データ仮想化における効率的なクエリ処理の実装と評価

米田 信之† 米川 慧† 齋藤 和広† 渡辺 泰之† 村松 茂樹† 小林 亜令†

†株式会社 KDDI 研究所 〒102-8460 東京都千代田区飯田橋 3-10-10 ガーデンエアタワー

E-mail: †{no-maita, ke-yonekawa, ku-saitou, yasuyuki, mura, kobayasi}@kddilabs.jp

あらまし 近年，企業におけるデータ分析の普及に伴い，更なる競争力向上のためのデータ統合が注目されている．データ統合技術の 1 つにデータ仮想化がある．これは複数のデータソースを仮想統合し，単一データソースであるかのようなアクセス性を提供する技術である．データ仮想化におけるクエリ処理は，データソースから中間データを抽出し，データ仮想化システム上で結合処理を実現する．しかしクエリによっては，中間データの肥大化する処理や，抽出データ量を削減できない処理が発生し，クエリ処理の著しい遅延や異常終了を引き起こす問題がある．そこで筆者らは，クエリ分解と中間結果再配置により，中間データを削減するクエリ処理手法を考案し，オープンソースソフトウェアの Presto に実装した．また，TPC-H ベンチマーク を用いた実験により，本手法の有効性を評価する．

キーワード データ仮想化，データベース，問合せ処理，Presto

1. はじめに

近年，企業におけるビッグデータ分析の普及に伴い，更なる競争力向上のためのデータ統合が注目されている．データ分析においては，1 つのシステムに閉じた単一データベースのみならず，企業内の様々なデータを組み合わせた分析をすることで新たな価値や知見を生むことが多い．その一方，データ分析を取り巻くシステム環境は，組織における部門や事業ごとにサイロ化されていることが多く，データ分析に取り掛かるまでのプロセスに多くの時間とコストが費やされている．したがって，組織内のデータ統合を実現することは，データ分析サイクルの迅速化ならびにコスト削減を可能とし，延いては企業の競争力向上に繋がるものと考える．

しかしながら，組織内のデータベースを 1 つに統合することは容易ではない．これは，データベースの収容変更に伴う既存アプリケーションの改修や，データ移行，関連業務システムへの影響が発生するためである．実現するためには，膨大な時間とコストが掛かることが想像できる．

そこで筆者らは，データ仮想化[1]（図 1）によるデータ統合の実現に着目している．これは，複数のデータソース（DS）を仮想統合し，あたかも単一のデータソースであるかのようなアクセス性を提供可能とする，データ統合技術である．データ仮想化によるデータ統合が実現すると，データベース利用者は，データ仮想化システム（DVS）に対して SQL 等のクエリを投稿することで透過的にデータアクセスが可能となる．この際，データベースシステム（DBS）のレイアウトや，固有のインタフェース，事前のデータ移行を意識する必要はない．そして，新たな DS が構築される場合には，ネットワークを介した DVS との接続により，データ統合環境に加えることができる．

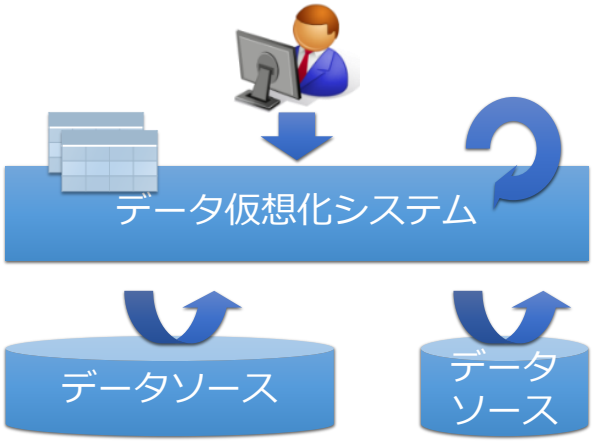

図 1 データ仮想化概要

しかしながら，データ仮想化におけるクエリ処理では，中間データが肥大化する処理や，抽出データ量を削減できない処理が発生し，著しい処理遅延を招くことがある．これらは，DVS の生成する DBS 向けのサブクエリが非効率であるため発生する．この問題は，昨今構築されているようなビッグデータを有する DBS が対象となる場合，影響が顕在化する．

そこで本研究では，データ仮想化におけるクエリ処理について，クエリ分解および中間データ再配置を用いる効率化手法を提案する．また，提案手法をオープンソースソフトウェアの Presto[2]に実装し，TPC-H[3] ベンチマークを用いて提案手法の有効性を評価する．

本論文の構成は以下の通りである．2 節で DVS におけるクエリ処理概要と課題について述べ，3 節で課題解決するためのクエリ処理手法を提案する．4 節では，提案手法を適用したプロトタイプの実装について述べ，5 節で評価実験，6 節で本研究を通じての考察，最後に 7 節でまとめと今後の課題を示す．

2. データ仮想化における課題

2.1. クエリ処理概要

データ仮想化は，複数の DS を仮想統合し，あたかも単一 DS であるかのようなアクセス性を提供する技術である．DVS が対象とする DS は，SQL をインタフェースとする DBS や CSV などのファイル，Web コンテンツなど様々である．なお本論文では，DS が DBS であるものとして議論を進める．

