (Complex Event Processing, CEP)である場合を考える．非決定性オートマトン(NFA)に基づく処理技法の場合，CEP 問合せは NFA による表現へ変換される．そして NFA を受理状態に遷移させる 1 つのイベントが到着した場合に，問合せに対応する結果として，シーケンスイベント集合がまとめて出力される．この量は場合によっては 100 万件を超えることもある．別の例として，演算子が数百万のノードから構成されるベイジアンネットワークである場合を考える．1 つのイベント到着に伴い確率伝播が行われ，条件（例：事象生起確率>閾値）を満たす最大数百万のノードが演算子から出力される．このようにデータマイニング・機械学習に関する演算子は 1 つの入力に対して膨大量の出力を生成する可能性がある．最後の例としてリレーショナルデータベースにおける分析的データ処理ベンチマークである TPC-H を考える．TPC-H では複数の結合演算を実行する問合せが存在する．もしも入力データ量が膨大であり，結合演算の選択率が高ければ，結合演算は膨大量の結果タプルを出力する．

この特徴に際して顕在化する問題は演算子間のデータ配送処理コストである．いまなお標準的な演算子間データ配送機構の実装は要求駆動方式，いわゆる Volcano スタイル [1]である．これは親演算子が子演算子にタプルを要求する方式である．演算子の処理はルートから開始する．子演算子が生成すべきタプルを有していない場合，それを生成する．あるいは生成すべきタプルの源が存在しない場合，その演算子がさらに一つ下の演算子に対してタプルを要求する. 換言すればタプルを pull するスタイルであると言える. 演算子を open-next-close イテレータ処理により実現される. Volcano では一度の next 呼び出しにおいて一つのタプルのみを要求する．そのために処理するタプル数が多い場合には演算子呼出の回数が増加する．演算子呼出は関数呼出として実装される．多数の関数呼出は性能劣化を引き起こす可能性がある．

そこで本論文では演算子呼出において 1 つではなく，複数のタプルを提供する方式を調査する．演算子呼出のみをシミュレーションすれば関数呼出の回数が性能に直接的に影響を与えることが考えられる．一方，現実的な SQL 処理系においてはストレージアクセス，タプル生成，タプル配送，演算子呼出，サーバからクライアントへのデータ転送など，様々な処理が実行される．それゆえ演算子呼出回数の削減が問合せ処理の性能に与える影響は明らかではない. そこで本研究では SQL 変換，演算子スケジューラ，各種リレーショナル演算子を有するプロトタイプ SQL 処理系において，タプル配送方式の影響を調査する．

本論文の構成は次の通りである．2 節では演算子出力制御方式を述べる．3 節ではプロトタイプ SQL 処理系を述べる. 4 節では評価を述べる. 5 節では演算子間データ配送方式の検討を行う．6 節では関連研究を述べる．最後に 7 節で論文をまとめる．

† 筑波大学システム情報系/筑波大学計算科学研究センター
Faculty of Systems and Information Engineering, University of Tsukuba / Center for Computational Sciences, University of Tsukuba

ⓒ 2016 Information Processing Society of Japan

2. 演算子出力制御

2.1 準備

データ処理系は演算木（演算子をノード，演算子間をエッジとする木構造）により論理的に表現される．いま，分析的データ処理の標準的ベンチマークである TPC-H の Query 3 や Query 19 のように，結合演算を複数行ってから集約処理をする問合せを考える．仮に結合演算が 3 つあり，その後に集約演算が実行される場合，その演算木は図 1 のように表現される．α は集約演算を表し，他の 3 つは結合演算を表す．各結合演算子の矢印の先には SCAN 演算子があるとする．SCAN 演算子はストレージからタプルを読み出す処理を担う．

本研究では結合演算以外を対象にしない．整列やグルーピングは頻繁に使用される重要な演算子だが，これらの演算子を実行するには全てのデータが必要になるため，本研究で扱う要求駆動方式は利用不可能になる．それゆえ本研究では結合演算のみを対象とする．

