データベース更新差分を用いた範囲検索の IO コスト推定

星野 喬 † 合田 和生 ‡ 喜連川 優 ‡

† 東京大学大学院 情報理工学系研究科 〒 113-0033 東京都文京区本郷 7-3-1
‡ 東京大学 生産技術研究所 〒 153-8505 東京都目黒区駒場 4-6-1

要 旨

本研究は，関係データベースシステム管理における再編成業務の自立化を目的とする．再編成は，構造劣化によって劣化した性能を回復するために表空間内のデータを再配置する．データベースが更新される限り構造劣化は避けられないため，再編成は不可欠な管理業務である．再編成自立化のために，データベースの構造劣化から性能劣化予測を行う必要がある．本稿では，ストレージ内の IO 性能特性を考慮した IO コストモデルを用いて構造劣化を表現することにより，データベースの範囲検索における性能の定量的推定を可能にし，再編成タイミングの判断に有用であることを示した．また，データベース更新差分を用いてわずかな性能オーバーヘッドで IO コスト推定が可能であることを，MySQL データベースに更新差分抽出機能を実装し，TPC-H ベンチマークを用いて評価することで示した．

Incremental IO Cost Estimation of Range Scan Using Update Difference of Database

Takashi Hoshino† Kazuo Goda‡ Masaru Kitsuregawa‡

†Graduate School of Information Science and Technology, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-0033, Japan
‡Institute of Industrial Science, University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo, 153-8505, Japan

Abstract

This research targets autonomic database reorganization for DBMS. Reorganization counteracts structural deterioration in tablespace to recover performance. Structural deterioration through data updates is inevitable, therefore reorganization is an essential task in database administration. Autonomic database reorganization requires prediction of performance degradation with structural deterioration. In this paper, we proposed a method to estimate IO cost of range scan of database considering IO behavior inside hard disk drive, which can be quantitative performance estiamtion for reorganization trigger. The method requires only database updates without fully table scan and it can keep estimated IO cost incrementally with little update overhead. We implemented the method on MySQL and evaluated it with TPC-H benchmark.

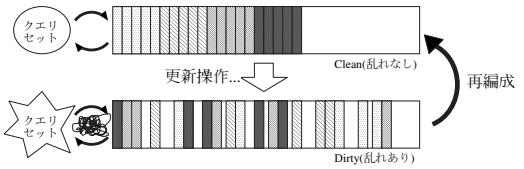

図 1: 再編成の概念図

1 はじめに

近年，人類の扱うデジタルデータの容量は飛躍的に増大しており，データ管理の基盤である DBMS は益々重要になっている．一方で，その管理はより困難化しており，コスト増大が問題となっている．この問題を解決するため，DBMS を自立化させる技術が求められている．

それに伴い，DBMS 及びストレージの管理負担を低減する試みが，増えてきた．例えば [3] などではこれからの DBMS の自立化に向けた取り組みが数多く紹介されている．

現在の DBMS で実際に動作している自立管理機構もいくつかある．オンライン表空間拡張機能はその一つであり，表空間の容量が足りなくなってきた時点で，予備領域の追加割り当てを自動的に行う機構である．また，自動バックアップも，あらかじめバックアップスケジュールを与えておけば，それに従ってテープドライブに自動的にバックアップするという自立管理機構である．このような単純な機構は動作しているが，データベース管理者に高度な判断を要する管理タスクを実際に自立化する試みは，未だ実現していない．自立化が難しいタスクには，データベース構成変更，性能チューニング，再編成などが挙げられる．

我々は再編成の自立化を目指し研究に取り組んでいる．再編成は，DBMS における主要な管理業務の一つであり，性能劣化を防ぐ [13] 重要な判断を伴う．

二次記憶空間上に確保された DB 表空間内のデータは，更新操作を繰り返すことによりその構造が劣化し，本来想定している配置と大きく乖離した状態となり，性能を大幅に劣化させる場合がある (図 1)．このため，DBMS の管理者は再編成を度々行うことにより，記憶空間上のデータを再配置し，性能の劣化を予め防ぐ必要がある．現状では，性能劣化を予測し再編成を開始することの判断は難しく，また，再編成自体にかかる時間は膨大であることから，再編成は高度の管理業務と考えられており，再編成を自立化させることの意義は大きい．