DVS におけるクエリ処理は図 2 のような処理フローとなる．DVS と DBS は，ネットワーク接続されている．DVS は，各々DBS が保持するデータベースを把握するためのスキーマ情報を持っている．これを仮想スキーマと呼ぶ．ユーザは，仮想スキーマに基づき，SQL などを用いたクエリを作成・投稿することができる．DVS は，ユーザから要求されたクエリを取得すると，クエリの構文解析，実行計画生成，最適化を施す．ここで生成された実行計画に基づき，クエリ処理に必要となるデータを，各 DBS より随時取得する．このとき DVS から DBS へ投稿されるクエリを，サブクエリと呼ぶ．サブクエリは，DBS のスキーマや，SQL などのインタフェース形式に応じて，動的に生成される．サブクエリ結果の取得後，DVS にて結合，集計，整列演算が実行され，最終的なクエリ結果をユーザに返却する．DVS における実行計画は，これらサブクエリによるデータ取得から，DVS 上での演算処理を含めたものとなる．

2.2. 基礎実験

データ仮想化技術の現状を把握するため，既存データ仮想化製品や技術についての調査を実施した．本調査では，MIND[4], Presto, Teiid[5]について，複数の DBS によるデータ仮想化環境を構築し，TPC-H ベンチマークを用いて，それぞれの振る舞いを確認した．

調査の結果，いずれの対象についてもクエリ処理が著しく遅くなる，あるいは異常終了するケースが確認された．これらのクエリ処理について，実行計画を詳細に確認したところ，DBS から DVS に転送されるサブクエリ結果が膨大となっていることが分かった．これにより，データ転送処理や DVS におけるサブクエリ結果を用いたクエリ処理に時間を要することとなり，著しい処理遅延を引き起こしたと考えられる．また，サブクエリ結果が大きすぎる場合には，DVS にて異常終了することも確認された．

このことから，DVS におけるクエリ処理の効率化を議論する上では，DVS にて生成される，サブクエリのデータ抽出量が重要になるものと考える．

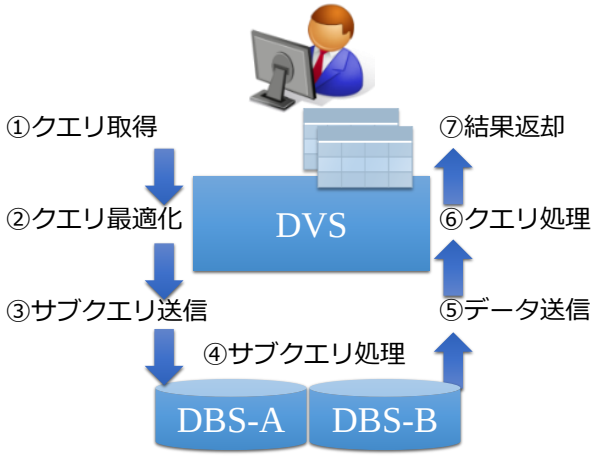

図 2　DVS におけるクエリ処理の流れ

2.3. 既存 DVS のクエリ処理方式

2.3.1. 単一 DBS 処理

調査対象の DVS について，サブクエリ生成の仕組みを調査した．

MIND の手法では，サブクエリに直積が混入することがある．これは，ユーザが投稿した SQL クエリについて，スキーマ定義情報を参照し，DBS との対応付けのみでサブクエリを生成することに起因する．これにより，同一 DBS であれば，結合条件が存在しないにも関わらず，サブクエリの FROM 句に，複数のテーブル名が定義される事象が発生する．結果として，サブクエリに直積や多対多結合が混入し，結果サイズの肥大化を招くことがある．

Presto では，データ取得対象となるテーブルごとに．サブクエリが生成される仕組みとなっている．Presto は，投稿されたクエリの実行計画生成において，Left-deep Tree を構築する．この Left-deep Tree におけるスキャン，範囲選択および射影演算までをプッシュダウンし，サブクエリとして生成している．結果として，単一テーブルに閉じたサブクエリが生成されるため，直積や多対多結合演算は混入しない．しかしながら，結合による絞り込みの恩恵が得られず，抽出件数も多くなる傾向にある．もし範囲選択がされない場合には，対象カラム全件を取得することとなる．

Teiid では，Bushy Tree による結合演算を含めたプッシュダウンを実現しているものの，多対多結合演算を排除できていない．Presto 同様に Left-deep Tree を生成後，クエリにおける結合条件とスキーマ定義を参照し，単一 DBS に閉じた結合演算について探索する．発見された場合には，実行計画を Bushy Tree へと更新し，結合演算をプッシュダウンする．しかしながら，多対多結合の場合にもプッシュダウンされるため，非効率となる場合がある．

