データベースにおけるエージングの局所性を考慮した問合せコストモデルの構築

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　データベースシステムにとって，更新処理がもたらすエージングによる性能低下は，多くの場合避けられず，特に更新が頻繁な領域において性能低下が見られることが少なくない．本論文では当該エージングの局所性を考慮した問合せ最適化のためのコスト計算手法を提案する．また，オープンソースのデータベースシステムである PostgreSQL を対象として，標準的なデータベースベンチマークを用いて行った実験を示し，当該手法の有効性を明らかにする．

キーワード　エージング，問合せ最適化，性能評価

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，その利活用の際に重要な役割を果たすデータベースシステム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，スキーマを構築しデータをロードした直後においては格納効率は高いが，データの削除や挿入といった更新処理を繰り返すと，論理アドレスと物理アドレスの不一致や，データの格納領域の拡大といった現象が発生し，格納構造の効率が低下することがある．当該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表の走査方法によって異なることから，論理的に同一の問合せに関しても，適切な問合せ実行計画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の変化は，これまでの問合せ最適化において必ずしも十分に考慮されているわけではない．特に，表の一部が局所的にエージングしている場合においては，当該現象が問合せ最適化へ与える影響は十分に解明されているとは言い難い．

本論文では，当該エージングの局所性を考慮した問合せ最適化のためのコスト計算手法を提案する．一般に，エージングは入出力コストの増大をもたらし，これにより問合せの処理性能が低下する．当該特性に着目し，データベースの部分空間ごとに入出力コストの変動を測定し，当該変動に基づき問合せ実行計画のコスト予測を行う．例えば，論理的には同量の入出力を必要とする問合せであっても，エージングが皆無の部分空間を対象とする場合と，エージングが進展した部分空間を対象とする場合とでは，本来，その入出力コストは異なるはずである．従来，これは十分に考慮されていなかったが，上述の提案手法によれば前者の場合と比べて後者の場合はより高い入出力コストが予測されることとなり，問合せ最適化における実行計画の選択の妥当性が向上することが期待される．

本論文では，オープンソースのデータベースシステムである PostgreSQL を用いた小規模実験環境において著者らが行った検証実験を示し，提案手法の有効性を明らかにする．

本論文の構成は以下の通りである．第 2. 章では，問合せ最適化におけるコスト計算手法を提案する．第 3. 章では，著者らが構築した試験環境を示し，当該環境において複数の問合せを用いて実施したケーススタディを述べ，提案手法の有効性を明らかにする．第 4. 章では関連研究を示し，第 5. 章では本論文をまとめる．

2. 局所的なエージングを考慮するコスト計算手法の提案

実運用されるデータベースの更新処理は一様ではない場合が多く，エージングを考慮する上で，その局所性を反映することは重要であると言える．本論文では，局所的なエージングを反映するコスト計算手法を提案する．提案手法は大きく二つのステップに分けられる．まず，テスト用問合せを実行し，その実行時間をもとに 1 レコードへのアクセスコストを割り出し，エージングの度合いとする．この測定は，データベース内の表毎に可能なアクセスメソッド毎に行う．例えばデータベース内に表が三個あり，それぞれの表毎に可能なアクセスメソッドが三個あった場合，実行するテスト用問い合わせは九個となる．

アクセスメソッドとしては，本論文では基礎的な二種類の方法である全表走査と索引走査を対象とした．前者は問合せの選択率に関わらず表内の全てのレコードを読み込むメソッドで，局所的に読むことはできないため，エージングの度合いは単一の値によって表される．一方，後者は索引を用いて表の一部のレコードを読むため，その索引毎に異なるエージングの局所性を持つと考えられる．

索引走査におけるエージングの度合いを得る際には，表を部分空間に分割し，それぞれの範囲を読むテスト用問合せを実行する．そしてその実行時間をもとに，それぞれの範囲内において 1 レコードへのアクセスコストを算出し，その変動をエージ

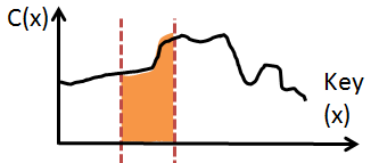

図 1　エージングしたデータベースのアクセスコスト

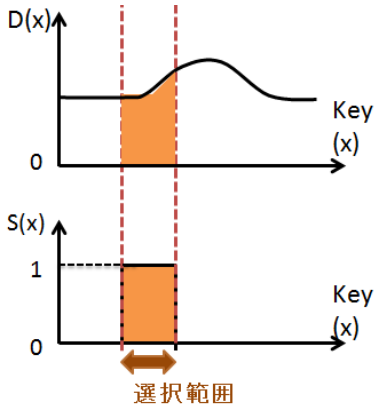

図 2　走査する領域とデータ分布密度

ングの度合いとして用いる．表の分割数を多くすることによって，より正確に局所的なエージングを測定することが出来る．索引の値を横軸に，部分空間内のアクセスコストを縦軸に取ることにより，例えば図 1 のようなグラフを得る．図 1 の x 軸は索引や番地の値，C(x) は一レコードへのアクセスコストである．このように，索引または番地とエージング度合いとの対応をアクセスメソッド毎に保持し，コスト見積りに活用する．

