関係データベースにおける構造劣化監視機構を用いた再編成スケジューラの提案

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 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，構造劣化している時間と空間を同定する必要がある．近年，データベースの大規模化による管理コスト増大，常時運用ニーズ増大などの背景があり，再編成のための監視，実施リソースが減り，従来行われていた管理者の経験則による判断がますます困難になりつつある．そのため，システム自身が自立的に再編成すべき時間，空間を判断する必要が出てきた．本研究では，データベース再編成の自立化を目的として，低オーバーヘッドなリアルタイム構造劣化モニタ及び，SLA ポリシーに基づいた再編成スケジューリングフレームワークを提案する．評価実験により，提案手法の有効性を明らかにする．

キーワード 関係データベース、自立管理、再編成スケジューラ、構造劣化

A Reorganization Scheduler on RDBMS Using Structural Deterioration Monitor

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 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 when the database reorganization should be invoked, and what region of the database should be reorganized. Nowdays, database size increases, database administration becomes costful, and full-time database operation is required, then system resources which can be spent to monitor and maintain the database decrease, and administrator's heuristic reorganization shceduling becomes more difficult. In this paper, we propose a structural deterioration monitor with a little overhead and a framework of reorganization schedule based on SLA policies and evaluate them.

Key words Relational Database Systems, Autonomic Database Administration, Reorganization Scheduler, Structural Deterioration

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) 管理はより困難化し，コスト増大が問題となっている．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なってきている．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きている．これらの背景により，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 [3].

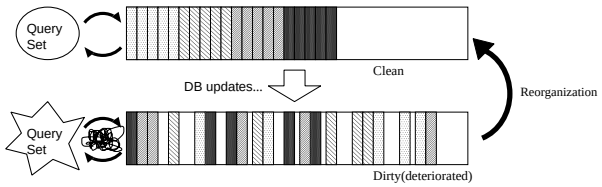

図 1　構造劣化と再編成

我々は DB 再編成管理に着目し，その自立化を目的としている．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．

これまで，DBMS における再編成では，DB 管理者が DBMS から得られる性能などの統計情報や，空間情報などから構造の劣化度合を推定し，再編成が必要であるかどうかを判断し，運用に合わせて再編成処理をスケジューリングする方法が一般的であった．これらの情報に基づく判断は，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，具体的に体系化されたものではない．結果として，DB 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が多い．しかし，DB 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る構造の劣化度合を越え，再編成スケジューリングの変更が必要になる場合が想定される．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，コストも高くなる．

DB 再編成契機の判断を明確に行うためには，構造劣化を定量的に表現し，常に把握できるようにすることが必要である．観測される性能のみを判断に用いた場合，構造劣化以外の原因との切り分けがしにくく，再編成契機の判断が難しい．また，バッチ処理などの時間のかかる処理の場合，実際にクエリを実行しないで構造劣化量を把握できることが望ましい．

そこで，我々は構造劣化量を直接性能指標として表現することを試みた．DBMS のクエリ処理における構造劣化は，以下の 2 つの視点から見ることができる．

空間効率の低下:　DBMS におけるデータ格納構造における空間効率の低下が，構造劣化となる．同じデータに対して，充填率が下がる等の理由によって空間効率が低下した場合，クエリ発行時に IO 数が増加し，性能が低下する．

IO 効率の低下:　B+Tree のような DBMS のデータ格納構造において，ストレージ上のデータ配置によって，IO 効率が異なり，同じデータ，クエリに対して性能が異なる場合がある．つまり，典型的なアクセスに対して IO 効率が低下することが，構造劣化となる．今日のハードディスクドライブを用いたストレージでは，非シーケンシャルアクセスに比べてシーケンシャルアクセスが著しく高速である．そのため，データが読み込まれる順番にストレージ上に配置されている場合，シーケンシャルアクセスが期待でき理想的であるが，データ更新によってデータ読み込み IO が非シーケンシャルアクセスにならざるをえないデータ配置になり，データ読み込み時の IO 効率が低下した場合，クエリ性能が低下する．

一般的に空間効率の低下は DBMS 内で把握することが可能であるが，IO 効率の低下は，DBMS 内の指標だけでなく，ストレージの構成及び性能特性に依存するため，より把握が困難である．正確な構造劣化同定のためには，性能に直接関係する，IO 効率を含めた視点から構造劣化を捉える必要がある．我々は，ストレージにおける性能特性を構造劣化量計算に用いることで，正確に構造劣化量を推定する手法を提案した [12], [13]．

DBMS を含む現状の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ストレージ IO に起因することが多い．そこで，これらのアプリケーションを対象に，構造劣化の度合を，DB データのストレージにおける配置の状態から期待される IO コストとして表現する．我々は，ストレージ内の IO 性能特性を考慮した上で，B+Tree 実装されている MySQL [8] クラスタ表の範囲検索を対象に，実際に範囲検索クエリを実行した場合に期待される IO コストを高精度かつ低オーバーヘッドでリアルタイムに推定する手法を構築し，構造劣化モニタを提案し，それを用いた再編成スケジューラを提案する．再編成スケジューラは，SLA ポリシーに基づく再編成実施判断基準を，構造劣化モニタによって得られた構造劣化量と比較し，再編成すべき時間と空間を出力する．

