D-022

列指向型データストア機能を有するRDBMSの問合せ処理コスト計算法

Calculating Query-Processing Cost of RDBMS with Column-oriented Data Store Function

塩井隆円 † Takamitsu Shioi
波多野賢治 ‡ Kenji Hatano

1. はじめに

これまで情報システムに利用されてきた Database Management System (DBMS) では，基幹系業務データを効率良く管理するために Relational Database Management System (以下，RDBMS) が採用されてきた．基幹系業務システムでは DBMS に求められるデータ処理の中でも Online Transactional Processing (OLTP) が多く，OLTP を効率良く処理するために行指向型ストレージが採用されている．表形式で管理するデータを一行ずつ格納する行指向型ストレージは，日常業務で発生するトランザクションを一行ずつ検索，格納する際の処理を高速に実行することが可能である．しかしながら，近年 DBMS に求められる処理として着目されている Online Analytical Processing (OLAP) が実行されるため，表形式で管理するデータの中でも集計処理を対象として少数の列データのみを利用することが多い [1]．そのため，一列ごとにデータを格納することで処理に不要な列データを取得せずに Random Access Memory (RAM) に読み込むデータ量を減らし，データ型の一致する列データをソートすることで格納するデータの圧縮率を高めることが可能な列指向型ストレージが OLAP に最適とされている [2,3]．そのため，近年では効率的な意思決定支援システムのために列指向型ストレージを採用した DBMS が運用されており，さらに，行指向型／列指向型の二種類のストレージを一つの DBMS に採用することで OLTP と OLAP 双方の処理最適化に焦点が置かれている [4,5]．

以上の背景を受け，近年の RDBMS では行指向型ストレージにデータを格納した上で列指向のインデックスを作成することで OLAP の高速な実行を図っており，列指向型 DBMS を参考にした二種類の列指向のインデックスが採用されている §．これらの列指向のインデックスは，RDBMS に格納される行指向型データのテーブルから列ごとにデータを分割することで列指向型データのテーブルを新たに格納している．また，各列の複数行のデータをまとめたセグメントと呼ばれる単位の中で最大値，最小値，行数をメタデータとして格納して問合せ処理に利用するといった，列指向型 DBMS を参考にした OLAP 高速化を図っている [6,7]．しかしながら，通常利用されている列指向のインデックスでは，列指向型データに対する統計情報を持たず，問合せ処理の際にコストベースオプティマイザが正確に問合せの実行計画を生成することができない．

そこで，本稿では行指向型ストレージと列指向型ストレージを効率良く運用することで柔軟な問合せ処理が可能になる [8,9] とされている点に着目し，RDBMS の行指向型テーブルから取得した統計情報を基に列指向型データのコストを見積もることで，RDBMS を利用した列指向型インデックス利用のための問合せ処理最適化を行う．

2. 予備実験

列指向ストレージで用いられるインデックスでは，通常，統計情報を取得しない代わりに，数千から数万行の列データが格納されたセグメント単位での最大値，最小値等の情報を用いて問合せ処理の実行計画を生成する．この場合，目的のデータを列指向型ストレージから大量のセグメントにまたがって取得する処理では，行指向型ストレージに比べて走査するデータ量の増加やキャッシュするデータ量増加の可能性があるため OLAP の処理速度が低下する可能性がある．そのため，行指向および列指向ストレージという二種類のストレージから，どちらがより効率良く問合せ処理を実現可能となるのかを明確にし，どちらか一方のストレージを選択するために，正確な処理コストを見積もる方法が必要である．また，列指向型ストレージの統計情報は利用できないため，行指向型ストレージに格納されている同じデータから列指向型ストレージの処理コストを見積もる必要もある．そこで本節では，列指向ストレージを用いた問合せ処理において，行指向ストレージを用いて見積もられる処理コストと，列指向ストレージを用いて実際の処理で要した処理コストを予備実験として比較することで，正確な処理コストの見積もりに役立てる．

