データベースにおけるリアルタイム構造劣化監視機構の試作

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

データベース更新が繰り返されると，二次記憶装置上のデータ格納構造が非効率になり，データベースアクセス性能が低下する．このような構造劣化を除去するためには，データベース再編成が不可欠である．再編成の実施においては，構造劣化している空間を同定する必要があるが，データベース容量，常時運用ニーズの増大などの背景からデータベース運用中に低オーバーヘッドでリアルタイムに構造劣化を監視できることが望ましい．本稿ではそのようなオンラインデータベース構造劣化モニタを試作し，TPC-C ベンチマークを用いて再編成管理例を示す．

Prototype of Real-time Structural Deterioration Monitor for DBMS

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

Database updates disorganize data stored physically in secondery storage, which is called structural deterioration. Structural deterioration causes performance degradation. Database reorganization removes structural deterioration. Reorganization scheme requires to know which section of the database should be reorganized. Nowdays, database size increases, database administration becomes costful, and full-time database operation is required, so it is desirable to monitor structural deterioration with little overhead in real time. In this paper, we make a prototype of online structural deterioration monitor and show an example of decision of reorganization target using TPC-C benchmark.

1 はじめに

近年，様々な分野において運用されているデータベース (以下，DB) が大容量化している．そのため，データベースシステム (以下，DBMS) の管理はより困難になり，人件費等のコストが増大している．DBMS 常時運用ニーズもまた増大しており，DB の状態監視および管理タスク実行に費やすことのできる時間やシステムリソースが少なく，DB 管理をさらに難しいものにしている．また，システムにおける人的操作ミスが，甚大な被害を引き起こす事例も増えており，人間の管理者のみによるきめの細かい DB 管理に限界が見えてきた．この問題を解決するため，DB 管理の自立化を目指す試みが増えている [4].

我々は DB 再編成管理に着目し，その自立化を

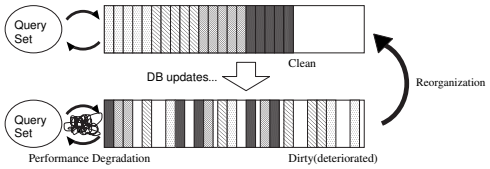


図 1: 構造劣化と再編成

目的としている．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，性能を大幅に低下させる場合がある (図 1)．このため，DB 管理者は再編成を行うことにより，ストレージ内のデータを再配置し，性能低下を予め防ぐ必要がある．現状では，構造劣化を同定し再編成を実施するべきか否かの判断は難しく，また，再編成自体にかかる時間は膨大であることから，再編成は高度な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．

常時運用に対応した再編成の自立化を実現するために，以下の課題を解決する必要がある．

- 性能指標としての定量的な構造劣化モデルがない
- 構造劣化監視を低オーバーヘッドで行う手段がない
- 再編成実施判断のための枠組みがない

まず，構造劣化を十分な精度，粒度で表現する手法，及び，構造劣化を常時監視する機構が必要であるが，観測される性能をそのまま構造劣化量として用いることは，構造劣化以外の性能低下原因との切り分けが難しく，再編成の判断材料としては使い辛い．また，バッチ処理など時間のかかる処理に対して，構造劣化による性能低下を実際にクエリを実行せずに構造劣化量を推定できることが望ましい．そのため，我々は DB におけるデータ配置から直接性能に影響する要因，特にストレージの性能特性に依存する構造劣化をモデル化した [12]．

当該構造劣化モデルを用いて DB を監視するためには，表空間を走査する必要があるが，DB の常時運用ニーズが高い背景から，監視に必要なリソース，特にボトルネックとなりやすいストレージへのアクセスは避けることが望ましい．そのため，我々はインクリメンタル構造劣化推定手法を提案し [13]，DB 更新時に必要な情報がメインメモリにある時に DB エンジンから抽出することで，リアルタイム監視可能なまでにオーバーヘッドを削減した．

