DEWS2005 5-B-02

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

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
E-mail: †{hoshino,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし 我々は関係データベースシステムにおける再編成タスクを自立化することを目的としている。再編成は、性能劣化の要因となる表空間内のデータの構造劣化を是正することにより、性能を回復する操作である。現状の関係データベースシステムにおいて、データの構造劣化を避けることはできず、再編成は必要である。現状では表空間内の構造劣化を性能劣化の原因として明確に表現する指標がないため、再編成契機を判断することは管理者でさえ難しい。本論文では、ストレージ内での IO 性能を考慮し、構造劣化指標を定義する。それを用いて、定量的な性能劣化同定モデルを提案する。また、それが再編成契機の決定に有効であることを示す。
キーワード データベース管理、自立再編成、構造劣化、性能劣化同定、ストレージ

Performance Degradation Detection for Reorganization Trigger on Relational Database Systems

Takashi HOSHINO†, Kazuo GODA†, and 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
E-mail: †{hoshino,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract We aim at autonomic reorganization for DBMS. Reorganization, which is one of the major database administration tasks, counteracts structural deterioration in tablespaces to recover performance. It is actually inevitable that data updates cause structural deterioration, therefore reorganization is an essential task in database administration. At present, there are only vague factors of disorganization for practical diagnosis whether administorator should trigger reorganization method or not. In this paper, we consider IO behavior of data updated at storage level and clarify causal relationship between factors of structural deterioration and performance degradation. We propose a quantitative method that detects performance degradation for reorganization trigger, evaluate it with empirical test studies.
Key words Database Administration, Autonomic Reorganization, Structural Deterioration, Performance Degradation Detection, Storage

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 [4] 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。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(注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。我々は、性能劣化と構造劣化の関係をストレージ内の IO を考慮した上でモデル化し、構造劣化指標としてシーケンシャリティと充填率を定義し、それらの指標を用いた性能劣化同定手法を構築した。その後、簡単な人工負荷実験と、実データベースを用いた実験にて評価を行ない、提案手法の有効性を示した。

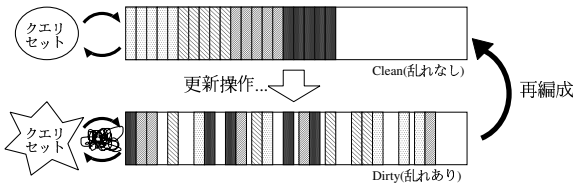

図 1　再編成の概念図

本論文は以下のように構成される。まず第 2 章で関係データベースにおける現状の再編成とその課題について述べる。次に第 3 章で我々の提案する自立再編成の枠組について、第 4 章では我々の提案する性能劣化同定手法について説明する。その後、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. 現状の再編成と課題

2.1 再　編　成

図 1 に、DBMS における再編成の概要を図示した。

DBMS を運用するにあたり、データを表空間にロードした直後は、データが整然と並んでいる。その後、運用によってデータベースの更新が繰り返され、徐々にデータが期待した通りに並ばなくなってくる。そのような状態になると、ロード直後と同様のデータセット、クエリセットにおいても、性能が悪化する。これをそのままにしておくと、要求される性能を下回るほど性能が悪化してしまう。我々はこれを、表空間におけるデータの構造劣化と呼ぶことにする。データの構造劣化は、許容できない性能劣化を引き起こす。性能要求を満たせなくなる前に再編成を実施し、構造劣化を改善する必要がある。再編成時には性能要求を安全なレベルで満たしつつ、少ない再編成コストで性能劣化を少なく抑えるようにすることが望ましい。

2.2 再編成の自立化に向けた課題

我々は再編成の自立化を目的としている。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 [3] 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。

我々は再編成スケジューリングを自立化することに焦点をあてる。その最初の段階として、構造劣化指標からクエリの性能劣化を予測することで再編成契機を判断できる機構を作ることを目指している。

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 [2], [6], [7], [9], [12] 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。

現状では、具体的な DBMS を対象にしてデータの構造劣化を定量的な性能劣化に直接結びつける方法論が存在しない。もっと単純に、実性能の劣化のみを監視し、再編成スケジューリン

(注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、本論文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。

