

修 士 論 文

エージングの局所性を考慮した
データベース問合せ最適化方式に関する研究

A Study on Database Query Optimization
with Awareness of Aging Locality

指導教員 豊田 正史 准教授



東京大学 大学院情報理工学系研究科
電子情報学専攻

氏 名 48-146410 加藤 千裕

提 出 日 平成28年2月4日

概要

データベースシステムは、更新処理を繰り返すことにより、格納効率が低くなり、実行性能が低下する。当該現象はエージングと呼ばれる。エージングが発生すると、表へのアクセスコストに影響が生じ、問合せの実行時間が変化する。そのため、コストベースの問合せ最適化を行う際には、エージングの影響を考慮し、コストを是正して最適な計画を選択する必要がある。しかし、従来の問合せ最適化機構は必ずしもエージングの影響を定量的に考慮していなかった。そこで本論文では、エージングの局所性に着目し、エージングによるアクセスコストの変動を考慮する問合せコスト見積りモデルを提案する。予備実験として、エージングが問合せ最適化に与える影響を詳細に観察し、それを踏まえ、実験によって当該モデルの有用性を検証し、従来手法によれば実行計画が誤って選択されていた場合に、本手法を適用することでより短時間の実行計画が正しく選択されることを明らかにする。

目次

第1章	はじめに	7
1.1	データベースシステムにおけるエージング	7
1.2	本論文の構成	8
第2章	関連研究	9
2.1	データベースのエージングと再構築	9
2.2	問合せ最適化	11
第3章	エージングを考慮するコストモデル	13
3.1	テスト用問合せによるエージングの測定	13
3.2	エージングを考慮したコスト計算	15
第4章	エージングを考慮するコストモデルの検証	17
4.1	予備実験	17
4.1.1	基礎的な二種類の走査方法	17
4.1.2	実験環境	19
4.1.3	問合せ最適化へのエージングの影響	20
4.1.4	入出力挙動の観測	27
4.2	コストモデルの有用性の検証と考察	32
4.2.1	実験環境	32
4.2.2	使用したデータベースとテスト用問合せ	33
4.2.3	相対的なエージング度合いの測定	34
4.2.4	有用性の検証	36

第5章 おわりに	48
付録A	50
A.1 ソースコードの変更点	50
A.1.1 Systemtap	50
A.1.2 DBgen	51
A.2 図表	53
参考文献	58
発表文献	61
謝辞	62

目 次

2.1	エージングの例	10
2.2	問合せ最適化機構	11
3.1	エージングしたデータベースのアクセスコスト	14
3.2	走査する領域とデータ分布密度	14
4.1	全表走査	18
4.2	索引走査	18
4.3	問合せ (A)	20
4.4	問合せ (B)	20
4.5	各カラムの相関係数	21
4.6	lorderkey を用いた初期状態データベースの問合せ実行時間	22
4.7	lorderkey を用いた 10%更新データベースの問合せ実行時間	22
4.8	フルテーブルスキャン実行時間比較	23
4.9	インデックススキャン実行時間比較 (lorderkey)	23
4.10	インデックススキャン実行時間比較 (lpartkey)	24
4.11	損益分岐点比較 (問合せ (A))	25
4.12	損益分岐点比較 (問合せ (B))	25
4.13	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	26
4.14	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 (lorderkey)	27
4.15	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 (lpartkey)	27

4.16	問合せ (C)	28
4.17	問合せ (D)	28
4.18	初期状態の問合せ (A) の入出力挙動	29
4.19	エージング後の問合せ (A) の入出力挙動	29
4.20	初期状態の問合せ (B) の入出力挙動	30
4.21	エージング後の問合せ (B) の入出力挙動	31
4.22	問合せ毎のエージングの影響比較	31
4.23	テスト用問合せ	33
4.24	l_orderkey を用いた lineitem 表の読み込み速度	34
4.25	l_partkey を用いた lineitem 表の読み込み速度	34
4.26	p_partkey を用いた part 表の読み込み速度	35
4.27	p_retailprice を用いた part 表の読み込み速度	35
4.28	全表走査による表の読み込み速度	36
4.29	検証用問合せ	37
4.30	検証用問合せ (A) の見積りと実行時間比較 (初期状態)	39
4.31	検証用問合せ (A) の見積りと実行時間比較 (エージング後)	40
4.32	検証用問合せ (B) の見積りと実行時間比較 (初期状態)	41
4.33	検証用問合せ (B) の見積りと実行時間比較 (4回更新後)	42
4.34	検証用問合せ (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	42
4.35	検証用問合せ (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	43
4.36	検証用問合せ (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)	44
4.37	検証用問合せ (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)	45
4.38	検証用問合せ (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	45

4.39	検証用問合せ (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	46
4.40	検証用問合せ (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)	46
4.41	検証用問合せ (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)	47
A.1	Lorderkey を用いた 20%更新データベースのスキャン時間	54
A.2	Lorderkey を用いた 30%更新データベースのスキャン時間	54
A.3	Lorderkey を用いた 40%更新データベースのスキャン時間	55
A.4	Lorderkey を用いた 50%更新データベースのスキャン時間	55
A.5	Lorderkey を用いた 60%更新データベースのスキャン時間	56
A.6	Lorderkey を用いた 70%更新データベースのスキャン時間	56
A.7	Lorderkey を用いた 80%更新データベースのスキャン時間	57
A.8	Lorderkey を用いた 90%更新データベースのスキャン時間	57

表 目 次

4.1 実験環境	19
--------------------	----

第1章 はじめに

1.1 データベースシステムにおけるエージング

データベースに対して、挿入や削除を行い更新処理を繰り返すと、例えば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肥大化の如き現象が発生することがあり、問合せ実行が必ずしも効率的と言えない状態になる場合がある、その結果、データベースの性能が低下する。当該現象はエージングと呼ばれる。

筆者は、エージングが問合せ最適化に与える影響に着目した。問合せ最適化機構は、データベースがユーザから問合せを受け取り、適切な実行計画に直す役割を持つ。実行計画の決定に際しては、可能な計画についてそれぞれのコストを見積り、そのコストが一番小さくなる計画を選択する。しかし、エージングが発生したデータベースにおいては、問合せコストに影響が生じ、適切な実行計画が変化する可能性がある。そのためコストベースの問合せ最適化を行う際には、エージングによるアクセスコストの変動を反映して問合せのコスト見積りを行うことが重要であるが、従来の問合せ最適化機構は必ずしもエージングの影響を定量的に考慮していなかった。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

1.2. 本論文の構成

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．

1.2 本論文の構成

本論文の構成は以下の通りである．

第2章 本論文で取り扱うテーマである，データベースの問合せ最適化機構と，エージングについて，詳細な説明を述べ，その関連研究を紹介し，本研究の立ち位置を明らかにする．

第3章 本論文の提案である，局所的なエージングを考慮する見積りコストモデルについて，詳細な解説を行う．

第4章 予備実験として，問合せ最適化へのエージングの影響を観察する．更に本論文で提案する見積りコストモデルが局所的なエージングの影響を反映できているかを実験によって観測し，その有用性について検証する．

第5章 全体のまとめと今後の課題について述べる．

第2章 関連研究

2.1 データベースのエージングと再構築

エージングとは、データベースの格納構造が、更新処理を繰り返したことにより劣化することである。その例を図2.1に示す。

基本的にデータベースを作成し、データをロードした段階では、データが実際に格納されているディスク上においても、論理アドレスに従って表ごとに理路整然と格納されている場合が多い。このため、論理的にシーケンシャルなアクセスをした際には、物理的にもシーケンシャルなアクセスが行われ、そのI/Oのコストは小さく抑えられる。しかし、図2.1の下部のように、更新処理を繰り返すと、エージングが発生する。この状態では論理的にはシーケンシャルなアクセスをしているつもりでも、物理的にはディスク上の距離が離れている場合があり、アクセスコストがランダムなアクセスに近づき、データベースの問合せのパフォーマンスが低下する。

データベースのエージングは、システムを運用する上で悪影響を及ぼす可能性が認知されており、解消するための研究も昔から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[1]がSockutとGoldbergによってなされている。この当時、一般に運用されていたデータベースは今に比べて極めて小規模で、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。

データセンタに集まるデータ量が増加すると、次第に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されることとなった。他方、データベースシステムの応用範囲が増えたことにより、継続した稼働を求められる場面が生まれ、システムの停止そのものが難しいケースも現れた。そのため、稼働中にオンラインによって

2.1. データベースのエージングと再構築

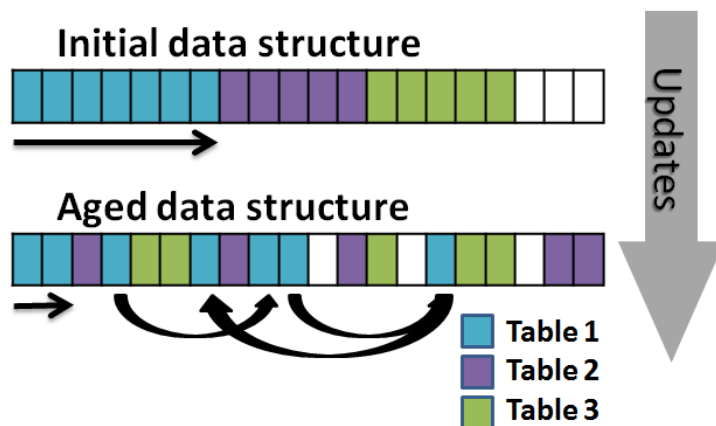


図 2.1: エージングの例

再構築をする方法が提案され、主流となった。

この方法には大きくわけて二つ存在する。一つは、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である。1997年に Sockut らによって提案されている [2]。この論文では、複製による再構成を行ったあと、新旧のデータベースの関係をマッピングした表を保持し、それを元に新たなデータベースに更新記録を適用している。もう一つは、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築する方法である [3][4]。データベースを小さな単位に分け、それぞれの単位内における再構成の時間を小さくするとともに、再構成を行っている部分にだけロックをかけることにより、継続したデータベースの運用を可能にしている。2005年には、ストレージの計算資源が豊富になりつつあることを受け、サーバの計算機資源を用いるのではなく、大規模ストレージ装置の有する計算資源をデータベースの再構築に用いる研究 [5] が発表され、翌年には複数のストレージ環境を想定し、システムの監視から再構成までを行う ORE というフレームワークを提案した研究 [6] が発表されるなど、オンラインによる再構成の研究は近年においても続けられている。

また、異なる視点からのアプローチも存在する。これまでの再構成は、そのタイミングが管理者に委ねられていることが多く、そのタイミングによっては効率のよ

2.2. 問合せ最適化

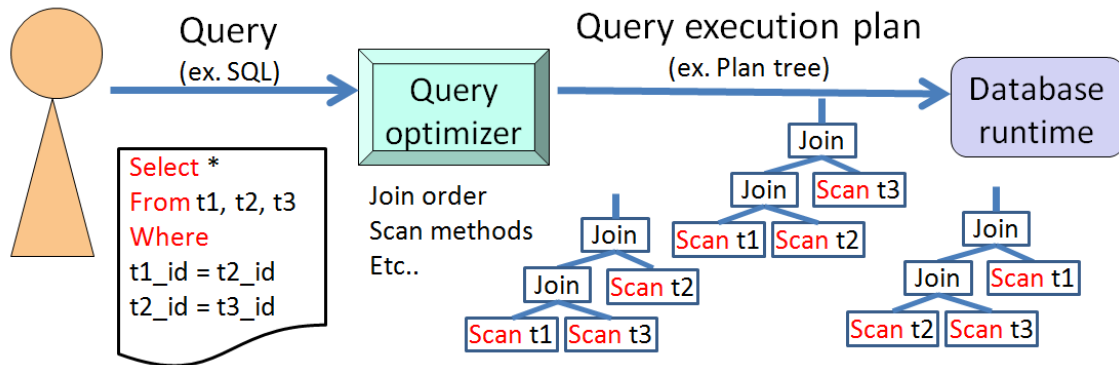


図 2.2: 問合せ最適化機構

い再構成と言い難い場合が存在したが、それを見極めるのは困難であった。そこで、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 [7] も行われている。

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クエリ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、本論文において、エージングしたデータベースを対象とした、クエリ最適化のアクセスコストのモデル化を提案した。

2.2 問合せ最適化

問合せ最適化は、大量のデータを運用するデータベースシステムにおいて、データに対する検索や統計処理などを効率よく実行するための鍵となる要素の一つである [8]。その概略図を、図 2.2 に示す。

ユーザがデータベースに問い合わせる際は、その問合せは、例えば図に示す SQL のような宣言的な形で与えられ、データベース内における実際の挙動に関しては、基本的には規定しないことが多い。問合せ最適化機構がユーザからの問合せを受け取

2.2. 問合せ最適化

り，データベースが処理できる実行計画に変換することで，ようやく問合せを実行することができる。この時，一つの問合せにおける実行計画は，データの走査方法や表の結合方法などの違いから複数存在する場合があります，問合せ最適化機構は，その実行コストを計算し，より高速に処理を行うことができると考えられる実行計画を選択し，データベースに実行させる役割を持つ。

コスト見積りと実際の実行時間との間に差異がある場合，問合せ最適化機構が選ぶ実行計画が必ずしも最適なものとはならない場合があるため，アクセスコストを変動させる可能性のあるエージングを考慮することは重要である。特に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，以前に比べて更にその重要性が増していると言える。

問合せ最適化の研究は，従前から様々に行われてきた [9][10]。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。[11]。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 [12] 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 コスト計算に関する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 [13] 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 [14][15][16]。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，モデルの提案と考察を行った。

第3章 エージングを考慮するコストモデル

3.1 テスト用問合せによるエージングの測定

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3.1のようなグラフを得る。図3.1の

3.1. テスト用問合せによるエージングの測定

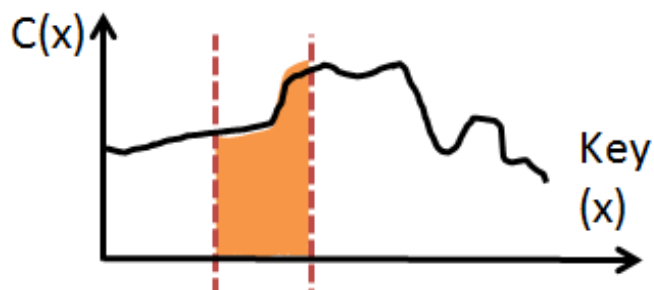


図 3.1: エージングしたデータベースのアクセスコスト

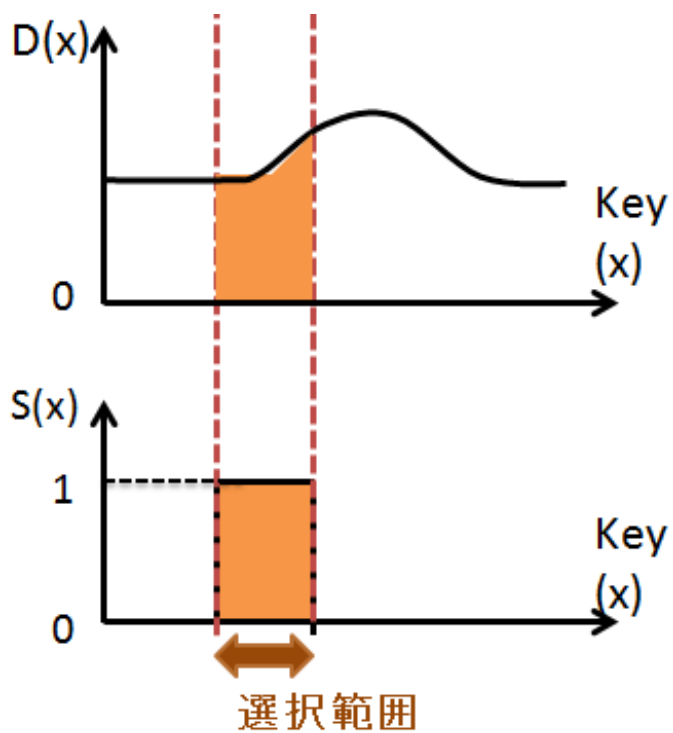


図 3.2: 走査する領域とデータ分布密度

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.