提案手法は，これまでの管理者による経験則に頼らない，より正確な構造劣化量同定による再編成判断が可能になり，定期的に再編成を行っていた場合に比べて余分な再編成の回数を減らすことが可能なため，再編成実施判断の自立化に大きく貢献すると期待される．

本論文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 章で自立再編成フレームワークについて述べる．第 4 章で構造劣化モニタについて述べ，第 5 章で再編成スケジューラについて述べる．その後，第 6 章で提案手法の評価を行い，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．[2], [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 [1], [5], [7], [9], [10]．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 モデルに基づいて再編成スケジューリング手法が考察された．

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．

著者らは，文献 [12], [13] において，DB の構造劣化の表現手法と，その定量的推定手法を提案し，インクリメンタルな構造劣化監視手法について提案した．本論文では，それらのモデ

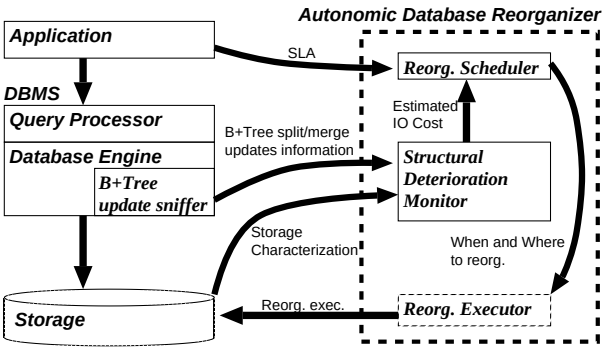

図 2　自立再編成フレームワーク図

ルを定式化し，高精度かつ低オーバーヘッドなリアルタイム構造劣化モニタと共に，それを用いた再編成スケジューラを提案する．

本手法では，これまであまり DBMS が考慮していなかったストレージにおける IO 効率低下という構造劣化を扱えるため，より正確な構造劣化量推定が可能になり，より高いレベルでのポリシー記述言語を今後適用することが可能であると期待される．我々は，システムの自立管理を実現するためには，システムにおける一番下位層の振る舞いをまず対象にし，ボトムアップ的な情報収集を行い，トップダウン的な管理を行うべきだと考えており，本論文では，限定的ながらも，そのアプローチに基づいた基本的なフレームワークを提案するものである．

3. 自立再編成フレームワーク

本論文では，DB の構造劣化量を同定し，再編成スケジュールを決定するために，構造劣化モニタ及び，再編成スケジューラを提案する．それらの機構を含む，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 つから構成される．

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(注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より，SLA ポリシー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，

(注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．

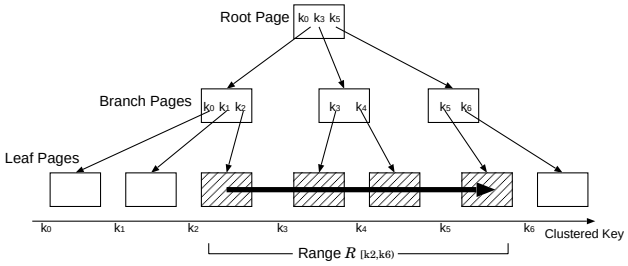

図 3　MySQL クラスタ表の B+Tree 構造

自立再編成機構が実現できる．提案手法によって，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，再編成回数の削減も期待できる．また，再編成空間を細かい粒度で同定することにより，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．

本論文では，構造劣化モニタ及び再編成スケジューラについて述べる．再編成エグゼキュータの詳細については議論しない．

4. 構造劣化モニタ

我々は，文献 [13] において，ディスクドライブの特徴に着目した静的な IO 性能モデルを構築し，当該モデルを用いて，範囲検索 IO コスト推定手法を構築した．また，DB 更新差分を用いてインクリメンタルに推定 IO コストの更新を行う手法を構築した．本章では，それらを用いて構造劣化の定式化を行い，再編成スケジューラで使えるように拡張し，構造劣化モニタを構築した．

本論文では，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 DBMS には MySQL などが挙げられる．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．範囲検索は，対象となる範囲 (例えば，図 3 における範囲 $\mathbf{R}$) に属する葉ページ群を，クラスタ鍵順に読み込む走査を伴う．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，更新によって，B+Tree の構造が変化する．即ち，葉ページの結合，分割が起きると，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．本論文では上記の構造劣化を議論の対象とする．

表 1 に本章および次章で用いる変数，定数および関数の一覧を示した．

まず，構造劣化モデルについて述べ，その後，監視する構造劣化を DB 更新差分を用いてインクリメンタルに更新する手法について述べる．

4.1 構造劣化モデル

まず，クラスタ表における範囲検索の IO コスト推定値を算出するモデルについて述べる．

クラスタ表の B+Tree 構造とそれに対する範囲検索クエリが

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
$\mathbf{S}$	$\mathbf{R}$ に対応する葉ページ列
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
X	許容構造劣化度
$\mathbf{T}$	再編成対象となる $\mathbf{S}$ の部分集合