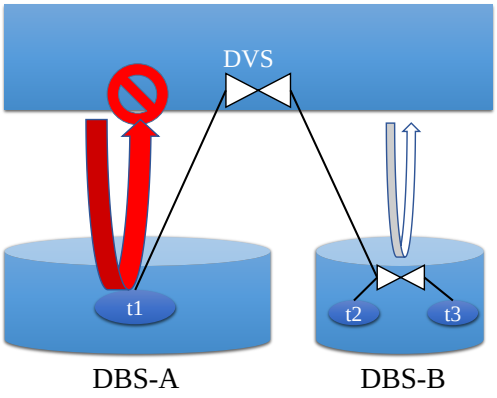

図 3　絞り込み不十分による影響

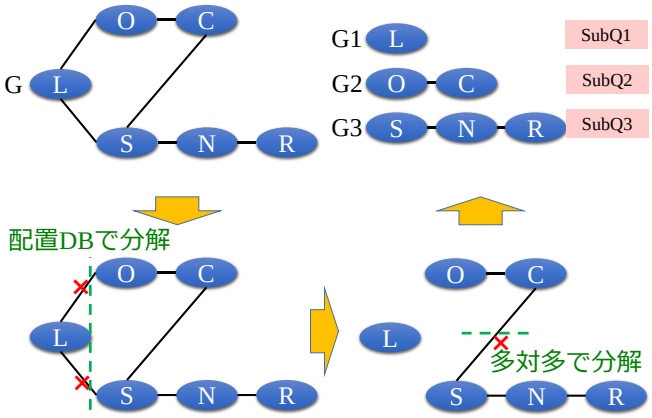

図 4　クエリ分解手法

2.3.2. 複数 DBS 間の処理

調査対象としたシステムについて，MIND と Presto では，DVS から DBS へサブクエリが投稿され，データを取得し，DVS 上で結合処理するアーキテクチャである．範囲選択や射影演算のプッシュダウンにとどまると，DBS から DVS へ転送せざるを得ない．例えば図 3 において，t1 が巨大なテーブルであった場合，大規模なデータ転送が発生しうる．結果として，転送による著しい遅延，あるいは DVS のメモリ不足による処理遅延や異常終了を引き起こす．片山ら[6]も課題として挙げているように，複数の DS を跨る場合の効率化が必要と考える．

Teiid では，Dependent Join と呼ばれる仕組みにより工夫されているものの，その効果は限定的である．1 つは Key Pushdown であり，先行するサブクエリで得た結果を，後続サブクエリの IN 句にキー情報として挿入する方法である．もう 1 つは Full Pushdown であり，前述した先行サブクエリの結果を，後続サブクエリが実行される DBS に，テンポラリテーブルとして書き込む方法である．前者は Query Injection，後者は Ship Join とも呼ばれる[1]．どちらも，後続サブクエリの抽出データ量を絞り込むことに有効であるが，Key Pushdown は IN 句への挿入件数が多い場合，却って DBS 側の処理が著しく遅延することがあり，大規模なデータ処理には不向きと考えられる．Full Pushdown は，前者と比較して効率的であるものの，ユーザ自身がクエリ内に記述することを必要としている．効果的に使いこなすためには，ユーザ自身が DBS の物理レイアウトやデータに対する知識を持たなければならない．

2.4. 既存 DVS におけるクエリ処理課題

以上 2 節での議論から，DVS におけるクエリ処理効率に関する課題は次のとおりである．

課題 1：直積や多対多を避けること

課題 2：サブクエリによる抽出データ量を削減すること

3. 提案方式

前節で述べた課題を解決する手法として，クエリ分解と中間データ再配置の 2 つを述べる．なお，中間データ再配置手法については，クエリ分解手法を前提としたものとなっている．

本節では，TPC-H ベンチマークにおけるクエリを題材として，手法について述べる．また DBS が 2 つあるものとし，一方にテーブル Lineitem，他方にそのほかのテーブルが配置されているものと仮定する．

3.1. クエリ分解手法

課題 1 を解決する手法として，クエリグラフ[7]の分解によりサブクエリ生成単位を決定する（図 4）．ここでは，一対一もしくは一対多の関係のみで連結したクエリグラフを求め，これらをサブクエリの最小単位とすることで，直積や多対多結合を回避する．

まず，ユーザにより投稿されたクエリについて，クエリグラフに変換する．クエリグラフは，ノードがテーブル，ノード間エッジがテーブル間結合を表している．得られたクエリグラフに対して，次の基準によりグラフ分解操作を実施する．

基準 1：DBS が物理的に異なるエッジ

基準 2：多対多条件となるエッジ

この操作の結果，分解された 1 つ以上のクエリグラフが得られる．分解されたクエリグラフをもととして，サブクエリを生成することができる．分解操作時に取り除いたエッジ（結合演算）については，後述する 3.2 節の手法により，処理方法を決定する．すべての結合後に，集計，整列演算を処理する．

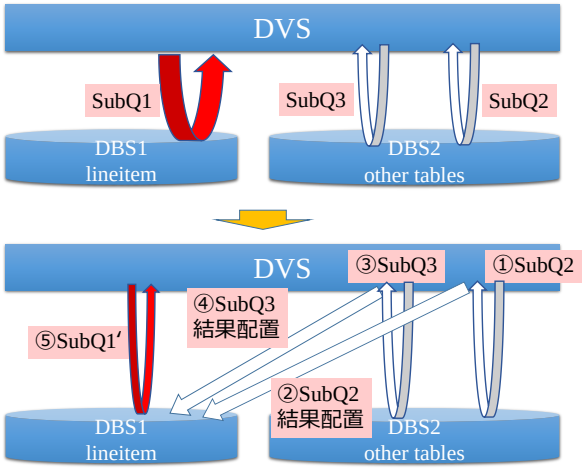

図 5　中間データ再配置手法

分解操作の例について，図 4 をもとに述べる．クエリグラフの初期状態が図左上である．まず，2 つの DBS におけるテーブル配置より，基準 1 の操作で L－O 間，L－S 間のエッジが分解される．次に，C－S 間が多対多の関係であったと仮定する．すると基準 2 により，C－S 間のエッジで分解される．この操作の結果，3 つの分解されたグラフが生成され，これらをサブクエリの単位とすることで，直積や多対多を回避することができる．ここでのサブクエリは，コネクテッドなグラフとなっているノード（テーブル）と，残っているエッジ（結合条件），そして削除された結合条件に含まれていたカラム，集計演算等に必要となるカラムを出力するものとなる．

