Vol.2016-OS-138 No.2
2016/8/8

並列データベースシステムにおける演算子間データ配送方式の検討

川島 英之† 建部 修見†

本論文では並列データベースシステムにおける演算子間データ配送方式の検討を行う．実体化方式とパイプライン方式を比較すると，パイプライン方式が安定的動作の為に必要であることを述べる．そして問合せから等結合演算子の系列が構築される場合にパイプライン方式を適用するには，(1)アルゴリズムとしては非分割並列ハッシュ結合あるいは並列整列併合結合を採用し，(2)スレッドはキャッシュミスを減らすべく協調的に動作することが望ましく，(3)オペレータ出力サイズは固定ではなく動的に調整することでメモリ不足によるスワップ発生やクエリキャンセルを回避することが望ましい，と考えられることを述べる．

1. はじめに

1.1 研究背景

複雑なデータ分析を行う際に一般的に採られる方式は，複数の演算子を接続する方式である．複雑な処理を行う 1 つの巨大な演算子を作成する方式に比べてこの方式は用途が広い．前者の方式はただ非常に限られた範囲の応用システム（ともすれば 1 つのみ）にしか適用できない．提供される巨大な演算子が提供するパラメータを変更することでしか，この方式の挙動は変えられない．従ってその挙動範囲が有益性と重なるアプリケーションのみが前者の方式の対象となる．一方，後者の方式は多様な演算子を準備してそれらの接続を変更することにより挙動を変える．アプリケーションの要求をシステムが満足できない場合，それに応える新しい演算子を導入し，それを既存の演算子と接合させることにより，アプリケーションの要求は満たされるようになる．前者の方式においては演算子自体を拡張しなければならないため，その維持と運用のコストが大きくなる．ここでは後者の方式をモジュール方式と記述する．

モジュール方式においては複数の演算子間でデータの授受を行う必要がある．ある演算子の出力結果を，それに続く演算子が受け取って処理をする必要がある．このデータ授受方式としては論理的には二つの方式が考えられてきた．第一の方式は実体化と呼ばれる．これは DAG(Directed Acyclic Graph)における各演算子を最下部から 1 つずつ動作させていく方式である．このとき，各演算子は入力データを全て処理する．第二の方式は反復方式と呼ばれる．反復方式は DAG の上部から処理が行われる．各演算子は一度に全てのデータを処理するのではなく，上部に要求されたサイズだけの結果を生成する．実体化方式を採用する有力なデータベースエンジンは Hyper の一実装 [14] である．

複数の等結合演算が実行される場合，この Hyper 実装は第一フェーズで各演算子におけるハッシュテーブルを作成する．次に第二フェーズにおいて全ての演算子の評価結果が真である場合のみ，結合結果を生成する．すなわち等結合演算子が N 個ある場合，第二フェーズは N 重ループを JIT (Just-In-Time) コンパイルにより構成し，ループの最深部において結果生成を行う．パイプライン方式に比べて関数呼び出し回数が極めて少なくなるため，この方式は高い性能が期待される．第一フェーズも第二フェーズもマルチスレッドにより並列処理がされる．第二フェーズでは 1 つの演算子を複数のスレッドで並列処理することで高性能化が達成される．

1.2 研究課題

上述のような実体化方式は高い性能を発揮する．その一方，実体化方式には 1 つの仮定があることに注意が必要である．それは演算子出力の結果がメモリに乗り切るという仮定である．通常の実体化方式ならば，ある演算の出力がメモリに収まることが仮定されているし，上記の Hype 実装のような多段の等結合演算を多段ループに変換する場合には, 複数演算の出力がメモリに収まることが仮定される．そもそも実体化方式が Volcano 以前に使われていた場合には，データ配送はファイルを用いることが前提とされていた．

近年の DRAM 価格の下落, ならびに一部のワークロードにおいて等結合演算の結合選択率が極めて低い事実を鑑みれば，そのような仮定はある程度成り立つと言えよう．しかしながら，その仮定が成り立たない場合も存在する．そのような場合には実体化方式ではなくパイプライン方式を使わざるを得なくなり，それが唯一の選択肢となる．それではパイプライン方式には近年のマルチコア CPU においてどのように設計すればよいのだろうか？これが本研究で検討する課題である．