本稿では，上記の構造劣化モデルおよび低オーバーヘッドな構造劣化監視手法を用いて，実際に構造劣化をリアルタイム監視できるオンライン構造劣化モニタを試作した．

本稿は以下のように構成される．まず，第 2 章で関連研究について述べ，次に第 3 章で構造劣化モデルおよび低オーバーヘッド構造劣化監視手法について述べる．第 4 章でオンライン構造劣化モニタの試作機について述べる．その後，第 5 章で試作したオンライン構造劣化モニタを用いて再編成管理の例について述べる．最後に第 7 章にて結論と今後の課題を述べる．

2 関連研究

DB 再編成に関する研究には，実行方式の高度化及び実行スケジューリングの最適化を中心に行われてきた．前者については，近年オンライン再編成方式についての研究が行われてきている．[3, 11]．後者については，再編成スケジュールの最適化に関する研究が行われてきた [8, 2]．再編成契機判断の自立化は，後者に属する．これらの研究では，表空間を構成するストレージの性能特性は考慮せず，比較的簡単な DB モデルに基づいて，近未来の性能低下量を予測し，運用時間あたりの再編成コストを最小化する再編成スケジューリング手法が考察された．

IO 性能モデルに関して，文献 [6] は，DB バッファ等の影響を考慮して，B+Tree 索引と表を走査するのに必要な IO 数をコストとしてモデル化している．この論文では，データがクラスタ化されていない状態を仮定しており，クラスタ化の度合いを構造劣化量としてモデル化する本研究とは扱う対象が異なる．

文献 [10] において，部分再編成手法が提案された．それまで再編成対象となる最小粒度は，表単位であるか，もしくは表データが配置されている表空間が分割されている場合は分割された表空間であった．この論文では，大容量の表に対して，表空間が分割されているかどうかに関係なく，構造劣化部分のみを再編成することにより，再編成時間を削減しつつ，構造劣化がほぼ回復する．本研究では，構造劣化監視においてリアルタイムに構造劣化部分の同

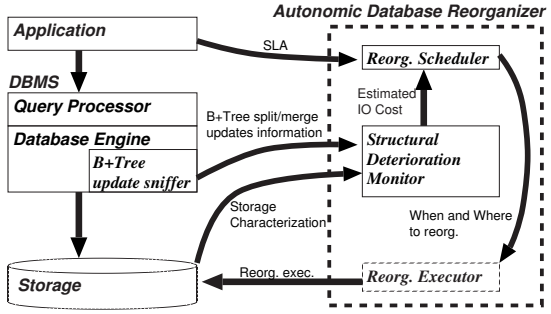


図 2: 自立再編成フレームワーク

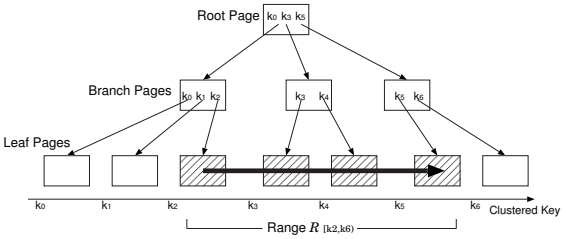


図 3: MySQL クラスタ表の B+Tree 構造

定を行うことが可能であるため，部分再編成対象の同定に貢献する．

3 構造劣化監視機構

本節では，本研究が目指す自立再編成フレームワークについて述べ，その構成の基本部分であるオンライン構造劣化モニタの試作に必要な理論的背景について，まとめる．

3.1 自立再編成フレームワーク

本研究で目指している自立再編成フレームワークの概要を図 2 に示した．

自立再編成機構(Autonomic Database Reorganizer) は，以下の 3 つの要素から構成される．

オンライン構造劣化モニタ: (Structural Deterioration Monitor) 構造劣化量を監視する．モデル化されたストレージの性能特性を用いて，構造劣化量を範囲検索 IO コストとして推定し，保持する．DB エンジンに実装された B+Tree 更新差分抽出器 (B+Tree Update Sniffer) から B+Tree 更新差分を抽出し，インクリメンタルに推定 IO コストを更新する．つまりリアルタイムに構造劣化量を把握可能である．