あったとき，読み込むクラスタ鍵範囲が $\mathbf{R} = \{k | k_0 \leqq k < k_1\}$ として与えられるものとする．k はクラスタ鍵である．$\mathbf{R}$ に対応する行を読み込む場合は，DBMS 内で B+Tree 構造における索引機構により，$\mathbf{R}$ に対応する葉ページの列として読み込まれる．この葉ページ列を $\mathbf{S} = \{p_0, p_1, p_2, ..., p_{N-1}\}$ とする．また，を $\mathbf{S}$ の大きさを，葉ページ数 N を用いて，$|\mathbf{S}| = N$ とする．i は葉ページをクラスタ鍵順に並べた場合の順序，つまり論理空間における葉ページの順序を示しており，範囲検索においてはこの順序で葉ページが読み込まれる．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．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，バッファキャッシュしづらく，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 $\mathbf{R}$ に相当する葉ページ列 $\mathbf{S}$ の走査 IO コストを範囲検索性能と近似してモデル化した．

このとき，各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 $p_i - p_{i-1}$ に依存する．これを関数 f で表す．関数 f の詳細については，文献 [13] で詳しく述べている．

上記より，ページ i を読み込む IO コスト $G[i]$ は，関数 f を用いて，以下のように与えられる．

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

ただし，G_0 は定数とする．

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

$$C_{\mathbf{R}} = \sum_{i=0}^{|\mathbf{S}|-1} G[i] \qquad (1)$$

となる．

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．実際，関数 f を導出したディスクモデルに従うと，ページ 0 を読み込む直前に読み込んだページのアドレスが必要になるが，これは実際に範囲検索を行わない限り事前には分からないため，

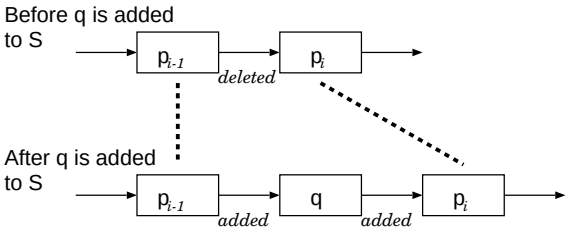

図 4　$\mathbf{S}$ の差分更新図: ページ追加時

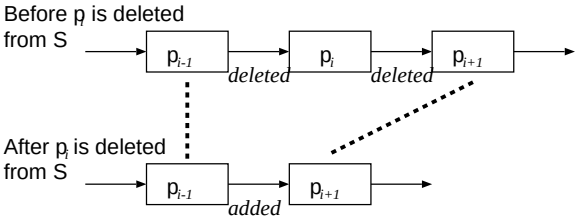

図 5　$\mathbf{S}$ の差分更新図: ページ削除時

$G[0]$ は不定である．ここでは，$G[0]$ の値が $C_{\mathbf{R}}$ の中で誤差に含まれるほど $|\mathbf{S}|$ を大きくとるものと仮定し，便宜上 $G[0] = G_0$ とする．$G[i]$ の列は，$\mathbf{G}$ と表し，これは葉ページ列 $\mathbf{S}$ に対する構造劣化分布であるといえる．

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．

4.2　構造劣化量のインクリメンタル更新

$\mathbf{G}$ 及び $C_{\mathbf{R}}$ を，DB 更新差分を用いてインクリメンタルに更新できるモデルについて，以下に述べる．

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，以下の 6 つの場合に分けられる．

（1）　ページ 0(左端) の左側にページアドレス q のページを追加

（2）　ページ $i_0 (0 < i_0 < |\mathbf{S}| - 1)$ の左側にページアドレス q のページを追加

（3）　ページ $|\mathbf{S}| - 1$(右端) の右側にページアドレス q のページを追加

（4）　ページ 0(左端) を削除

（5）　ページ $i_0 (0 < i_0 < |\mathbf{S}| - 1)$ を削除

（6）　ページ $|\mathbf{S}| - 1$(右端) を削除

$\mathbf{S}$ の境界以外での構造変化について，図 4 に追加した場合の，図 5 に削除した場合の $\mathbf{S}$ の変化の概要を示した．

DB 更新によって，$\mathbf{S} \to \mathbf{S}'$，$\mathbf{G} \to \mathbf{G}'$，$C_{\mathbf{R}} \to C'_{\mathbf{R}}$ と変化するものとする．このとき，$\mathbf{S}'$，$\mathbf{G}'$，及び $C'_{\mathbf{R}}$ は，以下の式で表現できる．

（1）　追加:ページ 0 の左隣 (左端に追加)

$$\mathbf{S}' = \{q, p_0, p_1, ...\}$$

$$G'[i] = \begin{cases} G_0, & i = 0 \\ f(p_0 - q), & i = 1 \\ G[i-1], & 2 \leqq i \leqq |\mathbf{S}| \end{cases}$$

$$C'_{\mathbf{R}} \quad = \quad C_{\mathbf{R}} + f(p_0 - q)$$

（2） 追加: ページ i_0 の左隣 $(0 < i_0 < |\mathbf{S}| - 1)$

$$\mathbf{S}' = \{..., p_{i_0-1}, q, p_{i_0}, ...\}$$

$$G'[i] = \begin{cases} G[i], & i \leqq i_0 - 1 \\ f(q - p_{i_0-1}), & i = i_0 \\ f(p_{i_0} - q), & i = i_0 + 1 \\ G[i-1], & i_0 + 2 \leqq i \leqq |\mathbf{S}| \end{cases}$$

$$\begin{aligned} C'_{\mathbf{R}} \quad = \quad & C_{\mathbf{R}} - f(p_{i_0} - p_{i_0-1}) \\ & + f(q - p_{i_0-1}) + f(p_{i_0} - q) \end{aligned}$$

