データベースにおけるエージングが問合せ最適化に与える影響に関する実験的考察

加藤　千裕† 　早水　悠登† 　合田　和生† 　喜連川　優†,††

† 東京大学生産技術研究所　〒 153–8505 東京都目黒区駒場 4-6-1
†† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{kato,haya,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　データベースは更新が繰り返されると，その格納構造の効率が低下し，問合せ処理に掛かるデータベースアクセスの性能が低下する場合がある．当該現象はエージングと称され，その影響は一般にアクセスの種別によって異なることから，論理的に同一の問合せに対して問合せ実行計画のコストが変化することがある．本論文では，データベースにおけるエージングが問合せ最適化に与える影響を検証し，考察する．

キーワード　データベース，問合せ最適化，エージング

1. はじめに

昨今では，ビッグデータと称される巨大なデータの登場に伴って，データを格納して管理するデータベース技術に注目が集まっている．データベースに巨大なデータを格納する場合，初期段階においては，例えば論理アドレスと物理アドレスが一致する如く，格納効率は高い．しかし，データベースに対して削除や挿入といった更新作業を繰り返すことにより，この例においては論理アドレスと物理アドレスの相関が低下し，即ち，格納構造の効率が低下する．当該劣化現象はエージングと呼ばれる．エージングの影響は一般にアクセスの種別によって異なることから，論理的に同一の問合せに対して問合せ実行計画のコストが変化することがある．

しかしながら，一般にエージングによるコスト変化は，問合せ最適化において必ずしも考慮されているとは言い難い．多くの更新処理を経て，エージングが進んだデータベースにおいては，問合せ最適化において想定されるアクセス性能と，実際のアクセス性能が乖離し，この結果，正しくない問合せ実行計画が選択される可能性がある．

本研究では，小規模なデータベース実験環境において TPC-H データセットを用いて問合せ処理の性能試験を行うことにより，エージングが問合せ最適化に与える影響を検証し，問合せ最適化のさらなる高度化の可能性を明らかにする．

本論文の構成は以下の通りである．第 2. 章において本論文の趣旨について詳細な説明を行う．第 3. 章において検証実験について説明し，第 4. 章で検証実験の結果を示し，第 5. 章で考察を行う．第 6. 章では関連する研究について記し，最後に第 7. 章で本論文のまとめと今後の課題について述べる．

2. エージングによる問合せ最適化への影響

本論文では，小規模なデータベース実験環境において問合せ処理の性能試験を行うことにより，エージングが問合せ最適化に与える影響を検証する．

本論文では，研究の初期段階として，単一表をアクセスする問合せを想定し，二つのアクセス種別を対象として，問合せ最適化を検証する．一つは，テーブル内の全てのレコードを読みだして検索条件に合致するレコードを抽出するフルテーブルスキャン，もう一つは，あらかじめ作成したインデックスを使い，検索条件に合致するレコードのみを読みだすインデックススキャンである．問合せ最適化においては，通常，問合せがテーブルの大部分にアクセスする場合には，フルテーブルスキャンが選択され，問合せがテーブルの一部分のみにアクセスする場合には，インデックススキャンが選択される．この選択は，それぞれのアクセス種別によるコスト見積りによって行われる．この際，フルテーブルスキャンは，データベースの初期状態においては，シーケンシャルなアクセスとなることが期待されるものの，データベースのエージングによってはランダムなアクセスに近付くことが知られており，これにより性能が低下する恐れがある．即ち，問合せ最適化においては，そのコストの見積りが不正確になることにより，最終的に実行される問合せ計画が，必ずしも効率のよいものとならない可能性がある．

本論文では，データベースのエージングの度合いに応じてこれら二つのアクセス手段の損益分岐点がどのように変化するかを測定することにより，エージングが問合せ最適化に与える影響を検証する．

