Vol.2013-DBS-157 No.19
Vol.2013-IFAT-111 No.19
2013/7/23

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

片山 大河[1,a)]　嶋村 誠[1,b)]　金松 基孝[1,c)]

概要：大量データの高速処理に向く NoSQL 型データベースとトランザクション処理に向く RDB など、用途に応じて複数のデータベースが併用されるが、アプリケーション開発者はそれらの API や SQL を意識して実装しなければならない。異なるデータベースの表に対する横断的な、データの結合や集計などが SQL1 文で実行可能なシステムを開発した。各データベースに対応するドライバが、SQL の言語仕様の違いやアーキテクチャの違いを感じさせない検索を実現する。性能は，データベース上である程度結果を絞れる検索では，直接データベースに接続する場合と比較して 4 倍程度のオーバーヘッドで，競合製品と比較して 1.5 倍高速である．

キーワード：データベースシステム，統合データベース，クエリ分割，高速化

Katayama Taiga[1,a)]　Shimamura Makoto[1,b)]　Kanematsu Mototaka[1,c)]

1. はじめに

NoSQL（Not only SQL）型データベースの普及に伴い，ひとつのシステムで複数の異なる種類のデータベースが併用されてきている．例えば，更新頻度が低いが一貫性の保障が必要なメタ情報などはリレーショナルデータベース（RDB）に格納し，時系列データなどの大量データは NoSQL 型データベースに格納するというように，用途に応じて格納先データベースを使い分けることがある．

このように異種データベースを併用する場合，アプリケーション開発者はそれらシステムが提供する API や SQL の方言を意識して実装しなければならない．さらに，データベース間を跨るデータに対する演算は，アプリケーション上で実現しなければならない．これを解決するシステムとして統合データベースがある．

統合データベースはアプリケーションとデータベースを中継するミドルウェアである．複数の異種データベースを仮想的な一つのデータベースとみなして，SQL の言語仕様の違いやアーキテクチャの違いを感じさせない検索が可能となる．一方，問題点として，アプリケーションから見ると通常のデータベースに対する検索と比べて 1 レイヤー多くなることから，検索性能の劣化が懸念される．今回，その性能劣化を抑えるために，データ取得の効率化により高速な検索を実現した．

第 2 章で統合データベース上の処理の流れを説明し，第 3 章で従来手法と高速化手法を述べる．第 4 章で提案手法の効果と競合製品との比較についての実験について述べて，第 5 章でまとめる．

2. 統合データベースの概要

統合データベースは，複数の異種データソースに分散されたテーブルへの検索（ジョイン，マージ，ソート，フィルタ，集計などの演算を含む）が 1 つの SQL 文で簡単に実現可能なシステムである．IBM 社の InfoSphere Federation Server[1] や日立製作所の Cosminexus[2] もその一種である．RDB に限らず，NoSQL 型データベース，その他非構造データをテーブルとみなして，それらのシステム（以降，データソースと呼ぶ）にアクセスできる．今回開発した統合 DB は，RDB では PostgreSQL[3] および IBM の DB2[4]，東芝ソリューションの NoSQL 型データベースエンジン [5] をサポートする．

[1] 株式会社東芝 ソフトウェア技術センター, Kawasaki, Kanagawa 212–8582, Japan
[a)] taiga.katayama@toshiba.co.jp
[b)] makoto.shimamura@toshiba.co.jp
[c)] mototaka.kanematsu@toshiba.co.jp

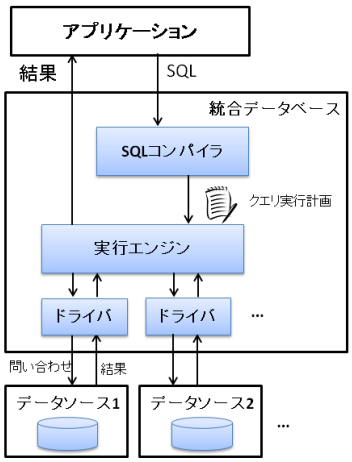

図 1 統合データベースのモジュール構成
Fig. 1 Module structure of integrated database.

