情報処理学会論文誌 Vol.59 No.2 372–383 (Feb. 2018)

データベース分割を目的とした データ仮想化によるデータベースの仮想統合

齋藤 和広[1,a)] 米田 信之[1,†1] 渡辺 泰之[1,†1] 黒川 茂莉[1] 村松 茂樹[1] 小林 亜令[1]

受付日 2017年5月9日, 採録日 2017年11月7日

概要：通信インフラの運用管理設備など，大規模なデータの逐次追加を要する DB においては，規模が大きくなることで，リソース設計が複雑となり，メンテナンスコストも増加する．そこで，単一の DB を複数の DB に分割しつつ，ユーザの利便性を下げない，データベースの仮想的な統合が求められている．データ仮想化は，複数の DB を仮想統合する技術の 1 つで，複数の DB のスキーマ情報を統合し，単一の DB と同様に SQL クエリでそれぞれの DB にアクセスできる．しかしそれぞれの DB からネットワークを介してデータを取得してクエリ処理をするため，大規模なデータを扱う際に，通信がボトルネックとなり，クエリ処理性能が低下するという課題がある．そこで本論文では，データ仮想化における通信ボトルネックを解消しうる 3 つの手法：クエリプッシュダウン手法，データ配置分析手法，中間データ再配置手法の有効性を，クエリ処理性能の観点から定量的に比較評価する．評価では，通信インフラの運用ログ（実データ）を利用し，通信品質を測定するクエリを対象に，クエリ実行時間を計測した．本評価の結果から，データ仮想化による DB 分割においては，クエリプッシュダウン手法の適用だけでなく，クエリログを用いたデータ配置分析手法や，クエリ実行時のデータ再配置手法を複合的に用いることで，通信ボトルネックをおおむね解消できることが明らかとなった．特に，12 種のクエリのうち 11 種で，3 つの手法適用前のデータ仮想化では DB 分割前の環境と比較して通信ボトルネックにより平均 2010 倍遅延していたが，3 つの手法を適用することで平均 1.8 倍の遅延に改善された．

キーワード：データベース統合，データ仮想化，データ配置，クエリ処理，クエリ最適化

Virtual Database Integration using Data Virtualization for the Divide of a Single Database

Kazuhiro Saito[1,a)] Nobuyuki Maita[1,†1] Yasuyuki Watanabe[1,†1]
Mori Kurokawa[1] Shigeki Muramatsu[1] Arei Kobayashi[1]

Received: May 9, 2017, Accepted: November 7, 2017

Abstract: As a scale of an integrated DB that has big data increasing day by day such as facility logs about telecommunication infrastructures gets bigger, its resource planning is more complex and its maintenance cost is higher. The virtual database integration technology is now necessary to divide such a big DB to multiple DBs without inconvenience for users. Data virtualization is one of such a technology that integrates only the schemas in multiple DBs and provides the integrated SQL interface for users like a single DB. However, because the data virtualization system accesses data in each DB through network, a communication between a DB and the data virtualization system degrades its query performance as a bottleneck. In this paper, to resolve the bottleneck problem of the data virtualization, we evaluate three methods: query pushdown, data placement analysis and replacement of intermediate data regarding query performance. In this evaluation, we use telecommunication logs as the real environment, and measure the execution time using analysis queries for network quality in 3DB environment integrated by data virtualization implemented three methods. As the evaluation result, we revealed to be able to resolve the network bottleneck problem by applying not only query pushdown but also data placement analysis using query logs and data replacement of intermediate data in the virtual database integration by data virtualization for the purpose of the divide of a single DB in real environment. Especially, in 11 queries of 12 queries, whereas the original data virtualization without three methods results in on average 2010 times slower than a single DB environment by network bottleneck, the data virtualization with three methods improves the network bottleneck to on average 1.8 times slower.

Keywords: database integration, data virtualization, data placement, query processing, query optimization

1. はじめに

データ分析を筆頭に，様々な種類のデータを統一的に活用すべく，データベース統合の需要が増加している．データベース統合は，複数のデータベース（DB）に分散するデータを 1 つの DB に集める物理統合が一般的である．しかし物理統合した結果，統合した DB のユーザが増え，多様なデータ分析のクエリによって DB の負荷が増加し，適正なリソース設計が困難になる場合がある．また部分的な障害が全ユーザに影響を与え，メンテナンス負荷が増大する．特にペタバイトスケールの大規模な DB で本現象が顕著となる．そこで，既存の DB を複数に分割でき，かつ既存ユーザの利便性を下げない仮想統合が求められている．データ仮想化 [1] は仮想統合を実現する 1 つの手法で，SQL インタフェースを備え，複数の DB のメタデータのみを集約し，ユーザから見て単一の DB に仮想的に統合する．これにより，統合するデータの規模や用途，利用頻度などで DB を複数に分けることが可能となり，それぞれのリソース設計の複雑化やメンテナンスコストの増加を最小限に抑えることができる．

一方，データ仮想化システムは，DB とネットワークで結合し，DB からデータを取得して，DB と同様に SQL の演算を処理する．そのため，大規模データに対するクエリや，複数のデータを組み合わせたクエリの処理において，DB とデータ仮想化システムの通信がボトルネックとなるという課題がある．

通信ボトルネックの解消手法は，DB 分割時に（静的に）DB 分割設計を行うデータ配置分析手法と，クエリ実行時に（動的に）クエリプッシュダウンやデータ再配置を行うクエリ処理手法に分類できる．

データ配置分析手法では，既存の単一 DB においてユーザが実行したクエリのログから，通信量が少なくなるように，複数の DB へのデータ分配を決定する [3]．

データ仮想化のクエリ処理手法では，ユーザが実行するクエリの演算処理の実行場所を最適化することで通信量を減らす．これには 2 つの手法，①できる限り 1 つの DB で処理して通信データ量を減らすクエリプッシュダウン手法 [1], [16], [17]，②複数の DB に存在するデータに対して処理するクエリにおいて，ある DB での処理結果をそのデータを必要とする別の DB に一時的に移動して処理する中間データ再配置手法 [1], [19], [20], [21]，が存在する．

本論文では，データ仮想化における通信ボトルネックを解消しうる，データ配置分析手法，およびクエリ処理手法の有効性を，クエリ処理性能の観点から定量的に比較評価する．評価では，通信インフラの運用ログ（実データ）を利用し，通信品質を測定するクエリを対象に，クエリ実行時間を計測した．本評価の結果からデータ仮想化による DB 分割においては，クエリプッシュダウン手法の適用だけでなく，クエリログ分析を用いた DB 分割手法や，クエリ実行時のデータ再配置手法を複合的に用いることで，通信ボトルネックをおおむね解消できることが明らかとなった．

本論文における貢献は，データ仮想化における通信のボトルネックを解消しうる 3 つの手法：データ配置分析，クエリプッシュダウン，中間データ再配置を複合的に評価できるシステムを構築し，実データを用いた評価により，有効性を定量評価した点となる．

本論文の構成は以下のとおりである．2 章で関連研究を述べ，評価対象の手法として 3 章でクエリプッシュダウン，4 章でデータ配置分析，5 章で中間データ再配置を述べる．6 章で 3 つの手法の実装について述べ，7 章で評価し，8 章で結論を述べる．

2. 関連研究

データ仮想化は，複数の DB を仮想的に 1 つの DB とするデータベース統合技術の 1 つである [1]．PostgreSQL における Foreign Data Wrapper [12] や Oracle における Database Link [13] などのように，DB 自身が他の DB と連携するデータベースフェデレーション [14] とは異なり，データ仮想化システムは DB と完全に独立で構築される．ユーザはデータ仮想化システムに対して SQL クエリを投稿し，データ仮想化システムは必要なデータを DB から取得してクエリ処理する（**図 1**）．

複数の DB を仮想的に統合するデータ仮想化システムは，マルチデータベースシステムや異種統合データベースシステムとも呼ばれ，古くから研究されている [29]．データ仮想化システムに関するプロジェクトとして，GRelC [34], OASIS [35]，および ViDa [36] は様々なプロトコルに対応したデータアクセスモジュール（Wrapper）を持つ．クエリ処理では，Wrapper を通じてデータを取得し，演算処

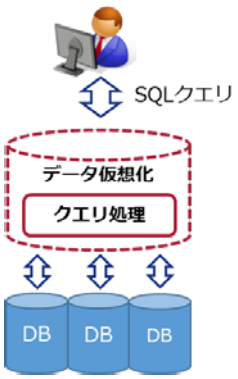


図 1 データ仮想化システム
Fig. 1 Data virtualization system.

