Vol.2014-OS-130 No.20
2014/7/29

分析用データ処理系における効率的なデータ配送機構

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．本論文ではまず in-DB 分析システムに特徴的な問題として，メモリ枯渇問題と多ノードデータ配送問題があることを示す．メモリ枯渇問題に対応するために，演算子制御方式として，要求駆動・異量ブロック・並列方式を提案する．これは要求駆動方式の利点であるメモリ利用量の正確な推定を活かしながら集合実体化方式の利点である多量結果出力を実行する方式である．提案方式はメモリが少ない場合には固定ブロックを用いる既存の要求駆動方式よりも 15 倍程度性能が良いことが実験により示される．多ノードデータ配送問題については参照表に基づく方式を提案する．提案方式は参照カウンタ方式に比べて最大で 51%の性能を改善することが実験により示される．

Efficient Data Transfer Mechanisms on Analytic Data Processing System

Hideyuki Kawashima[†]　Osamu Tatebe[†,††]

Based on the In DB analytics scheme, a DBMS should provide a variety of analysis methods. Then, the DBMS confronts new two problems: shortage of memory and multiple tuple dissemination. To cope with the first problem, we propose demand-driven/different size block/parallel execution method for operator control. The proposed method leverages an advantage of demand-driven method that allows us to estimate the amount of memory utilization, and executes partial set materialization. We show that proposed method shows the best performance when memory amount is limited through an experiment. The improvement is by a factor of 15 compared with fixed-size demand driven method. To cope with the second problem, we propose a reference based data dissemination method with table. The proposed method shows by a factor of 1.51 performance improvement through an experiment.

1. はじめに

様々な分析的データ処理が DBMS において行われてきた．RDBMS における古典的な分析的ワークロードとしては TPC-H が知られており, 新しいワークロードとしてはデータマイニング・機械学習などである．これらのワークロードが DBMS において実行されるとき, 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 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 CEP だとする．NFA に基づく技法の場合, 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，シーケンスイベント集合がまとめて出力される可能性がある．例えば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，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可能性がある．あるいは分析的データ処理ベンチマークである TPC-H における結合演算の選択率が直積に近い程高い場合に，結合演算は膨大量の結果を出力することが考えられる．

第二の特徴は，異なる複数の分析的演算子が入力を共有することがあり得る点である．例えばパケットデータからの知識発見・マルウェア検知を行う場合，パケットデータに対するスキャン演算子からの出力を，様々なデータあみマイニング演算子が読み込む可能性がある．

これらの特徴に際して顕在化する問題について考える．第一の特徴における問題点はメモリ溢れである．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からタプルを pull するスタイルであり，open-next-close イテレータ処理により実現される．ただし，近年性能向上のために push スタイル, すなわち子演算子が親演算子にデータを配送する方式も提案されており，そのような処理系では producer-consumer 処理によりデータ配送が実現される. さて push スタイルを分析的データ処理系においてナイーブに採用すると，演算子出力が大量であるためにメモリが枯渇する可能性がある．このときシステムはストールするかメモリ外部へデータを待避するかいずれかの方策を取る必要がある．

第二の特徴に際して顕在化する問題点はタプル複製コストである．子演算子は多数の親演算子に対してタプルを

† 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
†† 科学技術振興機構/JST CREST

配送する必要があるが，open-next-close スタイルにおいても producer-consumer スタイルにおいても，キューを用いてデータ配送を行うとき，N 個の親演算子が存在する場合にはタプルを N 回複製する必要がある．

本論文ではこれらの問題を解決する方式を述べる．まず，第一の特徴における問題を解決するために，我々は新しい演算子制御機構を提案する．同機構はメモリ枯渇を防ぐために，演算子の出力量を制御する．要求駆動方式を基本としながら高性能化を実現するためにブロック出力（即ち tuples-at-a-time or partial set-at-a time）と演算子内並列処理を導入する．出力ブロックを最大化しながらもメモリ枯渇を回避するため，上部ノードから下部ノードへと出力率（選択率）を用いて各ノードの出力量を計算する．