本研究では結合演算アルゴリズムとして入れ子ループ結合を用いる．高速な等結合演算アルゴリズムとしてマージ結合やハッシュ結合が存在するが，それらのアルゴリズムはメモリにデータが載りきらない場合には中間データをストレージに退避させる必要がある為，データサイズに比べてメモリサイズが小さい場合には不利となる．さらに，マージ結合とハッシュ結合は非等結合には利用できない一方，入れ子結合は非等結合にも適用可能であり，その適用範囲は比較的広いと考えられる．

本研究ではいずれの結合演算もある程度の選択率を有することにする．選択率は低いことが多くの場合に存在するが，低選択率でも入力データサイズが大きければ出力は巨大になる．本研究ではそのようにメモリサイズが問題になるような状況を想定して行われている．論文の以降では方式の説明において，図 1 の演算木を用いる．データマイニング演算子も結合率の高い結合演算子も，出力量が多いという意味では同一であるため，データマイニング演算子を使った演算木は本論文では具体的には扱わない．

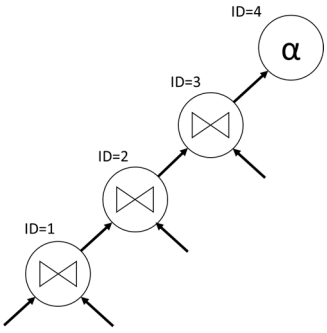

図 1　演算木

2.2 要求駆動・タプル：Volcano

Volcano [1]と呼ばれる要求駆動型のデータ配送方式が多くの DBMS において長きにわたって採用されてきた．図 2 に示されるような演算木がある場合，volcano 方式では上位ノードが下位ノードへタプルを要求する．要求を受けたノードは 1 つのタプルを生成して上位ノードへ配送する．例えば図 2 の場合，最初のタプル生成要求は ID=4→ID=3→ID=2→ID=1 と伝播する．リーフノード（ID=1）は 1 つのタプルを生成してその親(ID=2)に配送する．このタプル配送は ID=2→ID=3→ID=4 と伝わる．外表の要素であるこのタプルにより ID=3 が複数のタプルを生成する場合，その後は ID=4→ID=3 が幾度か続くことになる．そして ID=3 がタプル生成不能状態になったあと，また ID=2 へとタプル生成要求が伝えられる．この様子を図 2 に示す．

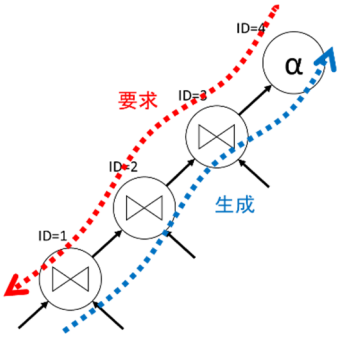

図 2 Volcano の要求・タプル生成過程

Volcano 方式の長所は配送に要するタプル量が少なくて済むことである．要求に基づいて 1 つのタプルを生成してそれを要求元へ配送した後，そのタプルは即座に消費される．一方 Volcano 方式の短所は並列性が余り考慮されていないこと（但し分散処理に際しては exchange 演算子なる複数タプルを一括送信する演算子が Volcano には存在する），ならびに多数の関数呼び出しが必要となり，それが高負荷になり得る事である．

2.3 要求駆動・ブロック

2.2 節で述べた要求駆動方式では，演算子が高々1 つのタプルのみを生成すると述べた．複数タプルを生成する方式は古くから行われてきた [3]．CPU キャッシュミスが多発する問題に着目し，演算子が複数のタプルを生成して処理を効率化する研究がある [2, 4]．MonetDB/X100 [2]においては出力タプルをベクタ化することにより高性能を達成している．CC-Optimizer は L1 キャッシュミスを削減することで PostgreSQL における問合せ処理を高性能化する [4]．BDQ はリモートプロキシを用いた演算子並列方式により高性能を達成している [5]．