2.1. 実験環境

RDBMS は通常，行指向ストレージを用いてデータを管理しているが，列指向ストレージの有用性が認識されつつある現在では，列指向ストレージをも用いてデータを管理できるようになりつつある．このとき，列指向型ストレージのインデックスは行指向ストレージ上のテーブルに列指向型ストレージ上のテーブル単位で作成される．こうした機能を有する RDBMS は現時点では PostgreSQL のみであるため，本予備実験では PostgreSQL 9.4.5 と列指向ストレージ機能を追加する cstore_fdw 1.4 を用いて実験を行う．この実験環境では，図 1 のように行指向型ストレージのデータからテーブルごとに列指向型データのテーブルを複製し，列ごとに圧縮して格納しているため利用するストレージは二種類となるが，総データ量が二倍になることは

†同志社大学大学院 文化情報学研究科，Graduate School of Culture and Information Science, Doshisha University
‡同志社大学 文化情報学部，Faculty of Culture and Information Science, Doshisha University
§SQL Server 2014 列ストアインデックスの説明 https://msdn.microsoft.com/ja-jp/library/gg492088(v=sql.120).aspx

ない．

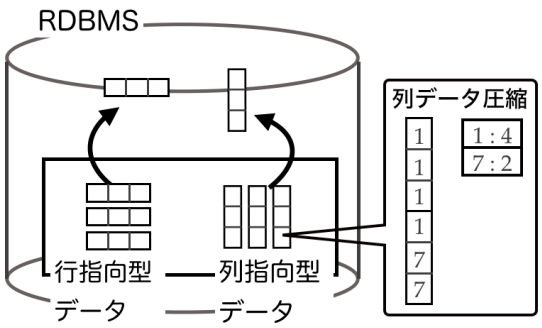

図 1: 列指向型インデックスを追加した RDBMS

2.2. 行指向型データと列指向型データの見積もりコストと問合せ処理時間の比較

予備実験で使用する PostgreSQL のコストベースオプティマイザでは，列指向型ストレージ上のデータに対するコスト見積もりは列データの集合であるセグメント単位で行われる．しかしながら，RDBMS の行指向型ストレージ上のデータに対しては取得する行数に応じて取得するデータ量等を考慮することで，キャッシュを効率的に利用するといった問合せ処理最適化が行われているため，本稿のように二種類のストレージを利用する場合には，列指向型ストレージ上の処理コストをそうした特徴を有していることを考慮して正確に見積もらなければ，問合せ処理に応じて二種類のストレージを効率よく利用した問合せ処理最適化を実現することはできない．

そこで，行指向型ストレージと列指向型ストレージから同じテーブルを取得する問合せに対して見積られるコストと，その問合せ処理時間の比較を行った．本予備実験では OLAP の性能評価のためのテストコレクションである TPC-H[¶] のデータから *lineitem* テーブルを使用した．その実験結果を図 2 に示す．

図 2 から，格納したテーブルの行数が 1,000 行を超えると行指向型データの方が見積もられたコストが大きいにも関わらず，実際に問合せ処理を行った時間は早いことが分かった．これは，列指向型データに対しては見積もられるコストが小さいにも関わらず問合せ処理時間が遅いため，二種類のデータ資源が利用できる RDBMS ではコストベースオプティマイザがコストの小さい列指向型データを選択することを示している．このような問合せの場合，行指向型データを利用した方が問合せ処理時間が早く，列指向型ストレージに対するコストの見積もり方法に誤りがあると考えられる．

列指向型ストレージ上のデータはセグメント単位でソートして圧縮することでそのセグメント内での最大値，最小値等は問合せ処理の高速化に役立てられるが，セグメント間でデータの最大値，最小値にばらつきが

¶米国トランザクション処理性能協議会 (Transaction Processing Performance Council: TPC) http://www.tpc.org/tpch/spec/tpch2.8.0.pdf