再編成スケジューラ: (Reorganization Scheduler) SLA[1] をユーザもしくはアプリケーションからあらかじめ取得しておき，リアルタイムにオンライン構造劣化モニタから得られた推定 IO コストと比較することで，再編成対象となる空間及び時間を判断する．

再編成エグゼキュータ: (Reorganization Executor) 再編成スケジューラから指示された空間及び時間に従って，再編成を実行する．

以上のフレームワークにおいて，SLA ポリシに基づき構造劣化を一定範囲内に留められる再編成を実施できるようになり，自立再編成機構が実現できると考える．十分な精度，粒度，低オーバーヘッドで構造劣化を監視することにより，定期的に再編成を行う運用に比べて無駄な再編成を省くことができ，再編成実施コストに対して性能改善効果の高い再編成が可能になることも期待される．

[1]Service Level Agreement．サービスレベル要求を指す．ここでは，許容できる性能低下の閾値を意味する．

表 1: 変数，定数及び関数の一覧表

変数	意味
$\mathbf{R}$	クラスタ鍵範囲
$\mathbf{S}$	クラスタ表 B+Tree 構造の葉ページ列
p	ディスク上のページアドレス
f	ページ列から IO コストを導く関数
$\mathbf{G}$	$\mathbf{S}$ に対する構造劣化分布
G_0	定数
$C_{\mathbf{R}}$	範囲 $\mathbf{R}$ の範囲検索 IO コスト

3.2 オンライン構造劣化モニタ

本節では，構造劣化モニタの動作原理である，構造劣化のモデルおよび，更新差分によるインクリメンタル構造劣化監視手法についてまとめる．表 1 に本節で用いる変数，定数および関数の一覧を示した．

3.3 構造劣化モデル

本稿では，DBMS において，クラスタ表を構成する B+Tree 構造を対象に議論を進める．このような構造は MySQL[7] のクラスタ表や，Oracle の索引構成表 [1] で使われている．

クラスタ表の B+Tree は，枝及び根ページにはクラスタ鍵 (Clustered Key) と下位ページへのポインタが，葉ページには，データ行が格納されている．(図 3) 葉ページは，論理空間ではクラスタ鍵順に並んでいる．この葉ページ列を $\mathbf{S} = p_0, p_1, ..., p_{N-1}$ ，$|\mathbf{S}| = N$ とする．p_i はディスクドライブ上のページアドレスであり，p_i に含まれるデータ行のクラスタ鍵の最大値は，p_{i+1} に含まれる鍵の最小値以下である (正順の場合)．

範囲検索は，対象となる範囲 (例えば，図 3 における範囲 $\mathbf{R}$) に属する葉ページ列，即ち $\mathbf{S}$ における連続部分ページ列を読み込む走査に相当する．一般的には，B+Tree における葉ページの論理順序とストレージ内の物理順序が対応し，シーケンシャルアクセスとなることが期待されているが，DB 更新によって，B+Tree の構造が変化すると，各ページの物理的な配置が乱雑化する．このため，範囲検索のディスクアクセスは非シーケンシャル化し，性能が大幅に低下する．範囲検索は，そのページ走査の中で，同じページを一度しか読み込まないため，DB のバッファキャッシュヒットを期待し辛く，一般的に IO ボトルネックになる．このとき，範囲検索クエリの応答時間は，対象としている葉ページ列の走査 IO 時間がそのほとんどを占める．そのため，対象範囲 $\mathbf{R}$ に相当する $\mathbf{S}$ の部分列の走査 IO コスト $C_{\mathbf{R}}$ は，範囲検索性能として近似できる．$C_{\mathbf{R}}$ は本稿で対象にする構造劣化をよく表しているため，$C_{\mathbf{R}}$ を範囲 $\mathbf{R}$ の構造劣化量として定義する．