要求駆動・ブロック方式の長所は単純な要求駆動方式である Volcano に比べて性能が極めて高いと考えられることである．この方式の主たる性能改善理由はベクタ化による CPU キャッシュミス率の削減だとも考えられる一方，関数

ⓒ 2016 Information Processing Society of Japan

呼出回数の削減による性能改善の影響も考えられる．

3. プロトタイプ SQL 処理系

3.1 設計と実装

性能評価のためにプロトタイプ SQL 処理系を設計・実装した．この処理系は基本的な機能を有するクエリ処理器である．同処理系の提供するデータ型は固定長文字列型，整数型，単精度浮動小数点型，倍精度浮動小数点型である．提供するリレーショナル演算子は，選択，射影，結合，集計，そして整列である．結合演算の処理アルゴリズムとしては，オンメモリアルゴリズムとして入れ子ループ結合と単純ハッシュ結合，外部アルゴリズムとしてブロック入れ子ループ結合である．

同処理系はユーザから SQL クエリを受け取ると，それをリレーショナル代数による表現に変換するために，中間言語に変換する．そしてその中間言語を解析することで演算木を木構造で構築する．このとき，結合演算が演算木に複数あれば左深木を構築する．TPC-H Query 9 のようにサブクエリが存在する場合には，サブクエリ毎に演算木を構築した後，それらを連結して単一の演算木を構築する．

演算木形状の決定後，演算木の実行スケジュールを決定する．クエリ処理器における演算子間のデータ配送方式としては大きく分けて要求駆動方式と実体化があるが，同処理系では要求駆動方式を実装している．

結合演算については，本実験では Nested Loops Join のみを用いる．なぜならハッシュ結合を用いる場合にはメモリが溢れる可能性があるからである．ハッシュ結合を行うにはまずハッシュテーブルをビルドする必要がある．このビルドされたハッシュテーブルが巨大である場合にはメモリに収まらず，ストレージアクセスが生じてしまう．本研究ではそのような場合を回避すべくハッシュ結合を用いない．

結合演算の実行は根から開始し，所望のタプル数を生成するよう下位のノードに存在する演算子に指令を出す．実装としては関数呼出を行っている．

ストレージマネージャはデータをファイル経由で取得する．データページのレイアウトは NSM (N-ary Storage Model)である．このモジュールはタプルを 1 つずつ取得する実装になっている．取得したタプルはバッファプールに置くこと無く，即座に演算木の葉演算子に渡され，処理が実行される．タプルはメタデータとして ID，スキーマ情報へのポインタ，最終タプルか否かを表すフラグ，タプルサイズ，ならびにデータ領域を有する．NSM であるためファイルからデータをメモリに読込めばよく，データ構造を再編成する必要はない．

実装言語は C++であり行数は約 5500 である．すべてのモジュールはスクラッチから作成された．

3.2 実行状態の保持

要求駆動・ブロック方式を実現するには実行状態を保持する必要がある．なぜならばある演算子が要求されたサイズのタプルを出力したあと，実行制御が親演算子に移動するからである．その演算子が再び親演算子から呼び出されたとき，前回終了時の状態から再開が行われる必要がある．