グラフ分解操作の基準判定について，いくつかの手法が考えられる．1 つはテーブル定義情報を用いたルールベースの分解である．基準 1 となる DBS ごとのテーブル配置は，各 DBS とテーブル間の紐づけ情報により確認できる．基準 2 に関しては，テーブル定義情報より，結合条件カラムが一意性制約（unique など）を有するか否かで判定できる．この手法については，DVS が DBS のテーブル定義情報を持つ必要がある．もう 1 つの手段として，サイズ推定技術の応用が考えられる．エッジに紐づくテーブルと結合条件ごとに結合後のサイズ推定をすることで，肥大化を招く結合であるか判断することが可能と考える．

3.2. 中間データ再配置手法

課題 2 に対しては，サブクエリの結果を他方の DBS に一時的再配置し，結合や集計，順序整列演算をプッシュダウンする手法を提案する（図 5）．

3.1 の手法により得られたサブクエリ群について，サイズ推定を用いてボトルネックとなりうるサブクエリを確認し，効率のよいサブクエリの組み合わせ順序を実行計画として組み立てる．本手法は Greedy Algorithm をベースとして設計した[8]．アルゴリズム

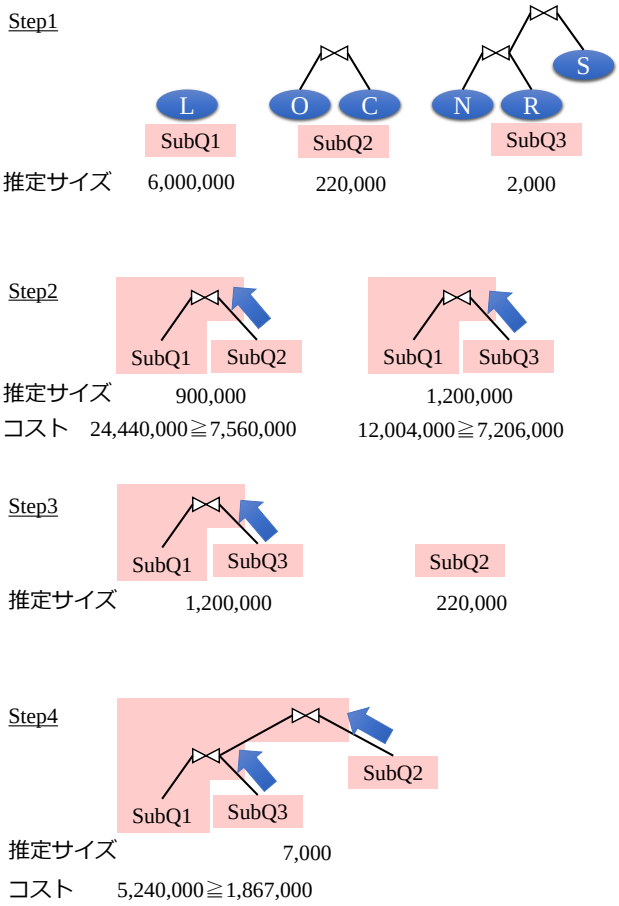

図 6　再配置プランの探索例

は，あるサブクエリ群が与えられたとき，コスト最大と推定されるサブクエリと，結合後にもっともコスト最小となるサブクエリの組み合わせ，そして結合演算の実行場所を判定し，実行計画を出力する．サブクエリ同士の結合演算の実行場所は，DVS，DBS 間のデータ処理をコストとして，もっとも小さくなるよう選択する．

ここで，DBS1 にサブクエリ SubQ1，DBS2 にサブクエリ SubQ2 があり，SubQ1 がコスト最大であったとする．このとき，以下の式により実行場所を選択する．

$$\begin{aligned}\mathrm{Read}(\mathrm{SubQ1}) + \mathrm{Ship}(\mathrm{SubQ1}) + \mathrm{Read}(\mathrm{SubQ2}) + \mathrm{Ship}(\mathrm{SubQ2}) \\ \geq \mathrm{Read}(\mathrm{SubQ2}) + \mathrm{Ship}(\mathrm{SubQ2}) * 2 \\ + \mathrm{Write}(\mathrm{SubQ2}) + \mathrm{Read}(\mathrm{SubQ1}) + \mathrm{Read}(\mathrm{SubQ2}) \\ + \mathrm{Ship}(\mathrm{Join}(\mathrm{SubQ1}, \mathrm{SubQ2}))\end{aligned}$$

上記式が成り立つ場合は，再配置処理（SubQ2 の結果を DBS1 に配置し，DBS1 で SubQ1 と SubQ2 を結合処理する）を選択する．式の左辺は，DVS で結合する場合のコスト，右辺は再配置処理時のコストである．ここでは，データ処理における読み込み，書き込み，転送処理をコスト要素としている．DVS の場合は，DBS1，DBS2 それぞれで読み込みと転送処理が発生する．DBS1 に中間結果を配置する場合は，DBS2 での SubQ2

の読み込み, DVS への転送処理, そして DVS から DBS1 への転送, 書き込み処理, その後 DBS1 において, SubQ1, SubQ2 がそれぞれ読み込まれ, 結合結果 Join(SubQ1,SubQ2)が DVS へ転送処理される.