実際の問合せが入力されると，テスト用問合せにより求めたエージング度合いを，実行する問合せが読み出す範囲と照らし合わせてコストを算出する．単一表の走査コストは以下のような式として表すことが出来る．

$$\Gamma = \int S(x)D(x)C(x)dx \tag{1}$$

- Γ：見積りコスト
- $S(x)$：走査する領域
- $D(x)$：データ分布密度
- $C(x)$：エージング度合い

D(x) と S(x) の関係を図 2 に示す．C(x) はテスト用問合せによって求めたアクセスメソッドごとの 1 レコードへのアクセスコストを用いる．複数表の結合を含む問合せに関しても，単一表の走査コストを組み合わせることにより計算を行う．例えば，索引走査を用いて二表のネステッドループ結合を行う場合は，以下の式で表わされる．

$$\Gamma = \int S_{t1}(x)D_{t1}(x)C_{t1}(x)dx + \int j_{t12}(x)\{S_{t1}(x)D_{t1}(x)\}C_{t2}(x)dx \tag{2}$$

表 1　実験環境

サーバ	Dell Power Edge R720xd
CPU	Intel(R) Xeon(R) CPU E5-2690 v2
メモリ	64GB
OS	CentOS release 5.8（64bit）
データベース	PostgreSQL ver 9.4.0
スキーマ構造	TPC-H (lineitem, part)
データ作成	Dbgen ver 2.17.0

- Γ：見積りコスト
- $S_{t1}(x)$：走査する領域
- $D_{t1}(x)$：データ分布密度
- $C_{t1}(x), C_{t2}(x)$：エージング度合い
- $j_{t12}(x)$：表 1 の 1 レコードに対する表 2 のレコード数

ネステッドループ結合は，結合元の表の 1 レコードごとに結合先の表にアクセスを行い，対応するレコードへのアクセスを行う．そのため，結合元の表の 1 レコードに対する結合先の表のレコード数の値 $j_{t12}(x)$ を用いて，このように表す．

一方，二表のハッシュ結合は，以下の式で表すことが出来る．

$$\Gamma = \int S_{t1}(x)D_{t1}(x)C_{t1}(x)dx + \int S_{t2}(x)D_{t2}(x)C_{t2}(x)dx \tag{3}$$

- Γ：見積りコスト
- $S_{t1}(x), S_{t2}(x)$：走査する領域
- $D_{t1}(x), D_{t2}(x)$：データ分布密度
- $C_{t1}(x), C_{t2}(x)$：エージング度合い

全表走査を用いるハッシュ結合の場合には，$S_{t1}(x)$，$S_{t2}(x)$ の値は常に 1 となる．索引走査を用いるハッシュ結合の場合はこの限りではなく，走査する範囲に応じて $S_{t1}(x)$，$S_{t2}(x)$ の値は変動する．

以上のコスト計算手法を用いることで，局所的なエージングに関しても，問合せ最適化の際に適切でない計画を選択する可能性を減らすことができると期待される．

3. 実　　験

著者らは，局所的なエージングを反映した提案手法の有用性について実験を行った．

3.1 実験環境

実験環境のサーバとして，Dell Power Edge R720xd（Intel(R) Xeon(R) CPU E5-2690 v2，メモリ 64GB，CentOS release 5.8（64bit））を用い，磁気ディスクドライブとしては，10Krpm で 900GB の容量を備えたものを用いた．データベースシステムとして PostgreSQL 9.4.0 を用いた．TPC-H 付属の dbgen を用いてスケールファクタを 100 として初期データと更新クエリの作成を行った．PostgreSQL の設定パラメータはすべて初期状態とした．

```
SELECT SUM(l_extendedprice) FROM lineitem
```

図 3　問合せ（A）

エージングによる問合せ最適化への影響を計測するため，初期状態のデータベースを TPC-H の定める更新関数で更新しエージングさせた．用意したデータベースは，初期状態のデータベース 1 つ，予備実験のために，データの 90%を 1 回更新したデータベースが 1 つ，局所性のあるエージングについて測定するために，lineitem 表の 10%に 1 回，2 回… 4 回更新処理を行った 4 つのデータベース，先頭から 10%，20%… 40%を 1 回更新した比較用の 4 つのデータベースの，計 10 個である．

更新前後でデータベースに格納されるレコード数は変化しない．また，それぞれのデータベースは異なる磁気ディスクに格納されている．よって，磁気ディスク内における格納位置の影響は生じないものとする．なお，いずれのデータベースも，実験前に vacuum コマンドにより不要領域の回収を行った．

3.2　予 備 実 験

エージングが問合せ実行に及ぼす具体的な影響を明らかにするため，初期状態のデータベースと，先頭から 90%を 1 回更新したデータベースに対し問合せを実行し，カーネルトレーサによって実行中のハードディスクの挙動を観察し，入出力命令の定量的な評価と考察を行った．詳細は [2] に記す．以下はその要旨である．

全表走査を行う問合せの実行中の入出力命令を観察した結果，エージングしたデータベースの全表走査は，初期状態の全表走査の際にアクセスしていなかった領域にアクセスしていることを観測した．データの削除と挿入を行ったことによりデータの格納領域が拡張し，このような結果になったと考えられる．

また，問合せ実行時の入出力発行回数，その総データ量，ディスクの総シーク距離をエージング前後で比較したところ，全表走査を行う問合せでは全ての値が増加した．データの格納領域が広がったことにより，シーケンシャルアクセスによる読み込み範囲が増大したためと考えられる．一方で，索引走査を行う問合せを実行した際には，エージングによる入出力発行回数と総データ量の変動は見られなかったが，総シーク距離が大きく増加した．索引走査は指定した領域のみを読み込むため，読んだサイズそのものに変動はなかったが，データの格納領域が分散したことによる大幅なシーク距離の増加という形でエージングの影響を受け，実行時間が増加したと考えられる．