これを実現するために各演算子に演算の進行状況を持たせる実装を行った．この実装を行った Nested Loops Join のコードを List 1 に示す．処理が中断されるのは 12 行目と 29 行目である．12 行目では外部表の入力タプルが消失したために処理が中断される．29 行目では求められたサイズ（この場合は SzPP）だけのタプルが生成された為に RETURN により処理が中断される．保持している内部表が全て無くなった場合，自分の子演算子に対して続きのタプルを提供するよう指示することが 36，44 行目(fetchLeft, fetchRight)に示されている．

```
1  extern void runJoin(NODE *n)
2  {
3     uint nGen = 0;
4
5     initRunJoin(n);
6     while (true) {
7        for (; n->j.itl != n->j.ltuple.end(); n->j.itl++) {
8           if ((*(n->j.itl))->isFin == true) {
9              pushFinalOne(n);
10             clearTupleList(n->j.ltuple);
11             n->j.ltuple.clear();
12             return;
13          }
14
15          // Right loop
16          bool isFin = false;
17          while (true) {
18             for (; n->j.itr != n->j.rtuple.end(); n->j.itr++) {
19                if ((*n->j.itr)->isFin == true) {
20                   isFin = true;
21                   break;
22                }
23
24                v = getNewTupleVal(n, *(n->j.itl), *(n->j.itr));
25                if (v) {
26                   pushTuple(n, v);
27                   if (++nGen == SzPP) {
28                      n->j.itr++;
29                      return;
30                   }
31                }
32             }
33
34             clearTupleList(n->j.rtuple);
35             n->j.rtuple.clear();
36             fetchRight(n);
37             n->j.itr = n->j.rtuple.begin();
38
39             if (isFin) break; // End of right rel ?
40          }
41       }
42       clearTupleList(n->j.ltuple);
43       n->j.ltuple.clear();
44       fetchLeft(n);
45       n->j.itl = n->j.ltuple.begin();
46    }
47 }
```

List 1: Nested Loops Join（要求駆動・ブロック方式）

ⓒ 2016 Information Processing Society of Japan

一点注意されるべきは，Volcano には実行状態の保持は必要ないことである．なぜなら Volcano の場合には必要なタプルがなければ毎回下の演算子を呼び出せばよいからである．後述する要求駆動・集合方式の場合には演算子を一度だけ呼び出せばよい．従ってこの場合にも実行状態を保持する機構は不要である．すなわち，ブロック単位での実行を行う場合にのみ実行状態の保持が必要になる．

4. 予備評価

4.1 設定

本節では要求駆動・ブロック方式の性能を評価する．評価には次の問合せ（Q1, Q2）を用いた．この問合せは 1 つの結合演算を生成し，それらが left-deep プランにより繋がれる．

```
Q1: select * from a, b where a.key = b.key;
Q2: select * from a, b, c where a.key = b.key and c.key = a.key;
```

ここで各表 a, b のリレーション濃度（タプル数）はいずれも同一とした．また，それぞれ key は 1〜1000 の値を設定した．すなわちこの問合せの結果として得られるテーブルのタプル数は 10000 である．

評価に用いたマシンのスペックは次の通りである．CPU: CPU:Intel Core i5 3.2GHz, RAM: 16GB．コンパイルに際しての最適化オプションは-O2 とした．