3. 検証実験

前章で説明したとおり，本章では，エージングが問合せ実行時間に与える影響について，問合せの最適化において一般的な二種類のアクセス種別を用いて計測を行う．

3.1 環　境

データベースへのエージングの影響についての評価を行うに当たり，今回の実験では，データベースとして PostgreSQL ver 9.4.0 を使用した．また，スキーマ構造は TPC-H の lineitem のみを利用し，付属の dbgen を用いてスケールファクター 100 として初期データと更新データの作成を行った．dbgen のバージョ

表 1　実 験 環 境

データベース	PostgreSQL ver 9.4.0
スキーマ構造	TPC-H (lineitem)
データ作成	Dbgen ver 2.17.0
OS	CentOS release 5.8 (64bit)
HDD	WD9001BKHG (900GB)

表 2　各カラムの相関係数

使用するカラム	相関
l_orderkey	0.129376
l_partkey	-0.00411825

ンは 2.17.0 である．また，OS は CentOS release 5.8（64bit），ハードディスクドライブは WD9001BKHG を使用した．容量は 900GB であり，スケールファクター 100 のデータベースは十分に容量内に収まる．PostgreSQL のパラメータはすべて初期状態とした．

データをロードした直後の基準となるデータベースと，10 パーセント刻みの更新用のデータを，dbgen を用いて作成した．更にデータベースに更新用のデータを適用し，更新処理を繰り返し，リフレッシュを行って人為的にエージングを発生させたデータベースを作成した．今回実験に用いたデータベースは，エージングしていない初期状態のデータベース（RF0）と，それぞれ 10 パーセントから 90 パーセントの更新処理を行ってエージングさせたデータベース（RF10，RF20，…RF90）である．また，PostgreSQL のデータベース不要領域の回収を行う VACUUM コマンドは，事前に全てのデータベースに対して実行した．

3.2　実 験 手 法

3.2.1　アクセスコストの変動

評価のため，以下の問合せを実行する．

```
SELECT SUM(l_extendedprice)
FROM lineitem
WHERE l_orderkey < x
```

この際，l_orderkey は lineitem テーブルの主キーである．スキャンを行う際には，x の値を増減させることで選択率を変化させた．また，次の問合せに対しても実験を行った．

```
SELECT SUM(l_extendedprice)
FROM lineitem
WHERE l_partkey < x
```

この実験では，l_partkey を使って選択率を操作する．この二つのカラムについての，物理的な行の並び順と論理的な列の値の並び順に関する統計的相関は表 2 に示される．

上記二種類の問合せを用い，フルテーブルスキャンまたはインデックススキャンを適用した場合について，それぞれ実行時間を計測した．

3.2.2　コスト見積りとの誤差

PostgreSQL のオプティマイザのコスト見積りがどの程度エージングの影響を反映しているか計測するため，前節と同様のクエリを使用し，実行時間とオプティマイザのコスト見積り