（3） 追加: ページ $|\mathbf{S}| - 1$ の右隣 (右端に追加)

$$\mathbf{S}' = \{..., p_{|\mathbf{S}|-1}, q\}$$

$$G'[i] = \begin{cases} G[i], & 0 \leqq i \leqq |\mathbf{S}| - 1 \\ f(q - p_{|\mathbf{S}|-1}), & i = |\mathbf{S}| \end{cases}$$

$$C'_{\mathbf{R}} \quad = \quad C_{\mathbf{R}} + f(q - p_{|\mathbf{S}|-1})$$

（4） 削除: ページ 0(左端)

$$\mathbf{S}' = \{p_1, ...\}$$

$$G'[i] = \begin{cases} G_0, & i = 0 \\ G[i+1], & 1 \leqq i \leqq |\mathbf{S}| - 2 \end{cases}$$

$$C'_{\mathbf{R}} \quad = \quad C_{\mathbf{R}} - f(p_1 - p_0)$$

（5） 削除: ページ i_0 $(0 < i_0 < |\mathbf{S} - 1|)$

$$\mathbf{S}' = \{..., p_{i_0-1}, p_{i_0+1}, ...\}$$

$$G'[i] = \begin{cases} G[i], & i \leqq i_0 - 1 \\ f(p_{i_0+1} - p_{i_0-1}), & i = i_0 \\ G[i+1], & i_0 + 1 \leqq i \leqq |\mathbf{S}| - 2 \end{cases}$$

$$\begin{aligned} C'_{\mathbf{R}} \quad = \quad & C_{\mathbf{R}} - f(p_{i_0} - p_{i_0-1}) - f(p_{i_0+1} - p_{i_0}) \\ & + f(p_{i_0+1} - p_{i_0-1}) \end{aligned}$$

（6） 削除: ページ $|\mathbf{S}| - 1$(右端)

$$\mathbf{S}' = \{..., p_{|\mathbf{S}|-2}\}$$

$$G'[i] = \{ \; G[i], \quad 0 \leqq i \leqq |\mathbf{S}| - 2$$

$$C'_{\mathbf{R}} \quad = \quad C_{\mathbf{R}} - f(p_{|\mathbf{S}|-1} - p_{|\mathbf{S}|-2})$$

以上で述べたページ追加，削除時の $C_{\mathbf{R}}$ の差分を表 2 にまとめた．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 (1) を用いて任意の範囲 $\mathbf{R}$ の IO コスト $C_{\mathbf{R}}$ が得られる．これで，リアルタイムに構造劣化を監視できる．

表 2　ページ追加，削除時の $C_{\mathbf{R}}$ 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	$-f(p_{i_0} - pi_0 - 1)$ $+f(q - p_{i_0-1})$ $+f(p_{i_0} - q)$	$-f(p_{i_0} - p_{i_0-1})$ $-f(p_{i_0+1} - p_{i_0})$ $+f(p_{i_0+1} - p_{i_0-1})$

5. 再編成スケジューラ

再編成スケジューラが行うべき再編成実施判断は，SLA を満たすために，再編成をすべき領域 $\mathbf{T}$ を求める問題に帰着できる．

前章で，構造劣化分布 $\mathbf{G}$ 及び構造劣化量 $C_{\mathbf{R}}$ をモデル化し，それらが DB 更新による構造変化差分からインクリメンタルに計算できることを示したため，リアルタイムに再編成すべき領域を判断することができるようになる．

まず，SLA ポリシーのインターフェースについて述べ，その後，SLA に基づいた再編成領域同定手法について述べる．

5.1 SLA ポリシーインターフェース

SLA は，ユーザもしくはアプリケーションから与えられるシステムが満たすべきサービス要件のことである．構造劣化モニタが対象にしている構造劣化が性能に及ぼす影響を定量的に把握できるようになったことで，ユーザ及びアプリケーションにとって分かりやすい基準である性能指標を用いて SLA を定義し，それに基づいて再編成管理を自立化することが可能になる．

構造劣化量の許容基準として，構造劣化していない場合に期待される範囲検索クエリの応答時間に対する相対比として，許容構造劣化度 X が，ユーザもしくはアプリケーションから与えられるものとし，以下の式で SLA を表す．

$$C_{\mathbf{R}} (= \sum_{i|p_i \in \mathbf{S}} G[i]) < G_0 |\mathbf{S}| X \qquad (2)$$