本論文では，特に局所的なエージングに着眼し，問合せ実行時間と入出力に与える影響を走査方法ごとに観測し，その重要性を示すとともに，見積りコストと実行時間との比較を行い，コスト見積り手法の有用性を検証する．

3.3　局所的なエージングの重要性に関する検証実験

著者らはまず，局所的なエージングを考慮する必要性を確認するため実験を行った．

3.3.1　使用した問合せ

全表走査を行う問合せとして図 3 に示す問合せ（A）を，先頭から 10%をクラスタ化索引によって読む問合せとして，図 4 に示す問合せ（B）を用いた．クラスタ化索引は表の並びに沿って作られた索引で，論理的にはシーケンシャルに近い走査が行えると期待される．問合せ（A）と問合せ（B）をそれぞれのデータベースに対して実行し，エージングの局所性による影響を観測した．

```
SELECT SUM(l_extendedprice) FROM lineitem
WHERE l_orderkey < 60000000
```

図 4　問合せ（B）

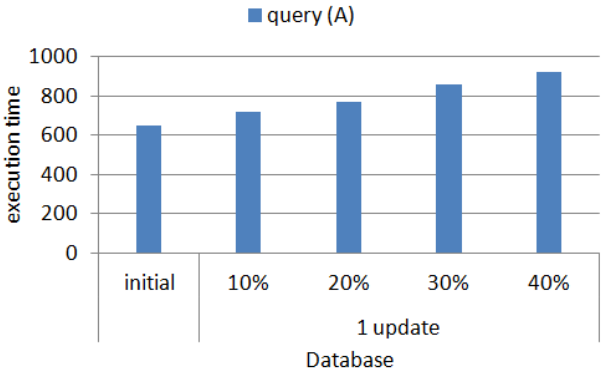

図 5　問合せ（A）の実行時間（エージング後）

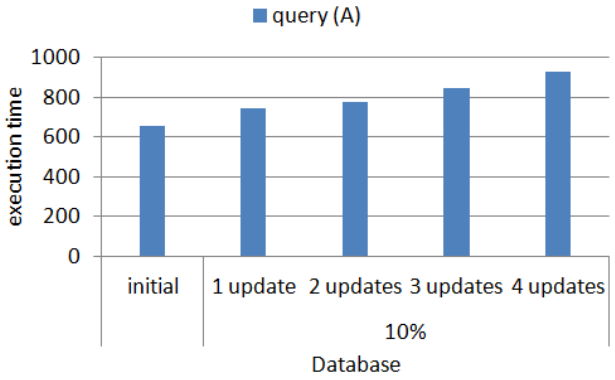

図 6　問合せ（A）の実行時間（局所的エージング後）

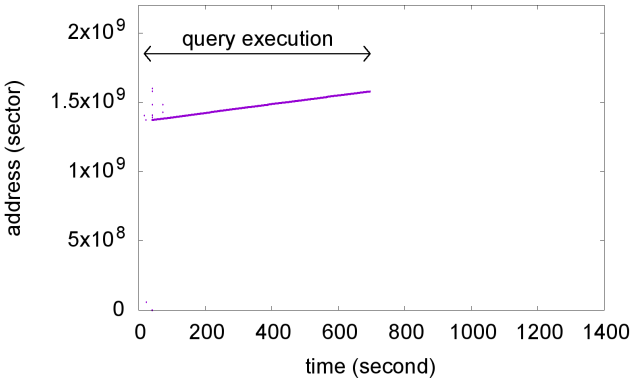

図 7　初期状態の問合せ（A）の入出力挙動

3.3.2　局所的なエージングの影響の観測と考察

全表走査を行う問合せ（A）を用いた場合は，局所的にエージングしたデータベース，幅広くエージングしたデータベース共に，同様の傾向で実行時間が増加するという結果になった．