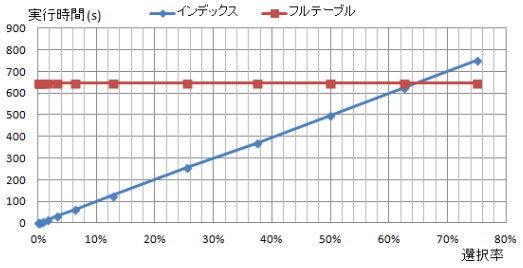

図 1　l_orderkey を用いた RF0 データベースのスキャン時間

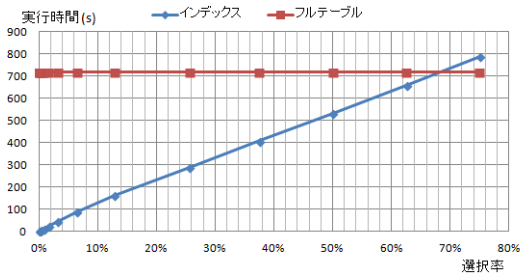

図 2　l_orderkey を用いた RF10 データベースのスキャン時間

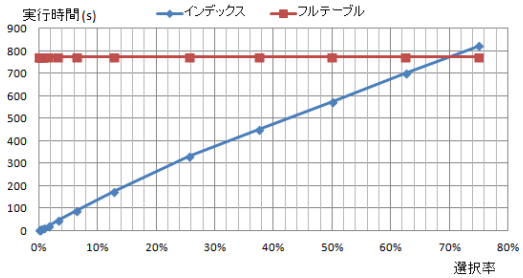

図 3　l_orderkey を用いた RF20 データベースのスキャン時間

との比較を行った．但しコスト見積りの単位は時間ではないため，更新率 0 パーセントの時の値を 1 とし，相対的な大小で比較を行った．

4.　結　　果

l_orderkey を用いた実験の結果の一部を図 1 から図 3 に示す．グラフの横軸が x で指定した選択率，縦軸が実行時間を表している．図から，更新処理を行う毎に，インデックススキャンとフルテーブルスキャンの実行時間が変化していくことが読みとれる．これ以上の更新率でも類似した傾向の結果を得た．

選択率を 1.6 パーセントで固定して，更新率を横軸に取りプロットしたグラフが図 4 と図 5 である．フルテーブルスキャンが更新率の増加に伴って右肩上がりに増加していることがわかる一方で，インデックススキャンに関しては一定の傾向は見られない．

以上のように実行コストの変化によって発生した，インデックススキャンとフルテーブルスキャンの損益分岐点の変動をまとめたものが，図 6 である．

同様の実験を，l_partkey を用いて行った．このカラムは表 2 の通り，テーブルの並びとの相関関係が少なく，ほぼ全てランダム I/O であることが期待される．更新率を横軸に取ってイン

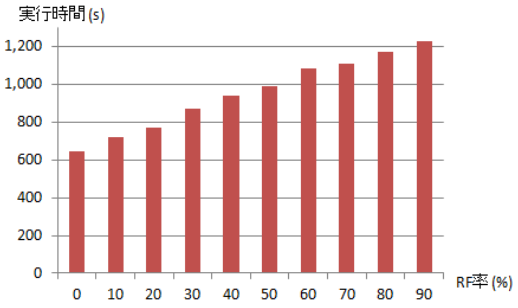

図 4　フルテーブルスキャン実行時間比較

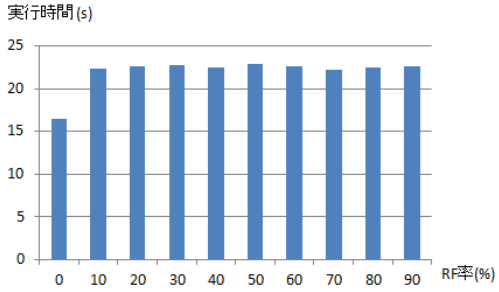

図 5　インデックススキャン実行時間比較（l_orderkey）

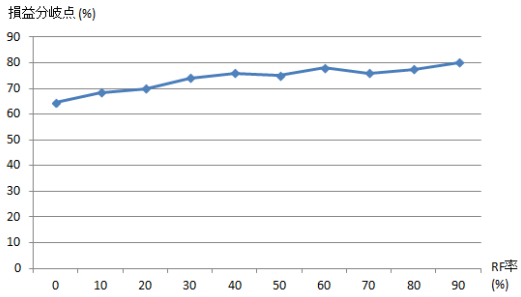

図 6　損益分岐点比較（l_orderkey）

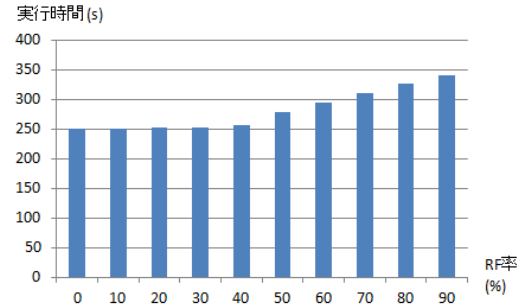

図 7　インデックススキャン実行時間比較（l_partkey）

デックススキャンについて実行速度を比較した結果が図 7 である．選択率は 0.01 パーセントで固定とした．但し，フルテーブルスキャンに関しては図 4 と同様の結果である．この結果に基づいて算出した損益分岐点の変動は，図 8 によって表される．