4.2 結果

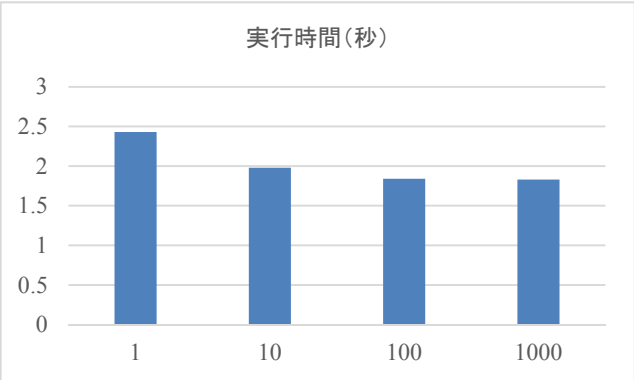

図 3-a：実行時間の比較（Q1，リレーション濃度=1000）

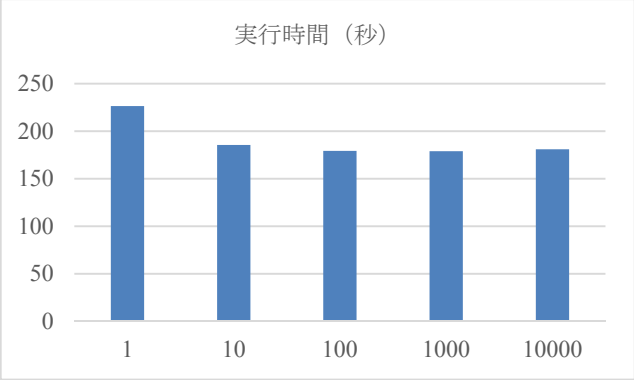

図 3-b：実行時間の比較（Q1，リレーション濃度=10000）

Q1 の問合せ処理の実行時間に関する実験結果を図 3-a,b に示す．対象としたリレーションの濃度はキャプションに示されているように異なっている．濃度が 10 倍違うことで性能が 10 倍異なっていることが観察される．図 3-a において，一度の演算子呼出で 1 つのタプルのみを取得する Volcano 方式（1）は，1 度の演算子呼出で全ての結果を取得する要求駆動・集合方式（100）に比べて 25%程度性能が悪いことが示されている．要求駆動・ブロック方式（10）は，要求駆動・集合方式（1000）よりは遅いが Volcano よりは高速な結果が得られている．図 3-b ではリレーション濃度を 10000 としているが，図 3-a と同様の傾向が見られている．リレーション数を 2 つから 3 つに増やした場合の結果は図 3-c に示されている．この結果は図 3-a, b と同様である．

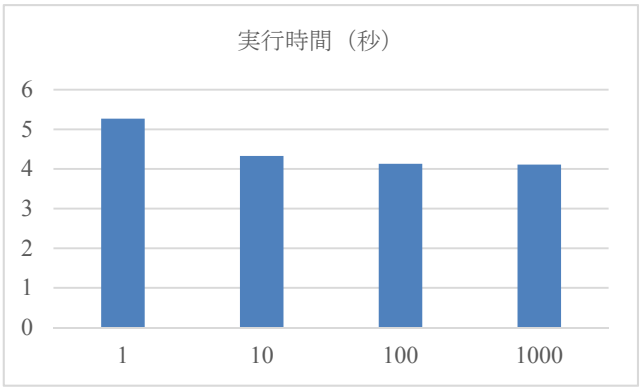

図 3-c：実行時間の比較（Q2，リレーション濃度=1000）

以上より，単純な Volcano 方式（1）の実装は SQL 処理系においては性能上好ましくないと考えられる．また，今回はメモリに乗り切るサイズの結果しか生成されなかった為に要求駆動・集合方式が最良の結果を示したが，メモリサイズよりも出力サイズが大きい場合には同方式によりストレージアクセスが発生し，急激な性能劣化が発現する可能性がある．

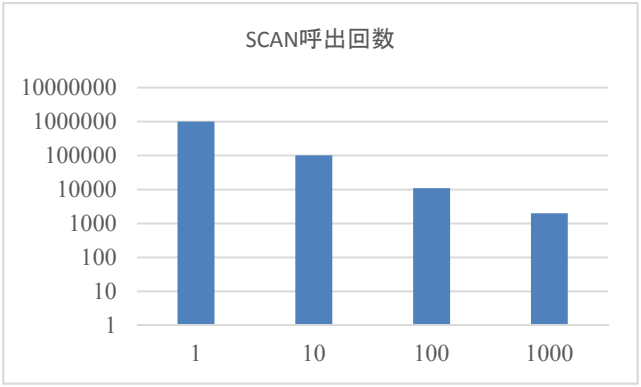

図 4：SCAN 呼出回数（図 3-a に対応）

このような性能差の理由の一つとして直感的に演算子

ⓒ 2016 Information Processing Society of Japan

呼出回数が関連するとの仮説が立てられる．この仮説を検証するために，演算木の最下層の存在する SCAN 演算子が呼び出される回数をリレーション濃度が 1000 の場合において（即ち図 3-a の場合において）測定した．測定結果を図 4 に示す．縦軸は対数値となっていることに注意されたい．また，今回対象とした問合せにおいては SCAN 演算子が 2 つ生成されるが，下記は全ての SCAN 演算子の呼出回数である．そのため右端の要求駆動・集合方式の SCAN 呼出回数は 2002 となっている．この内訳は 1 回の外表呼出，1000 回の内表呼出，1000 回の内表終了タプル呼出，1 回の外表終了タプル呼出である．この実験結果より，演算子呼出回数と処理性能の間には何らかの関係が存在することが示唆されるが，その関係は深くはないように考えられる．