現状においては，DBMS における再編成は，データベース管理者が DBMS から得られる性能などの統計情報や，空間情報などからそのデータベースに再編成が必要であることを判断し，再編成を実行する．これらの情報を判断する方法は，管理者の勘と経験によるノウハウに多くを依存しており，具体的にマニュアル化されたものではない．結果として，設計時に更新量を想定して，余裕をもって定期的に再編成を実行する運用も多い．しかし，データベースは一度設計すると長期的に使われることが多く，設計時に想定していた劣化を越えることがあり得る．また，無駄な再編成を行うとより高性能なシステムが必要になるため，コストも高くつく．

データベース再編成判断がこのような曖昧なものになっている主要因は，性能劣化と構造劣化の関係を記述する方法論が曖昧であることにある．再編成を起動するタイミングを自動化し，また，再編成後のデータ配置を効率化するためには，性能劣化の直接的な原因となる構造劣化を定量的な指標によって表現する必要がある．

それらの指標を明確化し，構造劣化に起因する性能劣化を推定することにより，ユーザやアプリケーションからの性能要求である SLO[1] を満たすように再編成タスクをスケジューリングすることが可能になる．また，ある性能劣化が構造劣化に伴うものかどうかを明らかにでき，他の原因による性能問題と区別できる．それは再編成の自立化に大きく貢献すると同時に，余分な再編成を減らすことにも繋がる．さらに，再編成タスクによって能動的に性能改善を図ることが可能になる．それに加えて，性能推定で得られた情報を DBMS にフィードバックすることで実行計画の効率化を図ったり，バッファの容量調整やアルゴリズム変更などによるメモリチューニング，スキーマ構成の改善による性能向上などが可能になると思われる．将来的には，IT システムにおける多くの自立管理技術への応用が期待される．

現状の，DBMS を含むデータインテンシブアプリケーションでは，性能のボトルネックがディスク IO に起因することが多い．つまり，データベースアクセスの多くを二次記憶装置への IO が占める．これまで我々は二次記憶装置の IO 性能特性を考慮

[1] Service Level Objective:一般的には，ユーザから要求される，製品が満たすべきサービスレベル目標のことを指すが，本稿ではクエリ性能に関する目標値に限定して使用している．

し，MySQL データベースのクラスタ表の範囲検索を対象に構造劣化指標を IO コストを用いて表現する手法を研究してきた [14, 15]．これまでの手法では，IO コスト推定が必要な場合は，常に表空間をスキャンしてデータ構造を取得する必要があった．データベースシステムの稼動中に表空間をスキャンすることは，他のオンラインアクセスと競合した場合，大きなオーバーヘッドになりうる．また，データベース更新差分を用いて，IO コストを最新のデータベース状態に追随できる手法を示唆した．本稿では，MySQL に更新差分抽出機能を実装し，評価を行った．

本稿は以下のように構成される．まず，第 2 節で関連研究について述べ，第 3 節において本研究が目指しているデータベース自立再編成の枠組みについて述べる．次に第 4 章でハードディスク内 IO 性能特性を用いた範囲検索 IO コスト推定手法について述べ，その後，今回提案するデータベース更新を用いた低オーバーヘッド IO コスト推定手法について説明する．第 5 章で提案手法の評価を行う．最後に第 6 章にて結論と今後の課題を述べる．

2 関連研究

データベース再編成の自立化にはオンライン再編成方式と，再編成スケジューリングの自立化とが必要になる．オンライン再編成方式は従来から多くの研究 [2] がなされており，それらを状況に応じて利用することが可能である．一部は既に現実の DBMS に実装されている．