グをする方法が考えられる。しかし、バッチ処理などの時間がかかり、クエリが発行される前に性能劣化を予測したい場合などには使えない。また、性能劣化は以下のような要因でも起きる。

- データの増大や質の変化
- 実行計画の変更
- バッファの振舞いの変化
- アプリケーション負荷の変動

実性能のみを監視する方法では、我々が今対象としている構造劣化に起因する劣化とそれ以外の要因による劣化との切り分けが不可能である。構造劣化が起きておらず、構造劣化以外の要因が性能劣化に大きく影響している場合、再編成による性能改善は見込めず、実性能の監視だけでは再編成契機の決定を高い確度で行うことは出来ない。

現実の運用では、熟練された管理者が経験則を元に構造劣化の量と性能劣化量との対応を推定し、再編成契機を判断している。構造劣化を示す指標と性能劣化の対応はいくつかが知られている。

- 行断片化や移行行による行取得性能劣化
- B 木の高さが必要以上に高くなることによる行取得性能劣化
- HWM 無効化による行挿入性能劣化
- クラスタリングプロパティ減少による範囲検索性能の劣化
- 充填率減少による検索性能劣化

などである。(注2) ここで挙げた例が全ての構造劣化を網羅しているかは分かっておらず、また、全ての例が個々の DBMS 実装で存在するわけではない。以上の対応関係は、DBMS 内では表空間や表、索引のデータ構造、データ更新戦略、実行計画の作成方法などに依存している。また、これらの指標は性能劣化との対応が曖昧なまま、性能劣化予測や再編成判断に使われており、指標同士の相関関係やトレードオフなども不明瞭である。

一般的にクエリの性能を予測するためには、DBMS の中だけでも、

- 実行計画の把握
- バッファの影響
- 表空間内のデータ配置

などの影響を考慮して始めて OS が提供する二次記憶空間に対するページ IO レベルの性能が予測できる。OS 内においては、

- ファイルシステムの影響
- ブロックデバイスのバッファなどの IO 戦略
- LV(注3)とのマッピング
- SCSI 層での IO キューイング

などの影響を考慮して始めてストレージデバイスが提供する二次記憶空間に対するページ IO レベルの性能が予測できる。ストレージデバイス内においては、

(注2)：各 DBMS のマニュアル等参照。[8]

(注3)：Logical Volume. 主に OS 層での抽象化されたディスク空間を表す。

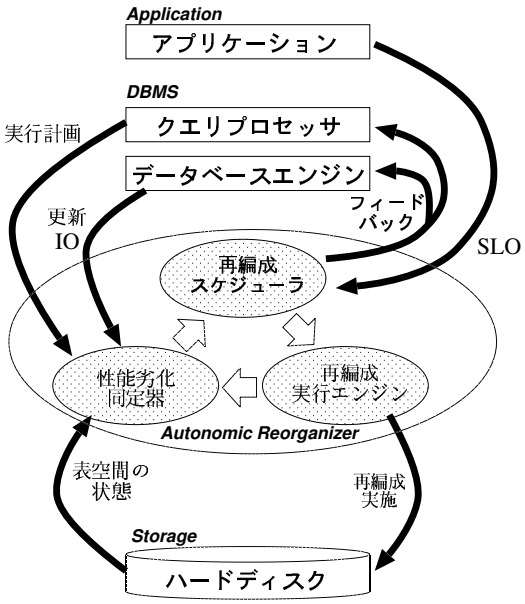

図 2　自立再編成の枠組

- LU(注4)とのマッピング
- RAID コントローラの IO 戦略の RAID レベルの違いによる影響
- ハーディスクの物理的な位置やシーク量、キャッシュアルゴリズムの影響

これらの影響を考慮して始めて実性能が予測できる。

構造劣化として規定される指標は、DBMS 層における表空間内のデータ配置より下の全ての層に存在し得る。DBMS 内での IO 振舞いに対しては、効率的な実行計画を作成するためのクエリオプティマイザのアルゴリズムがある。例えばデータの量や分布などを用いて性能を予測する機構は既に実装されている。しかし、構造劣化に起因する定量的な性能予測は DBMS 内部でもほとんど行われていない。また、ストレージ内 IO 性能を考慮した定量的な性能劣化予測を用いて再編成契機の決定に応用しようとする試みは未だ存在しない。