5. 演算子間データ配送機構の検討

演算子間データ配送機構において要求駆動方式を用いた場合，上位ノードが下位ノードに対して一定数以上のタプルを生成させることで性能が改善されることがわかった．一方，ブロックサイズはある程度まで引き上げられると性能改善は微増になる．このような結合木の処理を考えるとき，ブロックサイズが大きすぎるとメモリが枯渇してしまい，システムが停止する可能性がある．それゆえメモリが枯渇せず，かつ十分な性能を発揮するサイズのブロックを選定することが重要だと考える．

本研究においてはブロックサイズと性能との相関性が高くない実験結果が得られた．この理由として実験に用いたプロトタイプシステムが貧弱である点が考えられる．今回のシステムの問題点は次の通りである．第一の問題はストレージベースのシステムになっている点である．このために多数のファイルアクセスが発生し，IO ならびにシステムコールを多発し，性能が劣化した可能性がある．第二の問題点は演算子間データ配送がタプルベースで実現されている点である．複数のブロックをまとめて配送する形式，ならびに必要なカラムのみベクタライズして配送する形式を用いるならば，今回のブロックサイズに伴う性能変化がより大きく発現した可能性がある．第三の問題点は演算子処理が非並列的に実行されている点である．複数のスレッドならびに SIMD の利用により，問合せ処理の基本性能を向上させれば，ブロックサイズにより性能変動が露わになる可能性があると考える．

6. 関連研究

本論文では要求駆動方式を扱ったが，演算子間データ配送方式には他のものもある．本節ではそれらを述べる．

6.1 集合実体化

要求駆動方式においては，演算子はタプルを大量に生成することはない．これとは逆の考え方に集合実体化方式がある．集合実体化方式では演算子は全ての出力タプルを一括出力する．集合の意味するところは表全体，あるいは全タプルである．これを換言すれば “set-at-a-time” 方式となる．この方式は MonetDB において採用されている [3]．本論文においては，処理を演算木のリーフノード (ID=1) から開始してノードを上に辿り（ID=2→ID=3），ルートノード（ID=4）で終了させる方式とする．即ち要求駆動と逆に動作する．なお，本論文の実装においては出力結果はメモリに展開することとし，メモリ不足時にはストレージには待避することなく停止する．この様子を図 5 に示す．

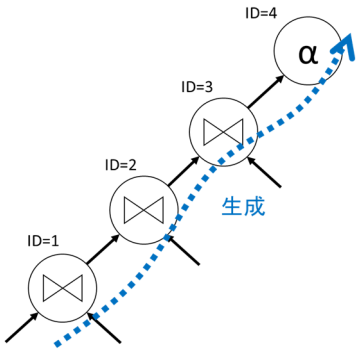

図 5 集合実体化方式

この方式の長所は演算子の並列処理が可能であり，メモリが出力タプルに対して大きい場合には高性能を達成できることである．逆にこの方式の弱点は，演算子出力よりもメモリが少ない場合にはタプルをストレージに出力する必要があり，I/O コストが生じてしまう点である．

6.2 集合実体化・並列

この方式は実体化方式を並列実行する．例えば図 5 の演算木においては，まず，ID=1 の演算子を複数のスレッドで並列処理する．各スレッドは自分の担当処理を終えたら処理を ID=2 の演算子に移し，処理を進める．全スレッドを同期させて実行させることも可能だが，利用可能メモリが十分大きい際にはスレッドを独立に動作させることで処理時間の短縮を図れる．