第二の特徴における問題を解決するために，我々は参照に基づくデータ授受機構を提案する．提案機構は参照表を利用することでラッチを取得することなく，データ授受を可能にする．

本論文の構成は次の通りである．2 節では研究の動機として in-DB 分析方式と，その方式ならではの課題を述べる．3 節では第一の問題であるメモリ枯渇に対処するために，演算子出力制御方式を述べる．4 節では多親ノードへの配送問題を解決するために，参照方式について述べる．最後に 5 節では本論文をまとめる．

2. 研究の動機

2.1 In-DB 分析アプローチ

分析的データ処理系には STORM のように処理内容を自由な言語で自由に記述できる方式と，DBMS，DSMS，そして Hive のように SQL ライクな問合せによりリレーショナルデータ処理を実行する 2 方式が存在する本論文では後者のリレーショナル処理を用いる方式を徹底し，種々の事象検出を行う為の分析演算子を DBMS に組み込み，分析機能をデータ基盤に組み込むシステムを考えるこれは DBMS 内部で分析等を行う方式だと表現できるため，これは一般的に in-DB 分析システムと呼ばれる [7]. In-DB 分析システムを図 1 に示す同図では分析・高性能化・安心化を all-in-one で提供することがわかる．

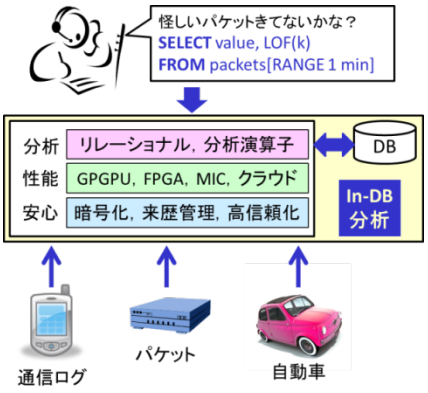

図 1　In-DB 分析システム

演算子組込み方式の利点は，利便性の向上，性能の向上，そして説明能力の向上である In-DB 分析方式により，性能を向上させられる．なぜならば演算子を DSMS 内部で分散化，差分計算処理，そして H/W の利用が可能になるからである．In-DB 分析方式により，説明能力を向上させられる．なぜならば入力データに対して行われたあらゆる演算子の処理内容を表面的なものに留まらず，その実現詳細についてまで提示可能になるからである．

演算子を組み込まないシステムにおいては，上記の利点が存在しなくなる．すなわち分析手法はユーザにより UDF を用いて作成されなければならず，UDF で記述されたプログラムは高性能を発揮することができず，さらに UDF 内部はシステムからはブラックボックスになるので説明能力は低下する．SQL MapReduce のように UDF において何らかの並列化支援を実現することでこの問題は多少の解決を見るが，並列化コードを書くという労苦は残る．

我々はリレーショナル演算子を軸として各種マイニング演算子を有し，かつ FPGA との連携を可能にする in-DB 分析システム Falcon を開発している．Falcon における研究開発を進める傍ら，下記に述べる問題が顕在化してきた．

2.2 In-DB 分析システム特有の問題点

In-DB 分析システムにおいては基本的に分析的データ処理が行われる．そのような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，そこには古典的な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，1 つのイベント到着に伴い確率伝播が行われ，条件を満たす最大数百万のノードが演算子から出力される可能性がある．しかも入力データが頻繁に到着する，あるいは大量にある場合，それらの出力がメモリを早晩枯渇させることは自明であろう．あるいは分析的データ処理ベンチマークとして知られている TPC-H における結合演算の選択率が直積に近い程高い場合が考えられる．TPC-H の Q3 や Q19 では複数の表を結合する．通常は選択率が低く表の結果が膨大になることは考えないが，分析的データ処理においてはそのような事態がしばしば発生する．