3. 自立再編成の枠組

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。

性能劣化同定器: 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における性能推定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

(注4)：Logical Unit. 主にストレージ層での抽象化されたディスク空間を表す。

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。

本論文では、このうち性能劣化同定機構について議論を行う。

4. 性能劣化同定方式

我々はデータの構造劣化を表す指標を定義し、ストレージ内 IO 性能を考慮した、定量的な性能劣化同定が可能な初歩的なモデルを構築した。

本章では、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。そのモデルを考慮して、DBMS 内のみならず、ストレージ内において性能に影響を及ぼしうる指標候補を列挙し、構造劣化指標を定めた。

構造劣化指標が、DBMS 内、ストレージ内にて性能にどのような影響を与えるかを示し、クエリコスト推定式を提案する。最後に性能劣化同定器の設計例として、低コストで性能劣化同定が可能な方式を提案する。

4.1 対象 DBMS 実装

本論文では、対象となる DBMS 実装形式を次に示すように限定して議論する。

対象の表、索引: B+木構造を持ち、主鍵によって論理的な順序性を保持する索引構成表 [1] もしくは索引を対象とする。ページ内に含まれる行の主鍵範囲は、他のどのページの主鍵範囲とも重複しない。葉ページの数に比べて、枝ページの数は無視できるほどに小さいか、論理的な隣り同士のページは相互に参照可能であることを前提とする。

対象とする更新戦略: 行挿入については、データの物理的な順序性を出来るだけ損わないような挿入を行うものとする。ページスプリットが起きる場合も、ページ確保はスプリット前のページ位置に出来るだけ近い空き領域を選択する。

行削除については、削除マークをつけるのみでその行が存在した領域をそのままでは再利用しない (論理削除)。ページ単位での領域再利用はされるものとする。

行更新について、論理削除と挿入を行うものとする。行分岐、行連鎖の類は起きないものとする。

対象とするクエリ: クエリの実行計画が DBMS から与えらえることを前提とする。実行計画は、基本アクセスの組み合わせで表現できると仮定する。基本アクセスは、全表検索、範囲検索、ランダム検索とする。本論文では範囲検索に限定して議論する。

対象とする性能要求: 性能要求 SLO は、対象クエリの応答時間を性能とし、最高性能時の α 倍以下を保証するものとする。α はユーザまたはアプリケーションによって与えられる定数である。

索引構成表や索引は、DBMS 実装において性能上重要な役割を果たしている。DBMS で定義される通常の表の形式は、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。索引構成表は、B+木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場合が多い。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、構造劣化パタンも多く、性能劣化の予測も通常表より難しい。しかし B+木を用いた実装は最も基本的かつ広く使われているため、索引構成表及び索引における性能劣化同定方式を実現することの意義は大きい。

各 DBMS にて実装の詳細は異なり、我々の対象実装は単純化している部分もあるが、基本的な実装方式は変わりない。

4.2 ストレージ内 IO モデル

データの構造劣化が、DBMS 内の IO の振舞いだけを考えていたのでは性能劣化を検知できないような指標が存在する。そのような構造劣化指標を導出するために、まず始めにストレージ内の IO モデルを構築し、どのような IO がストレージ層で性能の違いとなって表れるのかを定式化する必要がある。

ストレージ内 IO モデルを構築するにあたり、再編成契機決定のための性能予測を想定しているため、少ない性能予測コストで、実用的な精度の性能予測をしたい。また、常に表空間を全走査して情報を取得するのではなく、通常は更新 IO のみを用いて差分による状態保持ができるモデルが望ましい。

以上のことを考慮し、我々は、DBMS のストレージとして使われているハードディスクを対象に典型的な事象を網羅する、簡単な IO 性能モデルを構築した。

まずハードディスクに対するページ読み込みは、シーケンシャルなアクセスと非シーケンシャルなアクセスに分けて考えられるものとする。そして次のように C_s と C_r を定義する。