3.2 エージングを考慮したコスト計算

実際の間合せが入力されると、テスト用間合せにより求めたエージング度合いを、実行する間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できる。

$$\Gamma = \int S(x)D(x)C(x)dx \quad (3.1)$$

- 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 3.2 に示す。 $C(x)$ はテスト用間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間合せに関しても、単一表の走査コストを組み合わせるにより計算を行う。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。

$$\begin{aligned} \Gamma &= \int S_{t1}(x)D_{t1}(x)C_{t1}(x)dx \\ &+ \int j_{t12}(x)\{S_{t1}(x)D_{t1}(x)\}C_{t2}(x)dx \end{aligned} \quad (3.2)$$

- 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

3.2. エージングを考慮したコスト計算

- $C_{t1}(x), C_{t2}(x)$: 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。

$$\begin{aligned}\Gamma &= \int S_{t1}(x)D_{t1}(x)C_{t1}(x)dx \\ &+ \int S_{t2}(x)D_{t2}(x)C_{t2}(x)dx\end{aligned}\tag{3.3}$$

- 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。

第4章 エージングを考慮するコストモデルの検証

4.1 予備実験

本研究では、問合せ最適化に局所的なエージングの影響を反映するための見積りコストモデルを作成することを目的とする。そのための第一歩として、小規模なデータベース実験環境において基礎的な二つの走査方法である全表走査と索引走査を用い、問合せ処理の性能試験を行うことでエージングがデータベースにもたらす影響について走査方法ごとに詳細に観測する。また、実行結果を既存のデータベースによるコスト見積りと比較し、既存の問合せ最適化が影響を予測できているかについて考察する。更にカーネルトレーサを用い、実際に問合せを実行中に SCSI 層において発行された入出力の挙動を観察することで、具体的なエージングの挙動を観測し、如何にしてエージングが走査コストに影響を与えるかについて考察を行う。

4.1.1 基礎的な二種類の走査方法

本節においては、著者が実験において比較した二つの走査方法について、詳細な説明を行う。

4.1.1.1 全表走査

全表走査は、問合せにおいて指定された表を先頭から順に全て読み、問合せに合う行を検索する走査方法である。その簡略図を図 4.1 に示す。一般的に、全表走査においてはシーケンシャルな I/O が発行されるため、一つ一つの I/O の実行時間は

4.1. 予備実験



図 4.1: 全表走査

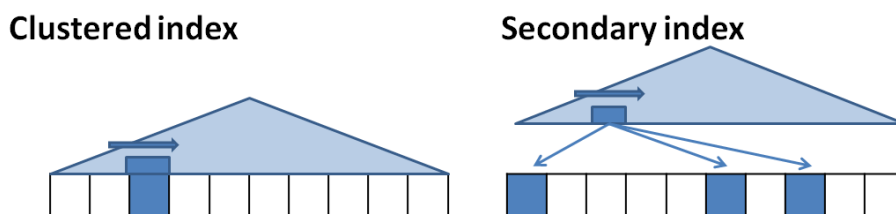


図 4.2: 索引走査

小さい。そのため、選択率が高く、表の大部分を走査する必要のある問合せに関しては、全表走査が有利になる場合が多い。一方、問合せの選択率に関わらず表の全てを読み込むため、選択率が小さく、読むべき行が少ない場合は、効率のよい方法とは言い難い。

4.1.1.2 索引走査

索引走査は、あらかじめ作成した索引を用い、問合せによって指定された表の一部を検索する走査方法である。そのため、問合せにおいて指定された範囲が小さいほど実行時間が短くなる。その挙動は、索引の性質によって二つに分類される。

一つ目のクラスタ化索引は、表の並びに沿って作成された索引である。一般的には主キーがこれに該当する場合が多い。その概略を図 4.2 の左図に示す。クラスタ化索引によって読み込む範囲を絞り込む場合、絞り込んだ範囲内においては読み込むデータがシーケンシャルに並んでいるため、シーケンシャルに近い I/O が発行され、高速に表の一部を読むことができる。

二つ目の二次索引は、表の並びとは関係のない列について作成された索引である。

4.1. 予備実験

表 4.1: 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)
データ作成	Dbgen ver 2.17.0

その概略図を図 4.2 の右側に示す。二次索引は基本的にランダムな I/O を発行すると期待され、一般には I/O 一回の実行速度は全表走査やクラスタ化索引による索引走査に劣る。一方で問合せによって指定された範囲のみを読み込むため、選択率の低い問合せの実行においては有効な走査手段となることがある。

4.1.2 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問合せの作成を行った。PostgreSQL の設定パラメータはすべて初期状態とした。

また、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、エージングしたデータベースを用意した。用意したデータベースは、初期状態のデータベース 1 つ、lineitem 表の 10%, 20%... 90% に 1 回更新処理を行った 9 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

4.1. 予備実験

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

図 4.3: 問合せ (A)

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

図 4.4: 問合せ (B)

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。

4.1.3 問合せ最適化へのエージングの影響

筆者は、エージングが問合せ最適化に与える影響に関して、複数の問合せを用い、基礎的な二つの走査方法について観測し、考察を行う。また、既存の問合せ最適化機構がエージングの影響を反映しているかについても観察を行う。

4.1.3.1 使用するデータベースと問合せ

計測に使用する問合せは、図 4.3 と図 4.4 に示す二つである。問合せ (A) は主キーである `l_orderkey` を用いた問合せとなっており、クラスタ化した索引による走査が行われるため、索引走査を実行する場合にも全表走査に近い動作を行う。問合せ (B) は二次索引を用いた問合せであるため、索引走査の際には、理想的には完全にランダムな I/O による走査であることが期待される。

それぞれ異なるエージング度合いを持つデータベースにおいて、問合せ (A) と問合せ (B) を全表走査と索引走査の二つの走査方法で実行し、索引の種類と走査の種類ごとにエージングが与える影響について観測を行う。

なお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図 4.5 に示される。

4.1. 予備実験

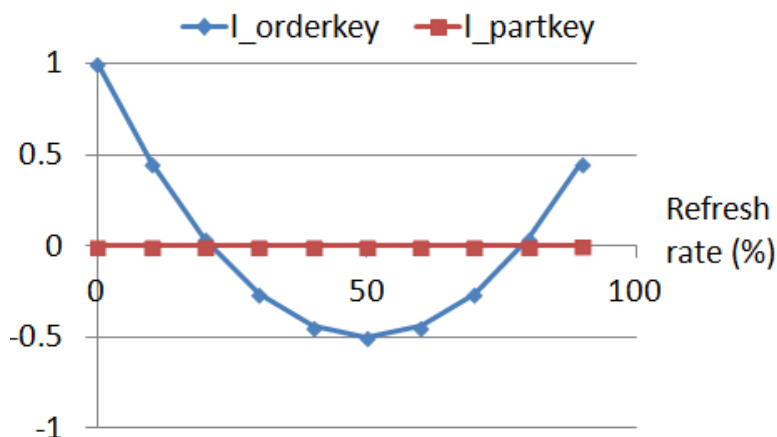


図 4.5: 各カラムの相関係数

更に PostgreSQL の最適化機構によるコスト見積りがエージングの影響を考慮出来ているかについて計測するべく、問合せ (A) と問合せ (B) について、最適化機構のコスト見積りを実際の実行時間と比較した。この時、最適化機構のコスト見積りとしては、PostgreSQL の explain コマンドを用いた結果を使用した。このコマンドによって出力されるコスト見積りの単位は時間ではないため、初期状態の値を 1 とし、相対的な大小で比較を行った。

4.1.3.2 実行時間の計測結果と考察

問合せ (A) を用い、更新率の違うデータベースごとに実行時間を計測した結果の一部を図 4.6 から図 4.7 に示す。グラフの横軸が x で指定した問合せ (A) の選択率、縦軸が実行時間を表している。図から、更新処理を行うことで、全表走査と索引走査にかかる時間が共に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。その結果は付録の図 A.1 から図 A.8 に記載する。

選択率を 1.6% で固定して、データベースの更新率を横軸に取りプロットしたグラフが図 4.8、図 4.9 である。全表走査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。全表走査においては、論理構造が実際のデータ構造と食い違っ

4.1. 予備実験

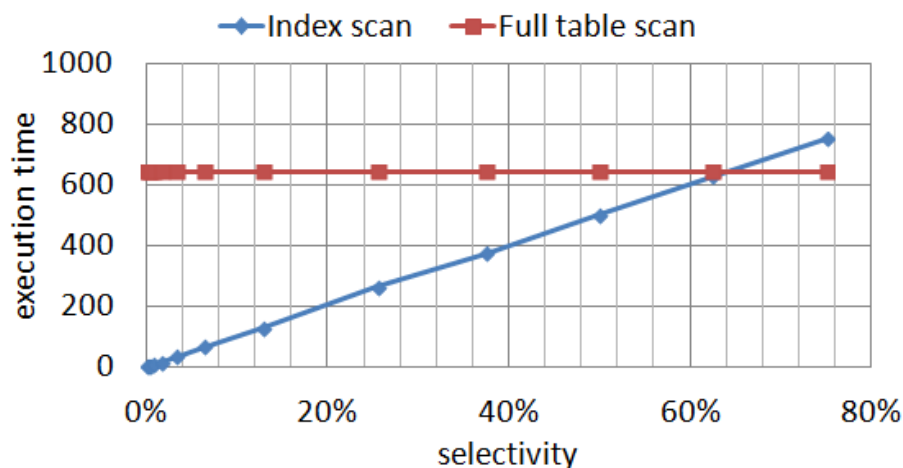


図 4.6: Lorderkey を用いた初期状態データベースの問合せ実行時間

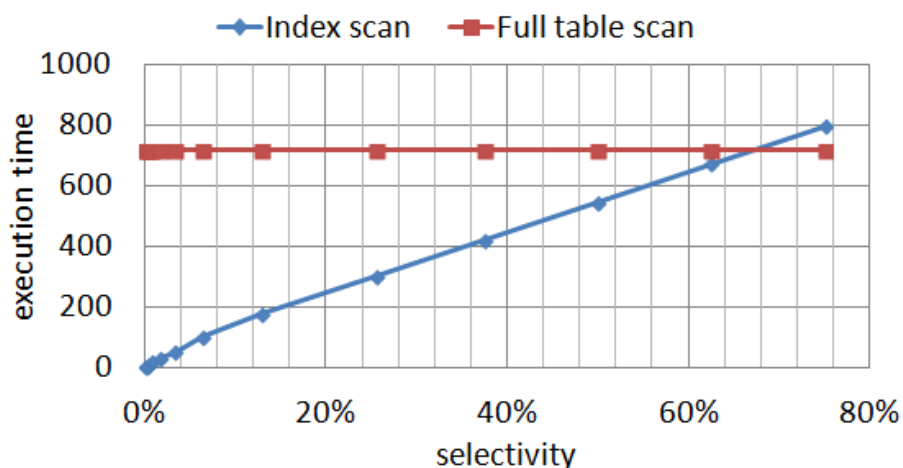


図 4.7: Lorderkey を用いた 10%更新データベースの問合せ実行時間

ているために、論理的にはシーケンシャルなアクセスを行っている場合にも、実際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、図 4.9 に示される、クラスタ化索引を用いた索引走査においては、初期状態とそれ以外のデータベースで顕著な違いが見られるが、エージングしたデータベース同士は一定の増減の傾向は見られない。これは DBgen によって作成した更新用関数が、先頭から Lorderkey に従って更新を行うものであったため