第二の特徴は，異なる複数の分析的演算子が入力を共有することがあり得る点である．例えばパケットデータからの知識発見・マルウェア検知を行う場合を考える．このときパケットデータに対するスキャン演算子からの出力を，様々なデータマイニング演算子が読み込む必要がある．なぜならばマルウェア検知を行うには多様な手法を同時に用いる必要があるからである．しかもデータマイニング演算子は往々にしてパラメータを持つ．そしてパラメータによって検知率が変動する．そのため，1 つのマイニング演算

子について複数の具体的演算子を稼働させることが好ましい．以下では in-DB 分析システム特有の問題について具体的に述べていく．

2.3 第 1 の問題：メモリ枯渇

第一の問題点はメモリの枯渇である．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からタプルを pull するスタイルであり，open-next-close イテレータ処理により実現される．下から上へ噴火するが如くであるために volcano の名が付けられている．しかしながら volcano は性能が低い点が近年指摘されており，性能向上のために volcano と逆である push スタイル，すなわち子演算子が親演算子にデータを配送する方式が提案されている [4, 9, 10]. このような処理は producer-consumer 処理により実現される. さて push スタイルを分析的データ処理系においてナイーブに採用すると，演算子出力が大量であるためにメモリが不足する可能性がある．このときシステムはストールするかメモリ外部へデータを待避するかいずれかの方策を取る必要がある．いずれにしても性能劣化を免れることはできない．

2.4 第 2 の問題：タプル複製コスト

第二の問題点はタプル複製コストである．分析的データ処理系においては，データマイニング演算子の存在のために，子演算子は多数の親演算子に対してタプルを配送する必要がある．要求駆動方式，実体化，供給駆動方式のいずれにおいても，キューを介してデータ転送をする場合，N 個の親演算子が存在する場合には，ある演算子はタプルを N 回複製し，各複製を各親演算子に渡す必要がある．N が小さい場合には問題がないが，N が数万件程度であり，かつ入力データがパケットストリームの如く高速にシステムへ流入するような場合には，このデータ配送は性能上のボトルネックとなる．

3. メモリ枯渇の対処：演算子出力制御

3.1 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，分析的データ処理ベンチマークである TPC-H の Q3/Q19 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 2 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．また，下記の結合演算子においては内表がメモリに全て載ることにする．また，結合演算アルゴリズムとしては入れ子ループ結合を用いる．また，いずれの結合演算も直積同様の選択率を有することにする．以降では方式の説明ならびに実験において，図 2 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．

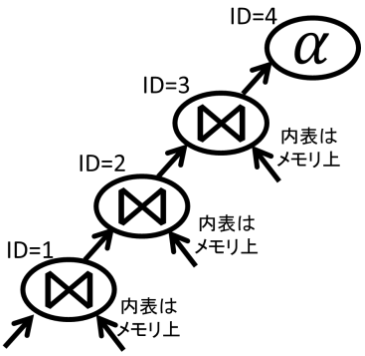

図 2　演算木

3.2 関連研究

3.2.1 要求駆動：Volcano

既に述べてきたように 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 3 に示す.

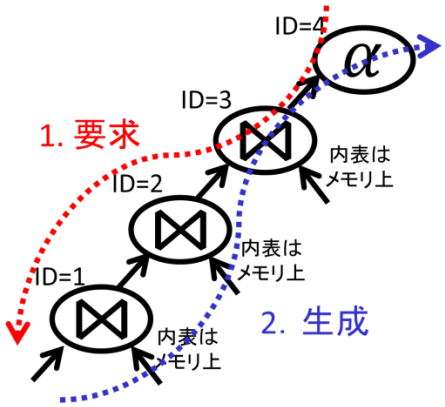

図 3 Volcano の要求・タプル生成過程

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，それが高価になり得る事である．

3.2.2 要求駆動・ブロック

3.2.1 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 [11]. 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, 12]．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を目的としていた[12]．BDQ で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 [8].