C_s: シーケンシャルアクセスとみなせるページ読み込み応答時間

C_r: 非シーケンシャルとみなせるページ読み込み応答時間

ページ群の読み込み性能 C は、シーケンシャルアクセスとみなせる読み込みページ数を N_s、非シーケンシャルアクセスとみなせる読み込みページ数を N_r、としたとき、

$$C = N_s C_s + N_r C_r \tag{1}$$

と表される。C_s と C_r はページ群の存在する空間の位置と大きさに依存した値とし、C_s はその空間におけるシーケンシャルページアクセスの平均応答時間を、C_r はその空間におけるランダムページアクセスの平均応答時間を使用する。

ハードディスクの性質から考えて、以下の点が我々のモデルの特徴として挙げられる。

- ハードディスクにおいてシーケンシャル読み込みはシークが発生しないため高速であるという事実に基づいた簡単な性能式である。
- データベースエンジンや SCSI 層などの上位層で、連続ページへの複数の IO が統合される場合がある。すると IO の振舞いが変わり、シーケンシャルアクセ スとみなせるページの読み込み性能に不均衡が生じる可能性がある。後の評価で使用したハードディスク (注5) においては、簡単な評価実験により、最外周ゾーン内に表空間を確保しページサイズを 16KB にしてい

(注5)：Seagate Cheetah 18LP ST318203FC [5]

た場合、IO まとめ効果によるページあたりの応答時間 C_s にはほぼ変化がないことを確認した。

- 非シーケンシャルアクセスとみなせるページに関して、表空間内でのランダムアクセスの平均応答性能をページあたりの読み込みコストとして使用するため、例えば、表空間内ページの読み込みにおいて、シークサイズの平均値が表空間でのランダムアクセス平均シークサイズよりも大幅に違う場合、C_r が変化し、定数として扱えない場合がある。

4.3 構造劣化指標

索引や索引構成表の実装で使われる B+木を使う理由は、

- 鍵属性を用いた範囲検索が高速
- 鍵属性を用いた 1 行検索が高速

であることが挙げられる。

本論文は、この範囲検索に限定して議論している。

範囲検索実行時に、DBMS 内及びストレージ内において性能に影響を与えると思われる指標がいくつかある。

- DB バッファヒット
- B+木の高さ
- B+木の葉ページの順序性
- ページ充填率

このうち、DB バッファヒットは構造劣化指標ではないことは明らかである。また、B+木の高さは 1 行検索には意味のある指標であるが、範囲検索時ににアクセスする葉ページの数に比べて枝ページのアクセスは小さいことを前提にして議論しているので、ここでは影響がほぼ無視できるものとする。よって、B+木の葉ページの順序性と、ページ充填率とを構造劣化指標として定量化する。

4.3.1 指標「シーケンシャリティS」

索引構成表や索引を構成する B+木における、葉ページにおける主鍵の順序をページの論理順序とする。範囲検索走査は論理順序に従い、ページが読み込まれる。

表空間のフォーマットが規定する物理アドレス順に並ぶページの順序をページの物理順序とする。物理順序は最終的にハードディスクの LBA 順序に対応することが期待される。

ページに対して、指標シーケンシャリティS(*Sequentiality*) を次のように定義する。

論理的に順方向に、ページ A、B、C、が並んでいるとする。物理的にページ A から順方向に L ページ以内にページ B が存在すれば、$S_1 = 1/2$ を、そうでなければ $S_1 = 0$ とする。同様にページ B から順方向に L ページ以内にページ C が存在すれば、$S_2 = 1/2$ を、そうでなければ $S_2 = 0$ とする。ページ B におけるシーケンシャリティ S を $S = S_1 + S_2$ と定義する。$0 \leqq S \leqq 1$ である。

A から B、B から C、へのアクセスを論理順序アクセスと呼ぶ。ここで、L は、ハードディスクの設計に依存して定める定数である。

B 木の論理的な順序性ストレージ内の順序性を対応させる再編成を行なう方式が研究 [11] で行なわれている。我々は枝ページを無視しているが、B 木のページ全てを深さ優先順位で順序づけしている。我々の前提では無視できることになっているが、隣同士の葉ページが相互参照を持っていなかった場合は、範囲検索はこの順序でページアクセスするため、自然である。