式 (2) の右辺は，構造劣化がない場合の範囲検索 IO コスト $G_0|\mathbf{S}|$，すなわちシーケンシャルアクセスでページを読み込めた場合の IO コストに対して X 倍の構造劣化量を閾値とし，左辺の範囲 $\mathbf{R}$ に対する範囲検索 IO コスト ($\mathbf{S}$ の読み込み IO コスト) がその閾値を越えないことを意味する．

より具体的に SLA を与えるインターフェースとして，以下の 2 つの場合が考えられる．

SLA(1): ユーザによって与えられた範囲 $\mathbf{R}$ に対する範囲検索 IO コストを $G_0|\mathbf{S}|X$ 未満に保つ．

SLA(2): 任意の範囲 $\mathbf{R}$ に対して，範囲検索 IO コストを，$G_0|\mathbf{S}|X$ 未満に保つ．

SLA(1) では，ユーザが明示的に範囲 $\mathbf{R}$ とその範囲検索 IO コストの閾値 $(= G_0|\mathbf{S}|X)$ の組を与える．SLA(2) では，特に範囲は指定しないで許容構造劣化度 X のみを，$X > 1$ で与える．SLA(2) においては，構造劣化がない場合に期待される範囲検索 IO コストの X 倍という相対的な表現方法を用いることで，範囲の大きさによらず，一意に再編成基準となる構造劣化量の

閾値を表現できる．

これまでは，構造劣化によって範囲検索 IO コストがいつの間にか X 倍以上に悪化してしまうことに管理者は気がつかないか，時間と労力をかけて推定するしかなかったところだが，本手法により，許容構造劣化度 X を指定するだけで，構造劣化に歯止めをかけ，自立的に再編成管理をしてくれる機構が実現できる．

5.2 再編成対象領域の同定手法

本フレームワークにおいて SLA に基づき，再編成を開始する時間を決定する再編成スケジューラは，以下の動作をする．ある時刻において，SLA における式 (2) が満たされなくなった場合は，もはや IO 効率が許容範囲よりも低下していることを意味するものとし，その場合，再編成スケジューラはどの領域を再編成すれば SLA を満たせるのかを決定し，管理者に提示するか，再編成エグゼキュータに再編成実施命令を出す．

まず，前提となる再編成操作について述べる．葉ページ列 $\mathbf{S}$ に対して部分集合 $\mathbf{T} \subset \mathbf{S}$ が再編成対象として与えられたとき，再編成操作によって，再編成後の $\mathbf{S}''$ が得られるものとする．このとき，$\mathbf{S}''$ からその構造劣化分布 $\mathbf{G}''$ が，

$$\mathbf{G}'' = \begin{cases} G_0, & p_i \in \mathbf{T} \\ G[i], & p_i \notin \mathbf{T} \end{cases}$$

になるものとする．本論文では，再編成実施手法及び，その結果であるページ列 $\mathbf{S}''$ については議論しない．

ここで，$\mathbf{R}$ と $\mathbf{T}$ が与えられたとき，$\mathbf{R}$ の範囲検索走査 IO コスト $C_{\mathbf{R}}$ が領域 $\mathbf{T}$ の再編成によって $C''_{\mathbf{R}}$ に改善するとして，以下の式が成り立つ．

$$C''_{\mathbf{R}} - C_{\mathbf{R}} = \sum_{i|p_i \in \mathbf{T} \cap \mathbf{S}} (G[i] - G_0) \qquad (3)$$

再編成スケジューラは，SLA を満たせるように常時，再編成をすべき領域 $\mathbf{T}$ を決定し，再編成実行器に指示する機構である．効率的な再編成スケジュールを行うため，与えられた SLA を満たしつつ，再編成対象領域の大きさ $|\mathbf{T}|$ (注2) を最小にするような領域 $\mathbf{T}_{min}$ が望ましい．

以下，SLA(1) と SLA(2) のそれぞれについて，最適な再編成対象領域 $\mathbf{T}$ の同定手法について述べる．

5.2.1 SLA(1) の場合

図 6 に，論理空間軸 i における構造劣化分布 $\mathbf{G}$ と，与えられたクラスタ鍵範囲 $\mathbf{R}$ に対応するページ列 $\mathbf{S}$ 及び，再編成対象 $\mathbf{T}_{min}$ の例を示した．ここでは，表全体に対して $\mathbf{G}$ を監視しているものとして，議論を進める．

ここで，$G[i] \geqq G_0X$ を満たす全ての p_i を集合 $\mathbf{T}$ として再編成すれば，SLA(1) を満たせることは明らかであるが，その場合 $|\mathbf{T}|$ は必ずしも最小ではない．実際は，$G''[i]$ と i 軸に囲まれた面積が G_0X と i 軸で囲まれた面積よりも小さくなるような最小の大きさの再編成対象領域 $\mathbf{T}$ が存在する．そのような領域は，$G[i]$ が大きいページ i から優先的に $\mathbf{T}$ に含めていけば，