再編成スケジューリングに関しては，古くからいくつかの研究 [10, 12, 7, 5, 1] がなされている．研究 [12] は，性能を監視して再編成契機を判断するアプローチである．

現実のデータベース運用では，データベースソフトウェア実装にも依存するが，いくつかの構造劣化を示すパラメータが規定されている．[2] これらのパラメータのうち，ストレージ内における IO 性能特性を示すのは，クラスタリングプロパティである．これは多くの実装で考慮されているが，精度の高い性能推定を行ってはいない．

クエリプロセッサはデータの量と分布から，DB バッファ等の影響も考慮して，期待される IO 数を推定することで，実行計画の性能を予想する [6, 9]．本稿で提案している，システム下位層におけるストレージ内の性能特性を考慮して IO コスト推定値を用いて構造劣化を定量的に表現する手法は行われていない．

[2] 各 DBMS のマニュアル等参照．MySQL については [8]

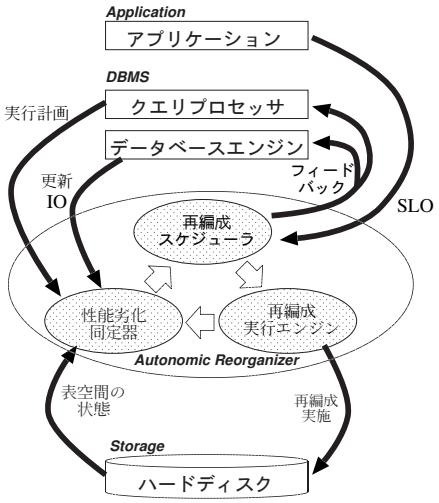

図 2: 自立再編成の枠組

3 自立再編成の枠組み

我々はストレージ内部の情報を有効利用して，再編成の自立化をすることを目指している．図 2 にその枠組を示した．

自立再編成器 Autonomic Reorganizer は，以下の 3 つの部分からなる．

構造劣化同定器: 表空間の状態を観測，保持して対象となるクエリの実行計画の性能を推定することによって構造劣化を同定する．

再編成スケジューラ: 性能推定や実性能などの監視などにより，少ないコストで SLO を満たすように再編成計画を立てる．

再編成実行エンジン: スケジューラの命令に従って，必要な再編成手法をストレージに対して実施する．

それぞれの部分がお互いに必要な情報をやりとりし，構造劣化の変化に対応した定量的な性能劣化予測を行い，SLO を保証するための最適な再編成手法と時間を判断し，再編成を実施する．また，性能予測や

構造劣化情報を DBMS のバッファや実行計画，スキーマなどのチューニングにフィードバックする機構も含む．

構造劣化同定器において，性能推定が必要になるが，IO コスト推定はこれに含まれる．データインテンシブアプリケーションにおいては，性能における IO の占める割合が大きいため，IO コスト推定を高い確度で行うことにより，性能推定も高い確度で行うことができる．

4 範囲検索の IO コスト推定手法

我々はストレージ内 IO 性能特性を考慮した，定量的な性能劣化予測が可能な IO コスト推定手法を構築した．

本節では，まず対象 DBMS 実装について説明する．次に，ストレージ内 IO 性能モデルを構築する．そのモデルを用いて，範囲検索の IO コスト推定手法を構築する．さらに，データベース更新差分を用いた推定手法について述べる．

4.1 対象 DBMS 実装

本稿では，MySQL のクラスタ表の範囲検索を対象に，その IO コストを推定する手法を構築した．MySQL のクラスタ表及び，索引は，B+Tree 構造を取っており，クラスタ表においては，葉ページにデータ行が格納されている．