この方式の長所は単純な要求駆動方式（Volcano）に比べて性能が極めて高いこと，ならびにメモリ使用量が一定量であることである．一方，欠点としては並列度がそれほど高くならない点，ならびに関数呼び出しが多く性能劣化を招く点である．

3.2.3 要求駆動・ブロック・並列

3.2.2 節で述べた要求駆動・ブロック方式を並列化することができる．ルートノードから駆動が開始される要求処理を複数のスレッドで実行すれば良い．図 2 に示される演算木の場合，ルートノードにおける集約演算子が 3 段の結合演算の結果を集約することになる．リレーショナルデータベースにおけるタプル独立性を利用することにより，選択・射影・結合などの演算については並列化が自明である．

3.2.4 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 4 に示す．

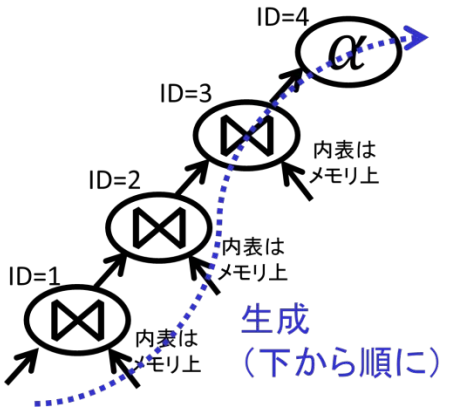

図 4 集合実体化方式

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．

3.2.5 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 4 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．

3.2.6 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 [4]．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．

本論文の実装においては 1 つの演算子に 1 つのスレッドを割り当てる．Morsel 駆動方式では複数演算子を 1 つのスレッドが担当するため，本論文の方式は morsel 方式よりも非効率的になり得る点を指摘しておく．また，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．本論文の実装形態を図 5 に示す．

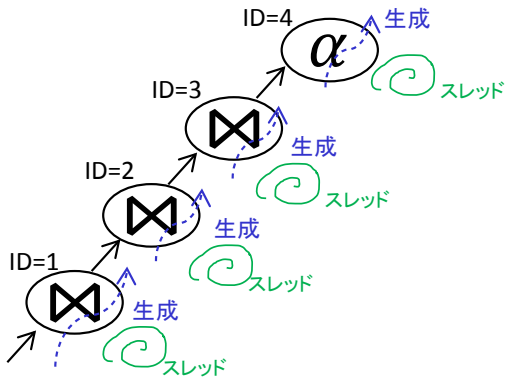

図 5 供給駆動方式

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．一方その長所は実体化方式同様にメモリである．高性能な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 [4]の 3.2 節に述べられているなど，メモリ枯渇については余り考えられてない．

3.2.7 供給駆動・ブロック

3.2.6 節で述べた供給駆動方式において演算子の出力を 1 タプルからブロックに変更したものがこの方式である．

3.2.8 供給駆動・ブロック・並列

3.2.7 節の方式を複数スレッドで並列処理する方式がこの方式である．これは morsel 駆動方式 [4]とも言える．

3.3 従来研究の問題点

分析的データ処理系においては 1 節で述べたように，ある演算子が大量の出力を行う．古典的なデータ処理系においては演算子が非常に多くの出力をする現象はさほどには考えられていない．例えば近年開発されている高性能データ処理系である HAIL [5] や PGBooster [6] においては実験データの選択率は 1%以下である．

逆に，本論文が対象とする in-DB 分析システムにおいては，入力データを数十倍～数千倍に増幅する演算子が多数使われることを想定している．このような場合，従来方式である，要求駆動，実体化，供給駆動のうち，要求駆動のみが適切に動作する．実体化方式は枯渇するまでメモリを使用し，場合によっては仮想記憶を呼び出してストレージアクセスを暗黙的かつ非効率的に行う可能性がある．