2.1 統合データベースの構成

統合データベースの主なモジュールは，SQL コンパイラ，実行エンジン，ドライバから構成される（図 1）．

SQL が与えられると，次のように動作する．まず，SQL コンパイラがクエリ実行計画（実行エンジンで実行されるプログラム）を生成する．この処理では，テーブルスキーマをもとに SQL で使用されている列の所属（どのデータソースのどのテーブルか）を判定したり，演算の実行手順を決定する．このとき，与えられた 1 つの SQL は，データソース次第で複数の検索要求に分割される．

その後，実行エンジンがクエリ実行計画を実行する．実行の過程でデータソース固有の処理が必要になると，ドライバに処理を委ねる．ドライバは，データソースのインスタンスと直接やり取りを行うモジュールである．各種データソース向けのドライバを用意することで，言語仕様やアーキテクチャの違いを吸収する．例えば，ドライバ経由でデータソースに問い合わせてデータを取得したりする．

ドライバが行う機能には，例えば以下に挙げるようなものがある．

- データソースへの問い合わせ
- 統合データベース向けのデータ型に問い合わせ結果を変換
- テーブルのスキーマ取得
- データソースへの接続および切断

ドライバによってデータソースからデータを取得した後，統合データベース上でジョインなどの演算を行って，最終結果を生成する．

3. 高速化手法

直接データソースに接続して検索する場合と比べて，統合データベースを導入することによる性能のボトルネックはデータ転送量である．不要なデータをデータソースから統合データベースに転送しないようにして高速化する．

表 1 データソース 1 上のテーブル $t1$

Table 1 Table t1 on source1.

$a1$	$a2$	$a3$	$a4$
1	A	1	6
2	A	2	5
3	B	3	4
4	B	4	3
5	C	5	2
6	D	6	1

表 2 データソース 2 上のテーブル $t2$

Table 2 Table t2 on source2.

$b1$	$b2$
A	111
B	555
C	999
D	222

表 3 最終結果

Table 3 Final result.

$a1$	$f(a3, a4)$
1	1
2	32
6	6

3.1 従来の手法

SQL が与えられると，その SQL に使用されている各列に対する全レコードをデータソースから取得する．そのデータを用いて，統合データベースが次のような処理を行なって最終結果を生成する．SQL 関数が含まれている場合は統合データベース上で計算したり，条件式がある場合は統合データベース上でその計算式の演算を行う．

例えば，データソース 1 およびデータソース 2 にはそれぞれ図 1，図 2 のようなテーブルがあるとする．入力 SQL として，

$$\begin{aligned}&\text{SELECT}\quad a1, f(a3, a4)\quad \text{FROM}\quad t1, t2\\ &\qquad\qquad \text{WHERE}\quad b2 < 123\quad \text{AND}\quad a2 = b1 \quad (1)\end{aligned}$$

が与えられるとする．このとき，統合データベースはまずデータソース 1 から $a1, a2, a3, a4$，データソース 2 から $b1, b2$ を取得する．次に，統合データベース上で $b2 < 123$ のフィルタリング，SQL 関数 $f(a3, a4)$ の計算，$a2 = b1$ によるジョインを行う．最終的に図 3 のような結果を生成する．

3.2 データ取得の効率化

データ転送量を削減して高速化するために，条件式と SQL 関数をデータソース上で計算させる．字句解析の過程では，SQL 中のトークン情報が取得可能なことを利用して，列名から条件式や SQL 関数の所属を決定しデータソースに問い合わせるように SQL コンパイラを工夫する．これから条件式や SQL 関数をデータソース上で計算させる方法について説明する．

入力された SQL 中の条件式を論理積の集合とみなす．その各項，つまり AND で分割したときの各要素（以下，

条件項と呼ぶ）に対して，所属データソースを判定して，適切なデータソース上で計算させる．

条件項の例を挙げると，条件式が「$a > 0$ AND $(b > 2$ OR $c > 3)$ AND $(d > 4$ OR $(e > 5$ AND $f > 6))$」の場合，3 つの条件項（「$a > 0$」，「$b > 2$ OR $c > 3$」および「$d > 4$ OR $(e > 5$ AND $f > 6)$」に分割される．

一方，SQL 関数については，引数からその関数の所属を判定する．

このようにして所属判定した結果，引数が同一データソースに所属する場合，その条件項や SQL 関数はそのデータソース上で計算させる．なお，リテラル値は無視するが，全てのトークンや引数がリテラル値の場合、統合データベース上で計算する．それ以外の場合，つまり所属データソースが複数の場合，統合データベース上で計算する．