各葉ページが読み込まれるときの IO コストはそれぞれ，読み込むべきページのアドレス p_i 及び，その直前に読み込んだページのアドレス p_{i-1} の差 $p_i - p_{i-1}$ から推定できる．これより，ページ p_i を読み込む IO コスト $G[i]$ は，ストレージ性能特性に基づいた関数 f を用いて，式 (1) で与える．

$$G[i] = \begin{cases} G_0, & i = 0 \\ f(p_i - p_{i-1}), & i > 0 \end{cases} \quad (1)$$

関数 f の詳細については，文献 [13] で詳しく述べている．G_0 は，ディスクドライブからのシーケンシャル読み込みにおけるページの読み込み IO コストを表す．ディスクドライブにおいては，$G[i]$ が取りうる値の中で G_0 が最小値である．

範囲検索において，最初に読み込まれる葉ページの IO コストは，当該モデルに基いた場合でも，直前に読み込まれるページが分からないため，推定不可能である．本稿では，対象とする範囲が十分大きい場合におけるモデル化を行うため，便宜上 $G[0] = G_0$ と置いた．

$G[i]$ の列は，$\mathbf{G}$ と表す．$\mathbf{G}$ は $\mathbf{S}$ に対する構造劣化分布を表す．

範囲 $\mathbf{R}$ に対する範囲検索 IO コスト $C_{\mathbf{R}}$ は，

$$C_{\mathbf{R}} = \sum_{i|p_i \in \mathbf{S}} G[i] \quad (2)$$

として表す．

範囲検索においては，B+Tree 構造の根及び枝ページも範囲を決定する過程及び，対象葉ページの位置を DB エンジンが知るために読み込まれるが，葉ページの数に対して大変少なく，誤差に含まれるため，本モデルにおいて考慮しない．

表 2: B+Tree 構造変化における $C_{\mathbf{R}}$ の差分

	ページ追加	ページ削除
左端	$+f(p_0 - q)$	$-f(p_1 - p_0)$
右端	$+f(q - p_{\|\mathbf{S}\|-1})$	$-f(p_{\|\mathbf{S}\|-1} - p_{\|\mathbf{S}\|-2})$
その他の位置 i_0	$-f(p_{i_0} - pi_0 - 1)$ $+f(q - p_{i_0-1})$ $+f(p_{i_0} - q)$	$-f(p_{i_0} - p_{i_0-1})$ $-f(p_{i_0+1} - p_{i_0})$ $+f(p_{i_0+1} - p_{i_0-1})$

3.4 インクリメンタル構造劣化推定

DB 更新により B+Tree の構造変化が起こったとき，それはページの分割もしくは結合である．そのときの $\mathbf{S}$ の変化に注目すると，ページが追加されるか，削除されるかのどちらかである．それぞれ境界での動作を含めて，構造劣化量 $C_{\mathbf{R}}$ の差分は，表 2 に示した 6 つの場合に分けられる．これにより，DB 更新に対して，インクリメンタルに $\mathbf{S}$，$\mathbf{G}$，$C_{\mathbf{R}}$ を保持することが出来る．とりわけ，全範囲を対象に $\mathbf{G}$ を保持しておくと，式 (2) を用いて任意の範囲 $\mathbf{R}$ の IO コスト $C_{\mathbf{R}}$ を推定可能である．