4.3.2 指標「充填率 F」

ページ充填率を F *(Fill Factor)* とし、

$$F = \text{ページ内データ量}/\text{ページサイズ} \tag{2}$$

と定義する。

分母をページサイズにしたことで、ヘッダやトレイラなど、または行サイズに満たない領域がページ内に存在したときの影響で充填率が低く計算されることになるが、性能劣化を推定する場合は、充填率の相対的な変化のみに注目すればよく、問題にはならない。

4.3.3 空間に対する指標 S、F

順序を持つ N ページからなる表空間内の部分空間があって、各ページのシーケンシャリティと充填率を $(S_1, F_1), (S_2, F_2), ..., (S_N, F_N)$ としたときに、その空間の S、F を、

$$S = \frac{\sum_{i=1}^{N} S_i}{N}, F = \frac{\sum_{i=1}^{N} F_i}{N} \tag{3}$$

つまり、その空間における平均シーケンシャリティS、平均充填率 F として定義する。

S、F はページ単位で微積分可能である。ここで、一緒に扱われるページは同一の表または索引に所属する。

次の節で、ストレージ内 IO を考慮した範囲検索性能推定式を立式する。

4.4 範囲検索コスト推定

構造化指標が及ぼす性能への影響を以下に述べる。

まず、シーケンシャリティにおける性能劣化の仕組みについて述べる。B+木の葉ページ群においてシーケンシャリティが低くなると、ストレージ内 IO において、シーケンシャルページアクセスが減り、ページあたりの読み込み性能が下がるため、範囲検索性能が劣化する。B+木における範囲検索は、B+木において論理的な順序でページがアクセスされる。表空間内の N ページが読まれるものとして、シーケンシャルページアクセスの割合を S と、非シーケンシャルアクセスの割合を $1 - S$ とすると、それぞれのページ数は NS ページ、$N(1 - S)$ ページとなる。これを式 (1) に代入すると、範囲検索コスト C^{range} は、

$$\begin{aligned} C^{range} &= C_s NS + C_r N(1 - S) \\ &= N \cdot (C_s S + C_r(1 - S)) \end{aligned} \tag{4}$$

となる。

次に、充填率における性能劣化の仕組みについて述べる。充填率が低くなると、同じデータを読むときに、より多くのページを読み込む必要がある。表空間の平均充填率が F だったとき、N_{data} ページ分のデータを読み込むときに実際に読み込むページ数 N は、

$$N = \frac{N_{data}}{F} \tag{5}$$

である。

式 (4) と式 (5) から、N_{data} ページ分のデータに対する範囲検

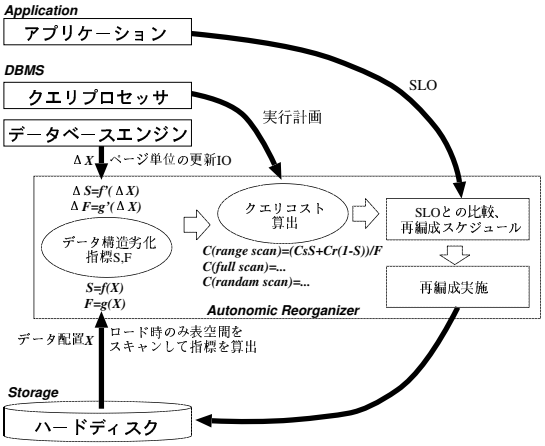

図 3　低コストで性能推定が可能な性能劣化同定器概要

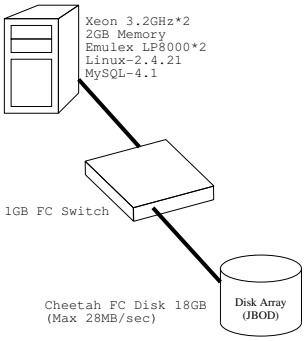

図 4　実 験 環 境

索コスト C^{range} が次のように導かれる。

$$C^{range} = N_{data} \cdot \frac{C_s S + C_r(1-S)}{F} \qquad (6)$$

4.5　性能劣化同定器の設計例

図 3 に、低コストな性能劣化同定器の設計図を示す。更新 IO による構造劣化指標 S, F の差分計算、S, F によるクエリコスト推定計算実施、クエリコストを SLO を比較して再編成契機を判断、再編成を自立的に実施することができる。