3.3 条件式および SQL 関数の所属判定

条件項や SQL 関数の所属判定方法について説明する．

条件項の所属判定では，要素中の列名からそれぞれのデータソースを取得する．SQL コンパイラでは，SQL パーサによって SQL 文がトークン（SQL を分割した際，意味のある最小単位）に分割される．それぞれのトークンに対して，所属データソースを判定する（図 2〜図 4）．

条件項で使用されるデータソースが同一である場合，その条件項の所属はそのデータソースとなる．

一方 SQL 関数の所属判定では，各引数に対して，所属データソースを判定して，全引数の所属データソースが同一である場合はその SQL 関数の所属はそのデータソースとなる．引数自体の所属判定方法は，条件項の判定方法と同様である．

例えば，式 (1) が与えられたとする．

条件項「$b2 < 123$」はデータソース 2 の列 $b2$ だけが使用されているので，データソース 2 で計算する．条件項「$a2 = b1$」はデータソース 1 の列 $a2$ およびデータソース 2 の列 $b1$ が使用されているので，統合データベース上で計算する．SQL 関数 $f(a3, a4)$」は，データソース 1 だけの列 $a3, a4$ が使用されるので，データソース 1 で計算する．

従って，データソース 1 に対しては「SELECT $a1, a2, f(a3, a4)$ FROM $t1$」，データソース 2 に対しては「SELECT $b1$ FROM $t2$ WHERE $b2 < 123$」のようにて問い合わせる．その結果，図 5，図 6 を統合データソースが取得し，最終的に $a2 = b1$ で結合した図 3 を生成する．なお，図 5 では，関数 f は power 関数としている．

このように，高速化した場合の方が，データソース 1 に対して取得する列数と，データソース 2 に対する行数および列数が小さくして，データ転送量を小さくできる．

4. 実験

開発した統合データベースの性能を評価するために実験を行った．

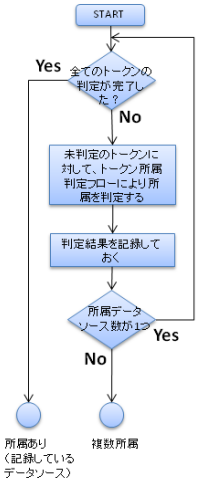

図 2 条件項の所属判定
Fig. 2 Affiliation judgement of term.

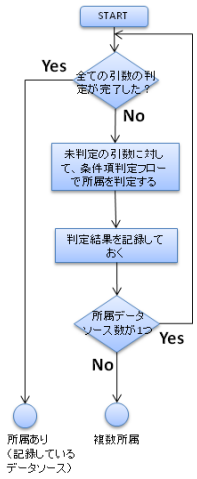

図 3 SQL 関数の所属判定
Fig. 3 Affiliation judgement of SQL function.

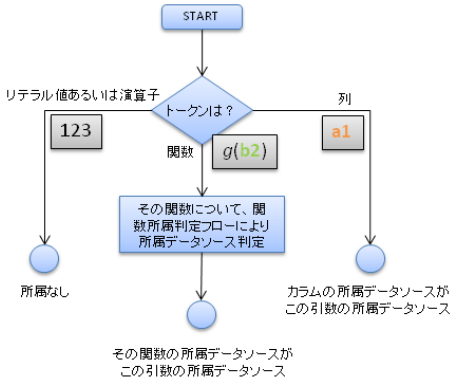

図 4 トークンの所属判定
Fig. 4 Affiliation judgement of token.

図 5 データソース 1 への問い合わせ結果
Fig. 5 Result of source1.

$a1$	$a2$	$f(a3, a4)$
1	A	1
2	A	32
3	B	81
4	B	64
5	C	25
6	D	6

図 6 データソース 2 への問い合わせ結果
Fig. 6 Result of source2.