図 6 は, 実行計画の探索プロセスを示したものである. Step1 では, 分解されたサブクエリについて, それぞれ実行後の推定サイズを求める. このとき, SubQ1 の推定サイズが最も大きく, ボトルネックになりうると判断する.

Step2 では, SubQ1 と SubQ2, SubQ3 それぞれの結果を結合した場合の推定サイズとコストを求めている. いずれも 6,000,000 件から削減されており, 効率化が見込まれる. また, コストは左辺が DVS での結合時, 右辺は再配置時のコスト値である. SubQ2, SubQ3 どちらを選択した場合も, 再配置時が効率的と判断する. この場合には, より低コストで実現可能な SubQ3 の再配置プランを選択する。

Step3 では, 残ったサブクエリを Step1 同様に列挙し, 推定サイズを求めている. ここでは SubQ1, SubQ3 結合後の推定サイズが大きく, ボトルネックになりうると判断する.

Step4 では, SubQ2 と結合した場合の推定サイズおよびコストを求める. 結果として, SubQ2 を再配置し結合するプランが効率的と判断し, Step4 で挙げたプラン候補が採用される.

このプラン探索は, すべてのサブクエリが特定の DBS に集約されるか, もしくは最適化されないと判断するまで繰り返す. 最後の結合演算が DBS にプッシュダウン可能と判断された場合には, 最上位の集計, 整列演算も含めてプッシュダウンすることができる. 最終的に得られる実行計画例を図 7 に示す.

4. 実装

3 節で述べた提案手法について, オープンソースソフトソフトウェアの Presto に拡張実装した. ベースとしたバージョンは, Presto-0.100 である. 本節では, 提案手法を Presto に適用する上での要点を述べる.

4.1. 実行計画の最適化機能

グラフ分解を用いたサブクエリ生成および再配置を含む実行計画の生成については, Presto の生成する実行計画を書き換えることで実現する.

Presto では, 最適化処理の中で Left-deep Tree となる実行計画を生成する. クエリグラフは, この実行計画より, テーブルおよび結合条件を抽出して生成する. DBS の物理配置は, 同様に実行計画内のテーブルアクセス情報から抽出可能である. 多対多判定については, 統計情報を用いたヒストグラム方式のサイズ推定を応用している[9]. 結合演算後のデータ件数を推定し, 入力となった 2 つのデータ件数いずれも上回る場合に多対多であると判定する. 上記操作で得られた情報をもとに、分解グラフ群を得る.

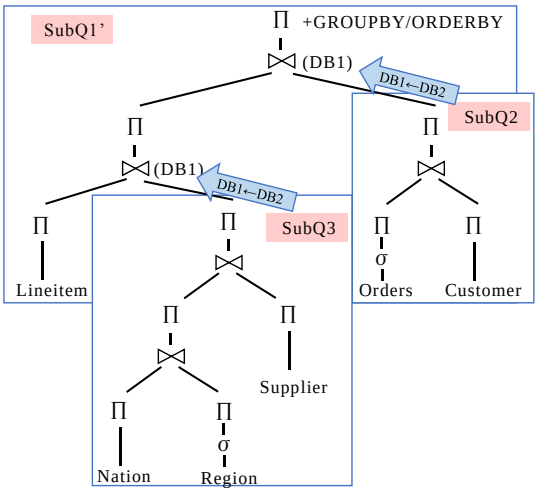

図 7 実行計画例

次に, 分解グラフ情報を用いて, Left-deep Tree の実行計画を, サブクエリ単位のサブツリーを有する Bushy Tree に変形する. その後, 3.2 節の手法に従い, これらサブクエリ結果と, サブクエリ同士の結合演算結果サイズを推定し, 中間データ再配置を考慮した実行計画に更新していく.

4.2. 実行処理

提案手法を実現するにあたり, Presto におけるプッシュダウン機能を改修している. 元となっている Presto では, プッシュダウン機能が範囲選択と射影演算に限定されている. これについて, 結合や集計, 整列演算もプッシュダウンできるよう改修している[10].

中間データ再配置処理については, サブクエリ実行結果をテンポラリテーブルに格納することで実現している. CREATE TABLE 句によりテンポラリテーブルを生成し, INSERT 句で結果挿入する. テンポラリテーブル名は, 複数クエリの発生時を考慮し, クエリごとに一意となるような識別子を付与する. クエリ処理の終了後, テンポラリテーブルを削除する.

5. 評価実験

提案手法の有効性を検証するため, プロトタイプを用いた評価実験を実施する. 評価実験では, 提案手法を実装した DVS と複数の DBS が接続されたデータ仮想化環境を構築し, TPC-H ベンチマークを用いて提案手法の効果を確認する. 比較対象として, 改修前の Presto-0.100, クエリ分解およびプッシュダウンのみを実装した DVS, 全提案手法を実装した DVS の 3 パターンで実行する. 評価観点として, クエリ実行時間と DVS－DBS 間のデータ転送量を観察する.