[1] 株式会社 KDDI 総合研究所
KDDI Research, Inc., Fujimino, Saitama 356–0003, Japan
[†1] 現在，KDDI 株式会社
Presently with KDDI Corporation
[a)] ku-saitou@kddi-research.jp

理はすべてデータ仮想化システムで実行する．その結果，データが大規模な場合に通信がボトルネックとなる．

MIND [18] は，データ仮想化システムの一種で，RDBMS を対象に，できる限り演算処理を委譲するクエリプッシュダウン手法 [1], [16], [17] を採用し，通信量の削減を図っている．しかし，異なる DB のデータに対する結合演算に対しては，MIND（データ仮想化システム）にデータ転送したうえでクエリ処理するため，大規模データの結合演算について，通信ボトルネックを解消できない．

そこで大規模データどうしの結合演算に対して，クエリプッシュダウンし，DB でクエリ処理可能とするため，以下に示す 2 種類のデータ配置手法の適用が考えられる．まず，クエリ実行前に，クエリログ分析を行い，通信ボトルネックとなるテーブルどうしをあらかじめ同じ DB に配置する，静的なデータ配置分析手法が提案されている [3]．次に，クエリ実行時に，動的に DB 間で中間データを移動して結合処理をクエリプッシュダウンすることで通信量を減らす，中間データ再配置手法 [1], [19], [20], [21] が提案されている．

データ配置手法をデータ仮想化に適用した事例としては，HP Labs の Pegasus [30]，ファイルなどの非構造データも対象とした TSIMMIS [31]，マルチメディアデータも対象とした IBM の Garlic [32]，推論システムの構築を目的とした HERMES [33] があげられる，これらの事例は，すでに複数存在する DB を統合することを目的として，中間データ再配置手法を適用しているが，DB 分割を想定しておらず，クエリログ分析による静的なデータ配置分析手法は未適用である．よって，クエリプッシュダウンを含めた両データ配置手法の比較評価や，それらを複合的に用いた場合における通信ボトルネックの課題解消可否について，論じられていない．

そこで本論文では，**図 2** で示す仮想統合環境として，データ仮想化における 3 つの手法：クエリプッシュダウン，データ配置分析，中間データ再配置を複合的に評価できるシステムを構築し，実データを用いた評価実験を行う．本評価では，単一の DB 環境でクエリを実行したときの実行時間に対する仮想統合環境における同クエリの実行時間の遅延を通信ボトルネックと定義し，3 つの手法の有無による実行時間の遅延を比較することで通信ボトルネックの課題解消を評価する．

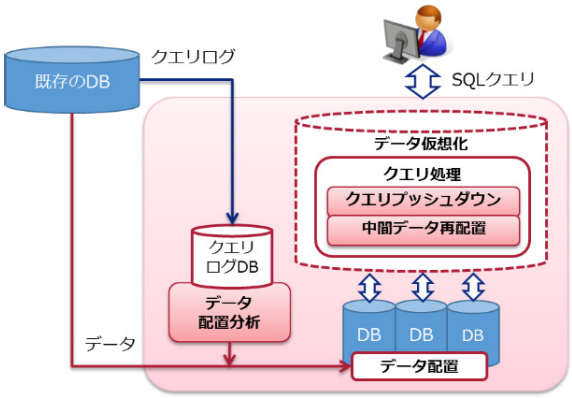


図 2 データ仮想化を用いた仮想統合環境

Fig. 2 Virtual integrated environment using data virtualization.

3. クエリプッシュダウン

SQL クエリの多くの演算は，処理することでデータサイズを減らす効果がある．DBMS において，選択演算や射影演算を，テーブル読み込み後可能な限りすぐに処理することでその後の演算処理の対象データのサイズを減らすルールベースの最適化手法がある [15]．同様にして，テーブル読み込み後の処理を可能な限り DB で処理することで，通信量を減らす効果が見込まれる [1], [16], [17]．クエリプッシュダウンは，この効果を利用し，同じ DB で処理可能な演算をすべて DB に委譲する手法である．他のクエリ最適化手法を施した後の実行計画に対して本手法を適用し，同一の DB 上で処理できる実行計画の部分木を DB へのクエリに変換する．データ仮想化システムは，このクエリ結果に対して実行計画の残りの演算を処理する．

一方で結合演算は，直積や，1 レコードが複数のレコードと結合する多対多の結合においてデータサイズが増えるため，クエリプッシュダウンによってむしろ通信量が増える．これは該当する演算をクエリプッシュダウンの対象外とすることで通信量の増加を回避できる．しかし，結合演算をクエリプッシュダウンの対象外にすることで，その後段の演算も同様にクエリプッシュダウンできず，結果として通信量が増える場合がある．そこでクエリからテーブルどうしの結合関係をグラフで表す Join Graph を作成し，クエリプッシュダウンするサブクエリ単位に分解する手法が提案されている [17]．これにより，Join Graph で表現されない直積が回避できるだけでなく，DB の異なるテーブルのエッジとともに，多対多の関係のエッジも分解することで通信量を削減できる．

4. データ配置分析

クエリプッシュダウンによって DB で演算処理する効果を高めるために，できる限り DB 間の結合処理を少なくしつつ，かつ DB 間の結合演算が必要な場合でもその通信コストが少なくなるようにデータを配置する [3]．これを実現するため，分割前の DB で実行された SQL クエリを分析し，特に通信量の大きい結合処理が行われているテーブルどうしを同じ DB に配置する．

SQL クエリの分析では，クエリ処理のコスト推定やスキーママッピングなどを目的に，SQL クエリやそのワークロードから特徴量として様々な情報を抽出する．最もシンプルな方法として，SQL クエリをテキストデータと

して解析して情報抽出する手法がある [4], [5]．また，SQL クエリから，ノードをテーブル，エッジを結合関係とする Join Graph を作成してクエリを分析する手法も存在する [6], [7]．一方で，SQL クエリ文で表現されている処理は，DB の最適化などにより実際の処理と異なる可能性がある．さらに上記の手法は結合演算の順序性を考慮できず，実際に処理されるときの情報が加味されていない．これらを考慮したクエリ分析手法として，クエリの実行計画を分析する手法がある [8]．

5. 中間データ再配置

クエリプッシュダウンするサブクエリを生成後，それぞれのサブクエリ結果の結合方法を決定する必要がある．データ仮想化システムの一種である MIND [18] では，サブクエリ結果をすべてデータ仮想化システムが取得して処理する．ここでサブクエリ結果が大規模な場合に，データの取得による通信がボトルネックとなる．一方，異なる DB で処理された 2 つのサブクエリの結果を結合することで，それぞれのサブクエリの結果よりデータサイズを小さくすることが可能な場合がある．中間データ再配置は，このように結合演算でデータサイズが小さくなる場合に，クエリの中間結果であるサブクエリの結果を，別のサブクエリの結果がある DB に移動して結合処理する手法である．中間データをデータ仮想化システムが取得して結合するときと比較して，通信量の削減が見込める場合にこの手法を適用する．