クラスタ表や索引は，DBMS 実装において性能上重要な役割を果たしている．DBMS で定義される通常表の形式は，行に順序性を持たず，そのままでは比較検索時に表全体を読み込む必要が生じるが，索引を作成すると，少ないコストで特定行，特定範囲にアクセスできる．MySQL クラスタ表は，B+Tree の葉ページに行への参照ではなくて行本体を格納する点以外はほぼ索引と同じ設計である．性能上の利点はあるが，木構造を持つのでその表空間内での配置は通常の表よりも複雑であり，構造劣化しやすく，性能劣化の予測も通常表より難しい．しかし B+Tree を用いた実装は最も基本的かつ広く使われているため，クラスタ表及び索引における構造劣化同定を可能にすることの意義は大きい．各 DBMS で実装の詳細は異なるが，基本的な実装方式は変わりない．

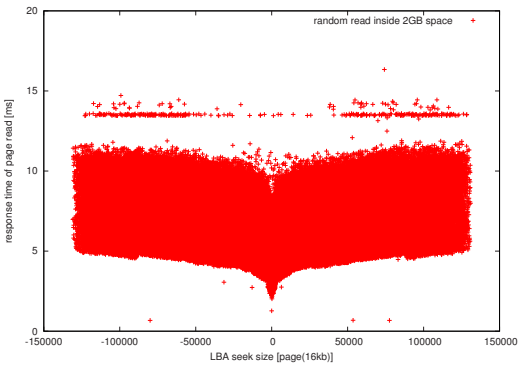

図 3: HDD 内の IO 性能特性 (ディスクの最外周ゾーン 2GB 空間)

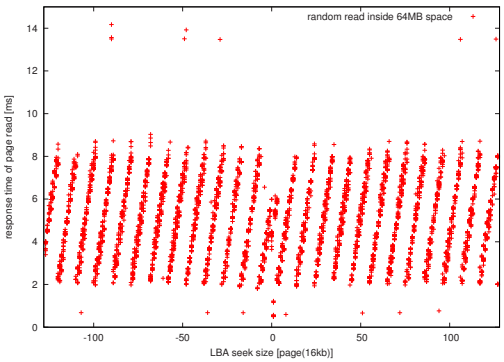

図 4: HDD 内の IO 性能特性 ($\Delta LBN = 0$ 近辺拡大図)

4.2 ストレージ内 IO 性能特性

ストレージの主要なコンポーネントはハードディスクドライブである．その IO 性能特性を考慮することにより，ハードディスク上に直接 raw デバイスを用いて表空間を作成した，基本的な場合における精度の高い IO コスト推定が可能になる．本稿ではこれ以降 Seagate のハードディスク[3] を用いる．

対象ハードディスクの性能特性を調査するため，ランダムリード負荷をかけると，図 3 の結果が得られた．横軸に 16KB ページ単位の論理ページ番号 LBN でのシーク距離を ΔLBN で示し，縦軸に各 IO のレスポンスタイムを示した．ここで，IO は全て 16KB ページ単位で発行されている．

ハードディスクのアクセスタイム T_{access} は，

$$T_{access} = T_{seek} + T_{rotation} + T_{transfer}$$

で近似される．図 3 において，縦方向に帯状に広がっ

[3]Seagate Cheetah 18LP ST318203FC[4]