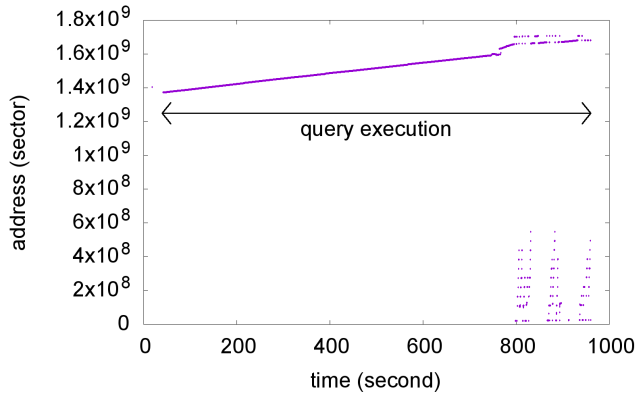

図 8　40%1 回更新データベースの問合せ（A）の入出力挙動

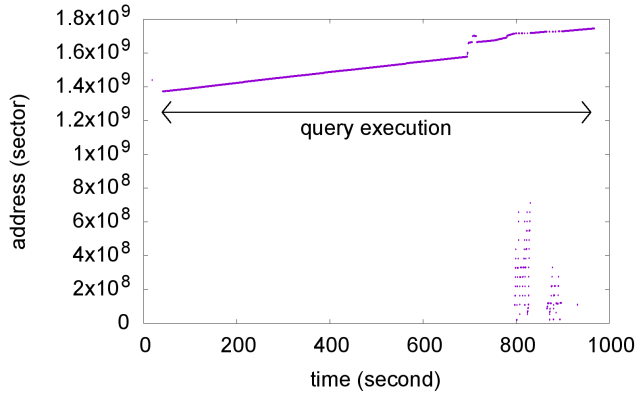

図 9　10%4 回更新データベースの問合せ（A）の入出力挙動

詳しい結果を図 5，図 6 に示す．図 5 は初期状態のデータベースと先頭から 10%，20%… 40%と幅広く更新したデータベースにおける問合せ（A）の実行時間を，図 6 は初期状態のデータベースと先頭から 10%を 1 回，2 回… 4 回と局所的に更新したデータベースにおける問合せ（A）の実行時間を表している．更新が行われた範囲に関わらず，更新された量に応じて実行時間が増加しており，全表走査の実行時間にエージングの局所性は影響しないと言える．

初期状態のデータベース，先頭から 40%を 1 回更新したデータベース，先頭から 10%を 4 回更新したデータベースに関して，ハードディスクの入出力を観察したグラフを，それぞれ図 7，図 8，図 9 に示す．横軸は経過時間，縦軸はセクタ単位のアドレスである．どちらの場合も，更新後のデータが似た領域に格納されており，その領域を読むために実行時間が増加していることが読みとれる．極端に局所的にエージングが発生したデータベースにおいて，全表走査を行う問合せを実行した場合には，局所性の少ないエージングと同様の動作を行うことが分かる．

一方，先頭から 10%のみをクラスタ化索引によって走査する問合せ（B）を用いた場合は，幅広くエージングしたデータベースはどのデータベースも先頭の 10%のエージング度合いは同じであるため，実行時間は図 10 に示す通り，10%更新以降は増加しないという結果になった．対して先頭の 10%が局所的にエージングしたデータベースは図 11 に示すように，更新回

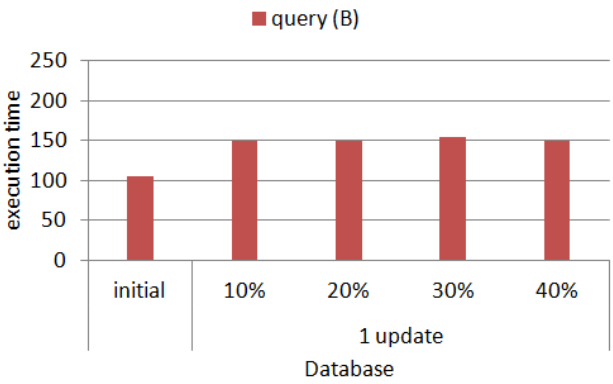

図 10　問合せ（B）の実行時間（エージング後）

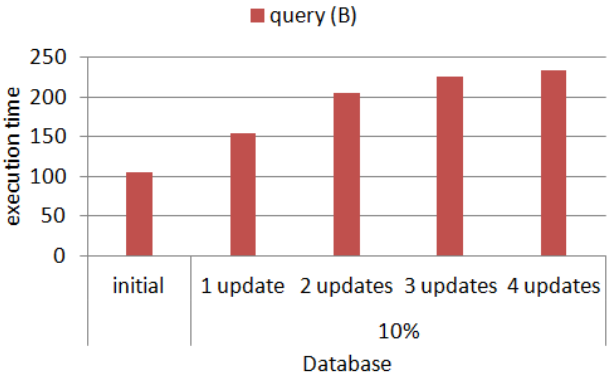

図 11　問合せ（B）の実行時間（局所的エージング後）

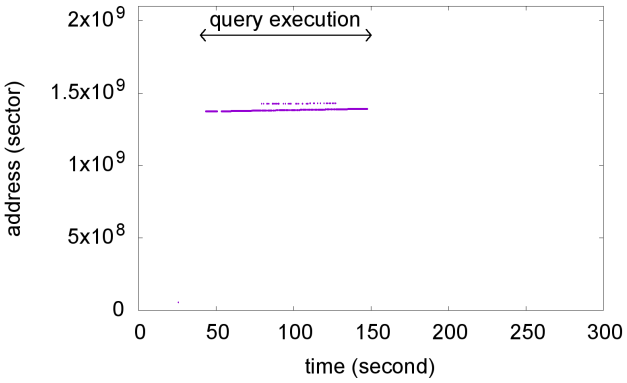

図 12　初期状態のデータベースの問合せ（B）の入出力挙動

数の増加に伴って実行時間が増加することが確認できた．

初期状態のデータベース，先頭から 40%を 1 回更新したデータベース，先頭から 10%を 4 回更新したデータベースに関して，ハードディスクの入出力を観察したグラフをそれぞれ図 12，図 13，図 14 に示す．初期状態では，クラスタ化索引を用いてシーケンシャルに近い読み込みを行っているが，先頭から 40%更新したデータベースにおいては，読み込む領域が分散し，ランダムに近いアクセスを行っていることが分かる．先頭から 10%を 4 回更新したデータベースは，更に広範囲にデータの格納場所が拡散しており，図 13 と比較してもエージングが更に進んでいることが観測される．