図 3 は，クエリプッシュダウンするサブクエリを対象に中間データ再配置する例である．サブクエリ SubQ1 はデータサイズが大きく，他 2 つのサブクエリ SubQ2，SubQ3 は比較的小さいと想定する．このときデータ仮想化システムで処理する場合に，SubQ1 の通信がボトルネックになる．そこで SubQ1 を実行する DB に，SubQ2 および SubQ3 の結果を再配置し，そこで結合処理を実行することで通信データ量を削減している．

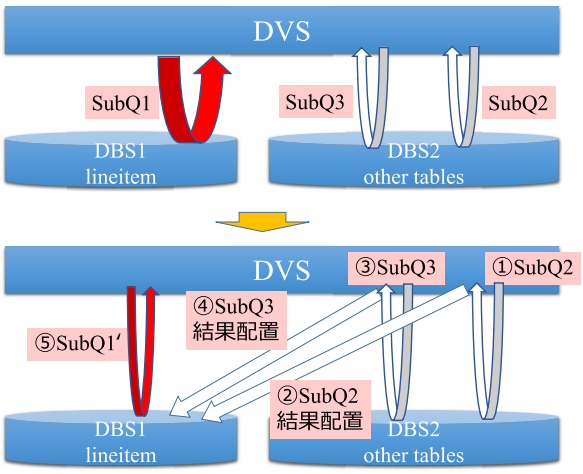


図 3 中間データ再配置
Fig. 3 Intermediate data replacement mechanism in DVS.

異なる DB で実行されたサブクエリ結果を結合する中間データ再配置は，処理方法として大きく 2 つの方法が提案されている [1], [19]．Ship Join は小さい方のサブクエリ結果 R をもう一方のサブクエリ結果 S がある DB に移動して結合処理する．このとき，S がある DB には，R を格納する一時テーブルの作成，R の挿入，そして R と S の結合処理後の一時テーブルの削除が追加処理として必要となる．Ship Join は R と S 双方が大きい場合，むしろ通信量が肥大化する．そこでもう 1 つの手法として，2-way Semijoin が提案されている．これは小さい方のサブクエリ結果 R のうち，結合キーに利用される属性のみを，もう一方のサブクエリ結果 S のある DB に転送して Semijoin を実行する．その結果を再び R があった DB に戻して R と結合処理をする．2-way Semijoin は R の属性数が多く，かつ R と S の Semijoin 後のサイズが小さい場合に，Ship join より通信量を削減できる．一方，2-way Semijoin は，R のレコード数や結合結果のレコード数が多い場合に性能が悪化するため，Bloomjoin など符号化して転送する手法も提案されている [19], [20], [21]．

複数のサブクエリ結果に結合演算が必要な場合，各結合演算の結合場所と順序を決める必要がある．従来の DBMS でも結合順序は古くから検討され，主に包括的に探索する Dynamic Programming（DP），ヒューリスティックに探索する Greedy Algorithm（Greedy），ランダムにテーブルの結合順序を変えて探索する Randomized Algorithm に分類される [22]．データ仮想化や分散 DB などの場合，実行順序に加えて実行場所も条件となるため，DP は探索範囲が広く計算量の観点で現実的ではない．そこで計算量を少なく最適解に近づけるため，DP と Greedy を組み合わせる Iterative Dynamic Programing（IDP）[22] や，Join Graph ベースで DP を実行する DPccp [23] など，様々な拡張が提案されている [24], [25]．

6. 実装

データ仮想化における 3 つの手法：クエリプッシュダウン，データ配置分析，中間データ再配置による通信ボトルネックの解消を比較評価するにあたり，図 2 で示す仮想統合環境を構築する．以下では，データ配置分析に必要なクエリログ DB の構築方法，データ配置分析の実装，およびデータ仮想化システムの実装について述べる．

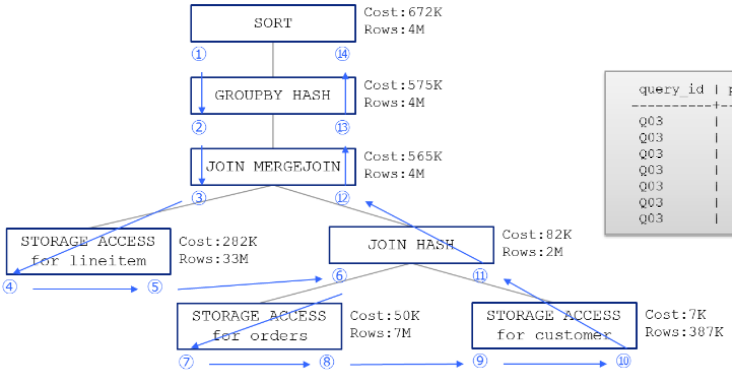
6.1 クエリログ DB

データ配置分析ではクエリ処理時に実際に行われる処理の中間結果のサイズが重要となるため，分析対象としてクエリの実行計画を分析する手法 [8] を活用する．SQL クエリの実行計画は，ツリー構造で表現されている．この蓄積

```
+-SORT [Cost: 672K, Rows: 4M] (PATH ID: 1)
|  Order: sum((lineitem.L_EXTENDEDPRICE * (1 - lineitem.L_DISCOUNT))) DESC, orders.O_ORDERDATE ASC
| +---> GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 575K, Rows: 4M] (PATH ID: 2)
| |       Aggregates: sum((lineitem.L_EXTENDEDPRICE * (1 - lineitem.L_DISCOUNT)))
| |       Group By: lineitem.L_ORDERKEY, orders.O_ORDERDATE, orders.O_SHIPPRIORITY
| |       Partially sorted keys: 1
| | +---> JOIN MERGEJOIN(inputs presorted) [Cost: 562K, Rows: 4M] (PATH ID: 3)
| | |       Join Cond: (lineitem.L_ORDERKEY = orders.O_ORDERKEY)
| | |       Materialize at Input: orders.O_ORDERKEY, orders.O_CUSTKEY, orders.O_ORDERDATE, orders.O_SHIPPRIORITY
| | |       Materialize at Output: lineitem.L_EXTENDEDPRICE, lineitem.L_DISCOUNT
| | | +-- Outer -> STORAGE ACCESS for lineitem [Cost: 282K, Rows: 33M] (PATH ID: 4)
| | | |       Projection: public.LINEITEM_super
| | | |       Materialize: lineitem.L_ORDERKEY
| | | |       Filter: (lineitem.L_SHIPDATE > '1995-03-12'::date)
| | | |       Runtime Filter: (SIP1(MergeJoin): lineitem.L_ORDERKEY)
| | | +-- Inner -> JOIN HASH [Cost: 82K, Rows: 2M] (PATH ID: 5)
| | | |       Join Cond: (customer.C_CUSTKEY = orders.O_CUSTKEY)
| | | | +-- Outer -> STORAGE ACCESS for orders [Cost: 70K, Rows: 7M] (PATH ID: 6)
| | | | |       Projection: public.ORDERS_super
| | | | |       Materialize: orders.O_CUSTKEY
| | | | |       Filter: (orders.O_ORDERDATE < '1995-03-12'::date)
| | | | |       Runtime Filter: (SIP2(HashJoin): orders.O_CUSTKEY)
| | | | +-- Inner -> STORAGE ACCESS for customer [Cost: 7K, Rows: 387K] (PATH ID: 7)
| | | | |       Projection: public.CUSTOMER_super
| | | | |       Materialize: customer.C_CUSTKEY
| | | | |       Filter: (customer.C_MKTSEGMENT = 'HOUSEHOLD')
```

(a) Vertica における SQL クエリの実行計画ログ

(a) SQL query plan log in Vertica



(b) 実行計画のツリー構造と位置情報付与手順

(b) Tree structure of query plan in (a), and procedure for giving location information on this tree

query_id	path_id	node	lft	rgt	cost	rows	table_name
Q03	1	SORT	1	14	672000	4000000	
Q03	2	GROUPBY HASH	2	13	575000	4000000	
Q03	3	JOIN MERGEJOIN	3	12	565000	4000000	
Q03	4	STORAGE ACCESS	4	5	282000	33000000	lineitem
Q03	5	JOIN HASH	6	11	82000	2000000	
Q03	6	STORAGE ACCESS	7	8	70000	7000000	orders
Q03	7	STORAGE ACCESS	9	10	7000	38000	customer

(c) Nested Set Model における実行計画

(c) Query plan described by nested set model

図 4 TPC-H Q3 の Nested Set Model 構築

Fig. 4 Construction expamle of nested set model in case of TPC-H Q3.