て分布しているのが$T_{rotation}$(回転待ち時間) である．ΔLBN の絶対値が大きくなるにつれ，全体的にレスポンスタイムが大きくなっているのは，T_{seek}(ヘッドシーク時間) である．$T_{rotation}$ の分布がこの図ではよく分からないので，$\Delta LBN = 0$ 近辺で拡大すると，図 4 になる．ここでは，$T_{rotation}$ が，ΔLBN に対して周期的に分布していることが確認できる．また，$\Delta LBN = 1$ の場合，ディスクはすぐ次のページにアクセスするため，部分的にはシーケンシャルアクセスをしていることになる．このときは，T_{seek} 及び $T_{rotation}$ は 0 であるため，$T_{transfer}$(転送時間) は，$\Delta LBN = 1$ のときのレスポンスタイムによって近似されていると見ることができる．ところどころ非常に小さい，$\Delta LBN = 1$ の場合と同様のレスポンスタイムのプロットが見られるのは，ディスク内のキャッシュにヒットした場合と考えることができる．ディスク内キャッシュは，基本的にはシーケンシャルな同期読み込みを高速化するためのプリフェッチのためのものが主である．LRU 機構に基づいてキャッシュアウトされるので，局所性が高いアクセスはキャッシュヒットするが，このディスクは 1MB しかキャッシュを持っていないため，ランダムアクセスに準ずる場合はキャッシュヒットを期待できない．その他，14ms 付近の帯は，OS の影響で IO が待たされたものだと考えられるが，少数であり，影響は無視できると考えることにする．これらの図から，16KB ページの IO レスポンスタイムは，ΔLBN の値から高い確度で近似することができることが分かる．このレスポンスタイムを各ページアクセスの IO コスト C として考えて，$C = f(\Delta LBN)$ とする．関数 f は，上図から算出される．

4.3 範囲検索の IO コスト推定

範囲検索は，クラスタ表もしくは二次索引の B+Tree 構造の葉ページを鍵順序に従い読み込むアクセスである．この場合，葉ページ群のディスク上での位置すなわちアドレスをあらかじめ知っておけば，読み込む順番が決まっているため，各読み込みページ間の論理シーク ΔLBN が全て予測できる．葉ページ列を鍵の順序に従い $R\{p_i : i = 0, 1, 2, ..., N-1\}$ として定義すると，各ページ間の ΔLBN が，$\Delta LBN_i = LBN_i - LBN_{i-1}$ として定義される．各ページ p_i の IO コスト C_i は，$C_i = f(\Delta LBN_i)$ として算出される．ここで，

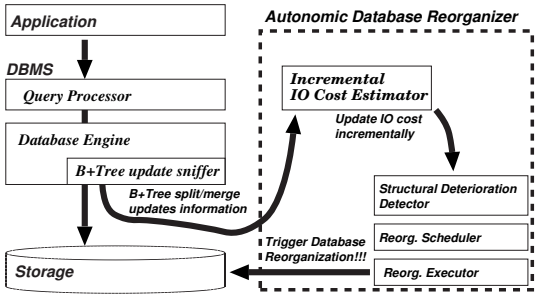

図 5: 低オーバーヘッドで IO コスト推定が可能なデータベース更新差分抽出器の概要

ΔLBN_0 は，不定であるが，本稿では大きな範囲を対象にしており，$N >> 1$ を前提とするので，$C_0 = 0$ としてもトータルの IO コストへの影響は無視できる．最後に任意の範囲に対して，対応するページ範囲 $[p_{i_1}, p_{i_2}]$ があり，その範囲検索の IO コスト C_{range} は，

$$C_{range} = \sum_{i=i_1}^{i_2} C_i$$

として推定できる．

4.4 データベース更新差分を用いた IO コスト推定手法

図 5 に，低オーバーヘッドで範囲検索の IO コストを抽出できる手法の概要を示した．表及び索引を構成する B+Tree の構造が変化するのは，split(分割) と merge(結合) によるときである．また，前節で示した IO コスト推定式は，ページ毎のコストの和で表現されていたため，ページ単位で差分計算が可能である．これらのことから，B+Tree の split と merge をリアルタイムに抽出して，ΔLBN_i の変化したページ p_i の分の IO コスト C_i を差分適用することができる．ある範囲検索 IO コストが C^{old}_{range} であり，差分適用をした後に C^{new}_{range} になるとすれば，差分適用式は，以下の式で表される．

$$C^{new}_{range} = C^{old}_{range} - \sum_{deleted\ i^{old}} C_{i^{old}} + \sum_{inserted\ i^{new}} C_{i^{new}}$$

i^{old} と i^{new} は，split もしくは merge の前後で，ページに対する i の割り当てが変化し得るため，区別している．削除も挿入もされないページで，ΔLBN の変化等で IO コストが変化する更新ページは，当該ページを削除し，その後，挿入したと考えれば良い．