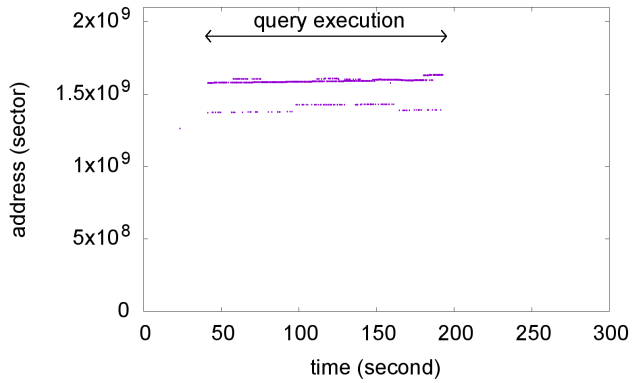

図 13　40%1 回更新データベースの問合せ（B）の入出力挙動

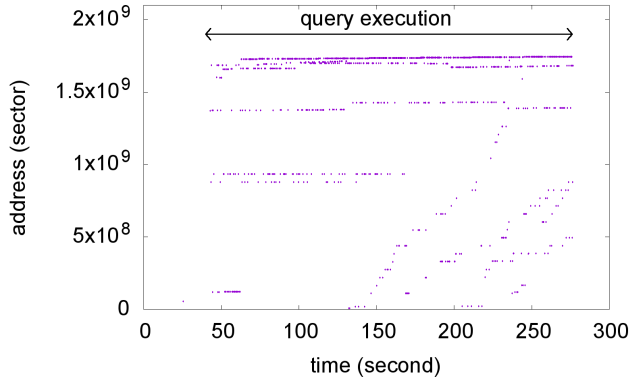

図 14　10%4 回更新データベースの問合せ（B）の入出力挙動

問合せ（A），(B) の実験結果から，表全体を走査する問合せはエージングの局所性を考慮しなくてもよいが，表の一部を読む問合せにおいては，データの更新量が同じであっても読み出す領域のエージング度合いに応じて問合せの実行時間に差異が生じるため，エージングの局所性を考慮し，エージングの度合いをアクセスメソッド毎に索引や番地の値に対応する形で持つ必要があることが確かめられた．そのため，あらかじめテスト用問合せを用いて表を分割して走査を行い，図 1 のような情報を得て，問合せ最適化に反映させる問合せコスト計算手法は重要であると考えられる．

3.4　コスト計算手法の有用性の検証

エージングの局所性を考慮した提案手法の有用性について，局所的にエージングしたデータベースを用い，いくつかの問合せについて実行時間との比較と考察を行った．

3.4.1　使用した問合せとデータベース

局所的にエージングしたデータベースの挙動を確かめるため，lineitem 表の先頭から 10%を 4 回更新したデータベースを実験に用いた．

提案手法のために用いたテスト用問合せを図 15 に示す．実験の際には，表をアクセスメソッドごとに 10 分割し，それぞれの部分空間に対してテスト用問合せを実行し，入出力コストの変化を得た．実行例として，エージング前のデータベースと，局所的なエージングが起きているデータベースにおいて，lineitem のクラスタ化索引である l_orderkey による索引走査に

```
SELECT SUM(l_extendedprice) FROM lineitem
WHERE l_orderkey < x AND l_orderkey > y          …(1)

SELECT SUM(l_extendedprice) FROM lineitem
WHERE l_partkey < x AND l_partkey > y            …(2)

SELECT SUM(p_size) FROM part
WHERE p_partkey < x AND p_partkey > y            …(3)

SELECT SUM(p_size) FROM part
WHERE p_retailprice < x AND p_retailprice > y    …(4)

SELECT SUM(p_size) FROM part or lineitem         …(5)
```

図 15　テスト用問合せ

```
SELECT SUM(l_extendedprice) FROM lineitem, part
WHERE p_partkey < x
AND l_orderkey > a AND l_orderkey < b
AND l_partkey = p_partkey
```

図 16　問合せ（C）

```
SELECT SUM(l_extendedprice) FROM lineitem, part
WHERE p_retailprice < x
AND l_orderkey > a AND l_orderkey < b
AND l_partkey = p_partkey
```

図 17　問合せ（D）

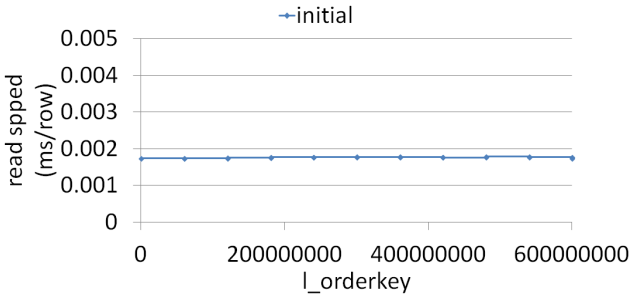

図 18　テスト用問合せにより測定した l_orderkey に対するエージング（初期状態）

ついてテスト用問合せ（1）を用いて入出力コストを測定したグラフを図 18 と図 19 に示す．同様にして全てのアクセスメソッドに対してテスト用問合せを実行し，以降の実験のコスト見積りに使用した．

提案手法の有効性の検証に使用した問合せは図 16，図 17 の二種類である．問合せ（C）はクラスタ化索引である p_partkey を用いて part 表を走査し，lineitem 表と結合する問合せ．問合