と分析を容易にするため，ツリー構造の実行計画を表形式で保存する Nested Set Model [9] を活用する．Nested Set Model では，実行計画上の演算を表すノードごとに，演算の情報に加えて実行計画上のツリー構造の位置情報を付与する．ツリー構造の位置情報は 2 つの番号で表現される．ツリーをルートノードから行きがけ順かつ帰りがけ順に走査し，ノードに到達したときに番号を付与することで，SQL クエリの実行計画におけるツリー上のノードの位置を表現する．

図 4 は，TPC-H [10] の Q3 における Nested Set Model の構築例である．図 4 (a) は Vertica [11] における実行計画ログであり，Vertica はこのクエリログのテキスト 1 行を 1 レコードとしてシステムテーブルに蓄積している．このクエリログのテキストをクエリ単位で解析し，図 4 (b) で示す演算のツリーを構築する．このツリーで示す矢印に従って，ルートノードである SORT 演算から順に走査して番号を付与し，各演算ノードに 2 つの番号を付与する．各演算で行きがけで付与した番号を lft，帰りがけで付与した番号を rgt として表形式にしたものが図 4 (c) である．

6.2 データ配置分析

Nested Set Model で表現されるクエリログ DB のデータを利用して複数の DB のデータ配置を分析する．入力はクエリログ DB にある過去の SQL クエリの実行計画と，配置対象のテーブル一覧，および配置先の複数の DB の情報となる．出力は配置対象である各テーブルの配置先 DB である．なお，テーブルの重複配置と，フラグメンテーションによるテーブルの分割配置は，データ整合性の維持によってメンテナンスの複雑化を招くことが想定されるため，DB 分割の目的であるメンテナンス性の向上の観点からここでは考慮しない．

各テーブルの配置先を決めるための条件は，DB 分割後の環境で通信データ量を最小化することである．ここでは，単一の DB で結合処理されていたテーブルどうしが，別の DB に分かれることによる通信を削減する．したがって，結合関係にあるテーブル群を同一の DB に集めることが目的となる．一方で，制約条件として各 DB のストレージ容量の上限があるため，結合関係にあるすべてのテーブルを必ずしも 1 つの DB に格納できるとは限らない．結合

関係があるテーブル群のうち，容量以下となるように結合関係を切り離す必要がある．そこで，結合関係にあるテーブル群の中で，頻度が多いテーブルの組から同一の DB に格納していき，ストレージ容量の上限となったら，残るテーブル群を別の DB に格納する．なお，テーブルの結合関係は，クエリログ DB にある実行計画から作成する．実行計画上で，結合処理に必要なテーブル群を 1 つの結合関係とし，その頻度を合計する．なお，マスタテーブルのように，小さいテーブルの結合は別の DB にしても影響は少ないため，結合関係として入力時の入力サイズやレコード数を条件にすることで，DB 間の通信ボトルネックとなりうるテーブルの組を優先的に同じ DB に配置する．

6.3 データ仮想化システム

データ仮想化システムの実装では，オープンソースソフトウェアである Presto [2] を利用した．Presto はインメモリの分散 SQL クエリエンジンで，Hadoop や様々な DB のデータを取得して処理可能なエンジンである．バージョンは 0.152 を利用した．本バージョンでは射影演算と一部の選択演算（数値型の比較演算）のみをクエリプッシュダウンし，他の演算は必ず Presto で実行する．そこでデータ仮想化のクエリ処理手法である 2 つの手法（クエリプッシュダウン，中間データ再配置）を実装した．中間データ再配置に関しては，再配置処理の実施を自動判定するため，ヒストグラムを用いた中間結果サイズ推定機能 [26] を実装し，そのための統計情報として接続先 DB のテーブルのヒストグラムを生成する機能を実装した．利用した中間データ再配置方式は Ship Join のみとし，結合順序と場所の判定アルゴリズムは探索時間を少なくするために Greedy をベースとした．

7. 評価

データ仮想化における通信ボトルネックを解消する 3 つの手法として，クエリプッシュダウン，データ配置分析，中間データ再配置を，通信インフラの設備ログデータと分析クエリを用いて複合的に評価する．評価では，クエリの実行時間を計測し，単一 DB から 3DB に分割してデータ仮想化で仮想統合することによる遅延を計測する．この遅延は通信ボトルネックによるものであり，この遅延が解消されている状態を，データ仮想化における通信ボトルネックが解消されている状態とする．以下では，まず評価環境としてシステム構成と，データの配置，評価クエリについて述べる．その後，本環境での評価結果を述べる．

7.1 評価環境

本評価の構成を**図 5** に示す．ハードウェアは 4 つのサーバを 1 GbitEthernet のネットワークで接続して構成した．サーバはすべて DELL R720 を利用し，スペックは

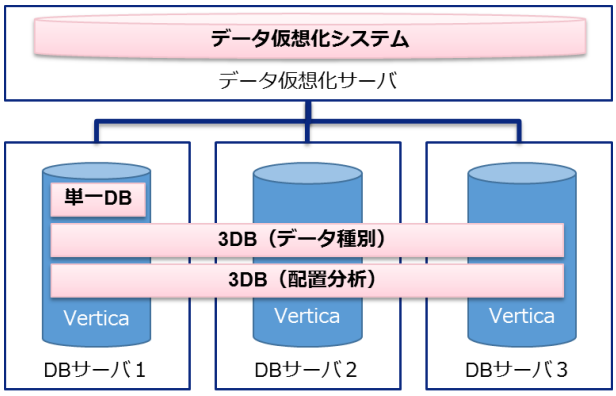


図 5 評価環境
Fig. 5 Evaluation environment.

表 1 各 DB における種類別のデータ配置と総データ容量
Table 1 Data placement by data type and total data size in each DB.

データ配置		テーブル数 ファクト 設備ログ	ファクト その他	マスタ	総容量
単一 DB	DB サーバ 1	3	4	25	957 GB
3DB（データ種別）	DB サーバ 1	3	0	0	306 GB
	DB サーバ 2	0	4	0	276 GB
	DB サーバ 3	0	0	25	375 GB
3DB（配置分析）	DB サーバ 1	3	0	2	348 GB
	DB サーバ 2	0	4	4	344 GB
	DB サーバ 3	0	0	19	266 GB

CPU が Xeon E5-2620 × 2（全 12 コア），メモリ 384 GB，HDD 4 TB × 3（RAID10）である．OS は CentOS 6.6 を利用した．DB として 3 つの DB サーバそれぞれに Vertica 7.2.3 [11] を構築した．分割前の単一の DB は DB サーバ 1 の Vertica のみに構築し，分割後の DB は 3 つの Vertica それぞれに構築した．なお分割後の DB 環境は，それぞれの DB サーバに配置するデータの違いから 3DB（データ種別）と 3DB（配置分析）の 2 つを構築した．結果として，DB サーバ 1 には，3 つの論理 DB を，DB サーバ 2 と 3 には，2 つの論理 DB を構築した．DB サーバ内の各論理 DB は排他的で，たとえば単一 DB を利用するときは，3DB（データ種別）のデータにはアクセスできない．また，各サーバの Vertica は分散 DB ではないため，DB サーバ間で別の Vertica のデータにはアクセスできない．

評価用のデータには，通信インフラに関する設備ログ，基地局データを全 32 テーブル，約 1 TB 分を利用した．3 つの DB 環境における種類別のデータ配置と総データ容量を**表 1** に示す．単一 DB にはすべてのテーブルを配置した．3DB（データ種別）では，ファクトテーブルであるログのうち最も大規模な設備ログ（3 テーブル，平均 59 億レコード，平均 79 カラム，平均 102 GB）を DB サーバ 1 に，

表 2 12 個の評価クエリに関する情報：利用するテーブルの種類と数，データ配置別の DB 間の結合処理，クエリの特徴（目的，処理内容，結果件数）

Table 2 Information about 12 evaluation queries: numbers of tables in each table type, existences or scales of join process across DB in each data placement, features of query (purpose, processing and number of results).