1.3 貢献

これまで様々なパイプライン方式が研究されてきた一方，そのメモリ溢れについては検討がされてこなかった．本論文ではメモリ溢れを含めてパイプライン方式の設計指

† 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba

針を整理する．ただしパイプラインが利用可能な演算子は選択・結合・射影（重複除去を行わない場合）など，一部に限定される．本研究では実用上重要である結合演算系列のみを取り扱う．即ち本研究の貢献は並列データベースシステムの結合演算系列におけるパイプライン方式の設計指針の整理である．このとき，設計指針として考慮すべきは次の 3 つの要素である．

1. 等結合演算のアルゴリズム
   我々は以前の報告 [12] において，パイプライン処理を用いる場合，結合演算系列に適用可能な等結合演算のアルゴリズムは入れ子ループ結合のみであると述べた．しかしながらある程度のメモリを使うことを許容すれば，高速であるとして知られている並列ハッシュ結合あるいは並列整列併合結合を利用可能であることを示す．
2. スレッドとオペレータの関係
   近年数千コアの共有メモリマシンを前提としてデータベースマシンの研究は展開されている [11]. それほど多数のコアがある場合，多数のスレッドを同時に走らせることが可能になる．そのような場合，スレッドとオペレータはどのように割り当てればよいのか定かではない．パイプライン実行方式においては，各スレッドが独立して動作する場合と各スレッドが協調的に動作する方式に 2 種類が存在することを示す．
3. 各オペレータの出力サイズ
   Hyper の近年の研究においては多段の結合演算を多重ループに変換せず，結合演算がある程度のサイズの結果をまとめて出力する方式（Morsel 方式） [6] がある．Morsel 方式では固定サイズが必ず出力されるためにメモリ溢れが必ずしも発生する訳ではないが，それが発生した場合にはクエリがキャンセルされる為に性能が大幅に劣化する．本研究ではオペレータ出力サイズもパラメータになり得ることを示す．

1.4 論文構成

本論文の構成は以下の通りである．2 節では研究背景を述べる．3 節では関連研究を述べる．4 節では設計指針を述べる．最後に 5 節では本論文をまとめる．

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．

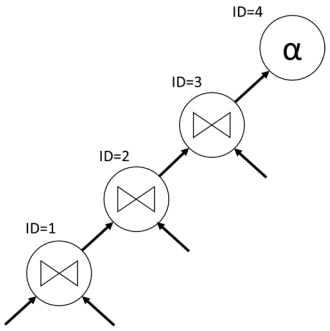

図 1　演算木

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．

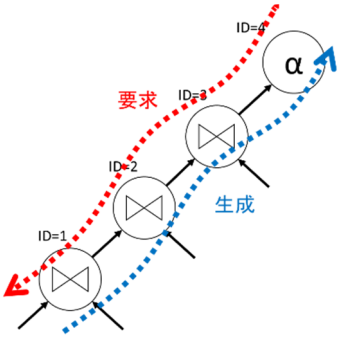

図 2 Volcano の要求・タプル生成過程

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，関数呼出回数の削減による性能改善の影響も考えられる．

3. 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．

3.1 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．

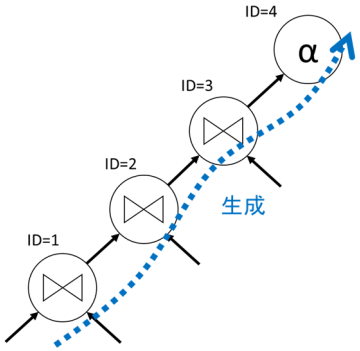

図 5 集合実体化方式

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．

3.2 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．

3.3 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．

4. 並列データベースシステムにおける演算子間データ配送方式の設計指針

本節では並列データベースシステムにおける演算子間データ配送方式の設計指針をまとめる．設計項目は 3 点からなる．

4.1 結合演算のアルゴリズム

我々は以前の報告 [12] において，パイプライン処理を用いる場合，結合演算系列に適用可能な等結合演算のアルゴリズムは入れ子ループ結合のみであると述べた．しかしながらある程度のメモリを使うことを許容すれば，高速であるとして知られている並列ハッシュ結合あるいは並列整列併合結合を利用可能である．

並列ハッシュ結合の場合, no-partitioning-hash-join (NPHJ) は適用可能だが，partitioning-hash-join (PHJ) は適用不能になる. なぜならば PHJ は入力テーブル R, S の両方についてハッシュテーブルを作成しなければならないからである．そのような方式では演算子を 1 つずつ処理していかざるをえない為，演算子間データ配送方式は実体化方式に限定され，パイプライン方式を適用できない．