(注2)：$\mathbf{T}$ に含まれるページ数を指す．

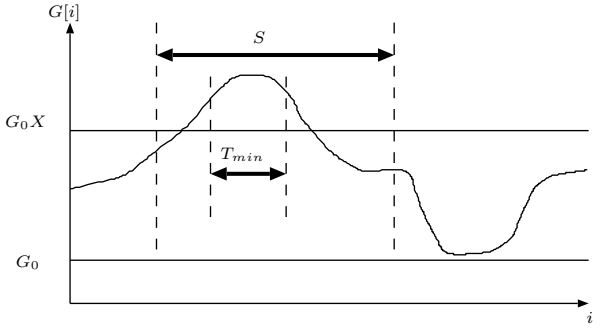

図 6　論理空間軸における構造劣化分布と再編成対象領域の例 (SLA1)

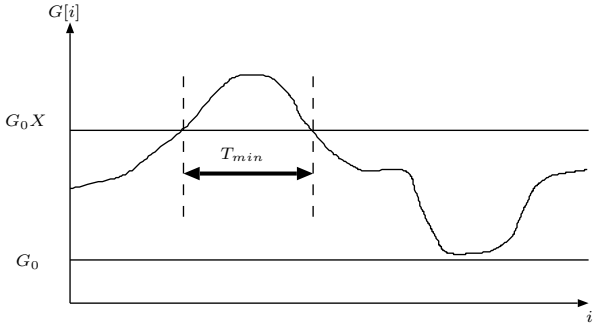

図 7　論理空間軸における構造劣化分布と再編成対象領域の例 (SLA(2))

あるところで，SLA(1) を満たすようになる．そのような $\mathbf{T}$ を $\mathbf{T}_{min}$ とすることで，再編成対象領域を決定できる．

具体的な $\mathbf{T}$ の同定手順を以下に述べる．このとき，$\mathbf{G}$ 及び $C_{\mathbf{R}}$ を構造劣化モニタによって監視している必要がある．

（1） $\mathbf{S}$ 及び $\mathbf{G}$ を，$G[i]$ が大きい順に並べ替えたものを，$\mathbf{S}'''$ 及び $\mathbf{G}'''$ とする．また，この時点での範囲検索コスト $C_{\mathbf{R}}$ を保持しておく．

（2） $\mathbf{S}'''$ を，先頭から順に再編成対象領域 $\mathbf{T}$ に加えていく．このとき，再編成によって減少する範囲検索走査 IO コスト C_{diff} は，式 (3) を用いて，$C_{diff} = \sum_{i=0}^{n'''-1} G'''[i] - G_0$ となる．ここで，n''' を 1 ずつ増やしていく．

（3） $C_{\mathbf{R}} - C_{diff} \leqq G_0X|\mathbf{S}|$ を初めて満たした n''' に対して，$\mathbf{S}'''$ の先頭から n''' ページを再編成対象領域 $\mathbf{T}$ として出力する．

再編成領域を決定するのに必要なコストのオーダーは，ソートを含んでおり，クイックソート等を用いた場合，$O(|\mathbf{S}| \log |\mathbf{S}|)$ である．

5.2.2 SLA(2) の場合

図 7 に，図 6 と同様に論理空間軸における構造劣化分布とその再編成対象領域を示した．任意の範囲 $\mathbf{R}$ に対する IO コスト $C_{\mathbf{R}}$ は，式 (1) より，$G[i]$ を，i において積分した値であると考えることができる．$\mathbf{R}$ に対応する i の範囲と，i 軸と $G[i]$ に挟まれた領域の面積が，$C_{\mathbf{R}}$ である．任意の $\mathbf{R}$ に対して式 (2) を満たすためには，図 7 の $\mathbf{T}_{min}$ のように $G[i] \geqq G_0X$ であるような領域を全て再編成対象領域にする必要がある．

具体的な $\mathbf{T}$ の同定手順は，

（1） $G[i] \geqq G_0 X$ であるような全てのページ i を，$\mathbf{T}$ として出力する．

のみである．このとき，$\mathbf{G}$ を構造劣化モニタによって監視している必要がある．再編成領域を決定するのに必要なコストのオーダーは，$O(|\mathbf{S}|)$ である．

SLA(1)，SLA(2) の両方について，$\mathbf{T}$ における論理空間 i が連続範囲でない場合に，再編成後の物理空間における位置 p_i および p_{i+1} が隣り合わない問題が発生しうる．このとき，ページ $i+1$ の読み込み IO コストは実際は，G_0 にならない．つまり，論理空間において連続したページ列を再編成したとしても，その先頭のページは，非連続になる可能性が高く，先頭ページに関しては構造劣化が是正されない．再編成操作モデルにおいては，簡単のため，対象となった領域 $\mathbf{T}$ については不連続なページであろうと IO コストが G_0 になるものとして扱ったが，論理空間上で不連続なページ群に対して再編成をしても，性能が回復しないことがあり得る．実際には，ある程度まとまった大きさの論理空間における連続領域に対して再編成を行う必要がある．このため，以上の手法を用いて実際に再編成を行う場合は，あらかじめ，$\mathbf{G}$ の平滑化処理を行い，連続領域が選択されやすいようにしておくか，再編成対象領域 $\mathbf{T}$ が得られた後に，$\mathbf{T}$ に含まれるページ i の近傍のページ (..., $i-2$, $i-1$, $i+1$, $i+2$, ...) を $\mathbf{T}$ に含めて再編成を実施する必要がある．