先述した構造劣化モデルは，ストレージにおけるデータ配置から直接構造劣化量を推定するため，単

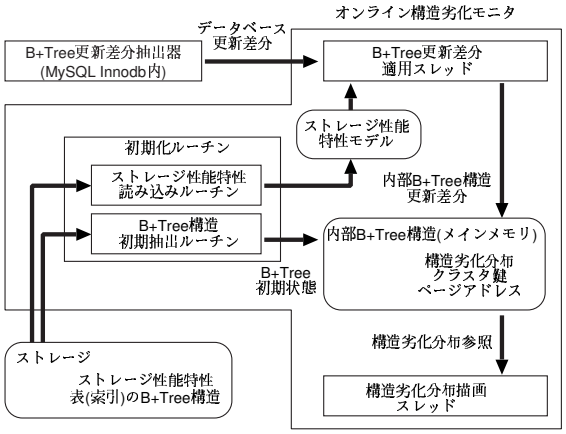


図 4: オンライン構造劣化モニタ試作機の構成

純に推定を行おうとすると表の構造を把握するためにストレージ空間を走査する必要が生じるが，DB 運用中のストレージ走査は，トランザクションの性能に多大な影響を与えることが懸念される．そのため，ボトルネックとなりやすいストレージから情報を得るのではなく，更新ページを DB エンジンで処理している間に，メインメモリから直接必要な情報を抜き出すことで，オーバーヘッドを小さくしながら，更新差分を抽出することが可能になる．この技術を用いることで，これまでにない精度，粒度でリアルタイムに構造劣化を監視できるようになる．

4 オンライン構造劣化モニタ試作機

前節に述べた，B+Tree 更新差分抽出器，及びオンライン構造劣化モニタを試作した．試作機は，MySQL 5.0 Innodb データベースにおける，クラスタ表のクラスタ鍵および二次索引の主鍵を論理軸として，構造劣化分布をリアルタイム取得，描画することが可能である．

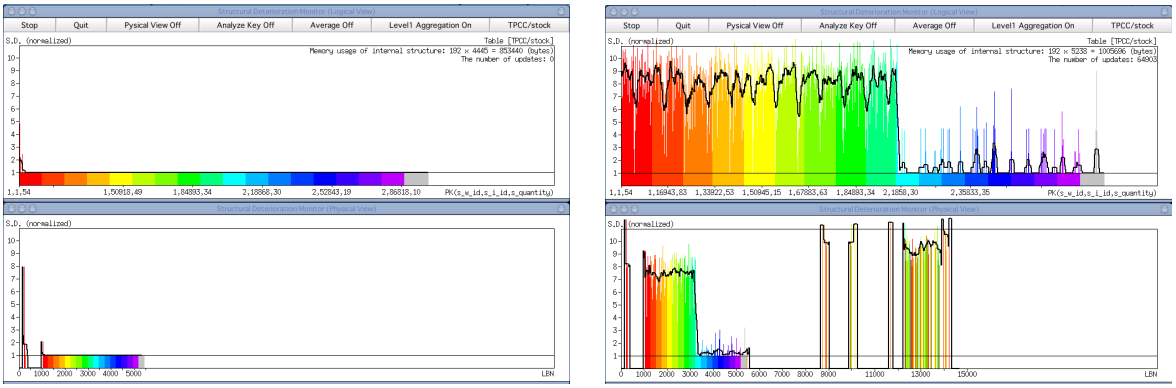
試作機の構成を図 4 に示す．試作機は内部データとしてメモリ上に内部 B+Tree 構造およびストレージ性能特性モデルを持ち，プログラムは主に初期化ルーチン，B+Tree 更新差分適用スレッド，構造劣化分布描画スレッドから構成される．

試作機を起動すると，初期化ルーチンにより，DB のディクショナリヘッダを読み込み，存在する表および索引のスキーマ，を認識する．その後，監視対象の表 (索引) を構成する B+Tree データ構造を読み込み，起動時における構造劣化分布構造をメインメモリ内に作成する．その後，B+Tree 更新差分適用スレッドが起動し，更新差分抽出器から DB 更新差分を取得するためのキューを作成し，更新差分読み込み待ちに入る．それとは平行して，構造劣化分布描画スレッドが起動し，DB 更新差分によって構造劣化分布が更新される度に，もしくは定期的に描画を行う．試作機において，B+Tree 更新差分適用スレッドおよび構造劣分布描画スレッドは並列に動作し，内部 B+Tree 構造を更新，参照適宜排他をとって更新，参照する．以下，各コンポーネントの詳細について述べる．