PostgreSQL のオプティマイザのコスト見積りとの比較結果は図 9 から図 11 に示される．但しインデックススキャンに関しては，l_orderkey は選択率 1.6 パーセント，l_partkey は選択

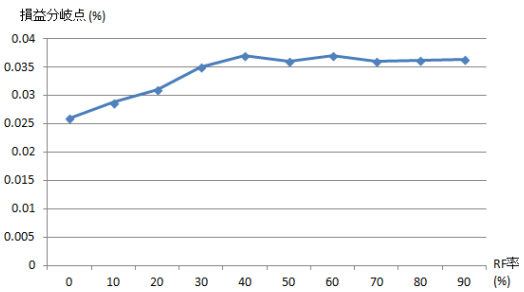

図 8　損益分岐点比較（l_partkey）

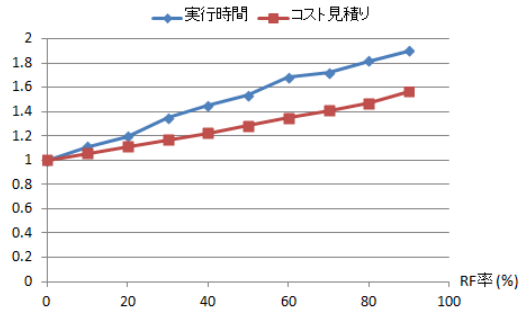

図 9　シーケンシャルスキャンに関するコスト見積りと実行時間の比較

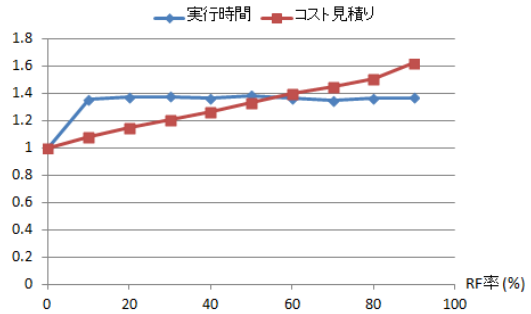

図 10　インデックススキャンに関するコスト見積りと実行時間の比較（l_orderkey）

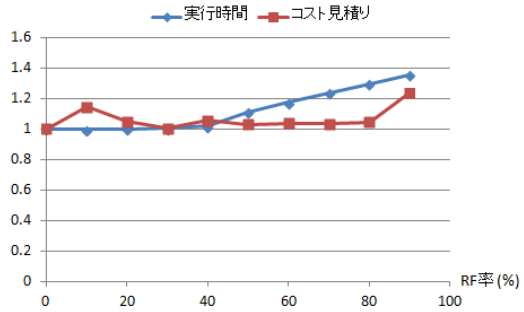

図 11　インデックススキャンに関するコスト見積りと実行時間の比較（l_partkey）

率 0.01 パーセントの時のグラフを載せている．そのほかの選択率に関しても，グラフの傾向はほぼ同様であった．

5.　考　　察

図 4 から，エージングの進展に従って，フルテーブルスキャ

ンの速度は徐々に劣化していくことが読みとれる．これは論理構造が実際のデータ構造と食い違っているために，論理的にはシーケンシャルアクセスを行っている場合にも，実際のスキャンとしてはランダムアクセスに近い傾向が生じており，その食い違いによって遅延が起こっていると推測できる．一方，インデックススキャンへの影響は図 5 と図 7 に見られる通り，使ったキーによって傾向が異なり，一定の解釈を与えることは難しい．どのようなパラメータが影響しているのか，今後更に調査を進めていく必要がある．