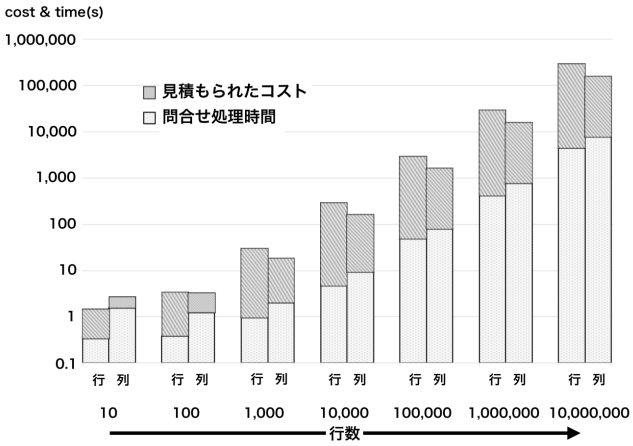

図 2: 行/列指向型データの見積もりコストと問合せ処理時間

生じてしまった場合には複数のセグメントからデータを取得する必要がある．また，列ごとに圧縮されたデータに対して見積もられたコストは，問合せ処理の際に解凍して復元されたデータの大きさを考慮することができないため，キャッシュを効率良く利用した問合せ処理の実行計画を生成しないという現状もある．そこで本稿では，行指向型ストレージ上のデータと列指向型ストレージ上のデータを問合せ処理の中で併用利用するために列指向型ストレージのコスト計算を正確に行う手法の提案を行う．

3. 提案手法

本節では，RDBMS を用いて行指向型／列指向型ストレージの二種類を効率良く併用利用するために，二種類のストレージ上のデータを比較してより小さいコストをオプティマイザが選択できるように，列指向型データが見積もるコスト計算を正確に行う手法について説明する．

本手法では，問合せ処理の中で各テーブルのデータをディスクから取得する処理である Sequential Scan の見積もりコストを二種類のストレージ上のデータから比較し，より小さいコストのストレージ上のデータを取得することを行う．この際，2 節で述べたように，常にコストが小さく見積もられる列指向型ストレージのみがコストベースオプティマイザによって選択されてしまうため，列指向型ストレージのコストを正確に見積もる必要がある．

3.1. 列指向型ストレージのコスト計算手法

PosgreSQL では行指向型ストレージの問合せ処理コストの見積もりを以下のような式から計算している[‖]．

$$C_{row} = 0.01 * n_{row} + p_r \quad (1)$$

n_{row} は，テーブルから取得したデータの行数であり，p_r は，読み込むディスクページの総数となっている．

‖http://www.postgresql.org/docs/9.4/static/using-explain.html

我々は，上記の行指向型ストレージのコスト計算式を参考に列指向型ストレージのコスト計算式を定義することとする．2 節で述べたように列指向型ストレージのコストが小さく見積もられる原因として，データ圧縮が考慮されていないことが考えられる．そのため，我々は格納された列指向型ストレージ上のデータのテーブルごとに容易に RDBMS で利用できる統計情報から以下のように圧縮率を定義し計算することとした．

$$R = s_{col}/s_{row} \tag{2}$$

s_{row} は行指向型ストレージ上のテーブルのデータ量であり，s_{col} は列指向型ストレージ上の同じテーブルのデータ量である．計算された圧縮率と定義した R を用いて列指向型ストレージのテーブルのコストを見積もる式を以下のように定義した．

$$C_{col} = 0.01 * n_{row}/(1 - R) + p'_r \tag{3}$$

この式で利用される p'_r は，問合せ処理の中で必要とされるテーブルの列データのみをコストに計算するために，問合せ処理に利用される列データのデータ型のサイズのみを合計している，問合せ処理に必要とされていない列指向型ストレージのテーブルのデータはコストに含まれないようにしている．

計算された C_{col} と，C_{row} を比較し，よりコストの小さい方のストレージを選択するようにコストベースオプティマイザの編集を行った．さらに，一つの問合せ処理の中で先ず最初に行指向型ストレージのテーブルを利用した場合の全ての Sequential Scan のコストを見積もり，次にそれぞれの Sequential Scan において処理に必要となるテーブルとその問合せ処理条件に合わせて列指向型ストレージのテーブルのコストを計算するようにコストベースオプティマイザの改良を行った．