5.1. 環境

本実験では, プロトタイプ実装した DVS 1 台および

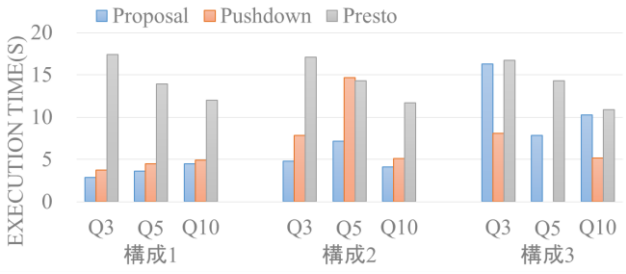

図 8　クエリ実行時間の比較

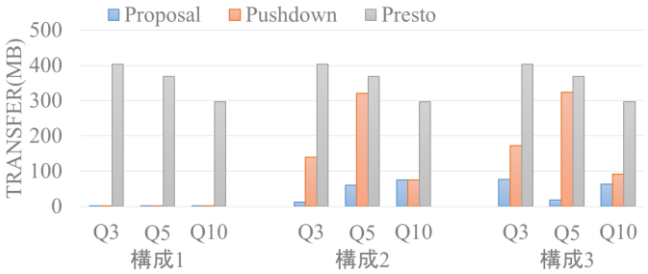

図 9　データ転送量の比較

DBS 1～3 台の複数構成とする．

実験に用いた主な環境仕様は次のとおりである．CPU：Intel Xeon X5675 3.07GHz，RAM：96GB，OS：CentOS 6.6．同仕様のサーバを 4 台用意し，DVS，DBS それぞれを構築する．また，各サーバ間は 1Gbps Ethernet によりネットワーク接続される．

DBS には，それぞれ PostgreSQL9.3.9 を導入する．実験データとして，TPC-H データセットを Scale 1 で生成した．データ配置については，DBS の構成による効果の差異を確認するため，以下のような 3 つのケースで構築する．

構成 1：DBS 1 つで構成し，全テーブルを配置

構成 2：DBS 2 つで構成し，lineitem のみ，それ以外のテーブルとして配置

構成 3：DBS 3 つで構成し，lineitem のみ，orders のみ，それ以外のテーブルとして配置

5.2. 実験結果

実験結果について，クエリ実行時間の比較を図 8，データ転送量の比較を図 9，また提案手法で生成された実行計画を図 10 に示す．図 8,9 の凡例について，Proposal は再配置を含む提案手法を実装した DVS，Pushdown はクエリ分解のみ適用した DVS，Presto は改修前による実行結果を表している．

図 8 は，各 DVS にて TPC-H Q3，Q5，Q10 を実行したときのクエリ処理に要した時間を表している．提案手法と他の実行時間を比較すると，構成 1，2 においては 2～6 倍の高速化となっている．構成 2 の Q5 においては，Proposal と Pushdown の差が観察され，中間データ再配置の有効性が確認できる．これに対し，構成 3

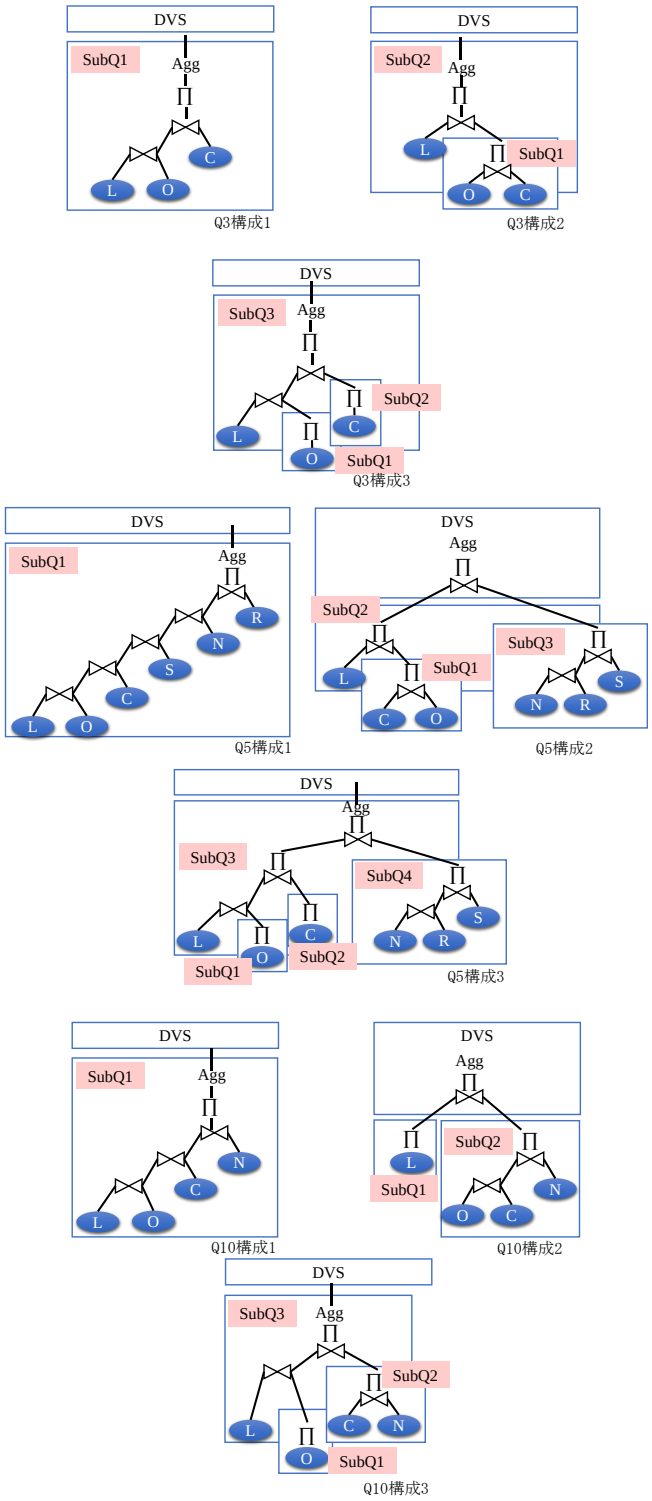

図 10　提案手法による実行計画

の Q3，Q10 を見ると，Proposal と Presto が同程度の実行時間となっている．また Presto については，構成 1～3 それぞれにおいて，実行時間がほぼ変わっていない．これは，Presto が構成によらずテーブル単位にサブクエリを生成するためである．なお，構成 3 の Q5 Pushdown についてのみ，実装上の不具合により，正し

いデータを取得できていない.