	利用するテーブルの種類			DB 間の結合処理		クエリの特徴		
クエリ	ファクト 設備ログ	ファクト その他	マスタ	3DB（データ種別）	3DB（配置分析）	目的	処理	結果件数
1	0	0	4	なし	なし	集計	マスタ同士の Union と結合	2453
2	0	0	4	なし	なし	集計	マスタ同士の Union と結合	7
3	0	0	2	なし	なし	中間データ作成	マスタ同士の結合	0
4	0	2	0	なし	なし	集計	ファクト同士の結合	3499
5	1	0	3	小対小	小対小	集計	ファクトに対するマスタのスター結合	3
6	1	0	1	大対大	なし	中間データ作成	ファクトとマスタの結合	1818
7	1	0	1	大対大	なし	中間データ作成	ファクトとマスタの結合	2842
8	0	1	1	大対大	大対大	集計	ファクトとマスタの結合	4903
9	1	0	4	大対大	小対小	集計	2 種類の集計結果の結合	74
10	0	1	4	大対大	大対小	集計	2 種類の集計結果の結合	18761
11	1	0	4	大対大	小対小	集計	2 種類の集計結果の結合	0
12	0	0	4	なし	なし	中間データ作成	マスタ同士の結合	343694

他のファクトテーブルである基幹網などのログ（4 テーブル，平均 8 億レコード，平均 81 カラム，平均 69 GB）を DB サーバ 2 に，基地局データのマスタテーブル（25 テーブル，平均 5 億レコード，平均 27 カラム，平均 15 GB）を DB サーバ 3 に配置した．ファクトテーブルは 2 つを除きファクトテーブルどうしでは結合せずマスタテーブルと結合する．マスタテーブルは ID キーを利用してファクトテーブルや他のマスタテーブルと結合する．3DB（配置分析）では，各 DB のストレージ容量を単一 DB の約 1/3 である 350 GB とし，通信品質管理に用いた 1 日分のアドホックなクエリ 377 個を対象にデータ配置分析した．ここでは，Vertica で実行された上記クエリのログをクエリログ DB に格納し，1,000 万レコード以上のテーブルどうしの結合演算を対象に，関連するテーブルを 1 つの DB に配置するようクラスタリングした．その結果として，マスタテーブルの 2 テーブル（平均 17 億レコード，平均 15 カラム，平均 21 GB）を DB サーバ 1 に，別のマスタテーブルの 4 テーブル（平均 14 億レコード，平均 36 カラム，平均 18 GB サイズ）を DB サーバ 2 に，その他を 3DB（データ種別）と同様に配置した．

評価で利用するクエリは，データ配置分析に利用した 377 個のクエリから大規模なデータの結合処理を含むクエリを 12 個抽出して実行した．**表 2** に，12 個のクエリそれぞれの情報として，利用するテーブルの種類とその数，クエリに含まれる DB 間の結合処理の種類，クエリの特徴を示す．DB 間の結合処理は，3DB（データ種別）および 3DB（配置分析）のそれぞれで発生する処理を記述している．入力データが 1,000 万レコード以上の場合を大，1,000 万レコード未満を小とし，「小対小」は小規模データどうしの DB 間結合を，「大対小」は片方が大規模なデータの DB 間結合を，「大対大」は大規模データどうしの DB 間結合を示している．クエリの特徴としては各クエリの目的，処理内容，結果件数を示す．各クエリは目的として，データの集計と，別クエリで利用するために一時テーブルに挿入する中間データ作成の 2 つに分類される．なお，評価時は一時テーブルへの挿入を行っていない．処理内容は，結合演算における種類として結合されるデータの種類を示す．

7.2 評価結果要約

図 6 に通信ボトルネックを考慮した場合と，考慮していない場合で 12 個のクエリを実行した結果を示す．縦軸は，単一 DB に対してデータ仮想化システムを通さずに直接実行したときの実行時間に対する比である．この値が 1 であれば，クエリの実行時間が単一 DB と同じであり，通信ボトルネックは完全に解消されている．なお縦軸は対数表示である．通信ボトルネックを考慮した場合として，3 つの手法：クエリプッシュダウン，データ配置分析，中間データ再配置を有効にしたデータ仮想化システムを利用し，考慮しない場合として，従来のデータ仮想化システム（Presto に実装した 3 つの手法を無効にしたもの）を用いた．図 6 から，3 つの手法を用いることで 12 個すべてのクエリで性能が改善され，クエリ 8 を除いた 11 個のクエリでは，従来のデータ仮想化システムで最大で 15,992 倍（クエリ 5），平均で 2010 倍遅延していたが，3 つの手法を用いることで最大で 3.8 倍（クエリ 5），平均で 1.8 倍の遅延に改善されることを確認できた．

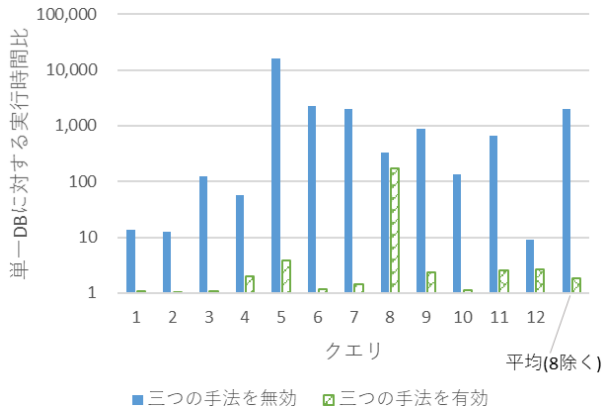


図 6 3 つの手法（クエリプッシュダウン，データ配置分析，中間データ再配置）を有効にした仮想統合環境の評価（対数表記）

Fig. 6 Evaluation results of virtual integrated environment with three methods (query pushdown, data placement analysis and intermediate data replacement) (log notation).

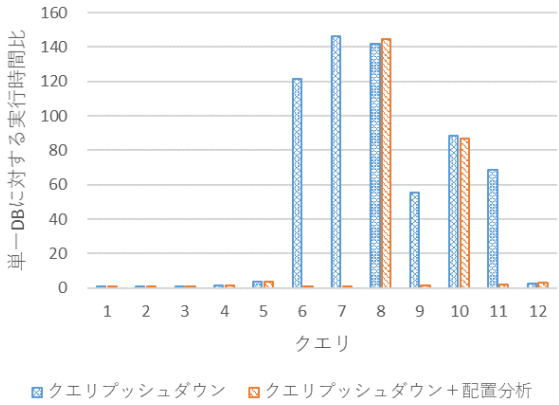


図 8 3DB（データ種別）と 3DB（配置分析）におけるデータ配置分析の評価

Fig. 8 Evaluation for data placement analysis by comparison of results with query pushdown in 3DB (data types) and 3DB (analyzed placement).