B+Tree 更新差分抽出器: MySQL 5.0 Innodb の DB エンジンのソースコードを改変する形で実装した．DB 運用時において，データの挿入，削除，更新等により，表 (索引) を構成する B+Tree 構造に変化があったとき，即ち，ページが追加及び削除された場合に，それらの処理を行うルーチン中で，対象ページ及びその論理空間における左右のページのクラスタ鍵値，ページアドレス，表 (索引) 番号を取得し，オンライン構造劣化モニタから取得するための共有メモリに確保したキューに挿入する．このため，構造劣化監視のためのストレージへのアクセスはほぼ発生しない．

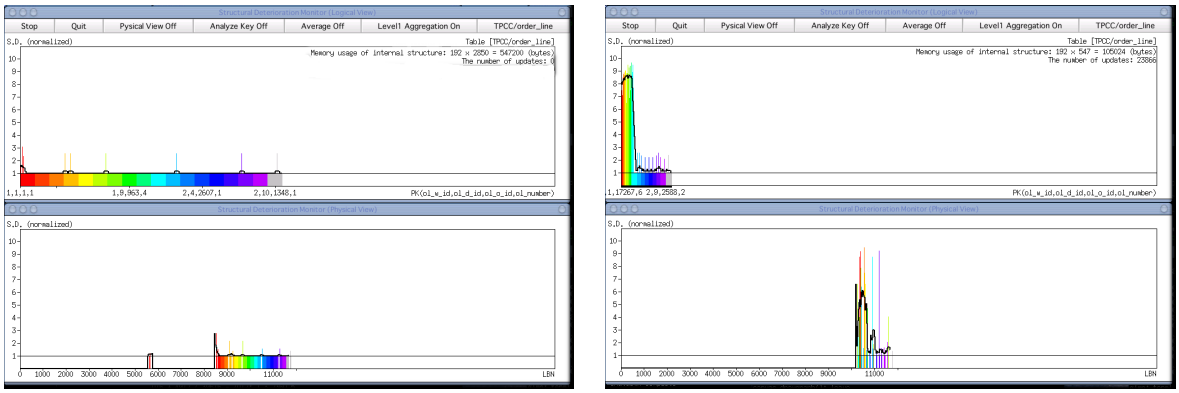
ストレージ性能特性モデル: 先述した，関数 f としてモデル化されている．現状の実装では，事前にパラメータを設定ファイルに記述する．

内部 B+Tree 構造: 試作機はメインメモリ内に実際の表を構成する B+Tree に対応する木構造を保持し，B+Tree の葉ページに相当するレベル 0 (最深部) のノードは，クラスタ鍵値，ページアドレス，構造劣化モデルから得られた IO コスト推定値を構造劣化分布として保持している．レベル 1 以上のノードは，当該ノードを頂点とする部分木における，レベル 0 のページ数，及びレベル 0 のページの IO コスト推定値の和を保持している．すなわち，レベル 1 以上のノードにおいては，当該ノードを頂点とする部分木における全てのレベル 0 ノードを含む範囲の構造劣化量の集約値を保持している．このため，広範囲における構造劣化量を高速に計算することができる．



(a) データロード直後　　(b) トランザクション発行後

図 5: TPCC ベンチマーク stock 表のオンライン構造劣化モニタによる表示



(a) データロード直後　　(b) トランザクション発行後

図 6: TPCC ベンチマーク order_line 表のオンライン構造劣化モニタによる表示

初期化ルーチン: 試作機起動時の初期化においては，既にある表の B+Tree 構造を読み込む必要があるが，現状の試作機においては，起動時における DB 状態を DB から直接読み込んでおり，MySQL サーバと通信して取得しないため，実装上の問題ではあるが，試作機が不整合な DB から情報を読み込まないように，MySQL サーバを起動する前に試作機を起動する必要がある．