供給駆動方式は残存メモリを制御しながら結果を出力することが困難であると筆者は考える．現状の供給駆動方式は複数のスレッドが様々な演算子をメモリ残存量も考慮せずに走らせる．メモリ残存量を考慮させるにはメモリに関する大域的ロック，あるいは事前メモリ割当などの措置が必要になると考えられるが，それによる性能劣化を避けることは容易ではないように思えるからである．

3.4 提案：要求駆動・異量ブロック・並列方式

メモリ枯渇問題に対処するために，筆者は演算子出力量を制御する方式を提案する．提案方式は 3.2.3 節で述べた要求駆動・ブロック・並列方式を拡張するものである．3 節で「ブロック」の名のついた方式はいずれも*同一サイズ*で*小規模*なタプル集合を演算子がメモリ枯渇を無視して出力することを意味してきた．すなわち，「ブロック」を正確に表現すれば「同量ブロック」となり，これを換言すれば “memory-*unaware* *same*-size tuples-at-a-time” 方式と表現できる．提案方式はこれを修正して，演算子の出力量を演算子ごとに変え，「異量ブロック」とする．換言すれば “memory-*aware* *different*-size tuples-at-a-time” 方式と表現できる．

提案方式を図 6 に示す．並列方式が用いられるために複数スレッドが動作する．1 つのスレッドは 1 つのイテレータ処理を行う．結合処理アルゴリズムは入れ子結合であるため，領域分割を行うことで並列処理を行う．スレッド毎にキューを持たせることでキュー操作はロック不要とする．各演算子は事前算出された領域量に見合うタプル集合をまとめて出力する．即ち部分集合実体化を意味する．図 6 における領域量の事前算出は次式により行う．ID=4 の入力量を k とすると，ID=3 の入力量は結合演算が直積同様となることから内表のサイズをrとすれば，それは k/rとなる．同様に ID=2 の入力量は k/r^2 となる．あるイテレータスレッドの利用可能メモリ量を M とすれば $M = k + k/r + k/r^2$が成り立つ．これをkについて解けば ID=4 の入力量が求まり，次いで他演算子の入力量も求まる．ただし入力量は少なくとも 1 になる．領域量算出の一般化については稿を改めて述べる．

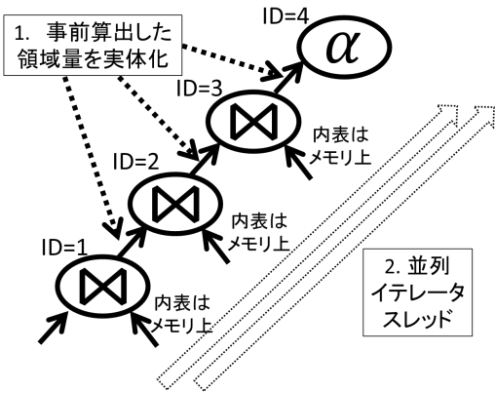

図 6 提案方式：要求駆動・異量ブロック・並列

提案方式の長所はメモリ枯渇問題を回避しながら，要求駆動方式におけるメソッド呼出回数を削減して高性能化を図っている点にある．一方短所はメモリが十分存在する環境においては集合実体化・並列方式よりも性能が劣る点である．

3.5 評価

提案方式の性能を評価するために図 2 に示される演算木に対して 3 節に述べた手法の一部を実装した．実装言語には C++，そしてコンパイラには gcc-447 を用いた．コンパイラの最適化オプションは O2 とした．実装環境は 8 コアのノードを用いた. CPU は Xeon E5620, 240GHz を用いた．スレッドにはアフィニティを設定した．表の属性は 4 バイトの 1 属性のみとした．内表のサイズはそれぞれ 120 とした．結合率は直積同様であり，結合演算は 3 つ存在するため，集約演算（カウント演算）が受けとるタプル数は $120^4 = 207360000$となる．