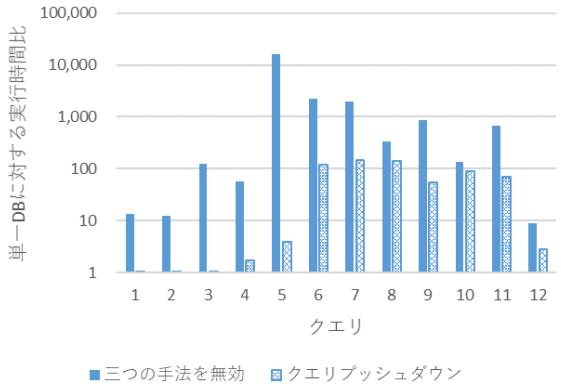


図 7 3DB（データ種別）におけるクエリプッシュダウンの評価（対数表記）

Fig. 7 Evaluation for query pushdown by enabling its function in 3DB (data types) (log notation).

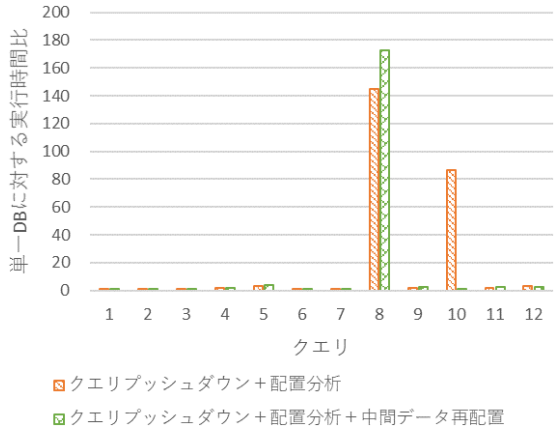


図 9 3DB（配置分析）における中間データ再配置の評価

Fig. 9 Evaluation for intermediate data replacement by enabling its function in 3DB (analyzed placement).

7.3 評価結果詳細

ここでは，7.2 節の結果における 3 つの手法それぞれの効果を評価する．

3 つの手法のうち，データ配置分析と中間データ再配置はクエリプッシュダウンを前提とした手法のため，まず初めにクエリプッシュダウンの効果を評価する．**図 7** は，7.2 節と同様に 3DB の環境において，従来のデータ仮想化システムと，クエリプッシュダウンのみを有効にした場合に，12 個のクエリを実行したときの結果である．この結果から，クエリプッシュダウンは 12 個すべてのクエリで効果が確認できる．なお，クエリ 6 から 11 の 6 個は，他のクエリと比較して効果が少なく，約 60 倍から 180 倍の大幅な遅延が見られた．

次にクエリプッシュダウンの機能を有効にした状態で，データ配置分析の効果を評価する．**図 8** は，クエリプッシュダウンのみを有効にした 3DB（データ種別），およびクエリプッシュダウンとデータ配置分析を有効にした 3DB（配置分析）で 12 個のクエリを実行した結果である．縦軸は図 7 と同様に実行時間比だが，対数表記ではない．データ配置分析の効果により，4 個のクエリで性能が改善されたが，2 つのクエリでは 3DB（データ種別）と同様に大幅な遅延が見られる．

最後にデータ仮想化の中間データ再配置を分析するため，データ仮想化の実行結果を比較する．**図 9** は，データ配置分析した結果である 3DB（配置分析）において，クエリプッシュダウン機能のみと，加えて中間データ再配置機能を有効にしたデータ仮想化システムを実行した結果で，データ配置分析の評価と同様に縦軸は単一 DB に対する実行時間比を表している．データ配置分析で大幅な遅延が回避できなかった 2 つのクエリのうち，クエリ 10 に関して，中間データ再配置によって遅延を回避している．一方で，クエリ 8 に関しては中間データ再配置が発生しているが，発生しない場合と比べてさらに遅延が大きくなっている．

7.4 考察

7.4.1 手法ごとの考察

評価結果として，データ配置分析とデータ仮想化におけるクエリプッシュダウンおよび中間データ再配置の 3 つの手法を用いることで，実環境における 12 クエリ中 11 個のクエリで大幅な遅延なく，単一の DB を 3 つの DB に分割できることを示した．

クエリプッシュダウンの効果は特に大きく，全クエリで通信ボトルネックの解消に寄与した．単一の DB で完結するクエリは結果の件数が最大でも 34 万レコードであり，これがクエリプッシュダウンにおける通信量となる．一方で全機能 OFF では，億単位のレコードのテーブルを通信するため，クエリプッシュダウンによって大幅に通信量を削減した．複数の DB にあるデータを使うクエリにおいても，それぞれの DB で処理を施すことで通信量を削減した．しかし，表 2 の DB 間の結合処理で示したように，1,000 万レコード以上の大規模なデータの通信が発生するクエリ 6 から 11 は，依然 10 倍以上の大きな遅延が発生している．したがって，各 DB で処理した結果が小さくならない限り，クエリプッシュダウン単体での改善は限定的である．

クエリプッシュダウンに加えて，データ配置分析によってデータ配置を改善することで，クエリ 6，7，9，11 の 4 つで大幅な改善が見られた．これは，DB サーバ 1 に移動したマスタデータによって，1,000 万レコード以上の大規模な結合演算をクエリプッシュダウン可能にしたことによる改善である．一方で，2 つのクエリは性能が改善されていない．クエリ 8 は利用するデータ配置が変わらなかったことに起因する．またクエリ 10 では，DB サーバ 2 へのマスタデータ移動が発生し，片方の DB でクエリプッシュダウンによって結果が小さくなったが，もう一方が支配的に大きく，依然として遅延が発生している．

ここでさらに中間データ再配置を追加することで，クエリ 10 は改善された．これは，片方のデータが 1,000 万レコード以上の大規模となる DB 間の結合演算において，もう一方の小規模なデータを移動して大規模なデータとともに処理することで，通信量を削減できたことに起因する．その結果，データ配置分析では回避できなかった大規模データ通信を含むクエリの通信遅延も，中間データ再配置で回避できることを示した．以上のことから，データ仮想化による仮想統合環境において，これら 3 つの手法の組合せが有効であるといえる．

7.4.2 クエリごとの考察

クエリ 8 に関しては，3 つの手法を用いたとしても大幅な遅延を回避できていない．3 つの手法によって 328 倍から 172 倍の遅延に改善されたが，依然遅延が大きく，特に中間データ再配置によりむしろ遅延が発生している．このクエリは，2 つのテーブルを Left Outer Join するクエリで，片方の入力は約 6 GB，もう一方が約 30 GB，出力は約 5 GB となる．なお結合後に集計処理があり，最終的なクエリの結果は約 70 KB（4,903 レコード）と小さい．大幅な遅延の原因は，データ配置分析と，データ仮想化の中間データ再配置の双方にあると考えられる．データ配置分析に関しては，データ配置の改良によってクエリ 8 を改善でき，データ配置分析に改善できる可能性があることが分かった．一方でデータ仮想化の中間データ再配置に関しては，他の処理方式を用いてもクエリ 8 の改善が見込めないことが分かった．以下で詳細を述べる．

データ配置分析に関しては，ストレージ容量の制約から，すべての大規模な結合関係にあるテーブルを 1 つの DB に集約できるとは限らない．本評価では，大規模な結合演算として 1,000 万レコード以上の通信が発生する結合演算を対象に，結合の頻度が多いテーブルの組を優先的に同じ DB に配置している．したがって，結合演算にともなう通信量の多寡は対象の結合演算の選択に用いるのみであり，特に通信量の多い結合演算の頻度が少ない場合の影響は考慮できていない．クエリ 8 においては，その他のファクトデータとマスタデータが，結合の頻度が少ないが通信量が特に多いテーブルの組に該当する．この影響を考慮に入れて，3DB（配置分析）環境において，クエリ 8 で利用される DB サーバ 2 のその他のファクトデータを，マスタデータが配置されている DB サーバ 1 に移動し，その代わりに容量制約に従って，クエリ 5 で利用される DB サーバ 1 の設備ログのファクトデータを DB サーバ 2 に移動し，それぞれのクエリを再計測した．なお，クエリ 5 と 8 以外のクエリには影響していない．再計測の結果，クエリ 8 は単一 DB に対する実行時間比が 1.06 倍に大幅に改善された．この結果から，頻度が少なくかつ通信量が多い結合演算を考慮することで，データ配置分析を改善できる可能性があると考えられる．