更新 IO はページ単位の IO として観測可能である。ページ単位の IO に関して、表空間に対して定義された S、F は差分計算可能である。ゆえに、指標 S、F は、更新 IO のみを用いて表空間の変化に追随可能である。性能劣化同定コストを計算するために、毎回表空間を読み込むのに比べて低いコストで状態保持が可能である。また S、F から C^{range} を計算することは容易なため、性能劣化同定コストが低いと言うことが出来る。紙面の都合上、証明は割愛する。

5.　評　　価

我々の提案する性能劣化同定方式を、現実のディスクや DBMS におけるデータ配置と負荷に適用したときの性能劣化の予測精度を 2 つの実験で評価した。

5.1　実 験 環 境

図 4 に、実験環境を示した。Linux 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 [5] で、最外周ゾーン (2GB)

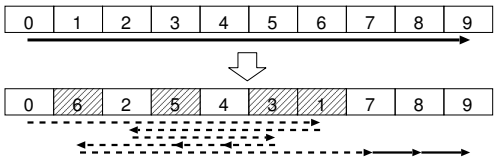

図 5　実験空間内における更新によるページ配置の変化

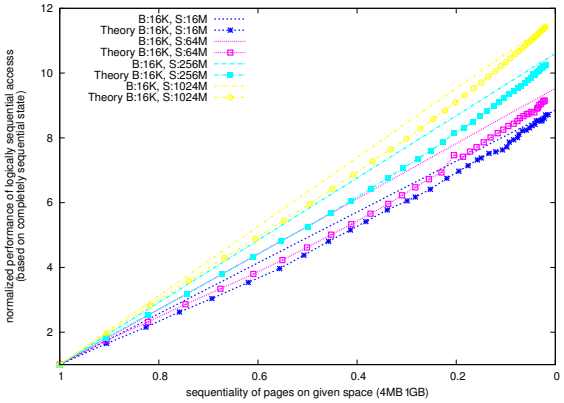

図 6　シーケンシャリティの変化に伴う性能劣化予測と実性能

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　人工負荷による評価

提案するシーケンシャリティの定義を用いた範囲検索の性能予測式と、実際の範囲検索の性能劣化について比較するため、次のような実験を行った。

大きさ一定、シーケンシャルリード性能一定の空間をディスク上に確保した。初期状態では大きさ 16KB (= *page_size*) のページが一様に並んでいるとし、各ページに LBA に対応するように 0 から $N-1$ の番号を割り当てた。

表空間サイズ (=*space_size*) として 16MB、64MB、256MB、1024MB の 4 通りについて実験を行った。N = *space_size*/*page_size* である。それぞれの空間について、あらかじめ C_s と C_r を測定し、性能推定式に使用した。

上記の空間において、以下の操作を 40 回繰り返した。

範囲検索操作:　番号順にページを読み込んで、応答時間を測定した。読み込むべき次のページが離れていればシークが発生し、性能が劣化する。

更新操作:　ランダムにページを 2 つ選び、入れ替えた。更新 1 回につき、5% の空間が変更されるように指定した。約 20 回の操作で空間内のほぼ全てのページが更新の対象になる。

図 5 に更新操作の概要を示した。更新に伴い空間のシーケンシャリティと及びシーケンシャリティを用いた範囲検索の性能予測値を算出し、実性能と比べた。ちなみに、この実験においては充填率は固定値 1 として性能予測式に代入した。

結果を図 6 に示した。横軸にシーケンシャリティをとり、縦軸に初期状態を基準に正規化した応答時間を性能としてとった。シーケンシャリティの減少に伴い、性能が劣化していることが確認できる。我々の性能劣化予測式は図 6 では直線として表現される。初期状態におけるシーケンシャリティは定義からほぼ