図 9 は，同様に実行した場合に，DVS－DBS 間で発生したデータ量を表している．データ量の算出には，DBS から DVS へ転送されるサブクエリ結果返却処理と，DVS から DBS へ転送される中間データ再配置処理の 2 つの要素を合算している．いずれの構成・クエリについても，提案手法ではデータ転送量は大幅に削減されていることが分かり，有効に機能したと考えられる．また構成 2,3 の Q5 より，中間データ再配置手法が有効に機能したものと分かる.

構成 1 について，提案手法が効率的である理由は，プッシュダウンによる効果である．単一 DBS で完結するため，クエリ分解操作の結果は 1 本のツリーとなり，集計や順序整列演算についても DBS にプッシュダウンされている（図 10）．これは単一 DBS 内でクエリ実行することと同様である．本構成の場合，クエリグラフは分解されず，また中間データ再配置処理も発生しない.

構成 2 については，Q3，Q5 について中間データ再配置の効果が得られ，Presto，Pushdown に対し実行時間およびデータ転送量が効率化されている．Q10 については，再配置処理されていないが，実行時間，データ転送量の観点でも目立って高くない．これは，SubQ1 と SubQ2 のサイズ推定結果から，DVS での結合が有利と判断されたためである．いずれにクエリについても，Presto，Pushdown（Q5）の実行時間は lineitem を取得するサブクエリがボトルネックとなっている．TPC-H において，lineitem はもっとも大きなテーブルであり，結果取得に時間を要する．これについて提案手法では，他方の DBS におけるサブクエリを先に実行し，lineitem が配置された DBS へ再配置することで，実行時間が短縮されている.

構成 3 については，提案手法によりデータ転送量が大幅削減されているものの，Q3，Q10 のクエリ実行時間は Presto と同等であった．提案手法の処理詳細を確認すると，中間データ再配置処理に時間を要していることが分かった（表 1）．タスクについて，PLAN は Presto 内での実行計画を生成するための所用時間である．それ以外については，DBS に発行される SQL 命令の所要時間であり，それぞれシーケンシャルに進められている．中間データ再配置処理には INSERT 命令を用いたが，データ量の増加と共に命令回数と時間が増加している．これは，Presto に提案手法を実装する上で，一命令を Presto におけるページ単位に生成したためである．別の要因として，データ配置が分離したことにより，サブクエリのプッシュダウン効果が減少したことである．例えば Q3 は，lineitem，orders，customer の 3 テーブルを結合するクエリである．構成 3 の場合はすべてのテーブルが分離される状態となるため，いずれのサブクエリも，範囲選択と射影演算のプッシュダウンにとどまり，データ転送量が大きくなった．このことを踏まえ，実行計画の生成に用いる判定式も，INSERT 命令のコストを踏まえてチューニングが必要と考える.

表 1　提案手法における Q3 処理詳細

構成	タスク	時間(s)	備考
Q03 構成1	PLAN	0.288	
	SELECT	2.610	
Q03 構成2	PLAN	0.361	
	CREATE	0.033	
	SELECT	0.594	orders, customer
	INSERT	1.943	INSERT 18回
	SELECT	1.869	
	DROP	0.022	
Q03 構成3	PLAN	0.377	
	CREATE	0.032	
	SELECT	0.827	orders
	INSERT	12.145	INSERT 89回
	CREATE	0.059	
	SELECT	0.091	customer
	INSERT	0.149	INSERT 4回
	SELECT	2.560	
	DROP	0.027	
	DROP	0.018	

実行計画の生成処理については，およそ表 1 の PLAN で示した程度の時間で完了しており，極端に処理時間を要することはなかった.

6. 考察

提案手法のクエリ分解は，実行計画の探索空間を削減することに有効と考えられる．データ仮想化環境や Multidatabase では，DS（DBS）のサイト間転送も考慮する必要があるため，テーブルの結合順序を決定するうえでは探索空間が広い．提案手法を用いることで，Bushy を含む効率的な実行計画が生成可能と考える．本手法と同様に，テーブルグループからサブツリーを生成し，効率的な Bushy プランを得る手法も研究されている[11].

一方で，データ転送量の観点では，実行計画を十分に探索しきれていない．図 9 の Q5 と Q10 の Proposal ついて，構成 2 と構成 3 を比較すると，僅かだが構成 3 のほうが少ない．構成 3 は，物理レイアウトが異なるため orders が分離されているものの，データ転送量は効率化されている．構成 2 の場合，クエリ分解次第では構成 3 と同様のサブクエリ単位とすることも可能であることから，提案手法のクエリ分解粒度について，更なる工夫の余地があるものと考える．また，転送量を効率化する手法として Semi-join などもある．カラム数が多くなる場合には有効と考えられ，プラン探索に組み込むことが望ましい.