4.1. 予備実験

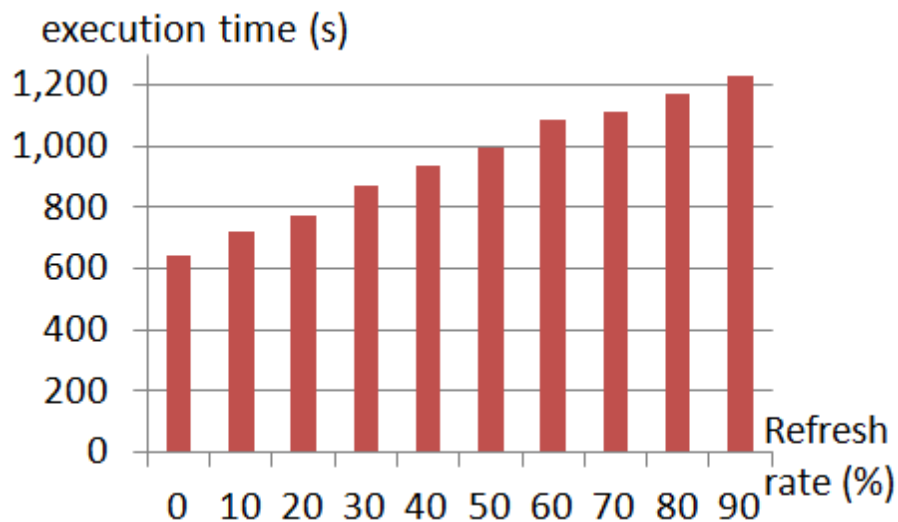


図 4.8: フルテーブルスキャン実行時間比較

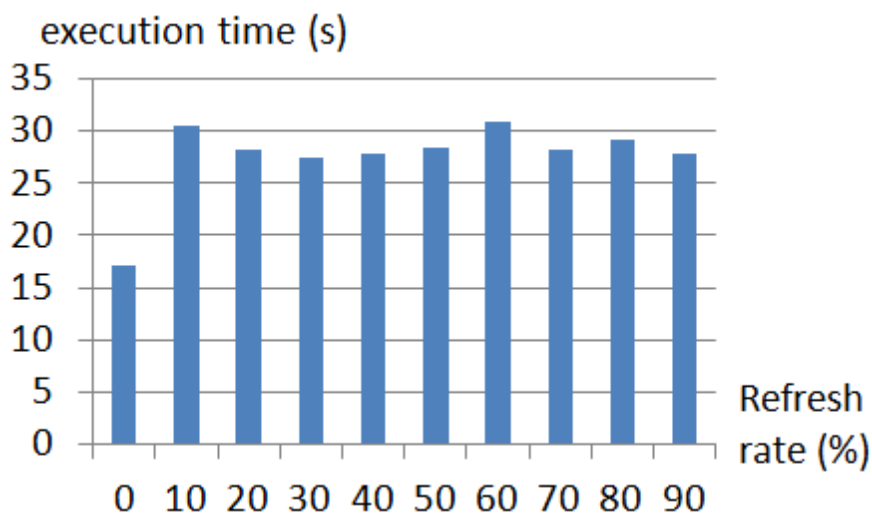


図 4.9: インデックススキャン実行時間比較 (Lorderkey)

に、表の先頭から選択率 1.6%の範囲を読む問合せ (A) を実行する際には、10%以上エージングしたデータベースの場合には読み込む範囲が全てエージングした状態になっていることが原因と考えられる。索引走査においても、初期状態と 10%エー

4.1. 予備実験

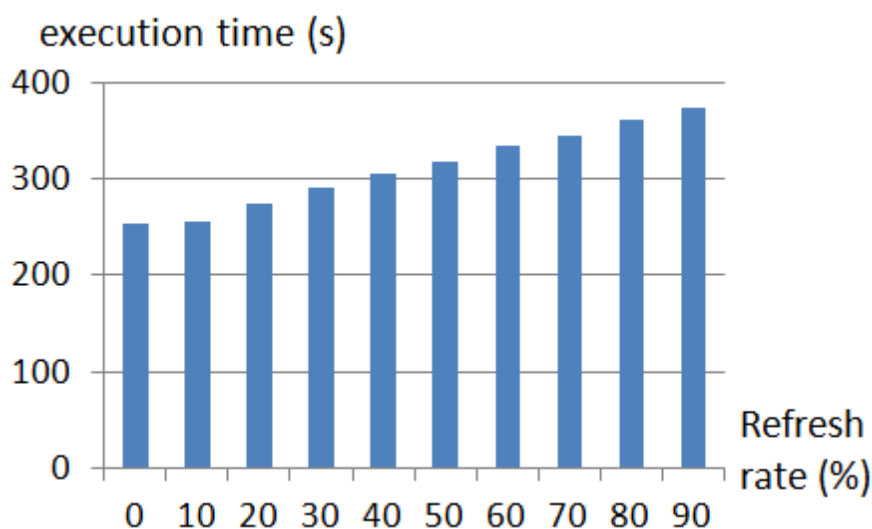


図 4.10: インデックススキャン実行時間比較 (l_partkey)

ジングしたデータベースを比較することにより、エージングの影響が存在することが観察できた。

同様の実験を、二次索引を用いる問合せ (B) について行った。この列は図 4.5 の通り、表の並びとの相関関係が小さく、ランダムなアクセスが行われることが期待される。更新率を横軸に取って索引走査について実行速度を比較した結果が図 4.10 である。選択率は 0.01% で固定とした。なお全表走査に関しては図 4.8 と同様の結果である。二次索引を使ったランダムなアクセスでも、エージングの影響は見られ、エージングした領域が増大するごとに、実行時間が増加するという結果となった。この原因については、次章で詳しく観察する。

この結果に基づいて算出した問合せ (A)、問合せ (B) の損益分岐点の変動を、図 4.11、図 4.12 に示す。図 4.11 では、全体的に、一般に知られている損益分岐点と比べ、索引走査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、索引走査でもシーケンシャル I/O に近い読み込みを行ったため、アクセスコストが小さく、データの範囲を限定して読むことができる。更新率によっては、初期状態と比べ最大で

4.1. 予備実験

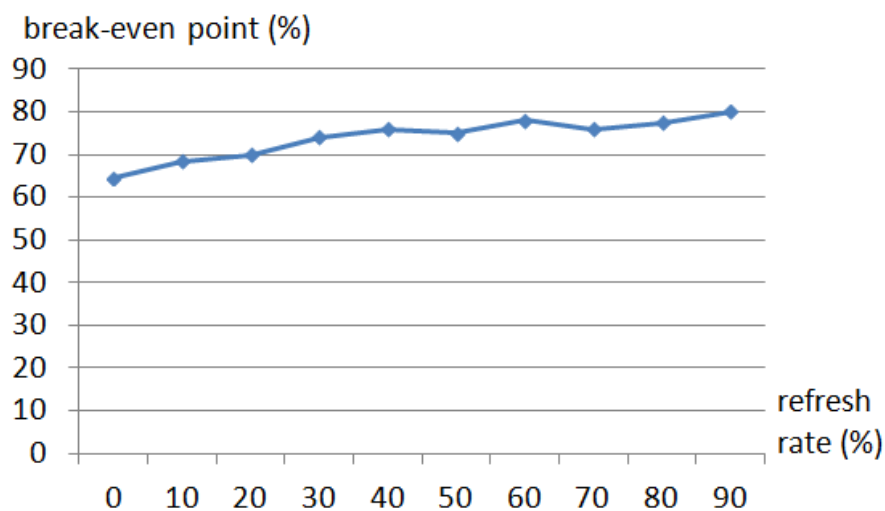


図 4.11: 損益分岐点比較（問合せ（A））

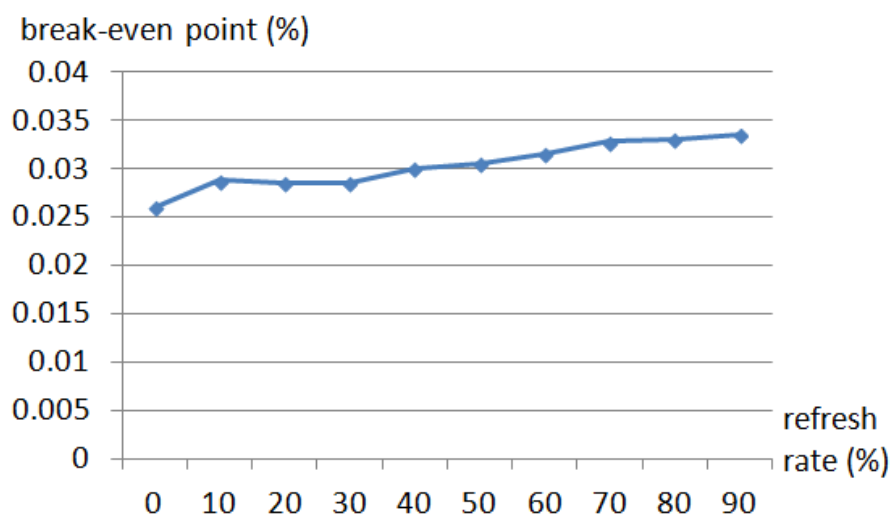


図 4.12: 損益分岐点比較（問合せ（B））

24%の誤差が生じており，アクセスコストの変動が走査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。

問合せ（B）を用いて計測した図 4.12 も同様の傾向を示した．損益分岐点の誤差

4.1. 予備実験

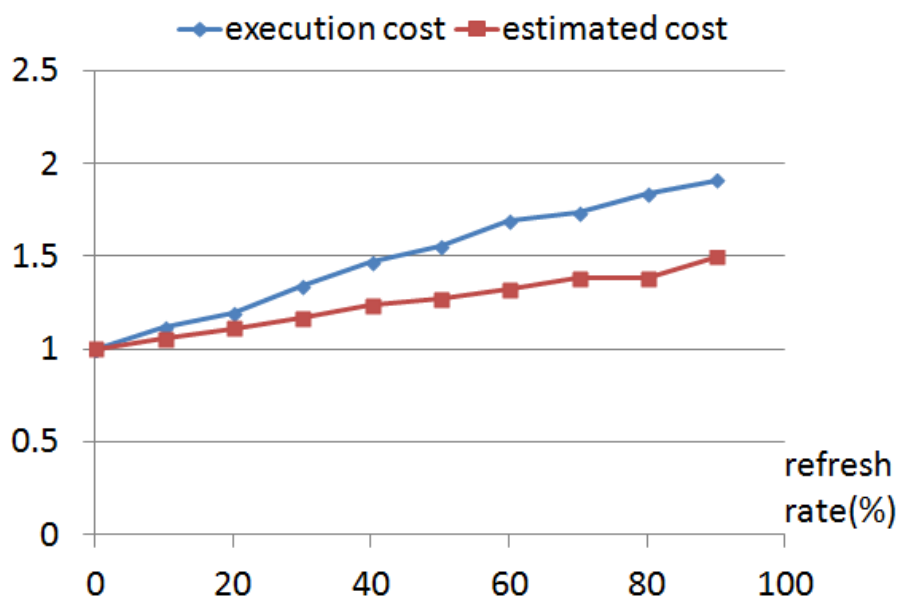


図 4.13: シーケンシャルスキャンに関するコスト見積りと実行時間の比較

としては初期状態と比べて最大で29%の変動が起きていることが分かり、問合せ最適化への影響が発生する可能性が確認できた。

4.1.3.3 問合せ最適化機構によるコスト見積りとの比較

PostgreSQLの問合せ最適化のコスト見積りとの比較結果を図4.13から図4.15に示す。但し索引走査に関しては、問合せ(A)は選択率1.6%、問合せ(B)は選択率0.01%としている。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。現状のPostgreSQLの問合せ最適化機構のコスト見積りと、実行時間には誤差があることが確認できる。全表走査に関する図4.13において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%ほどの誤差が生じた。索引走査に関しては、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%程度であった。現状のPostgreSQLの問合せ最適化機構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ると言える。

4.1. 予備実験



図 4.14:

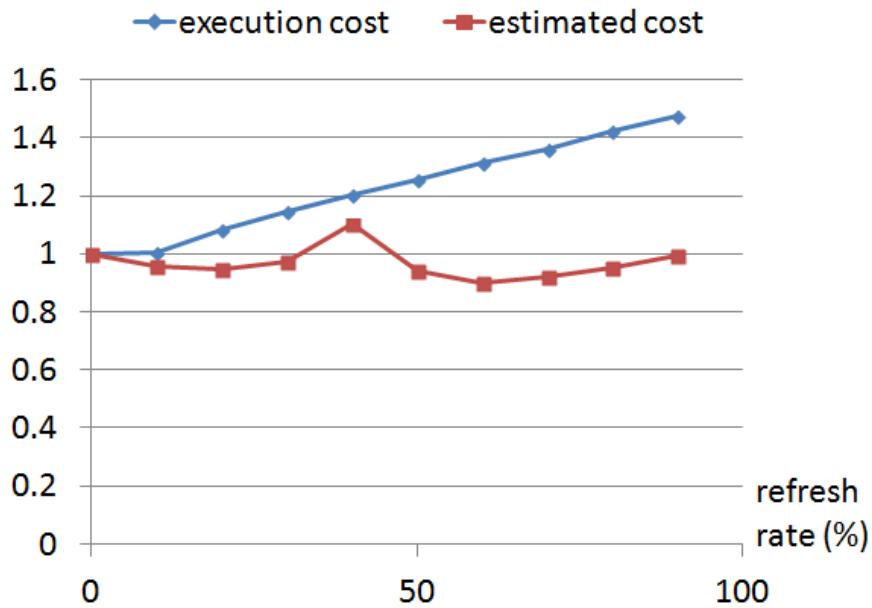


図 4.15: インデックススキャンに関するコスト見積りと実行時間の比較 (l_partkey)

4.1.4 入出力挙動の観測

次に本研究において、筆者は、SCSI層における入出力命令の挙動を観察し、エーシングの具体的な影響を考察する。

4.1. 予備実験

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

図 4.16: 問合せ (C)

```
SELECT SUM(l_extendedprice) FROM part  
JOIN lineitem ON p_partkey = l_partkey  
WHERE l_orderkey < 1024000
```

図 4.17: 問合せ (D)

4.1.4.1 使用したデータベースと問合せ

エージングによる入出力挙動の変化を計測するため、エージングしていないデータベース（初期状態）と、90%のデータを、TPC-H の定める更新問合せで更新しエージングさせたデータベース（エージング後）を使用する。なお、どちらのデータベースも、実験前に `vacuum` コマンドにより削除ページの回収を行った。問合せとしては、図 4.16 と図 4.17 に示す `lineitem` 表の一属性値の総和を求める問合せ (C) と、`lineitem` 表と `part` 表の二つを結合する問合せ (D) を用いた。これらの問合せを実行中に、`Systemtap` と呼ばれるカーネルトレーサを用いて、Linux カーネル内で SCSI 命令をトレースすることにより、入出力挙動を観測した。

なお、`Systemtap` はそのままでは SCSI 層の入出力を観察できないため、プローブを追加、変更している。これについては付録に記載する。

4.1.4.2 実行計画と実行時間

問合せ (C) は全表走査、問合せ (D) は `lineitem` 表と `part` 表を索引走査しネストドロープ結合を行う計画が実行された。

実行時間は問合せ (C) の場合には初期のデータベースにおいて 695 秒を要し、エージング後は倍近い 1281 秒を要するという結果になった。問合せ (D) は初期状態において 1805 秒を要し、エージング後は 2419 秒であった。どちらの問合せについても、エージングによって実行時間が増加する傾向が見られた。