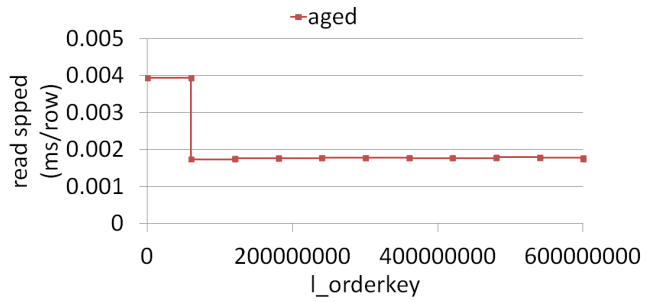

図 19　テスト用問合せにより測定した l_orderkey に対するエージング（局所的なエージング後）

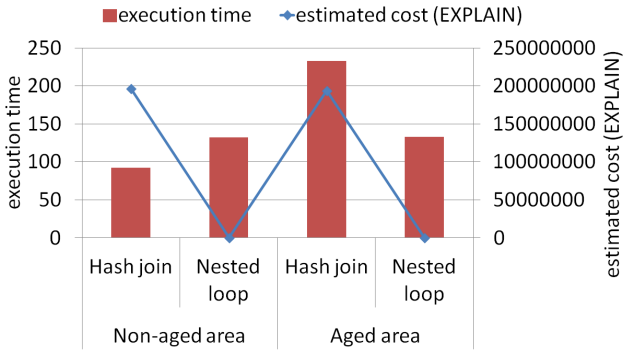

図 20　問合せ（C）の part 表選択率 0.005%における PostgreSQL による見積りと実行時間の比較

せ（D）は二次索引である p_retailprice を用いて part 表を走査し，lineitem 表と結合する問合せである．part 表の選択率は x の値で調整する．a と b の値を変化させることにより lineitem 表のエージングしている部分を読むかどうかの選択を行う．

二つの問合せについて，索引走査を用いるハッシュ結合と索引走査を用いるネステッドループ結合，それぞれの計画の提案手法による見積りコストと実行時間の比較を行った．また，PostgreSQL の問合せ最適化機構による計画選択と実行時間を比較し，既存の最適化機構がエージングの影響を反映できているかについての調査も行った．

3.4.2　検証用問合せによる有用性の検証

図 20 に問合せ（C）の part 表の選択率として 0.005%を用いた際の PostgreSQL による実行計画の見積りと実際の実行時間の比較を示す．PostgreSQL の見積りは explain コマンドの出力を用いた．この数値は実行時間ではないため，左の縦軸に実行時間，右の縦軸に explain による見積りコストの値を表示する．実行時間の結果より，この選択率においては，同一のデータベース内であってもエージング度合いの違いによって適切な計画が違うということが分かる．PostgreSQL によるコスト見積もりはハッシュ結合のコストがネステッドループ結合に比べて非常に大きく見積られており，エージングしている領域とエージングしてない領域どちらで問合せを実行した場合にもネステッドループ結合を選択するという結果になった．局所的なエージングの影響の反映は見られなかった．