図 6 と図 8 では，損益分岐点の入れ替わる選択率が示されている．図 6 では，全体的に，一般に知られている損益分岐点と比べ，インデックススキャンが優位という結果となっている．これは l_orderkey が lineitem の要素の並びとほぼ同じ順に並んでいるクラスタ化索引であったことで，インデックススキャンでもシーケンシャル I/O に近い読み込みを行ったことが原因と考えられる．更新率によっては，初期状態と比べ最大で 24 パーセントの誤差が生じており，アクセスコストの変動がスキャン方法の選択に影響を及ぼす可能性があることを示唆している．

l_partkey を用いて計測した図 8 は，更新率 40 パーセント以降横ばいに近いグラフとなった．このカラムを用いたインデックススキャンの場合，殆どランダム I/O であるため，インデックススキャンの計測結果がほぼ変動しないと期待されたが，図 7 に示される通り，RF40 以降大きくコストが増加している．結果として，フルテーブルスキャンのコスト増加とほぼ一致したために損益分岐点が横ばいとなった．この原因究明はこれからの課題としたい．一方で損益分岐点の誤差としては初期状態と比べて最大で 42 パーセントの変動が起きていることが分かり，やはりクエリ最適化への影響が発生する可能性が確認できた．

また，図 9 から図 11 の結果より，現状の PostgreSQL のオプティマイザのコスト見積りと，実行時間には誤差があることが確認された．シーケンシャルスキャンに関する図 9 ではグラフの形の傾向は類似しているものの，初期状態と比べ，最大で 35 パーセントほどの誤差が生じた．インデックススキャンに関しては，グラフの傾向も違うという結果になったが，最大誤差はどちらのキーでも初期状態から 25 パーセント程度であった．現状の PostgreSQL のオプティマイザはエージングがコストに与える影響を正確に反映できていない部分があり，どのようなパラメータを見ることで誤差を是正できるか考える必要があると言える．

6. 関 連 研 究

6.1 データベースのエージングと再構築

データベースのエージングについては，従前より，解消するための再構築に関する研究が多くなされてきた．1979 年には Sockut と Goldberg によって，データベースの再構築に関する研究 [1] がされている．当時はデータベースのデータ量は限定的であり，再構築はオフラインで，一旦データベースシステムの稼働を停止するアプローチが主流であった．

しかし，データ量が増えると再構築のために長時間データベースシステムを停止する必要が生じ，一方でデータベースシステムの継続した稼働も必要とされるようになり，オンラインでの再構築が求められるようになった．このような取り組みとして，一度データベースを複製してから再構成し，後から複製中の更新を反映させる方法の研究 [2]，ユーザの問合せと競合しないように考慮しながらデータベースをそのまま再構成する方法の研究 [3] [4] の二つが主になされてきた．

最近のオンライン再構成としては [5] などがある．また，この再構成を行うタイミングは従来は管理者の主観に委ねられていた部分が多く，どの程度構造が劣化しているかが明らかではなかったため，これを可視化する研究 [6] も行われてきた．

しかし常時データベースの構造を理想的に保つことは再構成を用いても極めて難しく，構造が劣化しているタイミングでは，問合せ最適化に悪影響が出ている可能性がある．本論文ではそれを明らかにするため，実験を行った．

6.2 問合せ最適化

問合せ最適化とは，ユーザからの問合せをもとに，効率のよい実行計画を生成する機能のことである [7]．データベースのデータ量増大に伴い，非効率な問い合わせ計画の実行が生み出すタイムロスも大きくなる．そのため，問合せ最適化の重要性はより高くなっている．

問合せ最適化に関する研究は昔からデータベースシステムの不可欠な要素として，研究が行われてきた [8] [9]．コンピュータのコア数増加の傾向に伴い，問合せ最適化の並列化も研究も進んだ [10]．近年の研究では，増大するストレージを有効利用して，メモリを犠牲に速度を上昇させる研究 [11] など，技術の進歩を取り入れてよりよい問合せ最適化を行う研究がなされている．