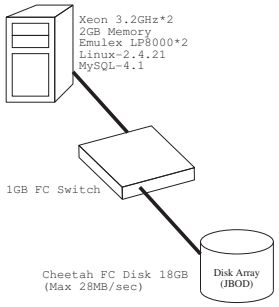

図 6: 実験環境

これにより，IO コスト推定の度に表空間をスキャンする必要がなくなる．オーバーヘッドは，split 及び merge 時に，削除されるページ及び挿入されるページの情報を取得するオーバーヘッドのみになる．これらのページは，更新される時点で既にメインメモリに読み込まれているため，差分取得操作に対してハードディスクへのアクセスは一切発生しない．このため低オーバーヘッドで実現できる．差分計算が可能なパラメータにこの手法を応用すれば，データベース診断の度に表空間をスキャンするメンテナンス IO を全くなくすことが出来る．

5 評価

我々の提案するデータベース更新差分抽出方式を MySQL [8] 用にプロトタイプ B+Tree update sniffer として実装し，全範囲検索の IO コストを推定予測精度とデータベース更新差分によって最新のデータベース構造に対して IO コストを常に推定出来ることを確認した．また，通常の MySQL と比べたオーバーヘッドを評価した．実験には TPC-H ベンチマーク [11] を用いた．

5.1 実験環境

図 6 に，実験環境を示した．Linux 2.4 が動作する PC 上でファイバチャネル経由のディスク 1 台を raw デバイスとして使用した．ディスクは cheetah 10K 18GB [4] で，最外周ゾーン (2GB) におけるシーケンシャルリードの最大スループットは 28MB/s であることを実測で確認した．

5.2 TPC-H ベンチマークによる実験

実験にあたり，トランザクション機能をサポートしている MySQL 4.1.12 InnoDB データベースエンジンを使用した．

TPCH スケールファクタは SF = 0.1 として実験を行なった．表空間は実験環境の raw デバイス上に先頭から 1GB 確保し，InnoDB 表空間を作成した．ページサイズはデフォルトの 16KB とし，各表はクラスタ表を用いて TPC-H スキーマを構築した．表空間へのデータロード時は，索引なども含めて約 220MB の空間を使用されており，更新が繰り返されることにより最大で約 300MB の空間にデータが分布した．初期ロード直後の平均ページ充填率は，95% 程度であった．

上記の空間において，以下の更新操作を 200 回繰り返した．また，以下の検索操作を適度な更新間隔で実行した．

更新操作: TPC-H ベンチマークの リフレッシュクエリ RF1(orders と lineitem のバルク挿入), RF2(orders と lineitem のバルク削除) をそれぞれ 1 回ずつ実行する．この更新で，表空間の 1% が更新される．この更新を 100 回繰り返すことで，orders 表と lineitem 表の全ての行が入れ替わり，400 回繰り返すことで，論理的には初期状態と全く同じデータに戻る．更新によってデータ量や主鍵値の分布はほぼ変わらない仕様になっている．

検索操作: orders 表と lineitem 表の全表検索を行い，その応答時間を測定する．

MySQL 4.1 における InnoDB データベースでは，全表検索も B+Tree の構造に沿った順序アクセスとして実装されており，範囲検索と同等であるので，我々の提案する IO コスト推定手法が適用できる．

また，データベース更新差分取得器を用いず，リリースされたままのソースコードからコンパイルした MySQL without sniffer と，データベース更新差分取得コードを実装したソースコードからコンパイルした MySQL with sniffer のそれぞれについてデータベース構造が変化する操作 2 つについてオーバーヘッドを測定した．

