データベース構造劣化によるOLTP性能低下に関する一考察

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，二次記憶装置内のデータ格納構造が劣化する．このような構造劣化はデータベースアクセス性能を低下させる．オンライントランザクション処理 (OLTP) は，電子金融取引や電子商取引などに不可欠なデータ処理方式である．従来のデータベースシステムにおける OLTP 高速化技術は，構造劣化による性能低下を防ぐという視点からはあまり行われてこなかった．近年，業務やサービスの電子化に伴い，データベース常時運用ニーズも高まってきたため，今後，構造劣化を監視し，性能低下を自動的に予防する技術の必要性は，ますます増大すると考えられる．本論文では，構造劣化が OLTP 性能低下に与える影響をベンチマークを用いて解析し，データ更新パターンに適応した，構造劣化の進行を抑えるデータ配置戦略について考察する．

キーワード データベースシステム、構造劣化，OLTP

A Study on OLTP Performance Degradation by Structural Deterioration of Database

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 secondary storage, which is called structural deterioration and causes performance degradation. Online Transaction Processing(OLTP) is essential data processing scheme for e-finance, e-commerce and so on. Conventional researches to improve OLTP perfromance lack a view from the aspect of preventing performance degradation due to structural deterioration. Recently, more and more business operations and services are being digitized then 24-hours-a-day operation of database is required. Needs for techniques to monitor structural deterioration and prevent performance degradation are increasing more and more. In this paper, we analyze how structural deterioration effects performance degradation in OLTP workload by using a benchmark. We also consider strategies of data updates which are adaptive to workload and prevent structural deterioration from being accelerated.

Key words Database system, Structural deterioration, OLTP

1. はじめに

オンライントランザクション処理 (Online Transaction Processing: OLTP) は，電子金融取引や電子商取引などにおいて，ACID 特性を保証するために不可欠なデータ処理方式である．そのため，OLTP の高速化は今も重要な課題である．加えて，近年，業務やサービスの急激なデジタル化が進んだため，データベースの常時運用を実現する技術が求められている．

OLTP の高速化には二つのアプローチがある．一つは，データベース構造や処理方式の改善により，高速化を達成しようとするものである．もう一つは，データベース更新による構造劣化による性能低下を防ぐ取り組みである．データベース構造劣化とは，データベース更新が繰り返されると進行するデータ格納構造の非効率化現象である．

前者に関して，これまで，IO を意識したデータベースの格納構造の最適化方式 [7], [8]，CPU 命令やキャッシュを意識した

データ処理高速化方式 [6]，そして，並列化によるデータ処理高速化方式 [5] などが OLTP 高速化のために研究されてきた．また，ハードウェア構成，データベーススキーマなどを負荷特性に応じて自動設定，自動調整することで高速化を目指す研究もなされている．[2], [13].

後者に関して，一般に，OLTP においてはデータ更新が大量かつ継続的に行われるため，構造劣化の必要以上の進行を阻止し，OLTP 性能を確保することが不可欠である．このため，従来，管理者はデータベース再編成を実施し，データの再配置を行うことで，構造劣化を除去し，継続的なデータベース運用における性能低下を防いできた．再編成は負荷の高い処理であり，データベース常時運用においてその負荷を分散するために，OLTP と小規模な再編成を同時に実行する技術 [14]，ストレージ技術を用いて OLTP 負荷に影響を与えずにバックグラウンドで再編成を実施する技術 [15] などの，オンライン再編成技術が近年研究されている．また，構造劣化を継続的に監視し，再編成契機を自動的に決定するための技術 [16], [17] が近年研究されている．以上のように，再編成によって構造劣化を除去する視点からの研究はなされているが，データ更新時に構造劣化を抑制するための技術，とりわけ，様々な負荷状況に応じて適応的に構造劣化を抑制する方法論はこれまで研究されてこなかった．

本論文は，データベースにおける OLTP 負荷において，構造劣化に起因する性能低下現象を，ベンチマークを用いて分析し，データ更新パターンに適応的に構造劣化を抑制するデータ更新戦略について議論する．トランザクションあたりの構造劣化量を抑制し，性能低下を最小限に抑えることで，OLTP 処理の高速化とともに，再編成負荷の低減にも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，OTLP と構造劣化の関係について述べ，性能低下の原因となる現象を説明する．次に，第 3 章で，ベンチマークを用いて構造劣化による OLTP 性能低下の分析を行い，第 4 章で，構造劣化を抑制するデータ更新戦略について議論する．第 5 章で関連研究について述べ，第 6 章で結論と今後の課題を述べる．

2. OLTP と構造劣化

本章では，OLTP の性質について述べ，起こりうる構造劣化の分類を行い，性能低下との関係について述べる．

2.1 OLTP 負荷の特性

OLTP をサポートするデータベースシステムとして，関係データベースシステムが主に用いられている．関係データベースは，複数の表から成り，表には複数のデータレコードが納められている．トランザクションの ACID 特性を満たすために，レコードの排他処理が行われる．レコードに対する同時平行処理を高めるために様々な格納構造，データ更新技術が用いられている．