3.2. 行指向型／列指向型ストレージを併用利用した問合せ処理最適化

列指向型ストレージ上のデータのコストを適切に見積もることで，行指向型ストレージと列指向型ストレージを問合せ処理に合わせて効率良く併用利用することが可能と考えられるが，問合せによっては処理に時間の掛かる処理が存在する．特に，列指向型ストレージ上のデータは，大量のセグメントに分散したデータを検索する処理が苦手とされており，大量のデータから特定の少数のデータを何度も検索するような問合せでは，行指向型ストレージを利用した方が高速に問合せを処理することが可能である．

そこで，行指向型ストレージを利用するルールを適用し，さらなる問合せ処理の高速化を図ることとする [10]．我々が行った調査では，相関副問い合わせが行われる際には検索条件が何度も変わるため複数回ディスクからデータを取得する Sequential Scan の必要があり，ディスクから取得するデータ量を減らすことでキャッシュ効率を上げる列指向型データの本来のメリットが薄く，検索対象のデータが少数であるほど多数行のデータを持つセグメントを何度も走査する必要があるため列指向型データのデメリットが色濃くなってしまうことが分かった [10]．そのため，本稿で提案した列指向型データのコスト計算を行う前に，相関副問い合わせの副問合せ内の検索条件に利用される列を含むテーブルは行指向型データを利用する，といったルールを適用した後に 3.1 節の列指向型データのコスト計算とその比較を行うように PosgreSQL のコストベースオプティマイザを改良した．

4. 評価実験

本節では TPC-H のデータをスケールファクター 100 で生成したおよそ 100GB のデータと 22 種類の問合せを用いて提案手法の評価を行った．また，行指向型／列指向型データのみをそれぞれ実行した場合を比較対象とし，22 種類の問合せをそれぞれ 10 回ずつ実行した．

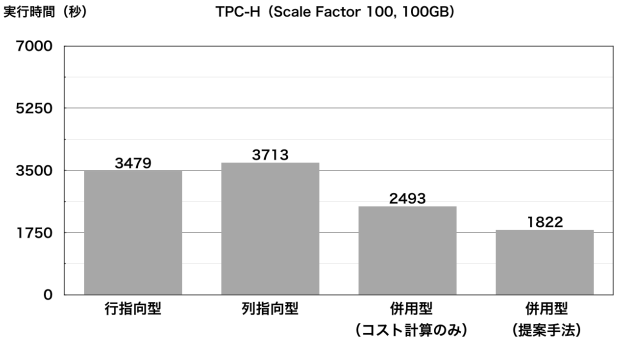

図 3: 二種類のデータと本提案手法を用いた場合の平均問合せ処理の比較

すべての問合せの平均実行時間は，図 3 のように本提案手法を用いた二種類のデータを併用利用した場合が最も問合せ処理時間を高速化できた．また，3.2 節で述べたルールと組み合わせることでより平均問合せ処理時間の短縮が可能となった．

本稿で提案したコスト計算では，列指向データの利点である列単位での圧縮されたデータが，問合せ処理の中間表を生成する際に解凍され元のテーブルの大きさに復元されるため，圧縮状態のデータを取得する Sequential Scan の際に復元後のコスト見積もりを図った．その結果，問合せ処理の際に効率良くキャッシュを利用することが可能となり，行指向ストレージのみを利用した場合に比べて約 28%，列指向ストレージのみを利用した場合に比べて約 33%，平均問合せ処理時間を高速化できた．また，3.2 節で述べたルールの適用を行うことで，一つの問合せの中で相関副問い合わせの処理では行指向型データのインデックスを利用して高速にストレージからデータを取得し，通常の問合せ処理ではコスト計算によって行指向型／列指向型のテーブルを効率良く選択できていたため，行指向型／列指向型ストレージのみを用いた場合に比べてそれぞれ約 48%，51%平均問合せ処理時間を高速化することができた．