中間データ再配置に関しては，通信量をベースに実施可否を判定しているため，それ以外のオーバヘッドが考慮できていない．クエリ 8 の通信量は，中間データ再配置する場合はデータ仮想化システムと各 DB 間で $6\,\mathrm{GB} \times 2 + 70\,\mathrm{KB} = 12\,\mathrm{GB}$ となり，データ仮想化システムで結合処理をする場合は $30\,\mathrm{GB} + 6\,\mathrm{GB} = 36\,\mathrm{GB}$ となった．通信量での計算上は中間データ再配置することで通信量の削減が期待できるが，データ仮想化システムにおける DB への転送処理や，DB への一時ストレージ書き込み（COPY），結合処理を含むサブクエリ実行時の一時ストレージ読み込みの遅延が無視できず，結果としてデータ仮想化システムで処理する場合より性能が劣化した．対応策としては，中間データ再配置する条件に上記のコストを含めて厳格化する点が考えられる．ただし，本対応策は中間データ再配置によるさらなる性能劣化を防ぐものであり，根本的な性能劣化の改善には至らない．別の改善策として，中間データ再配置手法の 2-way Semijoin の導入によ

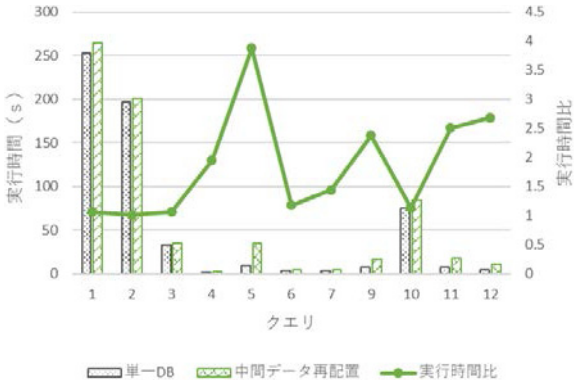


図 10 クエリ 8 を除いた単一 DB と中間データ再配置の実行時間比較

Fig. 10 Comparison of execution times of evaluation queries except for query 8 in single DB and 3DB (analyzed placement) with our three methods.

る改善が考えられる．2-way Semijoin では結合直後の容量が通信量に影響するため，本クエリでの通信量は結合キー（2 GB）の通信と，結合後の結果（5 GB）の通信となる．つまり通信量は 2 GB × 2 + 5 GB × 2 = 14 GB となり，本実装での中間データ再配置手法である Ship Join より通信量が大きくなる．したがって，本クエリに限っては改善を期待できない．

次に，クエリ 8 以外のクエリを分析する．中間データ再配置を有効にした場合のクエリ 8 を除いた 11 個のクエリの実行結果を**図 10** に示す．Vertica で直接実行した単一 DB と，3DB（配置分析）環境における中間データ再配置を有効にしたデータ仮想化システムの実行時間を棒グラフで，これらの実行時間比を線グラフで示している．クエリ 1，2，3，4，6，7，12 の 7 つは 3DB（配置分析）環境でも 1 つの DB にデータが集まっており，すべての演算をクエリプッシュダウンしている．このうちクエリ 4，6，7，12 の実行時間比が，クエリ 1，2，3 と比較して悪化している原因は，実行時間の短さに対して，通信量が大きいためである．特にクエリ 12 の結果は約 34 万件と大きく，データ仮想化システムと DB との通信によって，他のクエリより遅延が大きい．

クエリ 9，10，11 の 3 つは，異なる DB にデータがまたがっているクエリであるが，中間データ再配置が機能し，1.1 から 2.5 倍程度の遅延となった．クエリ 9，11 の遅延がクエリ 10 と比較して大きい理由は，実行時間が短く，中間データ再配置の通信の影響を相対的に大きく受けているためである．

クエリ 5 は遅延が 3.87 倍と，他と比較して遅延が大きい．これはデータ配置の影響ですべての結合処理がデータ仮想化システムで実行されたためである．中間データ再配置が行われなかった理由は，結合演算の入出力データサイズから行わないことで通信量を抑えられると判定されたためである．一方で，表 2 で示すように DB 間で発生する結合演算は小規模なデータの結合であり，最大でも約 250 MB（37 万レコード）と比較的小さいため，前述のクエリ 8 ほどの大幅な遅延は発生していない．またクエリ 5 の遅延はクエリ 8 と同様にデータ配置分析の工夫により回避することができ，前述のデータ配置を変更した追加評価により，3.87 倍の遅延から 1.33 倍の遅延に改善された．

本評価ではアドホックなクエリの単体実行を対象としているが，クエリのバッチ実行や，複数のクエリの同時実行を考慮に入れる場合には，データ仮想化システムにおける中間結果のキャッシング [27] や，複数のクエリで共通となる処理を統合するワークフロー管理 [28] の検討が必要である．また，本論文ではクエリログの分析を静的に DB 分割時にのみ利用したが，クエリの傾向変化に対応した動的なクエリ配置分析 [3] の検討も必要である．

8. おわりに

本論文では，データ仮想化における通信ボトルネックが 3 つの手法：データ配置分析，クエリプッシュダウン，中間データ再配置によって解消されることを示した．評価実験の結果，単一 DB の場合と比較して，12 個中 11 個のクエリで 4 倍を超える大幅な遅延を発生することなくデータ分析クエリを実行できることを示した．残り 1 つのクエリに関しても，頻度は低いが通信量が多い結合演算を考慮したデータ配置分析によって回避可能であることを示した．これにより，通信ボトルネックが解消できることから，単一 DB と異なりリソース設計の複雑化やメンテナンスコストの増加を抑制できる，単一の DB を複数の DB に分割してデータ仮想化で仮想統合するアーキテクチャが有効であると考えられる．

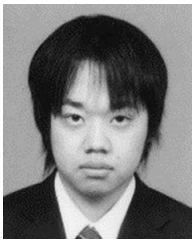
参考文献

[1] van der Lans, R.F.: *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann (2012).
[2] Presto, available from ⟨https://prestodb.io/⟩ (accessed 2017-04-28).
[3] Kossmann, D., Franklin, M.J. and Drasch, G.: Cache Investment: Integrating Query Optimization and Distributed Data Placement, *ACM Trans. Database Systems*, Vol.25, No.4, pp.517–558 (2001).
[4] Iyer, S. and Cheung, A.: Summarizing Source Code using a Neural Attention Model, *Proc. 54th Annual Meeting of the Association for Computational Linguistics*, pp.2073–2083 (2016).
[5] Yu, P. and Chen, M.: On Workload Characterization of Relational Database Environments, *IEEE Trans. Softw. Eng.*, Vol.18, No.4, pp.347–355 (1992).
[6] Yao, Q., An, A. and Huang, X.: Mining and Modeling Database User Access Patterns, *Proc. 16th International Symposium on Methodologies for Intelligent Systems*, *LNAI 4203*, pp.493–503 (2006).
[7] Elmeleegy, H., Elmagarmid, A. and Lee, J.: Leveraging Query Logs for Schema Mapping Generation in U-MAP,