以下に実験結果を示す．

図 7 と図 8 に，更新操作に伴う全表検索性能の提案手法による IO コスト推定値と，実際のクエリ実

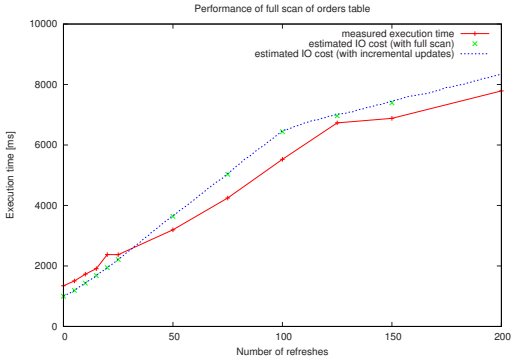

図 7: orders 表の全表検索

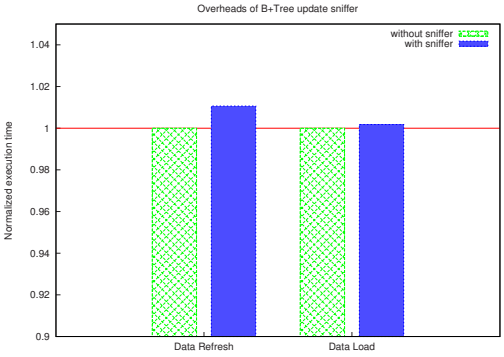

図 9: データベース差分取得器のオーバーヘッド

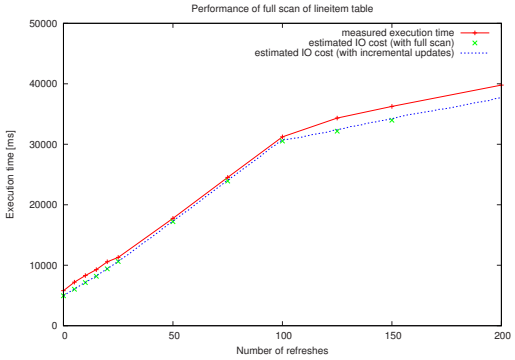

図 8: lineitem 表の全表検索

行時間を示した．全表検索は，IO インテンシブな操作なので，IO コストがほぼ実行時間を占めるものと考えて良い．ただ，時間の関係上小さいデータセットを用いて実験を行っており，特に orders 表の全表検索は数秒で完了するサイズなので，CPU コスト，プログラムロード等の時間が誤差に大きく含まれていることはあり得る．

更新操作に伴い，推定値，実測値が共に劣化している．推定誤差は orders 表については 17% 程度，lineitem 表の範囲検索については，6% 程度であった．

データベース更新差分抽出器のオーバーヘッドを図 9 に示した．Data Refresh は，更新操作 1 回分である RF1 と RF2 の実行時間の和を，実験を通して平均した値を用い，Data Load は全てのデータの初期ロード時間測定を 10 回行ない，平均値を比較した．Data Refresh は 1% 程度，Data Load は 0.2% 程度のオーバーヘッドであった．これ以外に，取得したデータベース更新差分を別プログラム IO コスト推定器 にて保持，適用，IO コスト計算等行っているが，別の CPU で実行されているため，この図には含まれていない．

6 おわりに

本稿は，関係データベース更新差分を用いて，クラスタ表の範囲検索における IO コスト推定を，低オーバーヘッドで行う手法を提案した．これにより，これまで IO コスト推定の度に必要だった表空間スキャンが必要なくなり，オーバーヘッドの大幅な削減を実現した．また，提案手法を MySQL データベースにプロトタイプとして実装し，TPC-H ベンチマークを用いて，これまでに提案した範囲検索の高精度な IO コスト推定機能はそのままで，低オーバーヘッドゆえにリアルタイムに IO コスト推定ができるようになることを示した．

本研究は再編成の自立化という大きなテーマを掲げているが，データベース状態把握手法の確立という道を経ることで，それが実現に近づくものと考えている．本研究の課題と今後の展望を，以下に挙げる．