$b1$
A
D

4.1 実験手法

以下の 4 種類のケースで実行時間を比較した．

(1) 高速化しない場合 (Original)
(2) 高速化した場合 (Proposed)
(3) 直接データソースに接続した場合 (Direct)
(4) 他社製品を用いた場合 (OtherProduct)

実験に使用したマシンの構成は図 7 に示す．使用した

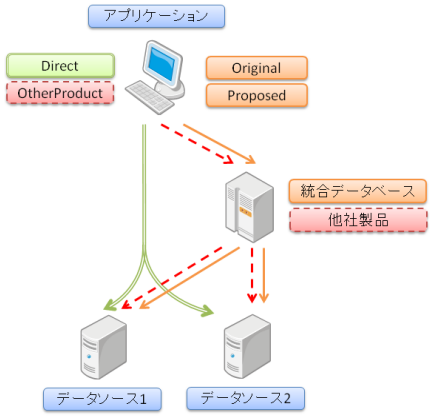

図 7 マシン構成
Fig. 7 Machine organization.

表 4 フライト情報テーブル
Table 4 Table of flight information.

機体	出発空港 ID	到着空港 ID
A333	NRT	TPE
B763	HNL	HND
B773	SIN	LHR
...	...	...

表 5 空港情報テーブル
Table 5 Table of airport information.

空港 ID	空港名	所在地
NHL	Honolulu International	Hawaii, United States
HND	Haneda	Tokyo,Japan
FSZ	Mt. Fuji Shizuoka	Shizuoka, Japan
...	...	...

データは，飛行機のフライト情報が格納されたテーブルと，空港情報が格納されたテーブルを用意した．フライト情報には，ある時刻における世界中に飛んでいる飛行機の機体名，出発空港 ID，到着空港 ID が 1987 件データソース 1 に（表 4）格納されている．一方，空港情報には，世界中の空港 ID，空港名，所在地が 6861 件データソース 2（表 5）に格納されている．なお，所在地の末尾に国名が格納されている．

以下に示すような，羽田空港から出発する国内線の各飛行機について到着空港の情報を取得する SQL の実行時間を測定した．

```
SELECT * FROM フライト情報 F, 空港情報 A
    WHERE A.ID = F. 到着地
    AND F. 出発地 = ' 羽田空港'
    AND RIGHT(所在地, 5) = 'Japan'
```

「F. 出発地 = ' 羽田空港'」の条件には 1987 件中 31 件，「RIGHT(所在地, 5) = 'Japan'」の条件には 6861 件 66 行ヒットし，最終的に 8 件の検索結果が得られる問い合わせである．

表 6 実行結果
Table 6 Experiment result.

Implementation	Time(msec)
Original	5354
Proposed	94.29
Direct	23.51
OtherProduct	137.7

4.2 実験結果

実験結果を表 6 に示す．提案手法による実行時間は 94.29msec であった．

高速化を実施しない場合は 5354msec であり，提案手法により約 57 倍の高速化を達成でき，データ転送量削減による効果が確認できた．

直接データソースに接続しアプリケーション上で演算を実施する場合の実行時間は 23.51msec であった．つまり，統合データベースを介するオーバヘッドは，4 倍の性能劣化程度であることが分かった．

また，高速化したことにより他社製品より約 1.5 倍高速であった．他社製品は条件式をデータソースで実行できていないことが性能低下の原因と思われる．しかし，高速化を実施しない場合の 5354msec よりもはるかに高速である．この性能差は，他社製品が高速な結合アルゴリズムを採用しているからだと考えている．

5. おわりに

本稿では，統合データベースおよびデータソース間のデータ転送量の削減による高速化手法を提案した．条件式の一部や SQL 関数の所属データソースを判定し，そのデータソースで計算させるように工夫した．その結果として，高速化効果は約 57 倍を達成し，統合データベースのオーバヘッドは 4 倍の性能劣化であることを示した．

今後の展望としては，今回の手法では条件項や関数の引数が複数のデータソースに跨る場合に高速化できないため，そのような場合にも有効な手法を検討したい．また，高速な結合処理アルゴリズムを導入し，さらに高速化したいと考えている．

参考文献

[1] IBM InfoSphere Federation Server, http://www-03.ibm.com/software/products/us/en/ibminfofedeserv/ (2013.06.12).
[2] 日立製作所 Cosminexus, http://www.hitachi.co.jp/Prod/comp/soft1/cosminexus/index.html (2013.06.13).
[3] The PostgreSQL Global Development Group, http://www.postgresql.org/ (2013.06.19).
[4] IBM InfoSphere Federation Server, http://www-01.ibm.com/software/data/db2/ (2013.06.19).
[5] 東芝ソリューション NoSQL 型データベースエンジン, http://www.toshiba-sol.co.jp/pr/t-soul/vol04_1.htm (2013.06.19).