本論文で取り上げるような，問合せ最適化に用いる I/O コスト計算に関する研究も行われている．コストベースの最適化は昔から行われており，1997 年にはシーケンシャルアクセスとランダムアクセスの I/O コストを区別してのコスト計算を行う研究 [12] が発表されている．これらはすべて HDD を基本として考えてきたが，近年は SSD の登場でランダム I/O のコストが大きく改善されることにより，従来のモデルを見直す研究が盛んである [13] [14] [15]．本論文ではデバイスの変化とは別の切り口で，問合せ最適化における I/O コストモデルの見直しを行っていく．

7. まとめと今後の課題

本論文では，データベースのエージングについて述べ，問合せ最適化の際に用いられる二つのスキャン方法について，どのように影響を及ぼすかについての実験を行った．その結果，フルテーブルスキャンには遅延が生じることを確認し，インデックススキャンにも一定の影響があるという結果を得た．また，現状の PostgreSQL のオプティマイザでは，エージングの影響をコスト見積りに反映できていない場合があることを確認した．

以上の食い違いにより，エージングしたデータベースにおいて二つのスキャン方法を選択する際に最適な方法を選べず，クエリ最適化の精度が落ちている可能性があると考えられる．今

後はエージングに対してよりロバストなコスト見積りを実現することを目標としたい．

文　献

[1] Gary H Sockut and Robert P Goldberg. Database reorganization-principles and practice. *ACM Computing Surveys (CSUR)*, Vol. 11, No. 4, pp. 371–395, 1979.

[2] Gary H. Sockut, Thomas A. Beavin, and C-C Chang. A method for on-line reorganization of a database. *IBM Systems Journal*, Vol. 36, No. 3, pp. 411–436, 1997.

[3] Chendong Zou and Betty Salzberg. On-line reorganization of sparsely-populated b+-trees. In *ACM SIGMOD Record*, Vol. 25, pp. 115–124. ACM, 1996.

[4] Edward Omiecinski and Peter Scheuermann. A global approach to record clustering and file reorganization. In *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 201–219. British Computer Society, 1984.

[5] Shahram Ghandeharizadeh, Shan Gao, Chris Gahagan, and Russ Krauss. An on-line reorganization framework for san file systems. In *Advances in Databases and Information Systems*, pp. 399–414. Springer, 2006.

[6] Takashi Hoshino, Kazuo Goda, and Masaru Kitsuregawa. Online monitoring and visualisation of database structural deterioration. *International Journal of Autonomic Computing*, Vol. 1, No. 3, pp. 297–323, 2010.

[7] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 34–43. ACM, 1998.

[8] Goetz Graefe. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, Vol. 18, No. 3, pp. 19–29, 1995.

[9] Goetz Graefe. *Encapsulation of parallelism in the Volcano query processing system*, Vol. 19. ACM, 1990.

[10] Florian M Waas and Joseph M Hellerstein. Parallelizing extensible query optimizers. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 871–878. ACM, 2009.

[11] Luis L Perez and Christopher M Jermaine. History-aware query optimization with materialized intermediate views. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pp. 520–531. IEEE, 2014.

[12] Laura M Haas, Michael J Carey, Miron Livny, and Amit Shukla. Seeking the truth about ad hoc join costs. *The VLDB journal*, Vol. 6, No. 3, pp. 241–256, 1997.

[13] Steven Pelley, Kristen LeFevre, and Thomas F Wenisch. Do query optimizers need to be ssd-aware? In *ADMS@ VLDB*, pp. 44–51, 2011.

[14] Daniel Bausch, Ilia Petrov, and Alejandro Buchmann. Making cost-based query optimization asymmetry-aware. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, pp. 24–32. ACM, 2012.

[15] Pedram Ghodsnia, Ivan T Bowman, and Anisoara Nica. Parallel i/o aware query optimization. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 349–360. ACM, 2014.