範囲検索以外の性能推定式の作成: 現状では，範囲検索のみの性能推定式しか評価していない．他のアクセスパタンについても同様の性能推定式を立式し評価を行う必要がある．また，それらの組み合わせや，結合などによる複雑な実行計画について性能劣化への影響を考察し，様々なクエリについて性能劣化同定を行えるようにしたい．また，クエリだけでなく，更新操作 (行挿入，行削除，行更新) の性能劣化についても同様に性能劣化式を立式し，評価したい．

表空間のリアルタイム分析の実現: 本稿で提案した手法が適用できるパラメータは，例えばデータ充填率などがあり，それらをリアルタイムに保持，分析できるツールを作成すれば，オンライン性能にほとんど影響を与えずに表空間状態を容易にかつ詳細に把握することができる．このツールを作成することで，自立再編成器への足掛かりになるだけでなく，能動的なデータベースチューニング，メンテナンスへの発展も目指すことが出来ると考えている．

高度ストレージへの対応: これまで，ハードディスクにおける IO 性能特性を用いて IO コスト推定を高精度にすることを目指したが，大容量キャッシュを積み，RAID，仮想化などの豊富な機能を有した，より高度なストレージにおける IO コスト推定が必要とされる．これまでの方式を発展させることで可能なのかどうかを含めて検討する必要がある．

謝辞

本研究の一部は，文部科学省リーディングプロジェクト e-society 基盤ソフトウェアの総合開発「先進的なストレージ技術」の助成により行われた．協力企業である株式会社日立製作所より多くの有益なコメントを頂戴した．深謝する次第である．

参考文献

[1] Don S. Batory. Optimal file designs and reorganization points. *ACM Trans. Database Syst.*, 7(1):60–81, 1982.

[2] (Ed.)D.Lomet. Special Issue on Online Reorganization. *IEEE Data Eng. Bull.*, 19(2):1, 1996.

[3] Sam Lightstone, Berni Schiefer, Danny Zilio, and Jim Kleewein. Autonomic Computing for Relational Databases: The Ten Year Vision. In *Proceedings of Workshop on Autonomic Computing Principles and Architectures (AUCOPA2003)*, August 2003.

[4] Seagate Technology LLC. *Cheetah 18LP FC Disk Drive Product Manual Volume 1*, 1999.

[5] Guy M. Lohman and John A. Muckstadt. Optimal policy for batch operations: Backup, checkpointing, reorganization, and updating (abstract). In *SIGMOD Conference*, page 157, 1977.

[6] Lothar F. Mackert and Guy M. Lohman. Index Scans Using a Finite LRU Buffer: A Validated I/O Model. *ACM Trans. Database Syst.*, 14(3):401–424, 1989.

[7] K. Maruyama and S. E. Smith. Optimal reorganization of distributed space disk files. *Commun. ACM*, 19(11):634–642, 1976.

[8] MySQL: The World's Most Popular Open Source Database. http://www.mysql.com/.

[9] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access Path Selection in a Relational Database Management System. In *SIGMOD Conference*, pages 23–34. ACM, 1979.

[10] Ben Shneiderman. Optimum data base reorganization points. *Commun. ACM*, 16(6):362–365, 1973.

[11] TPC: Transaction Processing Performance Council . http://www.tpc.org/.

[12] S. Bing Yao, K. S. Das, and Toby J. Teorey. A Dynamic Database Reorganization Algorithm. *ACM Trans. Database Syst*, 1(2):159–174, 1976.

[13] 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**, 46(TOD (SIG 8)):pp.1–18, 2005.

[14] 星野 喬，合田 和生，喜連川 優. 関係データベースシステムにおける自己再編成に関する一考案. In **夏のデータベースワークショップ** *DBWS2004*, 2004.

[15] 星野 喬，合田 和生，喜連川 優. 関係データベース再編成契機決定のための性能劣化同定方式. In **電子情報通信学会 第** *16* **回データ工学ワークショップ** *(DEWS2005)*, 2005.