B+Tree 更新差分適用スレッド: B+Tree 更新差分抽出器からキューに挿入された更新差分情報を，FIFO で取得する．その後，メモリ内の構造劣化分布構造のアップデートを先述したインクリメンタル構造劣化監視手法を用いて行う．レベル 1 以上の集約された更新差分は，レベル 0 における更新のあった全てのノードに対して，各ノードの先祖ノードすべてに対して，同様の差分計算を適用することでインクリメンタル更新が可能である．すなわち，ノード数 N に対して，木構造における $O(\log N)$ 個 (具体的には木構造の深さだけ) のノードの更新のみで，最新の状態に追随することが可能である．

構造劣化分布描画スレッド: 保持している構造劣化分布に変更があったとき，または定期的に，構造劣化分布の論理軸における描画および物理軸における描画を行う．論理軸とは，クラスタ鍵に対応する B+Tree の葉ページ列の順序である。物理軸とは，ストレージ上の位置，すなわち葉ページにおけるページアドレスである．これらを横軸にし，縦軸は構造劣化量 (S.D) を表示する．現状では，構造劣化がない状態における IO コスト、即ち G_0 を基準に正規化して表示する．描画の粒度が小さく、構造劣化分布の量的分布が目視では分かりにくいため，移動平均値を表示して，大きな視点から構造劣化を観察できるようになっている。また、表示においては、論理軸と物理軸のデータの分布の対応を

より分かりやすくするため、クラスタ鍵空間を18区間に分けて色分け表示している。

5 オンライン構造劣化モニタを用いた再編成管理例

本節では，オンライン構造劣化モニタ試作機を用いて再編成管理を行う事例を，オンライントランザクションベンチマーク TPC-C[9] ベースのベンチマークを用いて示す．

実験環境として，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ドライブ 1 台を raw デバイスとして使用した．ディスクドライブは，Cheetah 10K 18GB [5] 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，実測で 28MB/s であることを確認した．この値をページサイズで割った数値の逆数を最小ページ読み込み IO コスト G_0 として用いた．

上記の実験環境にて，MySQL 5.0 InnoDB データベースエンジンを用いて 2GB の表空間領域を確保した．ページサイズは 16KB とした．

上記の表空間にクラスタ表を用いて，TPC-C スキーマを作成し，初期データをロードした．全ウェアハウス数は 2 とし，主に更新されるアクティブウェアハウスは 1 つのみとした．約 200MB のデータが初期データセットとしてロードされる．

TPC-C はオンライン商取引 DB を模擬したベンチマークであり，注文 (new-order)，支払 (payment)，配達 (delivery)，注文確認 (order-status)，在庫確認 (stock-level) の計 5 トランザクションが定義されている．これらのトランザクションによって各表は更新され，構造劣化が進む．純正 TPC-C は更新によって，`Order`，`Order-Line`，`History` の各表のデータが単調増加するが，実運用システムでの長期的運用を考慮し，配達が完了した注文に関するデータを他の DB に移動するバッチ処理 move-old-orders (`Order` 表及び `Order-Line` 表からデータを移動) 及び，古い履歴を移動するバッチ処理 move-old-history(`History` 表から古いデータを移動) を追加した．これら計 7 トランザクションを，20 プロセスから並列に 1 時間実行した．各プロセスにおいて，7 トランザクションを new-order: 45%，payment: 45%，delivery: 6%，order-status: 1%，stock-level: 1%，move-old-orders: 1%，move-old-history: 1%の確率でランダムに選択し，トランザクション間の思考時間 0 にて投入した．本実験におけるトランザクションの発行率は，DB の大きさから考えると一般的な商取引 DB としては極度に多く，実際には数日〜数週間で投入されるトランザクションを想定している．

トランザクション投入後，試作機において顕著に構造劣化が確認された `Stock`，`Order-Line` 各表について，データロード直後及び更新後の DB において，構造劣化モニタの画面を図 5(a)(b)，6(a)(b) に示す．図中において，上図が論理軸 (クラスタ鍵)，下図が物理軸 (LBN [2]) における構造劣化分布 (S.D.) 表示である．