3.5.1 実験結果

メモリサイズに問題がない場合の結果を図 7 に示す．要求駆動方式，集合実体化方式，供給駆動方式に関する結果を示している．メモリサイズは 1 GB と設定した．縦軸は実行時間（対数軸）であり，短い程好ましい．括弧内の数字はブロックサイズあるいはスレッド数を表している．

結果より，集合実体化・並列方式が最も良い性能を示していることがわかる．一方，供給駆動方式の性能が悪いことがわかる．この理由は未調査であるが，キューの排他制御が問題である可能性は捨てきれない．

提案手法である要求駆動・異量ブロック・並列方式は，集合実体化（非並列）とほぼ同程度の性能を示している一方，他の要求駆動方式よりも高い性能を示している．提案手法は要求駆動・ブロック・並列方式より 15 倍程度の高性能を達成している．この結果より，本実験の条件においてメモリサイズに余裕がある場合には集合実体化方式が最も優れることがわかった．ただし同量ブロック方式においてブロックサイズを大きくした際の性能を検証することが今後望まれる．

一方，メモリサイズを 100 MB に制限した場合，集合実体化方式，集合実体化・並列方式，そして供給駆動方式はいずれもメモリ不足のために動作を停止し，動作を完了することはできなかった．これを図 8 に示す．この理由は本論文の実装では外部記憶へのデータ待避機能を実装しなかったからである．近年の高性能外部記憶(Fusion I/O 等)を利活用することで，待避問題は大幅に軽減される可能性はあるが，本稿ではその可能性までは調査していない．一方，要求駆動方式群はいずれも動作を正常に完了し，図 7 に示される値と同様の値を示した．