6. 評　　価

我々の提案する構造劣化モニタついて評価を行った．また，再編成スケジューラの手法が実際に適用可能か考察を行った．

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 [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，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．

構造劣化モニタが正常に動作することを確認するため，上記の実験環境にて，MySQL 4.1 InnoDB データベースエンジンを用いて表空間 3GB の領域を確保した．ページサイズは 16KB とし，DB バッファキャッシュを 1GB 確保した．

上記の表空間にクラスタ表 T1 を作成した．スキーマは単純に `T1(int id clustered key, int id2, char str[255])` とし，クラスタ鍵 id を，($\texttt{id}; 0 \leqq \texttt{id} < 6000000$) として，重複がないように 600 万行ロードした．1 ページあたり，約 50 行格納できる．データロード後，DB は 表空間内で 2GB 程度の大きさを占める．

上記の環境で，以下の DB 更新及び，範囲検索を実行した．

DB 更新:　クラスタ鍵に対して特定範囲を指定し，指定された範囲内でランダムにバルク行を削除，挿入する操作を，領域毎に異なる頻度で実行する．以上の操作を DB に対する更新 1 回とし，1000 回の更新を行った．1000 回後，各行が更新された回数を，横軸にクラスタ鍵値をとって図 8 に示した．鍵値の小さい領域が高い頻度で更新されるような設定になっている．

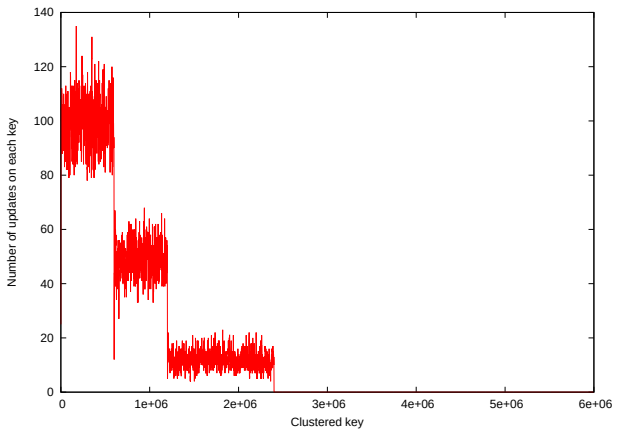

図 8　更新 1000 回後の DB 更新分布 (1000 クラスタ鍵毎に平滑化)

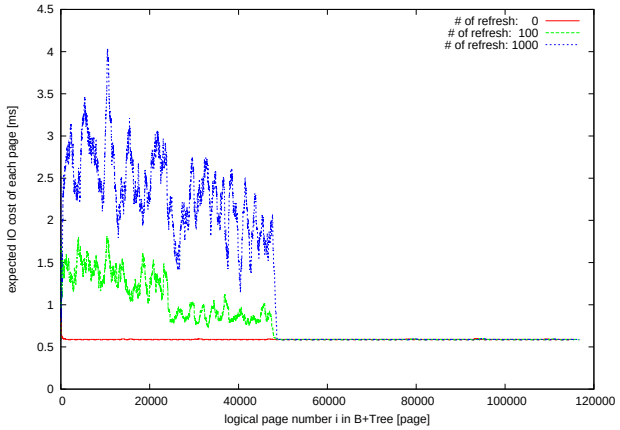

図 9　論理空間軸 i における，構造劣化分布 $G[i]$(i 方向に 1000 ページ毎に平滑化)

範囲検索:　全クラスタ鍵範囲を，10% ずつ 10 の範囲に分け，それぞれの範囲に対して，範囲検索クエリを実行し，その応答時間を測定した．

上記の DB 更新に伴い，構造劣化モニタを用いて，構造劣化分布 $\mathbf{G}$ を取得した．

DB 更新において，空間効率はほとんど変化せず，あるページが削除された後，大きく離れたディスクドライブ上の位置に配置されるなど，ページ列の分布 $\mathbf{S}$ が大きく変化した．

6.1 構造劣化モニタの精度およびオーバーヘッド

図 9 に，DB 更新に伴う，論理空間軸 i における構造劣化分布 $\mathbf{G}$ の変化を示した．更新分布 (図 8) と対応しているのが確認できる．初期状態 (更新 0 回) では，範囲検索を想定した場合，すべてのページは約 0.6[ms] の IO コストと推定されている．これがモデルにおける G_0 である．更新が繰り返され，更新 100 回後には，更新のあった領域の IO コストが更新の頻度に応じて増加している．つまり，その領域の構造が劣化していることを示している．更新 1000 回後には，さらに構造が劣化していることが確認できる．

各 10% の領域に対する範囲検索クエリの応答時間とその構造劣化モデルによる推定値を図 10 に示した．推定値との誤差は 8 ～ 17% 程度であった．ただし，推定モデルは範囲検索における IO 時間しか考慮していないことと，MySQL がメモリに