6.3 供給駆動

要求駆動と逆の考え方として供給駆動がある．マルチコア環境を利用して高い並列性を実現するために供給駆動方式の優れた具体的方式として，morsel 駆動方式が提案されている [6]．この方式では複数のスレッドが並列動作することを前提にしている．各スレッドは演算木の一部を割り当てられる．そして担当部分の入力から出力を一貫して行う方式である．

供給駆動方式の長所は，要求駆動方式と異なり，複数のスレッドをマルチコアで並列動作可能であるために高性能を達成しやすいことである．一方その短所は実体化方式同様にメモリの枯渇可能性である．近年発表された供給駆動方式である morsel 駆動方式においてさえ，メモリが枯渇した場合にはスレッドを停止するなどの処理が必要であることが文献 [6]の 3.2 節に述べられているなど，メモリ枯渇に

ⓒ 2016 Information Processing Society of Japan

ついては対策が取られていない．

また，Neumann ら Just-In-Time コンパイルを用いて問合せ処理の高速化を図っている [10]．この方式では多段結合が多段ループに変換されるために多段結合の出力が全てまとめて出力される．従ってメモリ不足に対応できないという問題が存在する．

7. まとめ

DBMS 内部で多様な分析処理を行うスキームは in-DB 分析と呼ばれる．本論文では in-DB 分析システムに特徴的な問題として，データ配送問題があることを提起した．この問題に対処するために，演算子制御方式として，要求駆動・ブロック方式を述べた．これは tuple-at-a-time 方式で実行される volcano に比べて演算子呼出回数が少ないために効率的だと考えられた．一方，様々なオーバヘッドの存在する現実的なデータベースシステムにおいて，どの程度の効率性が存在するかは明らかではなかった．そこでこの仮説を検証するためにプロトタイプ SQL 処理系を実装し，その処理系において提案方式と Volcano 方式を比較した．結合演算を含む問合せにおいて，要求駆動ブロック方式（ブロックサイズ=1000）は Volcano 方式（ブロックサイズ=1）に比べて 25%程度の効率性を示すことがわかった．

今後の課題はプロトタイプ SQL 処理系における他手法の評価，ならびにプロトタイプ SQL 処理系で処理可能な問合せを TPC-H の全クエリに対応することである．

謝辞

本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，JST CREST「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」による．

参考文献

1) Goetz Graefe: Volcano - An Extensible and Parallel Query Evaluation System IEEE Trans Knowl Data Eng 6(1): 120-135 (1994)
2) Peter A Boncz, Marcin Zukowski, Niels Nes: MonetDB/X100: Hyper-Pipelining Query Execution CIDR 2005: 225-237
3) Sriram Padmanabhan, Timothy Malkemus, Ramesh C. Agarwal, Anant Jhingran: Block Oriented Processing of Relational Database Operations in Modern Computer Architectures. ICDE 2001: 567-574
4) 辻 良繁，川島 英之，"CC-Optimizer: キャッシュを考慮した問合せ最適化器"，日本データベース学会 Letters，Vol. 6, No. 1. pp. 41-44 ． 2007 年 6 月．
5) 油井 誠，宮崎 純，植村 俊亮，加藤 博一.「Remote Proxy を利用した分散 XQuery 問合せ処理」情報処理学会論文誌：データベース，Vol. 2, No. 1 (TOD41), pp. 104-115，情報処理学会, 2009 年 3 月
6) Viktor Leis, Peter A. Boncz, Alfons Kemper, Thomas Neumann:Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. SIGMOD Conference 2014: 743-754.
7) Joseph M. Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. The MADlib analytics library: or MAD skills, the SQL. Proc. VLDB Endow. 5, 12.
8) Makoto Yui and Isao Kojima. ``Hivemall: Hive scalable machine learning library'' (demo paper), NIPS 2013 Workshop on Machine Learning Open Source Software: Towards Open Workflows, Dec 2013.
9) TPC-H: http://www.tpc.org/tpch/
10) Thomas Neumann: Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB 4(9): 539-550 (2011)

ⓒ 2016 Information Processing Society of Japan