図 21 に，提案手法の見積りと実行時間の比較を示す．提案手法は局所的なエージングの影響を取り入れ，アクセスコスト

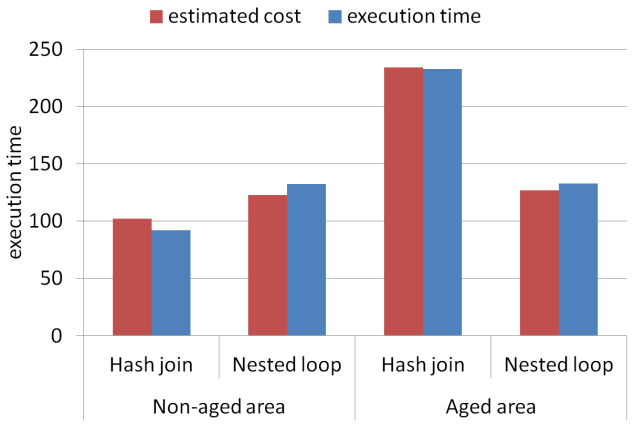

図 21　問合せ（C）の part 表選択率 0.005%における提案手法の見積りと実行時間の比較

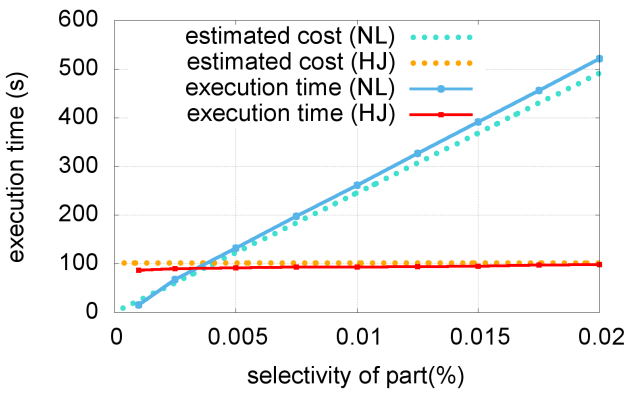

図 22　問合せ（C）の提案手法による見積りと実行時間比較（4 回更新後，エージングしていない部分の測定）

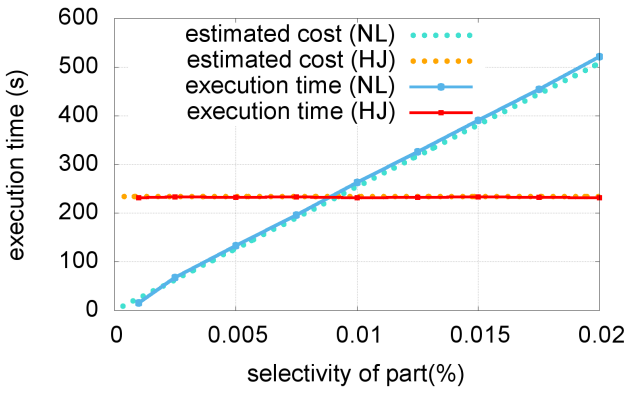

図 23　問合せ（C）の提案手法による見積りと実行時間比較（4 回更新後，エージングしている部分の測定）

の変化をコスト見積りに反映し，どちらの領域においても有利な実行計画を選択することが可能となっている．このため，本手法は問合せ実行計画の妥当性を向上させたと言える．

図 22，図 23 に，問合せ（C）を用いたハッシュ結合とネステッドループ結合それぞれの実行時間と見積りコストを示す．横軸が part 表の選択率，縦軸が時間を表す．図 22 は lineitem 表のエージングしていない部分で問合せを実行した場合，図 23 は lineitem 表のエージングしている部分において問合せを実行した場合である．

二つの図を比較すると，ハッシュ結合を用いた問合せがエー

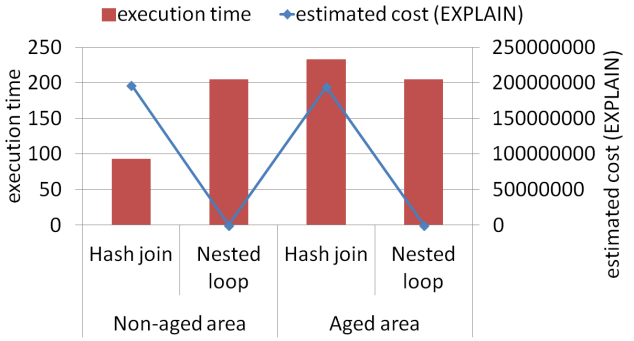

図 24　問合せ（D）の part 表選択率 0.0075%における PostgreSQL による見積りと実行時間の比較

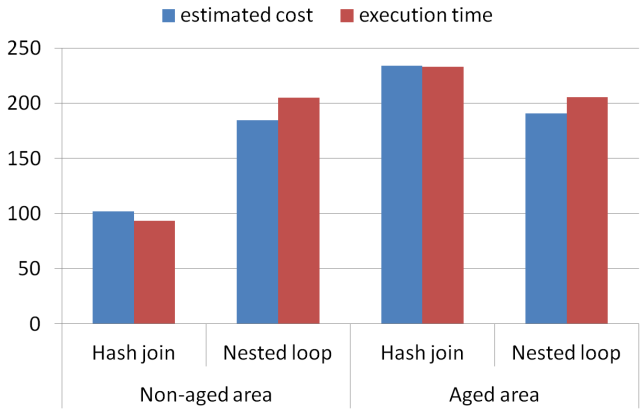

図 25　問合せ（D）の part 表選択率 0.0075%における提案手法の見積りと実行時間の比較

ジングに対してより高い感受性を示し，その結果，実行計画の有利不利が入れ変わる損益分岐点が右に移動したことが読みとれる．この現象によって図 20 や図 21 に示される，走査範囲内のエージングの有無による有利な計画の違いが発生したということが分かる．

問合せ（C）と同様に，問合せ（D）について part 表の選択率として 0.0075%を用いた際の PostgreSQL の見積りコストと実際の実行時間の比較を行った結果を図 24 に示す．こちらも，エージング度合いの違いにより適切な実行計画に違いが生じていることが読みとれる．PostgreSQL によるコスト見積りはやはりハッシュ結合の見積りコストが大きく，適切に計画を選択できていないことが分かる．

図 25 に，問合せ（D）について part 表の選択率を 0.0075%とした際の提案手法の見積りと実行時間の比較を示す．提案手法によって適切な実行計画のコストを小さく見積ることができており，この問合せにおいても実行計画の妥当性が向上している．

図 26，図 27 に，問合せ（D）を用いた提案手法による見積りと実際の実行時間を示す．問合せ（C）の際と同様，エージングにハッシュ結合が高い感受性を示していることが分かる．問合せ（D）においては part 表の走査に二次索引を用いたが，この問合せは lineitem を読むための時間が大きく，全体の実行時間内に対して支配的であったため，part 表の実行時間の増加が殆ど影響を及ぼさなかったと考えられる．