4.1. 予備実験

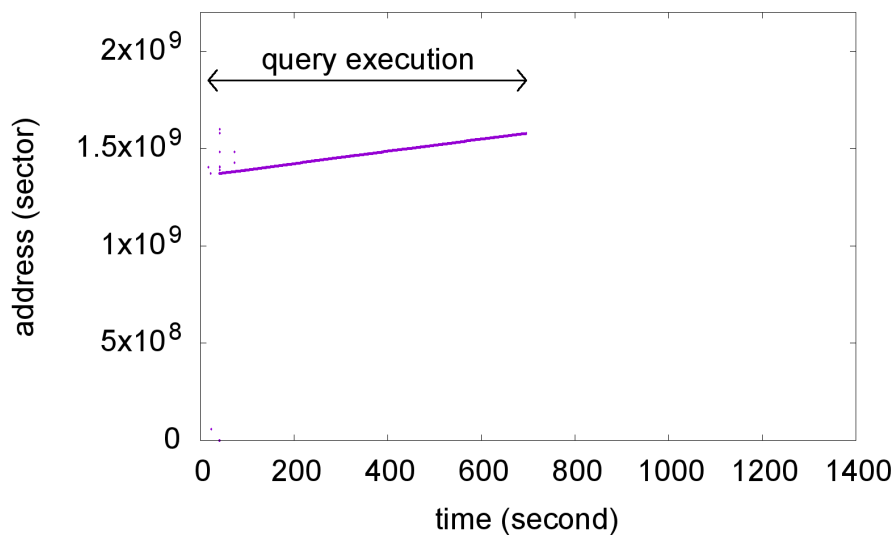


図 4.18: 初期状態の問合せ (A) の入出力挙動

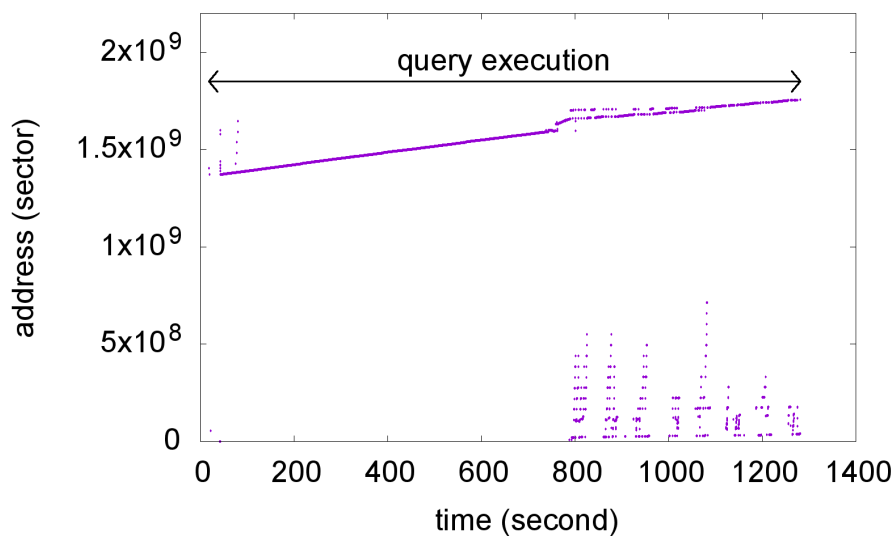


図 4.19: エージング後の問合せ (A) の入出力挙動

4.1.4.3 入出力挙動の観測結果

図 4.18 に、例として、問合せ (A) を実行中の入出力の挙動を示す。横軸は経過時間、縦軸はセクタ単位のアドレスである。全表走査を実行する問合せ (A) は、シー

4.1. 予備実験

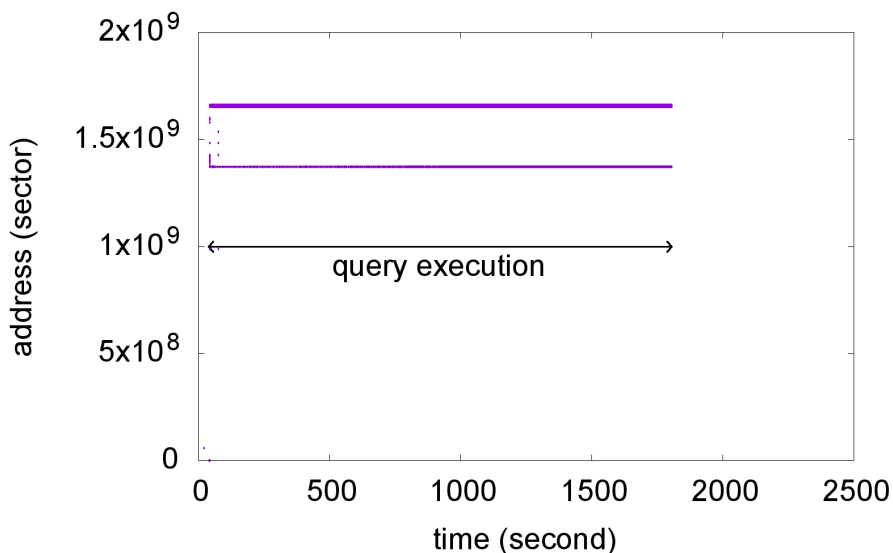


図 4.20: 初期状態の問合せ (B) の入出力挙動

ケンシャルに読み込みを行っていることが確認できる。対して、エージングしたデータベースにおいて、問合せ (A) の入出力挙動を観測した結果を、図 4.19 に示す。図 4.18 と図 4.19 を比較すると、実行時間が初期状態と比較して増加しており、初期状態の場合にはアクセスされなかった領域を読んでいることが観測された。データの更新を行った際に、これまでのアドレスの先にデータが格納されたことによって、全表走査で読む範囲が増大し、このような結果になったと考えられる。

図 4.20 と図 4.21 に、同様にして計測した問合せ (B) を実行中の入出力挙動を示す。問合せ (B) は part 表と lineitem 表を結合するため、初期状態の図 4.20 においてもシーク距離が大きく、どちらの表からも読み込みを行っていることが観察できる。エージング後の 4.21 においては更に読み込む領域が広がっていることが観察でき、全表走査の際と同じく、これまではアクセスしていなかった領域にアクセスしていることが読みとれる。

4.1. 予備実験

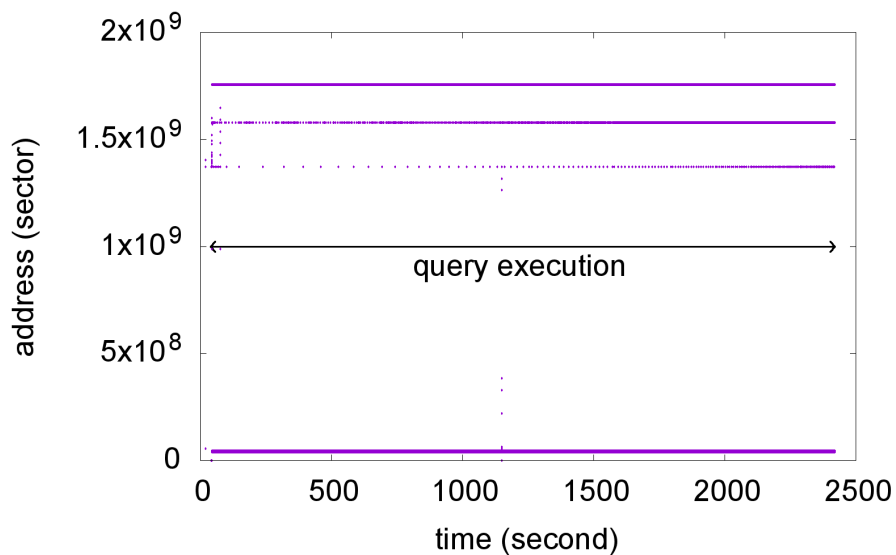


図 4.21: エージング後の問合せ (B) の入出力挙動

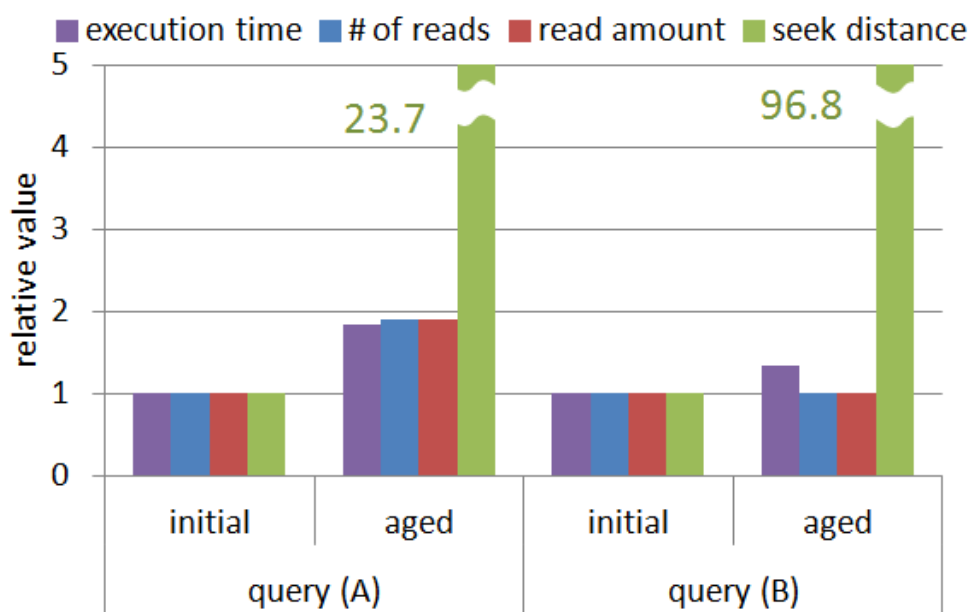


図 4.22: 問合せ毎のエージングの影響比較

4.2. コストモデルの有用性の検証と考察

4.1.4.4 入出力挙動の定量的な比較

二つの問合せについて、トレーサによる観測を基に、実行時間、読み込みの入出力発行回数、その総データ量、ディスク上の総シーク距離を比較した。結果を纏めたグラフを図 4.22 に示す。グラフの数値は全て初期状態のデータベースの測定値を 1 として正規化を行っている。

問合せ (A) においては、エージングによってデータが格納されている範囲が広がっているため、入出力発行回数と総データ量が增大する結果となった。入出力発行回数は、初期状態の 70.4 万回から、エージングにより 1.91 倍の 134 万回となり、92.2GB であった総データ量も 175GB と、1.90 倍に増加した。総シーク距離も、73 億セクタから 23.7 倍の 1740 億セクタになった。問合せ (B) に関しては、索引条件により指定された領域のみを読みだすため、入出力発行回数は、初期状態は 43.2 万回、エージング後も 1.00 倍の 43.3 万回と、殆ど差異が見られず、総データ量についても 3.71GB から 1.01 倍の 3.75GB と、変動が小さい結果となった。その一方で総シーク距離は 2.02 兆セクタから 195 兆セクタに増加し、96.8 倍となり、問合せ (A) 以上の大幅な増加が観察された。データの格納場所が分散したことによる総シーク距離の大増加が、問合せ (B) の実行時間が増加する主な原因になったと考えられる。

4.2 コストモデルの有用性の検証と考察

予備実験において、著者はエージングによるデータベースのアクセスコストへの影響が少なからず存在していること、そしてその影響を既存の最適化機構が十分に反映していない場合があることを確認した。本節では、著者が提案したコストモデルについて、実際にテスト用問合せを実行して測定し、別の問合せに適用してその実行コストを計算し、エージングの影響を反映したコストモデルの有用性を検証する。

4.2.1 実験環境

本実験の環境は、表 4.1 に示す前節の環境と同一である。

4.2. コストモデルの有用性の検証と考察

```
SELECT SUM(l_extendedprice) FROM lineitem
WHERE l_orderkey < x AND l_orderkey > y      ... (A)

SELECT SUM(l_extendedprice) FROM lineitem
WHERE l_partkey < x AND l_partkey > y        ... (B)

SELECT SUM(p_size) FROM part
WHERE p_partkey < x AND p_partkey > y        ... (C)

SELECT SUM(p_size) FROM part
WHERE p_retailprice < x AND p_retailprice > y ... (D)

SELECT SUM(p_size) FROM part or lineitem     ... (E)
```

図 4.23: テスト用問合せ

4.2.2 使用したデータベースとテスト用問合せ

特に局所的なエージングの影響を調査するため、初期状態のデータベース 1 個、lineitem 表の先頭から 10% に対し 4 回更新処理を行ったデータベース 1 個の合計 2 個を用意した。

使用したテスト用問合せを、図 4.23 に示す。それぞれ、l_orderkey によって制限した lineitem 表の一部範囲の走査時間、l_partkey によって制限した lineitem 表の一部範囲の走査時間、p_partkey によって制限した part 表の一部範囲の走査時間、p_retailprice によって制限した part 表の一部範囲の走査時間を測定する問合せで、l_orderkey と p_partkey がクラスタ化索引、l_partkey と p_retailprice が二次索引である。これらを初期状態のデータベースとエージング後のデータベースに対して実行し、それぞれの索引走査の一行ごとのアクセスコストを得た。また、範囲を指定しない問合せである問合せ (D) によって全表走査を行い、全表走査実行時の一行ごとのアクセスコストを算出した。

4.2. コストモデルの有用性の検証と考察

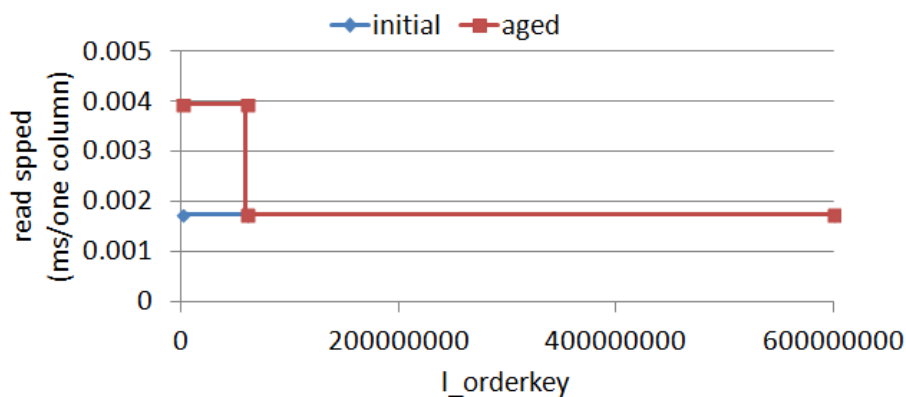


図 4.24: l_orderkey を用いた lineitem 表の読み込み速度

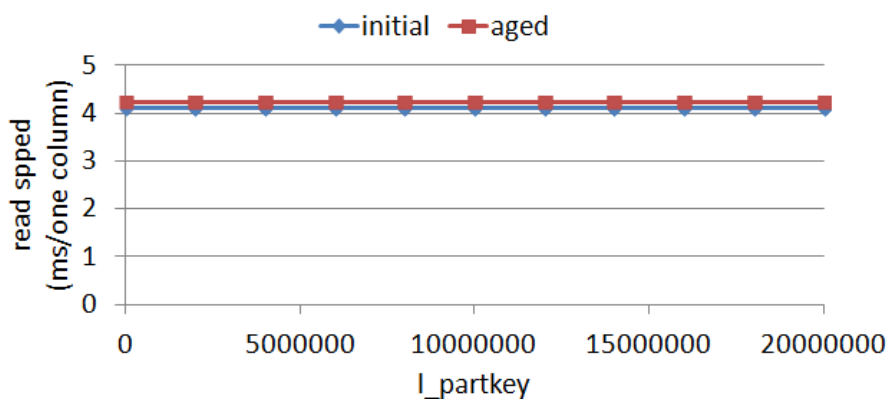


図 4.25: l_partkey を用いた lineitem 表の読み込み速度

4.2.3 相対的なエージング度合いの測定

筆者はまず、テスト用問合せを用い、索引別にエージングの偏りを調査した。結果を図 4.24, 図 4.25, 図 4.26, 図 4.27 に示す。横軸は索引を張っている列の値, 縦軸