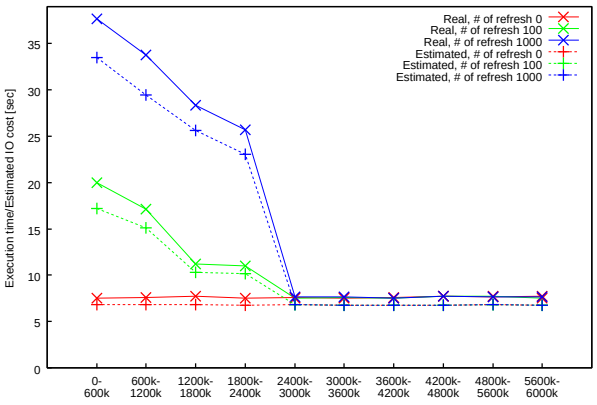

図 10　10%領域の範囲検索性能とその推定 IO コスト

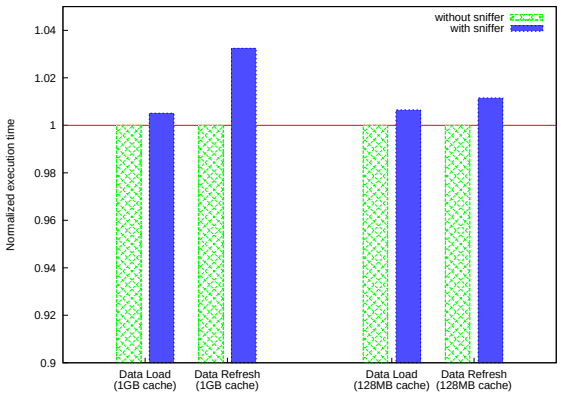

図 11　DB 更新差分抽出器のオーバーヘッド

ページを読み込んだ後のデータ処理時間が，実験環境だと，IO 時間の 10% 程度かかっていることを示すデータが得られているため，概算では，IO コスト推定誤差は 7% 未満である．

図 11 に，DB 更新差分抽出器のオーバーヘッドの評価結果を示した．バッファキャッシュを 1GB と 128MB の二通りの設定を用いて評価を行った．その結果，データロード時は 1GB キャッシュ時に約 0.5%，128MB キャッシュ時に約 0.7% であり，データ更新時は 1GB キャッシュ時に 約 3.2%，128MB キャッシュ時に約 1.2% のオーバーヘッドが測定された．データサイズが約 2GB で，更新されるデータがその 40% であるから，1GB キャッシュ時には，データページはほぼ全てバッファキャッシュ上にあることになり，オーバーヘッドも，取りうる最大値であると考えられる．また，大容量データベースの使用を想定した場合，128MB キャッシュ時の結果のように，バッファキャッシュヒット率が減少し，ディスクドライブに IO を頻繁に発行するため，ほぼメモリアクセスしかしない DB 更新差分抽出器が相対的にオーバーヘッドが小さくなることが期待される．また，データロード時は，B+Tree アクセス競合が起きないため，データ更新時よりもオーバーヘッドが少ないことが確かめられた．

以上により，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示された．

6.2　再編成スケジューラ適用の考察

図 9 において，例えば，SLA(2) に基づいて $X = 2$ と与えておくと，更新 100 回の時点で，論理空間 $\{i|0 \leqq i < 24000\}$ すなわちクラスタ鍵範囲 $\mathbf{R} = \{k|0 \leqq k < 600000\}$ の空間がもはや SLA を満たさないであろうことが分かる．この場合，再編成スケジューラはこの領域を再編成対象領域 $\mathbf{T}$ として同定できる．

7.　おわりに

本論文は、関係データベースにおいて自立再編成機構を実現することを目的として，低オーバーヘッドなリアルタイム構造劣化モニタ，及び SLA ポリシに基づいた再編成対象領域を同定するアルゴリズムを内包する再編成スケジューラを提案した．評価において，構造劣化モニタが低オーバーヘッドで高精度に構造劣化を監視できることを示し，再編成スケジューラの適用方法について考察した．

今後，提案した再編成スケジューラの有効性を示すべく，再編成対象領域の同定機構を構築し，評価を行う必要がある．また，再編成実施手法についても考察し，コスト対効果の高い再編成スケジューラの構築を目指して研究を進めたい．

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。

文　　献

[1] Don S. Batory. Optimal file designs and reorganization points. *ACM Trans. Database Syst.*, Vol. 7, No. 1, pp. 60–81, 1982.

[2] (Ed.)D.Lomet. Special Issue on Online Reorganization. *IEEE Data Eng. Bull.*, Vol. 19, No. 2, p. 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*, p. 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.*, Vol. 14, No. 3, pp. 401–424, 1989.

[7] K. Maruyama and S. E. Smith. Optimal reorganization of distributed space disk files. *Commun. ACM*, Vol. 19, No. 11, pp. 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*, Vol. 16, No. 6, pp. 362–365, 1973.

[10] S. Bing Yao, K. S. Das, and Toby J. Teorey. A Dynamic Database Reorganization Algorithm. *ACM Trans. Database Syst*, Vol. 1, No. 2, pp. 159–174, 1976.

[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, Vol. 46, No. SIG 8(TOD 26), pp. 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. 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), Vol. 4, No. 2, pp. 37–40, 2005.