Proc. 2011 ACM SIGMOD International Conference on Management of Data, pp.121–132 (2011).
[8] Ganapathi, A., Kuno, H., Dayal, U., et al.: Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning, *Proc. IEEE 25th International Conference on Data Engineering*, pp.592–603 (2009).
[9] Celko, J.: *Trees and Hierarchies in SQL for Smarties, 2nd Edition*, Morgan Kaufmann (2012).
[10] TPC-H, available from ⟨http://www.tpc.org/tpch/⟩ (accessed 2017-04-28).
[11] Vertica, available from ⟨https://www.vertica.com/⟩ (accessed 2017-04-28).
[12] PostgreSQL FDW, available from ⟨https://www.postgresql.org/docs/current/static/postgres-fdw.html⟩ (accessed 2017-04-28).
[13] Oracle Database Link, available from ⟨http://docs.oracle.com/database/122/ADMIN/distributed-database-concepts.htm#ADMIN12084⟩ (accessed 2017-04-28).
[14] Sheth, A.P. and Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, *ACM Computing Surveys*, Vol.22, No.3, pp.183–236 (1990).
[15] Ramakrishnan, R. and Gehrke, J.: *Database Management Systems, 3rd Edition*, McGraw-Hill (2003).
[16] Haas, L.M., Kossmann, D., Wimmers, E., et al.: Optimizing Queries across Diverse Data Sources, *Proc. 23rd International Conference on Very Large Data Bases*, pp.276–285 (1997).
[17] Ozsu, M.T. and Valduriez, P.: *Principles of Distributed Database Systems*, Third Edition, Springer (2011).
[18] Nural, S., Koksal, P., Ozcan, F., et al.: Query Decomposition, Optimization and Processing in Multidatabase Systems, *Proceedings of Symposium of the European Joint Conference on Engineering Systems Design and Analysis*, pp.41–52 (1996).
[19] Doan, A., Halevy, A. and Ives, Z.: *Principles of Data Integration*, Morgan Kaufmann (2012).
[20] Li, Z. and Ross, K.A.: PERF Join: An Alternative to Two-way Semijoin and Bloomjoin, *Proc. 4th International Conference on Information and Knowledge Management*, pp.137–144 (1995).
[21] Polychroniou, O., Sen, R. and Ross, K.A.: Track join: Distributed joins with minimal network traffic, *Proc. 2014 ACM SIGMOD International Conference on Management of Data*, pp.1483–1494 (2014).
[22] Kossmann, D. and Stocker, K.: Iterative Dynamic Programming: A New Class of Query Optimization Algorithms, *ACM Trans. Database Systems*, Vol.25, No.1, pp.43–82 (2000).
[23] Moerkotte, G. and Neumann, T.: Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products, *Proc. 32nd International Conference on Very large Data Bases*, pp.930–941 (2006).
[24] DeHaan, D. and Tompa, F.W.: Optimal Top-down Join Enumeration, *Proc. 2007 ACM SIGMOD International Conference on Management of Data*, pp.785–796 (2007).
[25] Fender, P. and Moerkotte, G.: Counter Strike: Generic Top-Down Join Enumeration for Hypergraphs, *Proc. VLDB Endowment*, Vol.6, No.14, pp.1822–1833 (2013).
[26] Bruno, N. and Chaudhuri, S.: Exploiting Statistics on Query Expressions for Optimization, *Proc. 2002 ACM SIGMOD International Conference on Management of Data*, pp.263–274 (2002).
[27] Perez, L.L. and Jermaine, C.M.: History-aware Query Optimization with Materialized Intermediate Views, *Proc. IEEE 30th International Conference on Data Engineering*, pp.520–531 (2014).
[28] Makreshanski, D., Giannikis, G., Alonso, G., et al.: MQJoin: Efficient Shared Execution of Main-Memory Joins, *Proc. VLDB Endowment*, Vol.9, No.6, pp.480–491 (2016).
[29] Lane, H., Thompson, G.R., Chung, C.-W., Barkmeyer, E., et al.: Heterogeneous Distributed Database Systems for Production Use, *ACM Computing Surveys*, Vol.22, No.3, pp.237–266 (1990).
[30] Ahmed, R., Smedt, P.D., Du, W., et al.: The Pegasus Heterogeneous Multidatabase System, *IEEE Computer*, Vol.24, No.12, pp.19–27 (1991).
[31] Chawathe, S., Garcia-Molina, H. and Hammer, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources, *Journals of Intelligent Information System*, Vol.8, No.2, pp.117–132 (1994).
[32] Carey, M.J., Haas, L.M., Schwarz, P.M., et al.: Towards Heterogeneous Multimedia Information Systems: The Garlic Approach, *Proc. 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management*, pp.124–131 (1995).
[33] Emery, R., Anne, S.A., Lu, J.J., et al.: *HERMES?: A Heterogeneous Reasoning and Mediator System*, Submitted for publication (1995), available from ⟨http://www.cs.umd.edu/projects/hermes/overview/paper⟩.
[34] Fiore, S., Negro, A. and Aloisio, G.: Data Virtualization in Grid Environments through the GRelC Data Access and Integration Service, *Proc. International Conference for Internet Technology and Secured Transactions*, pp.817–822 (2009).
[35] Salloum, M. and Dong, L.: Online Ordering of Overlapping Data Sources, *Proc. VLDB Endowment*, Vol.7, No.3, pp.133–144 (2013).
[36] Karpathiotakis, M., Alagiannis, I. and Heinis, T.: Just-In-Time Data Virtualization: Lightweight Data Management with ViDa, *Proc. 7th Biennial Conference on Innovative Data Systems Research* (2015).

齋藤 和広 （正会員）

KDDI 総合研究所．2010 年成蹊大学大学院工学研究科博士課程前期修了．同年 KDDI 株式会社入社．現在，株式会社 KDDI 総合研究所統合分析プラットフォームグループ研究主査．大規模データ活用に向けた分散システム，データベースシステムの研究開発に従事．日本データベース学会会員．



米田 信之 （正会員）

KDDI 総合研究所．2009 年岩手県立大学大学院ソフトウェア情報学研究科修士課程修了．同年 KDDI 株式会社入社．株式会社 KDDI 総合研究所クラウドプラットフォームグループにて，データベースシステム，ハイパフォーマンスコンピューティングの研究開発に従事．現在，KDDI 株式会社．

渡辺 泰之 （正会員）

KDDI 総合研究所．1995 年東京理科大学大学院理工学研究科情報科学専攻修士課程修了．同年国際電信電話株式会社（現，KDDI 株式会社）入社．株式会社 KDDI 総合研究所クラウドプラットフォームグループにて，分散システム，データベースシステム等の研究開発に従事．現在，KDDI 株式会社．日本データベース学会会員．

黒川 茂莉 （正会員）

KDDI 総合研究所．2005 年慶應義塾大学理工学部管理工学科卒業．2007 年同大学院理工学研究科修士課程開放環境科学専攻修了．同年 KDDI 株式会社へ入社．現在，株式会社 KDDI 総合研究所統合分析プラットフォームグループ研究主査．データマイニング，位置情報履歴の分析等の研究に従事．

村松 茂樹 （正会員）

KDDI 総合研究所．1999 年東京大学大学院工学系研究科電子情報工学専攻修士課程修了．同年 KDD 株式会社（現，KDDI 株式会社）入社．現在，株式会社 KDDI 総合研究所フューチャーデザイン 2 部門 3 グループアナリスト．位置推定，行動認識，データマイニング，データベースシステム等の研究開発に従事．本会 2011 年度論文賞受賞．

小林 亜令 （正会員）

KDDI 総合研究所．1973 年鹿児島生まれ．1998 年北海道大学大学院工学研究科修士課程修了．同年 KDD 株式会社（現，KDDI 株式会社）入社．現在，株式会社 KDDI 総合研究所統合分析プラットフォームグループリーダー．これまで XML，SVG，ITS，通信放送融合技術，センサデータマイニング，等の研究開発に従事．2003 年 FIT2003 船井記念ベストペーパー賞受賞，2008 年 ARIB 電波功績賞受賞．本会 UBI 研究会運営委員．