4.2. コストモデルの有用性の検証と考察

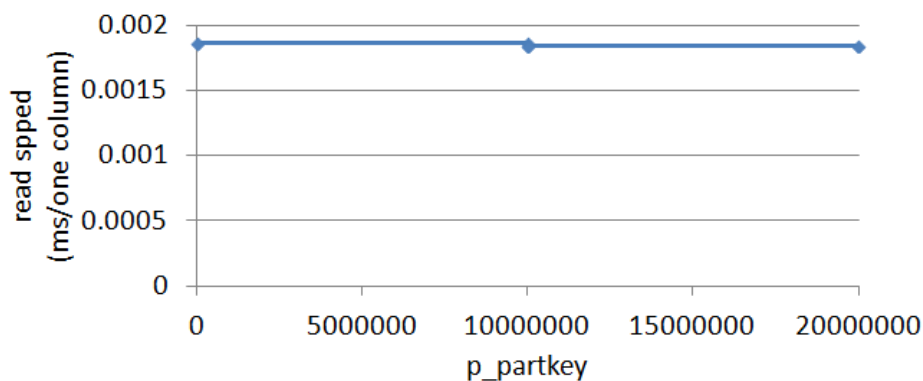


図 4.26: p_partkey を用いた part 表の読み込み速度

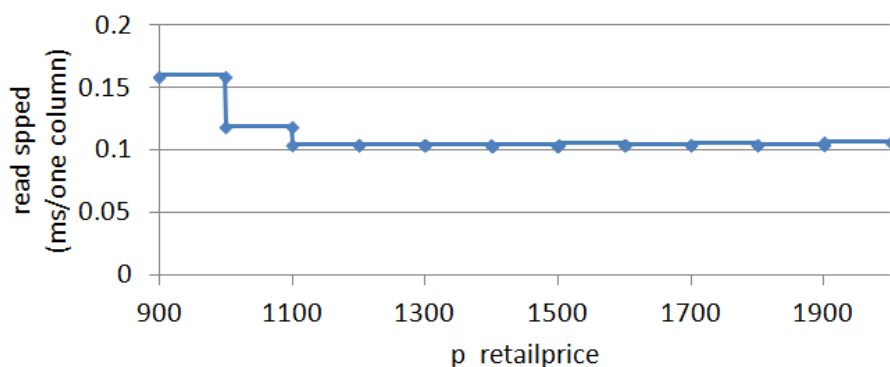


図 4.27: p_retailprice を用いた part 表の読み込み速度

は 1 行読む際にかかる読み込み時間を表す。今回用いた更新関数は l_orderkey の先頭から 10% を繰り返し更新するため、l_orderkey を横軸にとった図 4.24 では、エージング後のデータベースにおいて局所的に読み込みが遅くなることが観測できた。また、l_partkey を横軸に取ってエージングの度合いを測定した場合には、l_orderkey とは相関が小さいキーであるため局所的な変化は見られないが、一貫してエージン

4.2. コストモデルの有用性の検証と考察

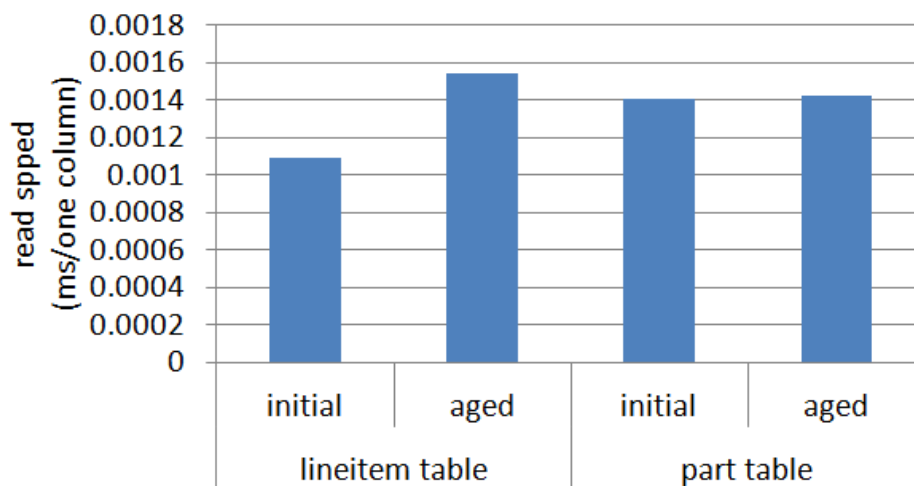


図 4.28: 全表走査による表の読み込み速度

グ後のデータベースの方が読み込みに時間がかかるという結果となった。part 表はエージングを一切行っていないため、p.retailprice の実行時間が領域によって変化する理由は不明であるが、図に示すような結果となったため、この数値のまま実験に使用した。また、part 表、lineitem 表の全表走査の一行あたりのアクセスコストは、データベースにつき一つの値で表せる。この値は 4.28 に示す。以降の実験では、以上のグラフを元に本論文で提案するコストモデルの見積りコスト計算を行った。

4.2.4 有用性の検証

これまでの測定によって、全表走査、及びそれぞれの索引に応じた索引走査の、一行読み込むごとの実行時間を得た。本節では、検証用問合せに対してコスト見積りを行い、実行時間との比較と考察を行う。

4.2.4.1 使用問合せとそのコスト見積り

本研究では、著者は図 4.29 に示す検証用の問合せを用い、実験を行った。これらは全て、lineitem 表と part 表を走査、結合する問合せである。

4.2. コストモデルの有用性の検証と考察

```
SELECT SUM(p_size) FROM lineitem, part
WHERE p_retailprice < x
AND l_partkey = p_partkey ... (A)
```

```
SELECT SUM(p_size) FROM lineitem, part
WHERE l_orderkey < x ... (B)
AND l_partkey = p_partkey
```

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

図 4.29: 検証用問合せ

問合せ (A) と (B) に関しては、モデル化の正確さを明らかにするため、全表走査によるハッシュ結合と索引走査におけるネステッドループ結合の実行時間を測定し、モデルとの比較を行った。問合せ (C) に関しては、局所性の影響とそのモデルへの反映を確かめるため、エージング後のデータベースにおいて、a と b の値を変更することにより、エージングしている部分のみ、またはエージングしていない部分のみに範囲を絞り、索引走査を用いたハッシュ結合または索引走査を用いたネステッドループ結合を用いて実験を行った。

また、lineitem 表、part 表ともに複数の索引走査が考えられるため、テスト用問合せの結果から、実行計画に応じて適当と思われるものを選択し、それを基にコスト見積りを行った。

4.2. コストモデルの有用性の検証と考察

4.2.4.2 実行計画

問合せ (A) は、全表走査によるハッシュ結合を行った場合には、lineitem 表と part 表をそれぞれ全表走査し、ハッシュ結合する計画となった。問合せ (B) における全表走査によるハッシュ結合の場合も同様である。

索引走査によるネステッドループ結合を行う場合には、問合せ (A) は part 表を外側とし、p_retailprice による走査を行い、読み込んだ行の p_partkey に対応する l_partkey から内側の lineitem 表にアクセスする問合せとなった。一方、問合せ (B) は lineitem 表を外側とし、l_orderkey による走査を行い、読み込んだ行の l_partkey に対応する p_partkey から内側の part 表にアクセスする問合せとなった。問合せ (C) に関しては、どちらの表も外側になりうるが、今回の実験で実行した範囲においては全て part 表を外側とし、p_partkey による索引走査を行い、対応する lineitem 表にアクセスする計画となることを確認した。

問合せ (C) については、索引走査によるハッシュ結合も実行した。この実行計画は、lineitem 表、part 表をそれぞれ l_orderkey と p_partkey によって索引走査し、ハッシュ結合を行う計画となった。

4.2.4.3 測定結果とコスト見積りモデルの比較

初期状態のデータベースにおける問合せ (A) のコスト見積りの結果と実際の実行時間を、図 4.30 に示す。用いた実行計画は、全表走査を使用するハッシュ結合と、索引走査を使用するネステッドループ結合である。見積りと実行時間の傾向は、ほぼ一致するという結果になった。実際の数値に関しても、大きな誤差なく予測できている。

更に、先頭から 10% のデータに 4 回更新を行ったデータベースにおける問合せ (A) の計測結果を図 4.31 に示す。初期状態のデータベースにおける結果の図 4.30 と比べると、エージングによる実行時間の変動が見られるが、見積りと実行時間の誤差は初期状態のものと同じ程度に収まっており、エージングによって増加した実行時間を正しく予測できていると言える。

4.2. コストモデルの有用性の検証と考察

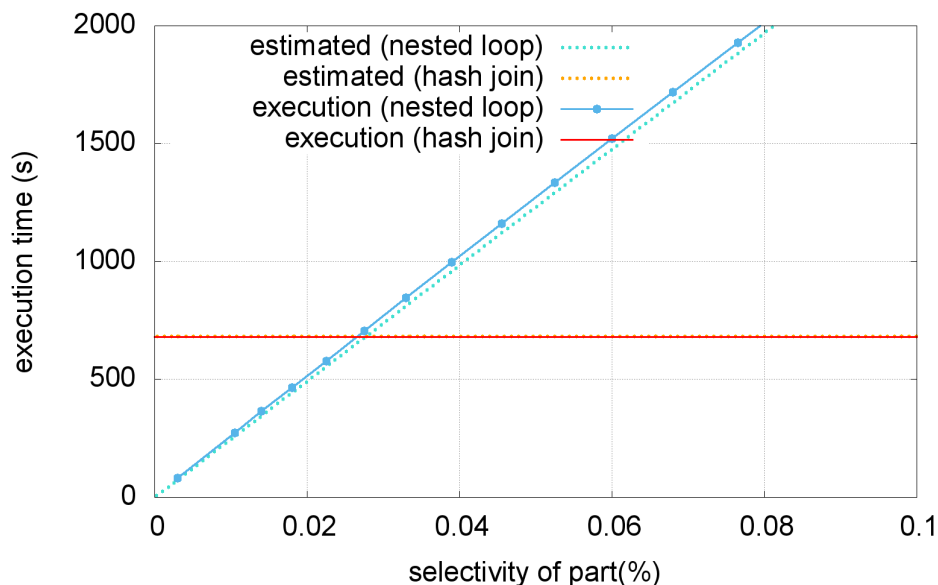


図 4.30: 検証用問合せ (A) の見積りと実行時間比較 (初期状態)

二つの結果から、結合に二次索引を用いるネステッドループ結合を行う場合は、エージングの前後においてほぼ実行時間に変化がないことが分かる。一方、全表走査によるハッシュ結合を行う計画の場合は、エージングに応じて顕著に実行時間の増加が見られ、その結果、損益分岐点が移動していることが観察できる。そのため、例えば選択率 0.03% 付近の問合せにおいては、初期状態のデータベースにて実行する場合には全表走査を用いたハッシュ結合を行う計画の方が迅速に問合せを実行できるが、エージング後のデータベースにおいて実行する場合には有利不利が逆転し、索引走査を用いたネステッドループ結合を行う計画の方が実行時間が短いという現象が発生している。この選択率 0.03% 付近の現象は当モデルにおいても予測できていることから、エージングの影響を考慮したことによって、単純なモデルであってもエージングによる損益分岐点の変化を推定し、実行計画の選択ミスを減らすことができる可能性が示されている。

問合せ (B) についても同様の実験を行った。初期状態のデータベースにおいてのコスト見積りの結果と実際の実行時間を、図 4.32 に示す。見積りは問合せ (A) と

4.2. コストモデルの有用性の検証と考察

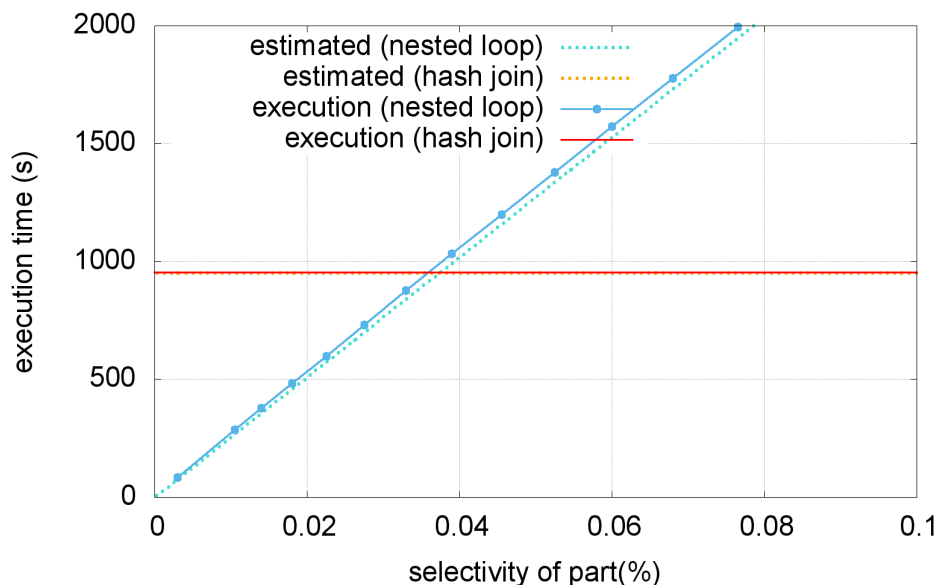


図 4.31: 検証用問合せ (A) の見積りと実行時間比較 (エージング後)