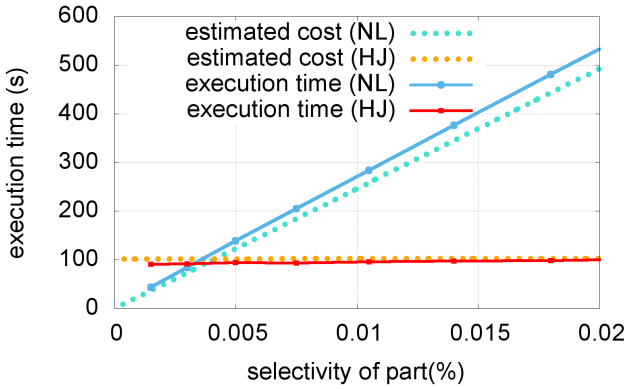

図 26　問合せ（D）の提案手法による見積りと実行時間比較（4 回更新後，エージングしていない部分の測定）

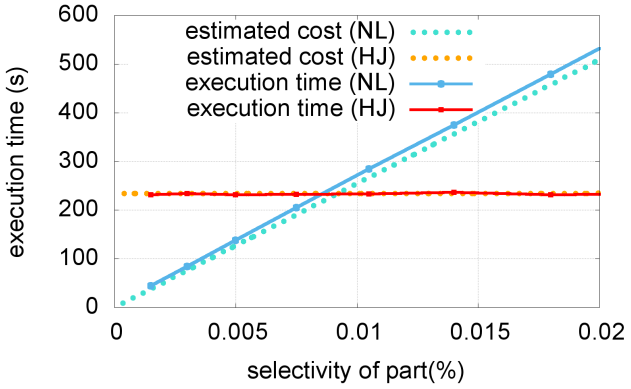

図 27　問合せ（D）の提案手法による見積りと実行時間比較（4 回更新後，エージングしている部分の測定）

以上より，局所的なエージングの影響を考慮した提案手法によるコスト見積りは，その影響を受ける問合せにおいて，よくエージングの局所性を反映し，正しい実行計画の選択に寄与しているということが言える．

4. 関 連 研 究

4.1 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は，システムを運用する上で従来から障害として認知されており，解消するための研究も，様々に行われてきた．1979 年には，データベースを再構築することでエージングを解消する研究 [3] が Sockut と Goldberg によってなされている．この当時，一般的に運用されていたデータベースは今に比べて小規模で，データ量も少なかった．そのため，一度データベースの稼働を停止し，再構築をオフラインで行う方法が有効であった．

取り扱うデータ量の増大に伴い，オフラインによる再構築は，長時間のシステム停止を要求するため次第に困難になった．更にデータベースが様々な場面で運用され，継続した稼働を必要とする場面も増え，システムの停止自体が困難になった．対して稼働中に行えるオンラインによる再構築が主流となり，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 [4]，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 [5] [6] の二系列が主だって考えられてきた．近年でも [7] [8] のようなオンラインによる再構成の研究は続けられている．

また，異なる視点からのアプローチも行われており，再構成を行うタイミングをデータベースを運用する側が正しく見極められるよう，エージングの程度を可視化する研究 [9] も存在する．

しかし，データベースの再構築によってデータ構造を理想的な状態にし続けることは現実的ではなく，エージングが発生しているタイミングでの問合せ最適化への悪影響は存在している可能性がある．よって，エージングの度合いを問合せ最適化側でも考慮することは，よりよい最適化の実行に繋がると考えられるため，著者らは，このコスト計算手法を構築した．

4.2 問合せ最適化

問合せ最適化とは，ユーザから受け取った問合せを，効率的な実行計画へと変換し，データベースシステムに渡す機能である [10]．取り扱うデータ量が増えるほど，実行計画の選択による実行時間の変動も大きくなるため，問合せ最適化は，以前に比べて重要性が増していると言える．

データベースシステムの重要な一要素である問合せ最適化の研究は，従前から様々に行われてきた [11] [12]．CPU のコア数増加に伴い，問合せ最適化を並列化する研究もなされてきた. [13]．更に近年では，豊富なストレージ資源を活用し，メモリを犠牲にして問合せの実行速度を上げる研究 [14] など，技術の進歩に応じた問合せ最適化の研究が行われている．また，データへのアクセス方法に着目した，問合せ最適化に用いる I/O コスト計算に関しての研究も行われている．コストベースの最適化について，1997 年にはシーケンシャルなアクセスとランダムなアクセスの I/O コストを分けてコストの計算をする研究 [15] が登場している．これらは全て，ハードディスクにデータを格納している前提で考えられてきたが，近年は SSD におけるランダムアクセスのコストがハードディスクに比べ小さいことに着目し，従来のモデルを見直す研究も盛んに行われている [16] [17] [18]．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アクセス方法の選択に関して，コスト計算手法の提案と考察を行った．

5. 終わりに

本論文では，著者らは，更新が頻繁な領域において起こり得る局所的なエージングを考慮した，問合せ最適化のコスト計算モデルを提案した．そして，その重要性を検証するために，PostgreSQL を対象とし，問合せ実行時間とその際の入出力命令の挙動を観測した．その結果，局所的なエージングは，表の全てを読む問合せにおいては幅広くエージングした場合と同じ挙動を示したが，エージングした領域を読む問合せにおいては，実行時間が更新回数に応じて増加するという結果を得た．更に構築したコスト見積り手法を用いて実行時間との比較を行った結果，よくエージングの局所性を反映し，実行計画の選択の妥当性を向上させることができているとの結果を得た．今後は，より複雑性の高い問合せへの適用と検証や，PostgreSQL への実装などに取り組んでいく．

文　献

[1] 加藤千裕, 早水悠登, 合田和生, 喜連川優. 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. 第 7 回データ工学と情報マネジメントに関するフォーラム, 2015.

[2] 加藤千裕, 早水悠登, 合田和生, 喜連川優. カーネルトレーサを用いた postgresql の入出力挙動の観測と一考察（to appear）. 情報処理学会全国大会, 2016.

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

[4] 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.

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

[6] 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.

[7] 合田和生, 喜連川優. データベース再編成機構を有するストレージシステム. 情報処理学会論文誌. データベース, Vol. 46, No. 8, pp. 130–147, 2005.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.