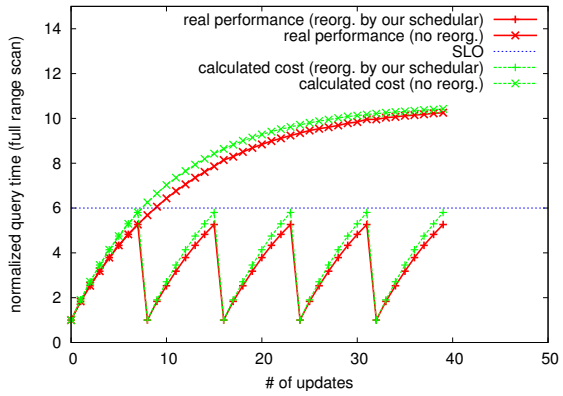

図 7　性能劣化予測による自立再編成運用例

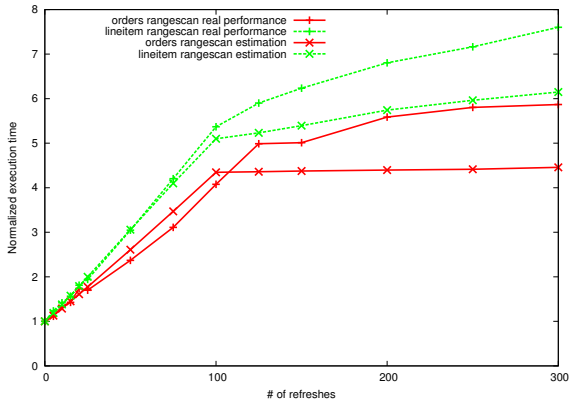

図 8　更新操作に伴う範囲検索性能劣化の推定値と実測値

1 であり、そのときの性能は C_sN、シーケンシャリティ 0 における性能は C_rN であるので、図 6 における切片は C_r/C_s になる。実負荷を見ると性能劣化していることが同様に確認できる。シーケンシャリティが 1 に近いときと、0 に近いときは予測値とほぼ同じ予測が出来ている。予測誤差は、最大でも 1 割程度である。異なるサイズの表空間においては、ランダムアクセスのコストの差に表れていることが確認できた。この図では正規化されて見えないが、シーケンシャルコスト C_s はほぼ同じだった。空間の大きさではなく、位置のみに依存するという前提と一致する。

図 7 に、再編成契機の自立判断の例を示した。256MB の表空間における実験データを使用した。最高性能時の 6 倍以内を保証するという SLO が設定された運用であると仮定している(注6)。横軸は更新回数、つまり時間経過を示している。我々のモデルを用いて次回のクエリ実行時の性能劣化を線形予測すれば、6 倍を越える時間は判断できるため、図 7 に示したような運用が可能になる。性能推定と実性能の誤差が 10%であるということは、自立再編成スケジューラが SLO を比較するときに 10%のマージンを与えて判断すれば十分に SLO を保証できるシステムを構築出来ることを示している。

5.3　TPCH ベンチマークによる評価

オープンソース DBMS である MySQL [8] と標準的な意思決定ベンチマークである TPCH [10] のデータを用いて実験を行ない、提案手法を評価した。

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 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 4GB 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、各表は索引構成表を用いて TPCH 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 300 回繰り返した。また、範囲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。

範囲検索操作:　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+木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、我々の提案する性能劣化式が適用できる。C_s、C_r については、実測データにおけるシーケンシャルアクセスと見なせるページアクセス応答時間の平均値を C_s として、非シーケンシャルアクセスと見なせるページアクセス応答時間の平均値を C_r として用いた。

シーケンシャリティ S の計算において、簡単のため $L = 1$ とした。充填率 F は定義通り算出した。

図 8 に、更新操作に伴う範囲検索性能の提案手法による推定値と、実性能を示した。更新操作に伴い、推定値、実測値が共に劣化している。推定の精度に関して、更新操作が 100 未満の間は高い精度で性能を推定できている。誤差は lineitem 表の範囲検索については、4% 程度、orders 表については 10% 程度である。しかし、更新操作が 100 以上の領域では、推定値と実測値の乖離が目立ち、30%程度になっている。

図 9 に、更新操作数と S の対応を、図 10 に、更新操作数と F の対応を、それぞれ示した。

更新回数 100 までは S、F、共に直線的に減少した。その後更新 IO の振舞いが変化し、orders 表の S はほぼ 0 に収束した後、変化が見られなかった。充填率 F は全体で 10 % 程度の変化しかないので、図 8 における性能推定値においても 10 % 程度の寄与をしていることになる。しかし、F の変化率のみでは説明できない orders 表の範囲検索性能劣化が実性能では観測さ