OLTP において，各データレコードのサイズがページサイズに比べて小さい場合を考える．このとき，ページあたり数〜数十レコードを格納できる．各トランザクションがアクセスするデータレコードは高々数〜数十であり，データマイニング処理やバッチ処理に比べて少なく，すなわちトランザクションあたりの IO 数も少ない．ただし，大規模なシステムでは同時実行されるトランザクションが多いため，全体では大量の IO が必要になる．このため，OLTP においても IO はボトルネックになりやすいため，性能を確保するために IO を減らす努力が必要である．

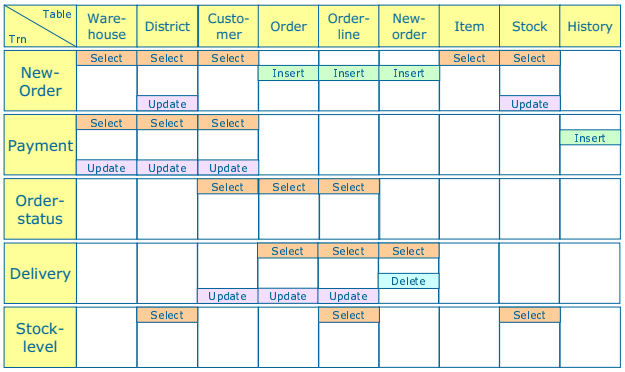

図 1　各 TPC-C トランザクションのアクセス対象とその種類

以下，具体的な OLTP アプリケーションにおいて説明する．TPC-C ベンチマーク [4] は卸売り会社のトランザクション処理を模擬したベンチマークであり，OLTP の標準ベンチマークとして今日広く用いられている．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．各トランザクションの実行比率に関しては，new-order 及び payment トランザクションがほとんどを占め，delivery，order-status，及び stock-level トランザクションはそれぞれ数パーセント程度の比率で実行するのが一般的である．

図 1 に，各トランザクションがアクセスする表とそのアクセスの種類を示した．主にアクセスされるデータの視点から見た処理の流れやアクセスパタンの特徴を以下に示す．

- new-order トランザクションによって order 表に新しい注文を表すレコードが挿入され，order-line 表にその注文内容が平均 10 レコード挿入される．new-order 表には，配達が完了して delivery トランザクションが実行されるまで注文情報を保持する．delivery トランザクションによって，new-order 表から未配達の注文が取り出され，削除される．そして，対象注文の order 表のレコードに対して配達完了マークをつける．その後，order 表と order-line 表の当該注文のデータは基本的にアクセスされない．
- warehouse 表，district 表は，レコード数がその他の表に比べて極端に小さい．これらの表は，注文に対して，注文番号を連番で付加していくために排他制御が主に行われる表である．ウェアハウスを識別する属性 w_id と各ウェアハウスに存在するディストリクトを識別する属性 d_id が存在し，ウェアハウス数は TPC-C のデータセットの大きさを調整するために可変であるが，ディストリクト数は，ウェアハウス毎に 10 と定められている．
- customer 表，item 表，stock 表においてレコード数は変化しない．各トランザクションによるレコード更新および参照が行われる．
- history 表は，payment トランザクションによって支払の履歴が記録される．ベンチマーク内ではその後アクセスされない．
- payment トランザクションは，new-order トランザクションと warehouse 表，district 表においてほぼ常に排他制御を行う以外は他のトランザクションとはほとんど干渉しない．
- order-status トランザクションと stock-level ト

ランザクションはレコード操作を行わない．

- order, order-line, history 表のデータが単調増加し，他の表に関してはトランザクションによってレコード数は変化しない．

各表において，アクセスパターンは必ずしもランダムアクセスに近似できるわけではなく，一連のトランザクションが一部の表データを主鍵順にアクセスする，すなわちシーケンシャルなレコードアクセスも実施されている．具体的には，order 表，order-line 表は注文番号 (o_id) 順に挿入され，アクセスされる．また，データアクセスの局所性も大きいことが分かる．new-order 表に存在している未配達注文に関するデータレコードは order 表においても頻繁にアクセスされるが，配達済み注文に関するデータレコードはほとんどアクセスされない．これらの性質は，一般的な OLTP アプリケーションにも当てはまると考えられる．

2.2 データベース格納構造と構造劣化

構造劣化は，データベースの格納構造が変化することで，特定のアクセスパターンにおける IO 数の増加もしくは，IO あたりの応答時間増加という形で性能低下を引き起こす．一般的なデータベースシステムでは，ページ単位でデータを管理し，各ページに格納されているデータレコードへのアクセスはメインメモリ上に確保されたバッファキャッシュを経由して行う．データベース格納構造にもよるが，一般に以下の構造劣化現象が存在する．