一方，NPHJ では 2 つのフェーズを取ればパイプライン処理を実行可能である．第一フェーズでは，全ての結合演算子についてハッシュテーブルを作成する．そして第二フェーズにおいては複数のタプルを演算子が出力すれば良い．ここで全てのタプルを出力する必要はない．

等結合演算を効率的に処理可能であるとして名高いもう一つの方式である並列整列併合結合もパイプライン処理に適用可能である．並列整列併合結合の場合も NPHJ 同様に 2 つのフェーズを取れば良い．第一フェーズでは，全ての結合演算子についての整列を行う．N 件のデータの整列処理は通常は O(NlogN)が求められると考えられているが，近年の技術では N 件について O(N)で実行可能 [13] であるため，並列整列併合結合は第一フェーズにおいて NPHJ と性能的に比肩する技術だと考えられる．第二フェーズでは二分探索等を用いて照合処理を行う．ここで複数のタプルを演算子が出力することで，パイプライン方式を実現可能である．

4.2 スレッドとオペレータの関係

近年数千コアの共有メモリマシンを前提としてデータベースマシンの研究は展開されている [11]. それほど多数のコアがある場合，多数のスレッドを同時に走らせることが可能になる．そのような場合，スレッドとオペレータはどのように割り当てればよいのか定かではない．パイプライン実行方式においては，各スレッドが独立して動作する方式と各スレッドが協調的に動作する方式の 2 通りが存在する. これらの是非を考える際には CPU の構造を考慮する必要がある．CPU は複数ソケットから構成される．キャッシュミスを考えればあるソケットで処理されるデータは同一であることが好ましい．その観点からはスレッドは協調的に動作する方式が好ましいと考えられる．但し，スレッドの同期待ち時間は短いことが好ましいため，適切な負荷分散が求められる．

4.3 各オペレータの出力サイズ

Hyper の近年の研究においては多段の結合演算を多重ループに変換せず，結合演算がある程度のサイズの結果をまとめて出力する方式（Morsel 方式） [6] がある．Morsel 方式では固定サイズが必ず出力されるためにメモリ溢れが必ずしも発生する訳ではないが，それが発生した場合にはクエリがキャンセルされる為に性能が大幅に劣化する．本研究ではオペレータ出力サイズもパラメータになり得る．

Morsel 方式では 20 万タプル程度を出力単位としているが，場合によってはそれがメモリを食い尽くしてクエリがキャンセルされる場合もある. そのような事態を防ぐには，出力タプル数を動的に調整することが好ましい．動的調整の頻度が多いとそのコストが高くなり性能劣化を引き起こす可能性があるため，動的調整はある演算子を実行する直前に実行され，そこで出力サイズを決定することが好ましいと考えられる．出力数がそのサイズに満たない場合，あるはそのサイズに達した場合には，制御を上位へ戻す処理が行われる．

以上をまとめると，等結合演算子の系列が構築される場合にパイプライン方式を適用するには，(1)アルゴリズムとしては NPHJ あるいは並列整列併合結合を採用し，(2)スレッドはキャッシュミスを減らすべく協調的に動作することが望ましく，(3)オペレータ出力サイズは固定ではなく動的に調整することでメモリ不足によるスワップ発生やクエリキャンセルを回避することが望ましい，と考えられる．

5. まとめ

本論文では並列データベースシステムにおける演算子間データ配送方式の検討を行った．実体化方式とパイプライン方式を比較すると，パイプライン方式が安定的動作の為に必要であることを述べた．そして問合せから等結合演算子の系列が構築される場合にパイプライン方式を適用するには，(1)アルゴリズムとしては NPHJ あるいは並列整列併合結合を採用し，(2)スレッドはキャッシュミスを減らすべく協調的に動作することが望ましく，(3)オペレータ出力サイズは固定ではなく動的に調整することでメモリ不足によるスワップ発生やクエリキャンセルを回避することが望ましい，と考えられることを述べた．

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」，科研費基盤研究(C) (#16K00150)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)
11) Hideaki Kimura, FOEDUS: OLTP Engine for a Thousand Cores and NVRAM. SIGMOD Conference, pp. 691-706, 2015.
12) 川島 英之，建部 修見, "演算子間データ配送方式の検討"，情報処理学会第 137 回 OS 研究会報告 (OS137), volume 2016-OS-137
13) Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey. Fast Sort on CPUs and GPUs: A Case for Bandwidth Oblivious SIMD Sort. SIGMOD Conference, pp. 351–362, 2010.
14) Thomas Neumann, Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB 4(9): 539-550 (2011)
15)