同様であり、索引走査を用いたネステッドループ結合の実行時間は線形に増加すると思われていたが、計測では選択率が上昇する毎に傾きが小さくなるという結果になった。実行時には PostgreSQL 側のキャッシュは使わないよう設定を変更していたが、OS のファイルキャッシュはそのままであったため、読む量が多い場合には内側表の part がファイルキャッシュに乗り、part 表自体が小さいため多くキャッシュヒットし、線形の増加が見られなかったと考えられる。全表走査についても問合せ (A) に比べて時間がかかる傾向が見られたが、この原因は不明である。

また、先頭から 10% のデータに 4 回更新を行ったデータベースにおける、コスト見積りの結果と実際の実行時間を図 4.33 に示す。エージング後のデータベースにおいても、索引走査を用いたネステッドループ結合の実際の実行時間は傾きが小さくなる結果となった。こちらもキャッシュに当たっていると考えられる。全表走査に関しても見積りとの誤差が生じているが、こちらに関しては不明である。

初期状態とエージング後の実行時間を比較すると、全表走査の実行時間が増加したことで損益分岐点がずれているが、このモデルによってエージングの影響を考慮

4.2. コストモデルの有用性の検証と考察

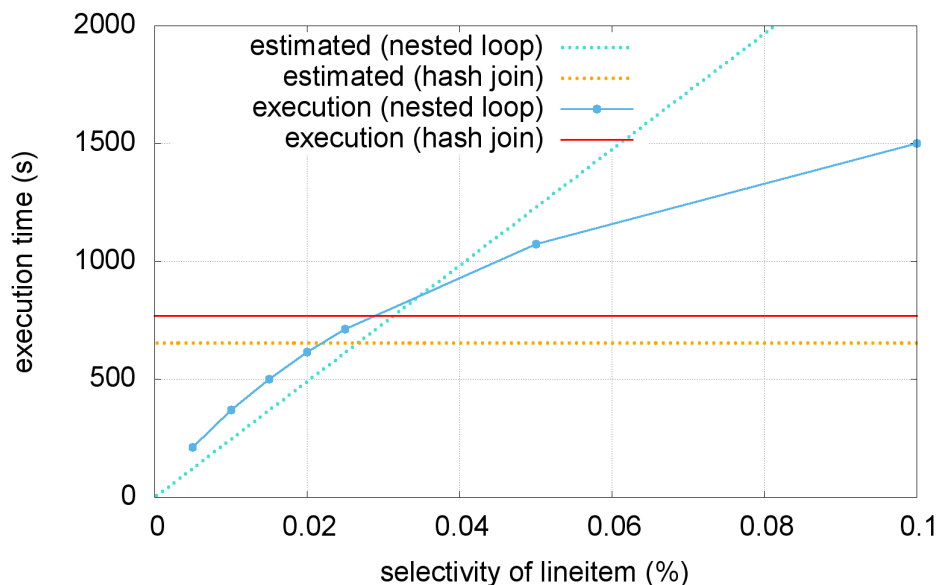


図 4.32: 検証用問合せ (B) の見積りと実行時間比較 (初期状態)

したことにより、多少の損益分岐点のずれには対応できている。但し、キャッシュの影響により実際の実行時間が線形に増加しなかったため、問合せ (A) のように上手く予測するには至らず、損益分岐点の変動の見積りとしては少し不十分な結果となった。本論文のコストモデルではキャッシュは影響しないものとして考えてきたため、これに関しては以降の課題としたい。

4.2.4.4 局所的エージングの反映

図 4.34 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

4.2. コストモデルの有用性の検証と考察

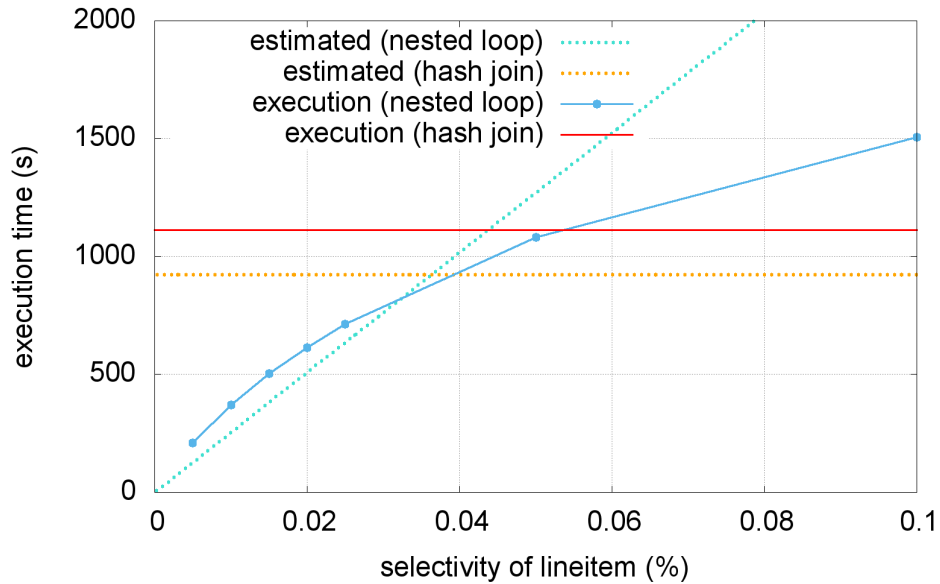


図 4.33: 検証用問合せ (B) の見積りと実行時間比較 (4回更新後)

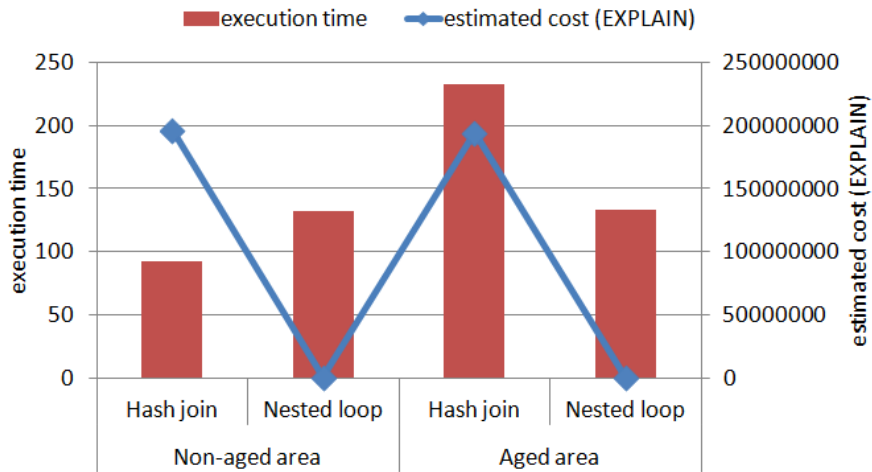


図 4.34: 検証用問合せ (C) の part 表選択率 0.005%における PostgreSQL による見積りと実行時間の比較

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

4.2. コストモデルの有用性の検証と考察

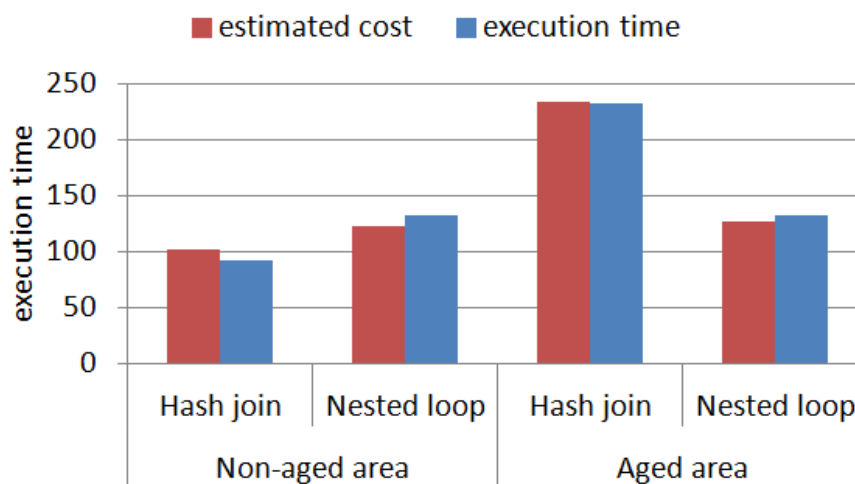


図 4.35: 検証用問合せ (C) の part 表選択率 0.005%における提案手法の見積りと実行時間の比較

エージングの影響の反映は見られなかった。

図 4.35 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 4.36, 図 4.37 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 4.36 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 4.37 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 4.34 や図 4.35 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

4.2. コストモデルの有用性の検証と考察

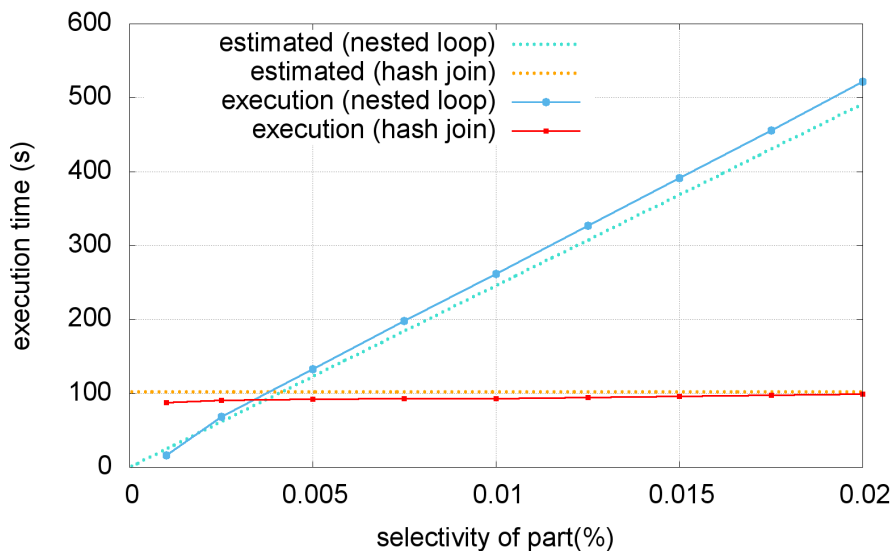


図 4.36: 検証用問合せ (C) の提案手法による見積りと実行時間比較 (4 回更新後, エージングしていない部分の測定)

4.38 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 4.39 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 4.40, 図 4.41 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.

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,

4.2. コストモデルの有用性の検証と考察

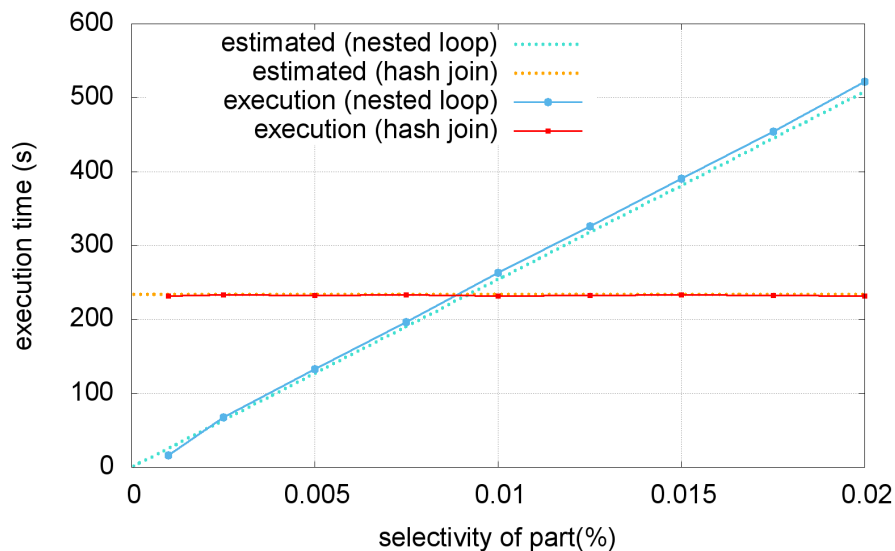


図 4.37: 検証用問合せ (C) の提案手法による見積りと実行時間比較 (4 回更新後, エージングしている部分の測定)

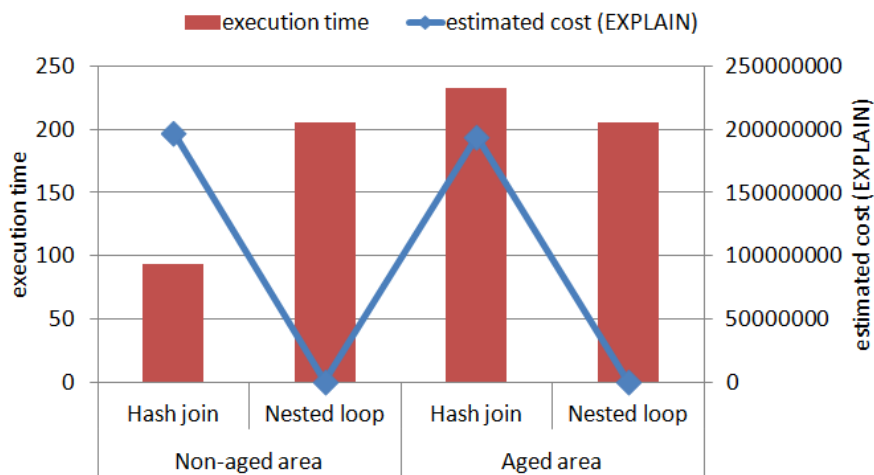


図 4.38: 検証用問合せ (D) の part 表選択率 0.0075%における PostgreSQL による見積りと実行時間の比較

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.