以上より，メモリサイズが不足する場合には提案方式が最も優れること，ならびに集合実体化方式群・供給駆動方式は外部記憶へのデータ待避が求められることが示された．

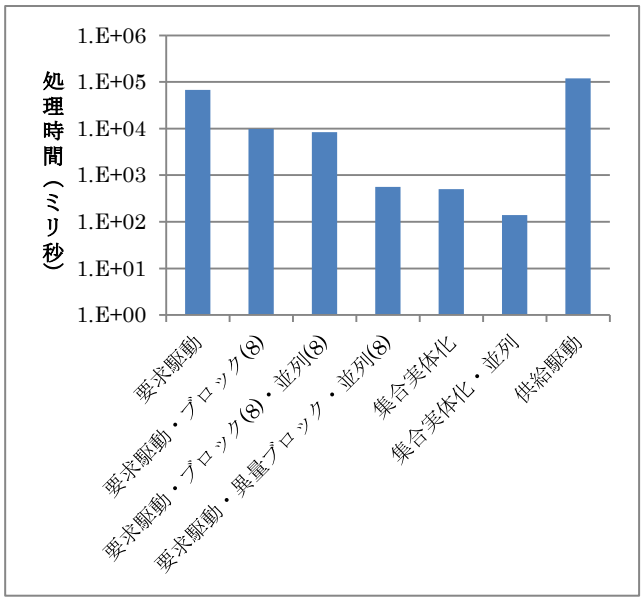

図 7 演算子処理の実験結果（メモリが枯渇しない場合）

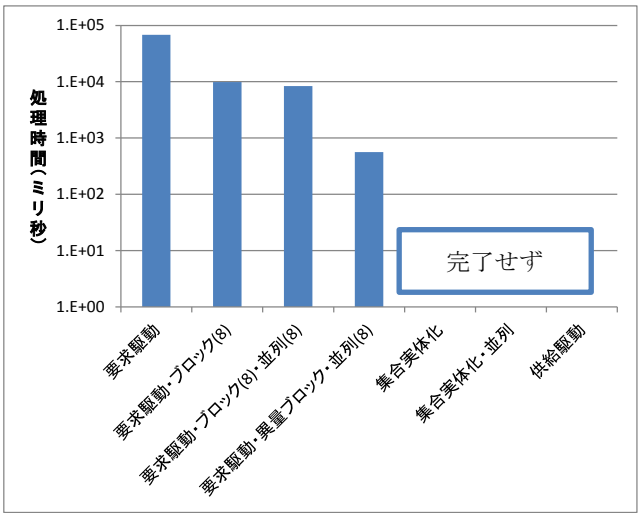

図 8 演算子処理の実験結果（メモリが枯渇する場合）

4. 多複製への対処：参照方式

4.1 研究課題

既に 2 節で述べたように，in-DB 分析システムではタプル複製コストが問題になり得る．なぜならば子演算子は多数の親演算子に対してタプルを配送する必要があるが，要求駆動方式，実体化，供給駆動方式のいずれにおいてもキューを使う限りは，N 個の親演算子が存在する場合には，ある演算子はタプルを N 回複製し，各複製を各親演算子に渡す必要があるからである．この例を図 9 に示す．図 9 においてはシステム管理者がマルウェア検知を in-DB 分析システムで行う状況を描いている．in-DB 分析システムに到着したパケットに対して N 件のマルウェア捕獲処理器（いわゆる機械学習など）が用意されている場合，到着パケットは N 件全ての捕獲処理器へ転送される必要がある．N が

小さい時にはさほど問題にならないが，大きい場合には性能を下げる要因となりうる．

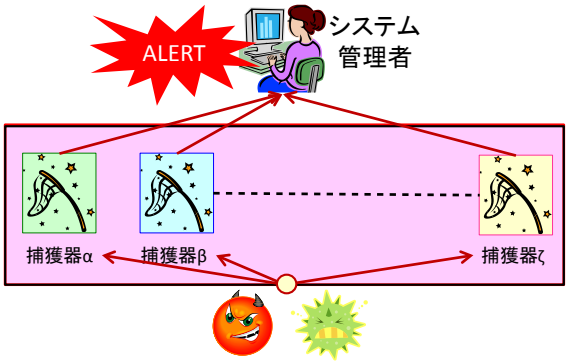

図 9　複数タプル配送の必要性

4.2 提案：参照方式

この問題に対処するため，親演算子（捕獲処理器）に入力キューを持たせることをやめることを提案する．代わりに子演算子が出力キューを持ち，親演算子が子演算子の出力キューの中身を参照することによりデータ配送を行う．これを図 6 に示す．提案方式ではメモリ確保ならびにキュー挿入処理が不要になる．

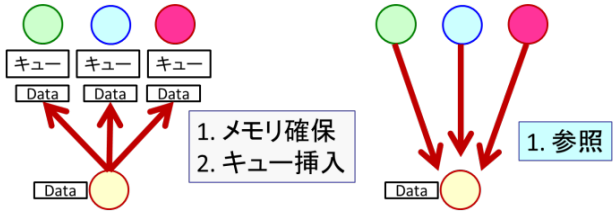

図 10 従来方式（左）と参照方式（右）

参照に関するナイーブな方式はN個の親ノードは未知データを読むと参照カウンタを 1 つ上げる方式である．参照カウントがNとなったデータについては子ノードが削除を行う．これにより従来行われていたメモリ確保処理が不要になるため，性能とメモリ利用率の両面で性能向上が実現される．

参照処理に際して参照カウンタを用いるナイーブな方式では，子ノードの出力キューを排他制御する必要がある．この性能を高めるために本論文では参照表を用意する．N 個の親ノードがあり，M 件のデータがある時，参照配列のサイズは NM となる．参照表方式ではカウンタをインクリメントする代わりに該当部分にフラグを立てる．子ノードは全ての親ノードがフラグを立てたデータについて削除処理を行う．

4.3 評価

提案機構を評価するために C++言語により実装を行った．コンパイラには gcc を用い，最適化オプションは O2 とした．CPU として Xeon X5570 (293GHz)を用いた．ノード上のコア数は 16 であり，10 件のスレッドを用いた．子ノードの数は 1 である．