中間データ再配置処理については，効率的な実現方法も検討すべきと考える．評価実験において，提案手法によるクエリ処理方式によって，中間データ量を削減できることを確認した．しかしながら，中間データ再配置処理のデータサイズが大きい場合，INSERT 命令がボトルネックとなり，クエリ処理の遅延が確認された．これについては，INSERT 命令単位のデータ量変更や，DBMS 製品ごとに INSERT 命令の効率化検討，あるいは中間データをファイルとして転送し，DBS にロードさせることで，より高速に処理することも可能と考えられる．

データ仮想化において，提案手法のクエリ処理はシーケンシャル性が高い．提案手法の狙いは，サブクエリの実行結果を再配置することで，よりデータ量が大きくなる後続のサブクエリを効率化することである．このため，順次実行する形態となることが特徴的である．一方 Presto では，DBS へのサブクエリを同時実行し，データが取り込まれ次第，順次処理を開始する特徴を持つ．多数の DBS にまたがり，データ量がボトルネックとなるほど大きくない場合は，却って提案手法の処理速度が遅くなる可能性がある．

異常終了を引き起こすクエリについて，提案手法は根本的な解決となっていない．クエリ処理におけるデータ量を削減し効率化することは可能だが，クエリによっては効率化できないことが考えられる．サービスレベルが高く，異常終了が好ましくないユースケースに対しては，確実な実行方式を備えることが望ましいものと考える[12].

提案手法は，大規模データを保有する DBS に対して，ネットワークリソースは効率化が期待できるものの，そのトレードオフとして CPU 処理負荷を与えることが多くなる．データ転送量の小さくなる中間データ再配置を目指すと，Fact テーブルを有する DBS に対し，外部キーとなる Dimension テーブルへのサブクエリ結果を再配置し，結合，集計や順序整列演算をプッシュダウンする実行計画が多くなるものと考えられる．結果として，大規模データを保有する DBS の処理負荷を高くすることになる．この反面，従来データ仮想化のようなデータ転送量が多い場合，IO やネットワークリソースを消費するものと考えられる．

実用においては，DBS への負荷影響を踏まえ，リソース制御機能を有することが望ましい．企業内で用いられる DBS のリソースは，その本来業務を対象としてサイジングされており，常時 DVS の要求にこたえられるとは限らない．リソース状態を考慮した実行計画の生成，実行制御が必要になるものと考える．また，DBS の特性によっては，データ挿入を得意とする製品もある．コストモデルをチューニングすることで，DBS 資源を有効活用した全体最適化が期待できる．

7. おわりに

本研究では，データ仮想化環境におけるクエリ処理について，既存データ仮想化製品および技術を調査した．調査の結果，サブクエリの処理結果が肥大化する事象と要因，そして DVS におけるクエリ処理の課題を明らかとした．本課題について，クエリ分解と中間結果再配置により，中間データ量を削減する，効率的なクエリ処理手法を考案した．考案手法について，オープンソースソフトウェアの Presto に実装し，TPC-H を用いて評価実験を実施した．その結果，中間データ量を削減できること，クエリ実行時間を短縮できることが確認された．一方，DBS の分離が多くなり，プッシュダウン効果を得にくい状態になると，中間データ量が増加し，データ挿入処理がボトルネックとなることが確認された．

今後の課題は，中間データの再配置方法について効率よい手段を取り込むこと，並列実行も考慮した実行計画生成をすること，実用化する上でのリソース制御手法を検討することである．

参 考 文 献

[1] Rick F., van der Lans , “Data Virtualization for Business Intelligence Systems”, Morgan Kaufmann 2012.

[2] Presto, https://prestodb.io/.

[3] TPC-H, http://www.tpc.org/tpch/.

[4] S.Nural., et al., ”Query Decomposition and Processing Multidatabase Systems”, Object Oriented Database Symposium of the thrid European Joint Conference on Engineering Systems Design and Analysis, pp.41-52, France 1996.

[5] Teiid, http://teiid.jboss.org/.

[6] 片山大河，嶋村誠，金松基孝，”異種データソースを透過的にアクセス可能とする統合データベースシステム”，情報処理学会研究報告，Vol.2013-DBS-157，2013

[7] M.Tamer Özsu. and P. Valduriez., “Principles of Distributed Database Systems Third Edition”, Springer Science, 2011.

[8] D.Kossmann and K.Stocker., “Iterative Dynamic Programming: A New Class of Query Optimization Algorithms”, ACM Transactions on Database Systems, Vol.25, Issue 1, March 2000.

[9] N. Bruno and S. Chauddhuri., “Exploiting Statistics on Query Expressions for optimization”, in Proceedings of 2002 ACM SIGMOD international conference on Management of data, pp.263-274, 2002.

[10] 齋藤和広，米田信之，渡辺泰之，村松茂樹，小林亜令，“データ仮想化システムにおける効率的なクエリプッシュダウンの実装と評価”，情報処理学会研究報告，Vol.2015-DBS-162, 2015.

[11] R. Ahmed., et al., “Of snowstorms and bushy trees”, Proceedings of the VLDB Endowment, Vol.7, Issue 13, August 2014.

[12] 齋藤和広，米田信之，渡辺泰之，村松茂樹，小林亜令，“データ仮想化システムにおけるクエリ分割実行の実装と評価”, DEIM2016.