4.2. コストモデルの有用性の検証と考察

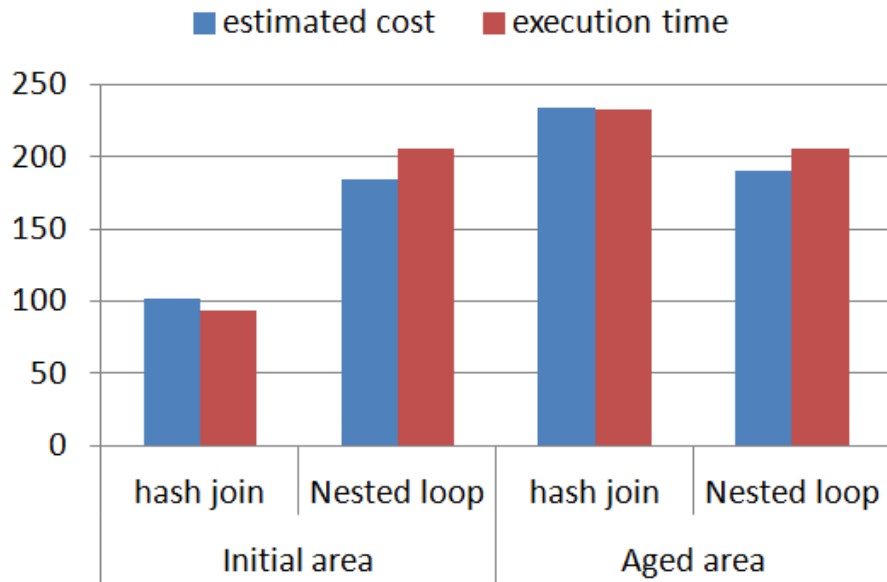


図 4.39: 検証用問合せ (D) の part 表選択率 0.0075%における提案手法の見積りと実行時間の比較

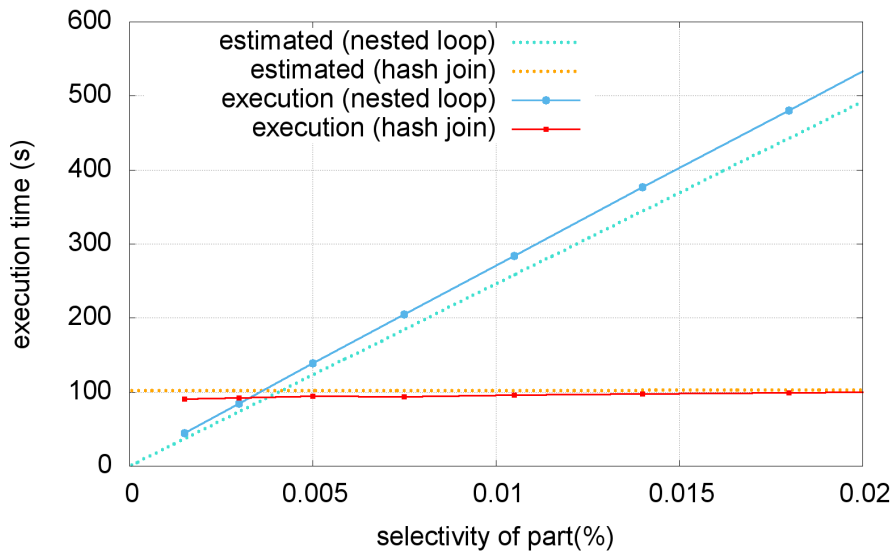


図 4.40: 検証用問合せ (D) の提案手法による見積りと実行時間比較 (4回更新後, エージングしていない部分の測定)

4.2. コストモデルの有用性の検証と考察

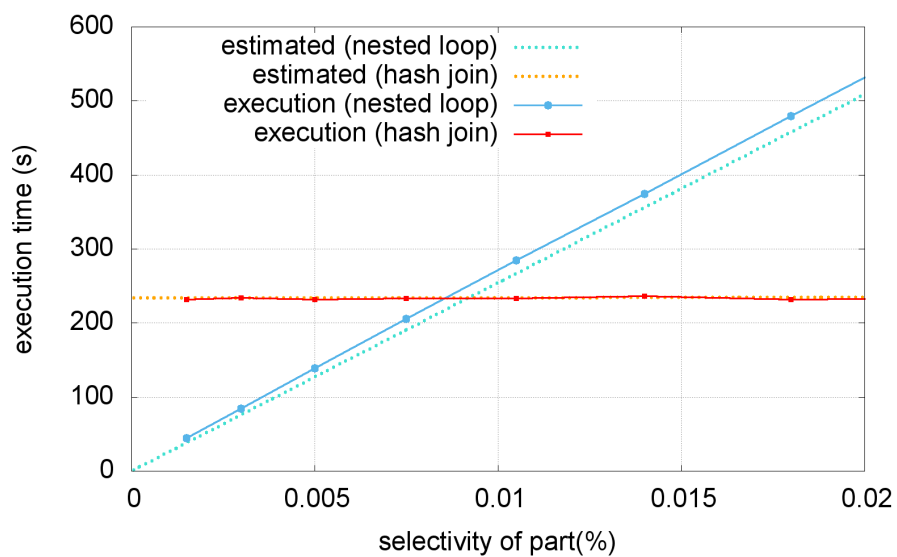


図 4.41: 検証用問合せ (D) の提案手法による見積りと実行時間比較 (4回更新後, エージングしている部分の測定)

第5章 おわりに

著者は、データベースの性能を低下させる要因の一つである、エージングに着目した。エージングは更新処理を繰り返したことにより発生する、格納構造の劣化による、データベースの実行性能の劣化現象である。筆者は、問合せ実行のコア技術である問合せ最適化の実行に際し、エージングの局所性を考慮することを提案した。まずテスト用問合せを用いてエージングによる I/O コストの変化を測定し、それを用いて問合せ実行計画のコスト見積りを行うことでより適した問合せ実行計画を選択するという問合せ最適化方式を構築し、その有用性についてオープンソースのデータベースシステムである PostgreSQL を用いた小さな実験環境で検証を行った。

提案手法の検証に先立ち、データベースのエージングがどのように発生し影響を及ぼしているかについて、実行時間の計測と入出力挙動の観察という二つの観点から観測を行った。その結果、エージングしたデータベースにおいては同一の問合せの実行時間が増加しており、データベースの性能が低下していることが確認された。また、入出力挙動の観察により、格納領域が肥大化していることを確認し、全表走査においては読み込むデータ量の増加が、索引走査においてはシーク距離の増大が性能低下の原因であるということを得た。

また、PostgreSQL の問合せ最適化機構がどの程度エージングを考慮できているかについても実験を行った。その結果、PostgreSQL の問合せ最適化機構は、エージングの影響を十分に考慮しておらず、特に局所的なエージングに関しては殆ど考慮出来ていないことを確認した。

以上を踏まえ、提案手法の有用性について検証を行った。局所的にエージングを発生させたデータベースを用意し、単一表を走査するテスト用問合せを用いてエージングによる I/O コストの変化を調査したのち、複数の表を結合する検証用問合せ

について、提案手法に基づいてコスト見積りを行い、実際の実行時間との比較を行った。その結果、エージングの局所性が大きく影響を及ぼす一部の問合せにおいては、エージングしている領域か否かで適切な実行計画が変化する場合が存在したが、提案手法はその差異を予測して適切なコスト見積りを行うことができ、局所性の考慮により実行計画の妥当性が向上したことを確認した。

今後は、更に複雑度の高い問合せにおける有効性の検証を行うとともに、実際に PostgreSQL へ実装し、評価を行いたい。

付録A

A.1 ソースコードの変更点

実験の際に用いたオープンソースのプログラムについて、変更を加えた点を示す。

A.1.1 Systemtap

systemtap/tapset/scsi.stp 内 95 行目の scsi.iodispatching に、SCSI 層へのリクエストの中身を観察するための、構造体へのアクセスを追加した。

```
probe scsi.iodispatching
    = module("scsi_mod").
      function("scsi_dispatch_cmd@drivers/scsi/scsi.c")!,
      kernel.function("scsi_dispatch_cmd@drivers/scsi/scsi.c")?
{
    host_no = $cmd->device->host->host_no
    (中略)
    req_addr = $cmd->request

    //以下, SCSI 層 I/O 観察のための追加部分
    req_addr_sector = $cmd->request->sector
    req_addr_nr_sectors = $cmd->request->nr_sectors
    req_addr_current_nr_sectors = $cmd->request->current_nr_sectors
    req_addr_hard_sector = $cmd->request->hard_sector
```

A.1. ソースコードの変更点

```
req_addr_hard_nr_sectors = $cmd->request->hard_nr_sectors
req_addr_hard_cur_sectors = $cmd->request->hard_cur_sectors
}
```

A.1.2 DBgen

DBgen において、10 回目以降の更新データの作成に際しバグが発生したため、print.c 内の pr_drange 関数と build.c 内の mk_order 関数について修正を行った。

```
//print.c
int pr_drange(int tbl, DSS_HUGE min, DSS_HUGE cnt, long num)
{
    static int last_num = 0;
    static FILE *dfp = NULL;
    DSS_HUGE child = -1;
    DSS_HUGE start, last, new;
    static DSS_HUGE rows_per_segment=0;
    static DSS_HUGE rows_this_segment=0;
    if (last_num != num)
    {
        if (dfp) fclose(dfp);
        dfp = print_prep(tbl, -num);
        if (dfp == NULL) return(-1);
        last_num = num;
        rows_this_segment=0;
    }
    //追加部分ここから
    if(num > 10){
        min = min - (10 * (int)(num/10) * cnt);
```

A.1. ソースコードの変更点

```
    }
    //ここまで
    start = MK_SPARSE(min, (num-1)/ (10000 / UPD_PCT));
    //num を num-1 に変更
    (中略)
    return(0);
}
```

```
//build.c
long mk_order(DSS_HUGE index, order_t * o, long upd_num)
{
    DSS_HUGE      lcnt;
    DSS_HUGE      rprice;
    long          ocnt;
    DSS_HUGE      tmp_date;
    DSS_HUGE      s_date;
    DSS_HUGE      r_date;
    DSS_HUGE      c_date;
    DSS_HUGE      clk_num;
    DSS_HUGE      supp_num;
    static char   **asc_date = NULL;
    char          tmp_str[2];
    char          **mk_ascdate PROTO((void));
    int           delta = 1;
    static int    bInit = 0;
    static char   szFormat[100];
    if (!bInit)
    {
```

A.2. 図表

```
        sprintf(szFormat, O_CLRK_FMT, 9, HUGE_FORMAT + 1);
        bInit = 1;
    }
    if (asc_date == NULL)
        asc_date = mk_ascdate();
    //追加部分ここから
    if(upd_num > 10){
        index = index - (10 * (int)(upd_num/10)
            * (int)(tdefs[ORDER_LINE].base / 10000 * scale * UPD_PCT));
    }
    //ここまで
    mk_sparse(index, &o->okey,
        (upd_num == 0) ? 0 : 1 + (upd_num-1) / (10000 / UPD_PCT));
    //num を num-1 に変更
    (中略)
    return (0);
}
```