(注6)：運用にもよるが、オンラインクエリに関しては、2 倍でも大きな性能劣化であることが多い。バッチ処理に関しては再編成コストと性能劣化の和を最小にするような判断が必要になる。

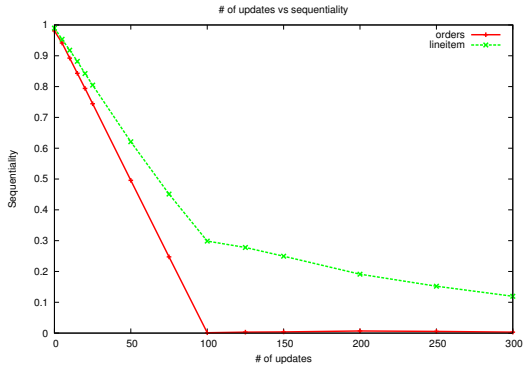

図 9　更新操作に伴うシーケンシャリティ S の変化

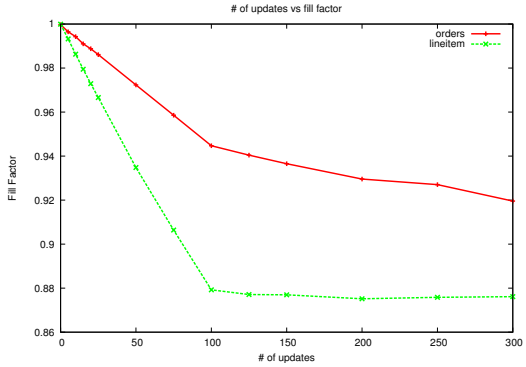

図 10　更新操作に伴う充填率 F の変化

れている。

ここでも C_s は実験を通じて一定だったが、4.2 節において述べた、C_r が、実験を通して変化しており、もはや定数として扱えなくなっていることが分かった。

よって、この実験では更新回数 100 付近で更新パタンが変化し、S と F だけでは記述できない現象が起きていることを示してみる。それは非シーケンシャルアクセスにおけるシーク量の分布の違いに起因する、ページあたりの読み込みコスト C_r が変化する現象である。結果、性能劣化推定誤差が更新回数 100 以上で大きくなった理由が説明できる。

この実験においても、3 割の性能誤差をマージンとして想定すれば再編成判断に利用できる。

6. おわりに

本論文は、関係データベースにおいて再編成契機を決定するための性能劣化同定方式の基本的な枠組とアイデアを提案し、ストレージ IO 性能を考慮したモデルを構築した。そのモデルを用いて、低コストで維持可能な表空間内の構造劣化指標を定義した。さらに定量的な性能推定式を基本的なクエリに関して立式し、その有効性と可能性について示した。

本研究の課題と今後の展望を、以下に挙げる。

ストレージ内非シーケンシャル IO 性能のモデル化:　本論文で我々が提案したストレージ IO 性能モデルでは、非シーケンシャルページアクセスを同じシークコストで読み込めるとして一元的に扱っていたが、シークコストの分布が構造によって異なるので、一部の構造劣化に対応できていない。何らかの形でシーク距離の分布を指標として保持することで対応できると思われる。

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。他の基本アクセスである、全表検索とランダム検索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。

空間の並列性や位置の違いをモデル化:　現状では、データ順序性と充填率を考慮しているが、他にもマクロに見たときのデータ位置、空間の並列アクセス性などが現在網羅していない指標候補であることが分かっている。現状の再編成の枠組を超越して、性能チューニングの域に入る可能性はあるが、構造指標が性能に影響を与えるという点では、同じものとして扱うことが出来る。

再編成方式の効率化:　我々が提案したモデル化により、更新戦略や再編成時のアクセスパタンなどに応じた効率化が可能である。再編成が不可欠なタスクであることを考えると、少ない再編成コストで、性能要求を満たす方式を実現することが期待される。

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。

文　献

[1] Oracle 9i Index-Organized Tables Technical Whitepaper. White paper, Oracle, 2001.

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

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

[4] 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.

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

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

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

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

[11] Satoshi Watanabe and Takao Miura. Reordering B-tree files. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 681–686. ACM Press, 2002.

[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.