5. おわりに

本研究では，近年 RDBMS で着目されている列指向型インデックス利用のために，行指向型ストレージと

列指向型ストレージの二種類をコストベースオプティマイザが効率良く選択するための列指向型データのコスト計算手法を提案した．

列指向型インデックスは，更新可能な列指向型データとして運用するためにテーブル単位で行指向型データから生成し，行指向型のテーブルに格納されるデータを列指向型データへと更新する必要がある．そのため，運用するデータを二種類持つことになるが，近年のハードウェア技術の進歩により安価に大量のディスクを利用可能である点からデータ量の増加がさほど問題にされず，OLAP の実行が苦手とされる RDBMS において注目される技術となっている．

そこで我々は行指向型ストレージと列指向型ストレージに同じデータを持たせ，OLAP の高速な問合せ処理を実現するために，一つの問合せの中で二種類のストレージから柔軟にデータを選択することが可能なコストベースオプティマイザ実現を図った．RDBMS では行指向型ストレージの利用のために行指向型のデータを見積もるコスト計算が可能であるが，その一方で列指向型のデータに関してはセグメント単位でのデータの検索，取得が一般的である．

その際，圧縮されたテーブルから複数の列を取得して問合せ処理を行う際には解凍後のデータサイズが分からないため，キャッシュを効率良く利用した問合せ実行計画が生成できないというデメリットがある．本稿ではそのような列指向型のデータに対して取得するデータのコストを計算することで，コストベースオプティマイザがストレージの選択を行いキャッシュを効率良く利用した実行計画の生成が可能となり，RDBMS を通常利用する場合に比べて OLAP の平均問合せ処理時間を約 48%高速化することができた．

今後の課題として，更新可能な列指向型インデックスの利用のために，行指向型ストレージに格納されるデータを効率良く列指向型ストレージへと更新しつつ OLAP を高速に行う方法を提案する．

謝辞

本研究の一部は JSPS 科研費 26280115，16H02908，文部科学省私立大学戦略的研究基盤形成支援事業 S1411030 の助成を受けたものである．

参考文献

[1] Yoo-Myung Choi. Web-enabled OLAP Tutorial. http://www.cis.drexel.edu/faculty/song/courses/info%20607/tutorial_OLAP/index.htm, April 2005. Retrieved May 1, 2015.

[2] Daniel J. Adadi, Samuel R. Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 967–980. ACM, June 2008.

[3] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, and Chuck Bear. The vertica analytic database: C-store 7 years later. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, pp. 1790–1801. VLDB Endowment, August 2012.

[4] Ute Baumbach, Patric Becker, Uwe Denneler, Eberhard Hechler, Wolfgang Hengstler, Steffen Knoll, Frank Neumann, Guenter Georg Schoellmann, Khadija Souissi, and Timm Zimmermann. Accelerating Data Transformation with IBM DB2 Analytics Accelerator for z/OS. http://www.redbooks.ibm.com/redbooks/pdfs/sg248314.pdf, December 2015. Retrieved April 29, 2016.

[5] Franz Färber, Sang-Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA Database: Data Management for Modern Business Applications. *ACM SIGMOD Record*, Vol. 40, No. 4, pp. 45–51, December 2011.

[6] Boncz Peter, A., Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proceedings of Conference of Innovative Database Research*, pp. 225–237, 2005.

[7] Mike Stonebraker, Daniel J. Asadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. C-Store: A Column-Oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 553–564. VLDB Endowment, August 2005.

[8] Daniel Abadi. Hybrid Row-Column Stores: A General and Flexible Approach. http://blogs.teradata.com/data-points/hybrid-row-column-stores-general-flexible-approach/, March 2015. Retrieved June 20, 2016.

[9] Ravishankar Ramamurthy, David J. DeWitt, and Qi Su. A Case for Fractured Mirrors. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 430–441. VLDB Endowment, August 2002.

[10] Takamitsu Shioi and Kenji Hatano. Query Processing Optimization using Disk-based Row-store and Column-store. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications and Services*, pp. 524–532. ACM Interational Conference Proceedings Series, December 2015.