A.2 図表

A.2. 図表

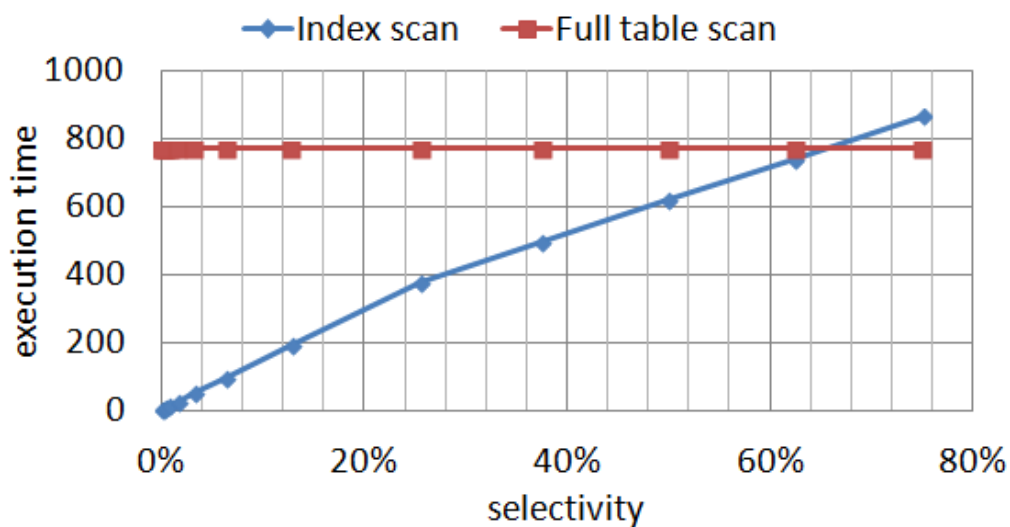


図 A.1: Lorderkey を用いた 20%更新データベースのスキャン時間

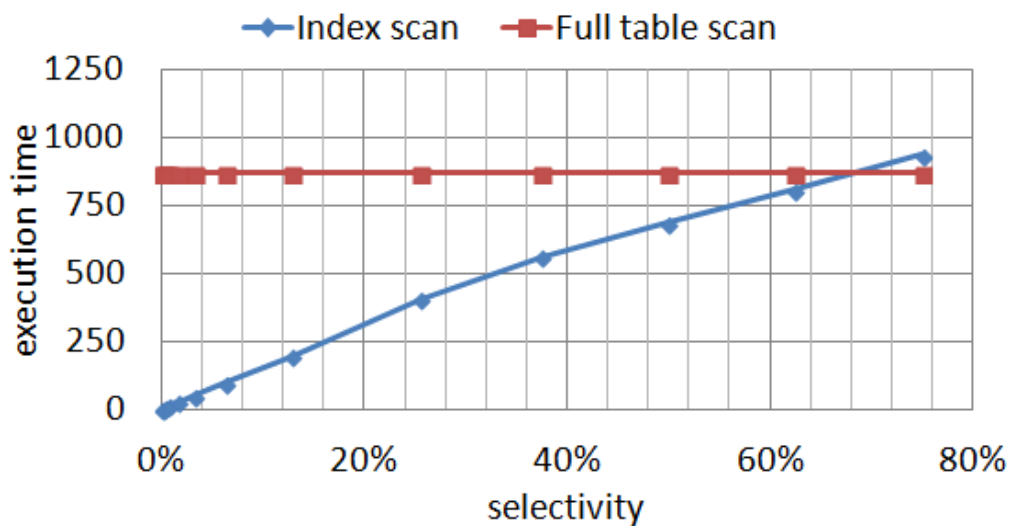


図 A.2: Lorderkey を用いた 30%更新データベースのスキャン時間

A.2. 図表

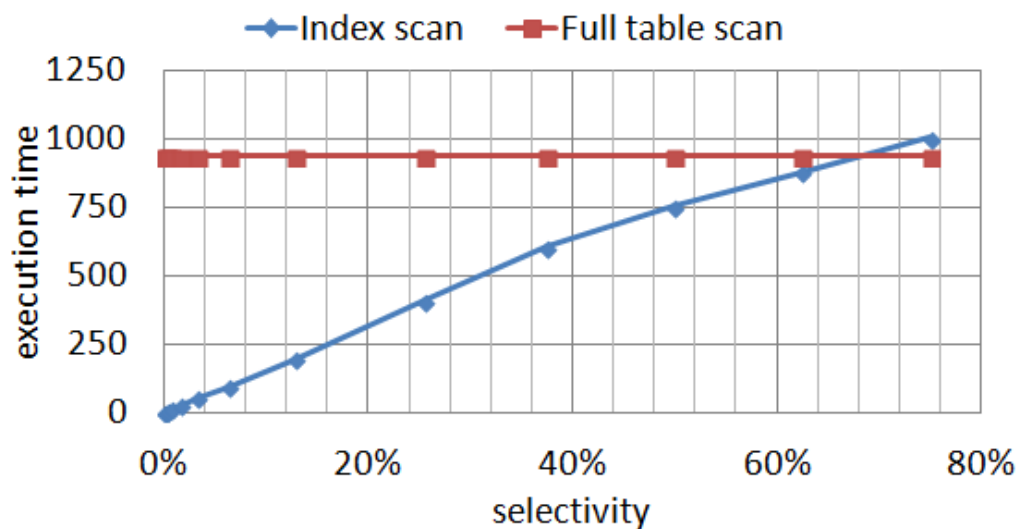


図 A.3: Lorderkey を用いた 40%更新データベースのスキャン時間

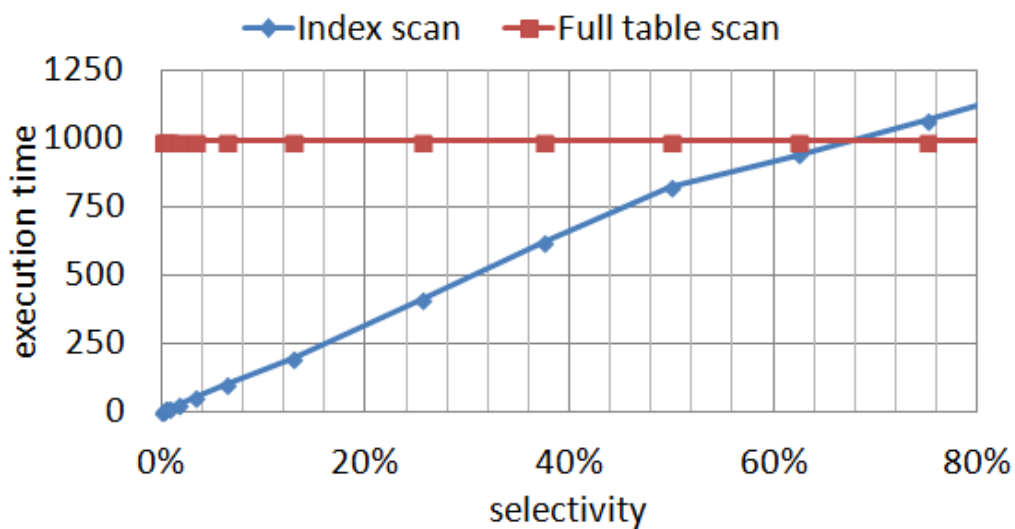


図 A.4: Lorderkey を用いた 50%更新データベースのスキャン時間

A.2. 図表

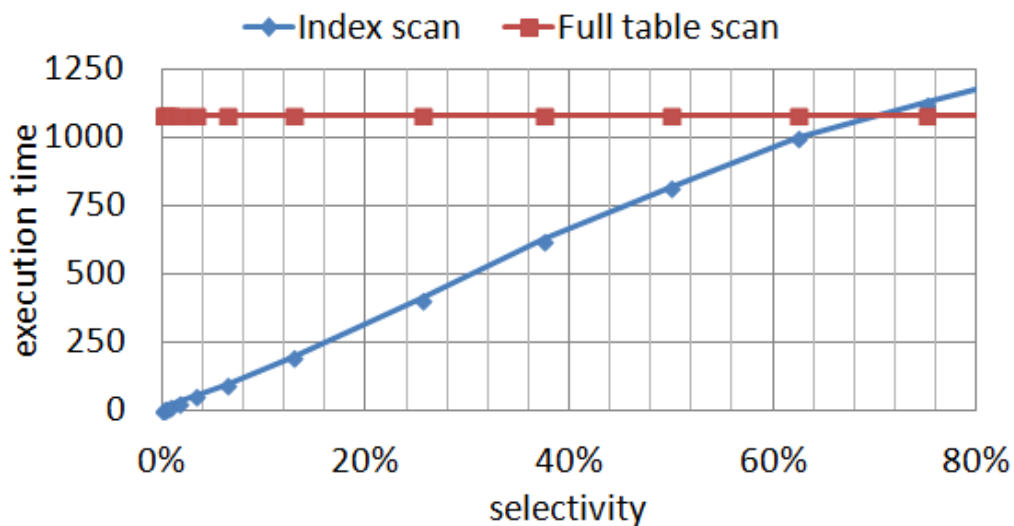


図 A.5: Lorderkey を用いた 60%更新データベースのスキャン時間

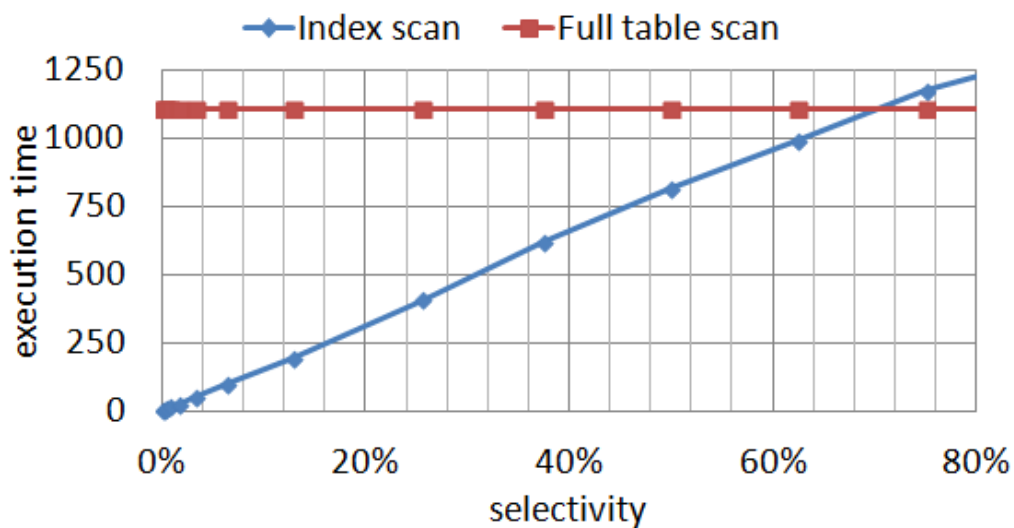


図 A.6: Lorderkey を用いた 70%更新データベースのスキャン時間

A.2. 図表

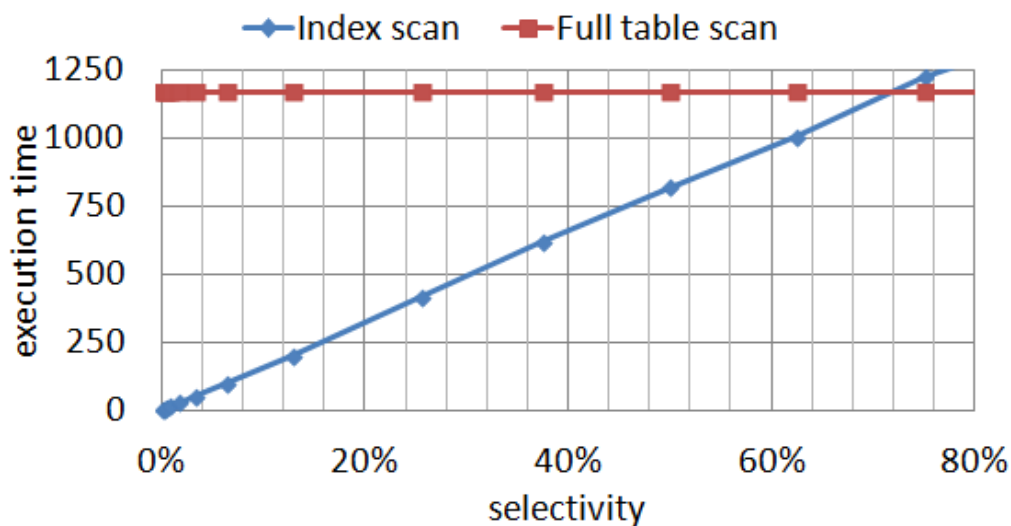


図 A.7: Lorderkey を用いた 80%更新データベースのスキャン時間

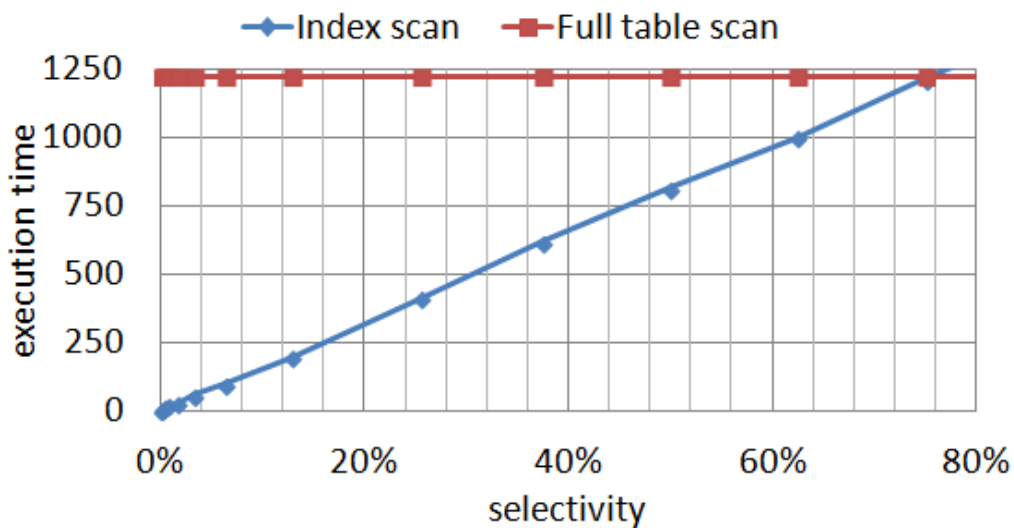


図 A.8: Lorderkey を用いた 90%更新データベースのスキャン時間

参考文献

- [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 Chung-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 *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pp. 115–124. ACM, 1996.
- [4] Edward Omiecinski and Peter Scheuermann. A global approach to record clustering and file reorganization. In *Proceedings of the seventh annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 201–219. British Computer Society, 1984.
- [5] 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 (TOD) , Vol. 46, No. 8, pp. 130–147, 2005.
- [6] 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.

- [7] 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.
- [8] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 34–43. ACM, 1998.
- [9] Goetz Graefe. The cascades framework for query optimization. *Data Engineering Bulletin*, Vol. 18, No. 3, pp. 19–29, 1995.
- [10] Goetz Graefe. Encapsulation of parallelism in the volcano query processing system. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pp. 102–111. ACM, 1990.
- [11] 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.
- [12] Luis L. Perez and Christopher M. Jermaine. History-aware query optimization with materialized intermediate views. In *2014 IEEE 30th International Conference on Data Engineering (ICDE)*, pp. 520–531. IEEE, 2014.
- [13] 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.
- [14] Steven Pelley, Kristen LeFevre, and Thomas F. Wenisch. Do query optimizers need to be SSD-aware? In *Second International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, pp. 44–51, 2011.

- [15] 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.
- [16] 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.

発表文献

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クエリ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, G3-2. 2015.
2. Chihiro Kato, Yuto Hayamizu, Kazuo Goda, Masaru Kitsuregawa. An Experimental Study of Aging Influence on Query Cost estimation. 19th International Database Engineering & Applications Symposium, pp. 220-221, 2015
3. 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の局所性を考慮した問合せコストモデルの構築. 第8回データ工学と情報マネジメントに関するフォーラム, 2016 (to appear)
4. 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. 情報処理学会 第78回全国大会, 2016 (to appear)

謝辞

本研究の実行，および本論文の執筆にあたり，様々なご指導ご支援を賜りました。

合田和生特任准教授に心より感謝を申し上げます。お忙しい中，たびたびミーティングの時間を設けていただき，研究に関する指針や様々な助言を賜りました。また，研究が上手くいかず，行き詰った時にも，励まして新たな切り口や展望を示してくださいました。論文投稿や学会発表の前には，丁寧な訂正やご指導をくださいました。心より感謝いたします。

早水悠登特任研究員には，多くのミーティングに同席していただき，研究の様々な面でサポートしていただきました。校正や発表練習にもご参加いただき，優しいながらも鋭いアドバイスを賜りました。心より感謝いたします。

豊田正史准教授にも深く感謝申し上げます。日頃からミーティングにて研究の方向性の相談にのっていただき，鋭いご指摘を多数いただきました。また，研究室内では気さくに話しかけてくださり，時には励ましの言葉をいただき，心の支えとなりました。心より感謝いたします。

喜連川優教授には，日頃の業務が多忙を極める中，研究室の集まりにおいて，研究への心構えを教えてくださいました。心より感謝いたします。

吉永直樹特任准教授は，分野外でありながらミーティングなどで多くの意見やご指摘，質問を賜りました。また，時には煮詰まっている私を親身になって励まし，積極的に相談に乗ってくださいました。心より感謝いたします。

鍛冶伸裕特任准教授（現・Yahoo! JAPAN 研究所 上席研究員），伊藤正彦特任准教授，横山大作特任助教にも，心より感謝いたします。修士ミーティングや発表練習などに，お忙しい中参加していただき，様々な視点からのご指摘や助言を賜りました。

修士一年生の時にお世話になった先輩方にも、大変感謝いたします。生産技術研究所に来たことも殆どなく、右も左も分からない新入生だった私に、様々な研究のノウハウや、過ごし方を教えてくださいました。また、研究の合間には楽しく他愛もない話をしていただき、研究室を活気づけてくださいました。

また、修士二年生になって研究室に加わった後輩の皆にも、心より感謝を申し上げます。自分の研究の合間にこちらの研究に様々な角度から鋭い意見をしてくださり、大変参考になりました。また、プライベートにおいても様々な交流していただき、心の支えとなりました。

そして、同期の石渡祥之佑君、川本貴史君、小矢島諒君、谷川祐一君に心からの感謝を申し上げます。同期の皆に励まされ、ここまで来ることが出来ました。研究で行き詰った時、辛いことがあって落ち込んだ時、皆と話す時間が最大の薬になりました。締め切り前には、皆で励まし合うことで、なんとか乗り越えていくことが出来ました。時には研究に関して鋭いご指摘もいただきました。本当にありがとうございます。

最後になりますが、家族に感謝いたします。母は全く知らない分野の研究にいそしむ私を、根気強く支え、励ましてくださいました。父は就職活動で戸惑う私に優しく声をかけて、助けてくださいました。そして弟には、感謝してもしつくせません。僅かな時間の他愛もない会話が、最高に楽しいひと時であり、私の生きる希望でありました。素敵な家族に心よりの感謝を申し上げます。