実験結果を図 11 に示す．縦軸は処理時間（対数軸）であり，小さい程好ましい．親演算子数は 100，10000 の二つの場合についての実験結果を示す．参照カウンタ方式としては排他制御方法として mutex と CAS を用いる実装を行った．参照表方式は親演算子数が 10000 のときに参照カウンタ(CAS)に比べて 51%程度の性能改善，配送方式に比べて 230 倍程度の性能改善を示した．性能から見れば配送方式は問題外であるが，高信頼性を考えたときには，多数の複数実体が存在することから配送方式が明らかに優れる点を指摘しておく．

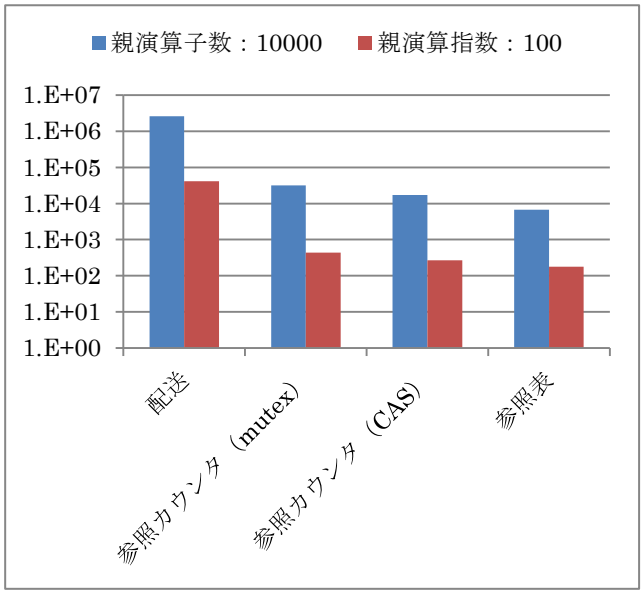

図 11 参照方式の評価

5. まとめ

本論文では in-DB 分析システムに特徴的な問題として，メモリ枯渇問題ならびに多ノードデータ配送問題があることを述べた．メモリ枯渇問題に対しては要求駆動・異量ブロック・並列方式を提案し，集合実体化方式が使えない程メモリが少ない時には最も性能が良いことを示した．多ノードデータ配送問題については二つの参照方式を提案した．参照表方式は参照カウンタ(CAS)に比べて最大で 51%の性能改善を示した．

今後の課題は領域計算アルゴリズムの確立，集合実体化に際する高性能ストレージの利用，そして我々が開発している in-DB 分析システム Falcon において提案方式を検証することである．

謝辞 本研究の一部は科研費#24500106 および JST CREST による．ここに記して謝意を表す.

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) Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K Sjoerd Mullender, Martin L Kersten: MonetDB: Two Decades of Research in Column-oriented Database Architectures IEEE Data Eng Bull 35(1): 40-45 (2012)
4) 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
5) 早水 悠登, 合田 和生, 喜連川 優,「アウトオブオーダ型クエリ実行に基づくプラグイン型データベースエンジン加速機構」, WebDB Forum 2013 年 12 月.
6) Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Stefan Richter, Stefan Schuh, Alekh Jindal, Jörg Schad: Only Aggressive Elephants are Fast Elephants PVLDB 5(11): 1591-1602 (2012)
7) Joseph M Hellerstein, et, alChristoper Re, 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," VLDB Endow 5, 12, 1700-1711
8) 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.
9) Thomas Neumann: Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB 4(9): 539-550 (2011)
10) Yannis Klonatos, Christoph Koch, Tiark Rompf, Hassan Chafi: Building Efficient Query Engines in a High-Level Language. PVLDB 7(10): 853-864 (2014)
11) Sriram Padmanabhan, Timothy Malkemus, Ramesh C. Agarwal, Anant Jhingran: Block Oriented Processing of Relational Database Operations in Modern Computer Architectures. ICDE 2001: 567-574
12) 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.