図 5(a) の `Stock` 表において、更新前に構造劣化はほぼない。即ち S.D. が 1 を指している。物理軸でも論理軸における分布とほぼ一対一対応をしており、範囲検索においてストレージのシーケンシャルアクセスを期待できることが確認できる。トランザクション発行後、図 5(b) においてウェアハウス ID (`s_w_id`) がクラスタ鍵の最初の属性であるため、アクティブウェアハウスであるウェアハウス 1 の部分のみ構造劣化が主に起きており，モニタ画面の目視のみで 7〜9 倍の構造劣化量が確認できる。また、トランザクション発行後に物理軸において、ウェアハウス 1 のデータ分布が、元あった位置以外に、LBN 9000〜14000 付近に点在していることが確認できる。このことから、ウェアハウス 1 のデータが、ストレージ上でクラスタ鍵順にもはや連続して配置されていないことが推察でき、構造劣化が進んでいる事実を確認できる。

同様に、図 6(a)(b) において、`Order-Line` 表もクラスタ鍵の最初の属性 (`ol_w_id`) がウェアハウス ID を示しており、ウェアハウス 1 が構造劣化していることが確認できる。トランザクション発行後にデータ量が少なくなっているのは、トランザクション `move_old_history` によって、配達済みの注文が削除されているからである。物理軸を見ると、`Stock` 表に比べて、データの存在位置は大局的にはあまり移動していないように見えるが、構造劣化 8〜9 倍になっていることから、小さい粒度ではページの配置が乱れていると推察される。

このような状況において、以下の 2 つのバッチ処理を考える。

Promising-Sales: ウェアハウスにおける配達待ち

[2]Logical Block Number。ストレージ上のページアドレス

の注文による売上期待額の合計を算出する。このバッチ処理は、Order-Line 表の範囲検索を伴う。本モニタで示される構造劣化推定量は、範囲検索の IO コストを推定しているため、本バッチ処理の応答時間の精度が高い推定値として使用可能である。図 6 から応答時間が 8〜9 倍になる恐れがある。

Stock-Quantity: ウェアハウスにおける在庫数を算出する。このバッチ処理は、Stock 表の範囲検索を伴う。Promising-Sales と同様に、本実験におけるトランザクション発行後はウェアハウス 1 において図 5 で応答時間が 7〜9 倍になる恐れがある。

本例において，上記 2 つのクエリの性能低下は，通常の運用においては許容されない場合が多い。オンライン構造劣化モニタを用いて DB のリアルタイム監視を行うことで，管理者が性能要求をもはや満たさない程に構造劣化が進んでいることを判断することが出来る。その際，範囲検索を主とするクエリの性能に比較的近い値として構造劣化量を定義しているため、再編成判断が容易になることが期待される。また、構造劣化が起きている空間が本モニタによって常に示されているため、過不足なく再編成すべき対象を決定することができる。

6 おわりに

本稿は、ストレージ性能特性を考慮した構造劣化モデルおよびインクリメンタル構造劣化推定手法によるオーバーヘッドの小さいオンラインデータベース構造劣化モニタを試作した．その後，実際にオンライントランザクションベンチマークを用いて試作機による再編成管理例を示した．

今後も自立再編成機構の実現を目指すべく，研究を進めたい．

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

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

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

[10] 合田 和生，喜連川 優．構造劣化の局所性を活かしたデータベース部分再編成の提案．In **電子情報通信学会 第** *17***回データ工学ワークショップ** *(DEWS2006)*, 2006.

[11] 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(SIG 8(TOD 26)):pp.130–147, 2005.

[12] 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.

[13] 星野喬，合田和生，喜連川優．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．**日本データベース学会論文誌** *(DBSJ Letters)*, 4(2):37–40, 2005.