高水準位 (HWM): 非クラスタ表(注1) で主に使われるデータ構造であり，データ保持空間の一番後ろのページを指すポインタとして HWM を保持しておき，空きページ探索を行うことなく高速なデータ挿入を可能にする．ただし，データベース空間の終端まで HWM が到達した場合，空きページ探索を行うため，データ挿入あたりの IO 数が増加し，性能が低下する．

空き領域断片化: データベースシステムは一般に，空き領域管理のため，空間における空きページの位置をビットマップもしくはフリーリストを用いて保持する場合が多い．ページ単位で空き領域が存在する場合は，これらの構造を用いて空き領域を探索すればよいが，全てのページにデータが格納されている場合，レコード挿入のためのページ内空き領域を探索するため，IO 数が増加し，性能が低下する．空きページが存在しない場合はレコード挿入が出来ないデータベースシステムも存在する．

レコード断片化: レコード更新によってレコード長が増加するときにページ内に必要な空き領域がなく，増加分のデータを別のページ内に保存する機能をサポートするシステムにおいては，レコードが断片化する．当該レコードにアクセスする場合，レコード断片が存在する全てのページにアクセスする必要があるため，レコードアクセスあたりの IO 数が増加し，性能が低下する．これは，データベーススキーマにおいてレコード長が可変の表にのみ起こる．この機能をサポートしていないデータベースシステムも存在する．そのような場合，同等の機能としてレコード削除および新レコード挿入を行うが，断片化する場合に比べてレコード更新のコストは高くなる．

(注1)：ここで，クラスタ表とは，レコードがクラスタ鍵を用いて，一意にアクセスされる，もしくは範囲を指定してクラスタ鍵順にアクセスされるために最適化された表を指す．このため，ストレージ上においても鍵順にデータを配置しようとする．非クラスタ表は，レコードの挿入順にデータが配置される場合が多い．

充填率低下: ページ内にレコードが十分格納されておらず，空き領域が多いとき，ページ内に十分レコードが格納されている場合に比べて，レコードあたりに必要なアクセスページ数が多くなる．すなわち，格納ページサイズに対する，レコードデータの占める空間サイズの比を充填率と呼ぶ．充填率が低下すると，レコードアクセスあたりの IO 数が増加し，性能が低下する．

クラスタ化度低下: クラスタ表や二次索引などのデータ構造(注2) で，データが鍵順にアクセスされる場合，すなわち範囲走査において，ストレージ上で連続してデータが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るが，ストレージ上で連続していない場合，非シーケンシャルアクセスが発生する．現状のデータベースシステムは永続的ストレージにディスクドライブを用いており，シーケンシャルアクセス性能は非シーケンシャルアクセス性能に比べて最大数十倍も高速であるため，IO あたりの応答時間が変化する．クラスタ化度は，このシーケンシャルアクセスの発生頻度を，1 次元の指標として表現したものである．すなわち，クラスタ化度が低いと，シーケンシャルアクセスが期待できなくなり，IO 効率が低下し，性能が低下する．クラスタ化度よりも正確な構造劣化指標として [17] が提案されている．

上記に挙げた構造劣化において，クラスタ化度低下を除いては，構造劣化に伴いデータアクセスのための IO 数が増加することで性能が低下する．前者 3 つは，監視が比較的容易で，構造劣化時の IO 数の増加量も予測しやすい．また，データベースシステムによってはサポートしていない場合もある．充填率低下及びクラスタ化度低下は，多くのデータベースシステムで起こる構造劣化であり，高解像度かつ正確な監視が難しい．充填率監視については [11] などでサンプリング等を行い状況を把握する手法が存在する．クラスタ化度低下の監視については, [16], [17] で研究されている．また，クラスタ化度低下は，数十ページ以上のアクセスを伴う大規模な範囲走査を含むクエリもしくはトランザクションの性能を低下させるが，そのような範囲走査はバッチ処理やデータウェアハウス系のアプリケーションにおいて主に想定されるアクセスパターンである．OLTP 負荷ではそのようなアクセスパターンは通常存在せず，TPC-C ベンチマークでも規定されていない．よって，クラスタ化度の低下が OLTP 性能低下に影響することは考えにくい．このため，本論文では充填率の低下に注目して，OLTP 性能低下との関係を分析する．

3. 構造劣化に起因する OLTP 性能低下分析

本章では，OLTP アプリケーションにおいてトランザクションによるデータベース構造劣化が，トランザクションの性能低下を引き起こす現象を実験を通して分析する．

3.1 対象データベース構造

クラスタ表を構成する B+木構造における充填率の低下を対象構造劣化として評価を行う．これは，MySQL [9] InnoDB のクラスタ表や Oracle の索引構成表などで採用されているデータ構造である．本論文では，以後 MySQL InnoDB を用いる．

図 2 に，対象とする B+木構造の概要を示した．ページは固定サイズであり，データ行は葉ページのみに格納されている．

(注2)：主に B+木や，それに準じた構造が使われる

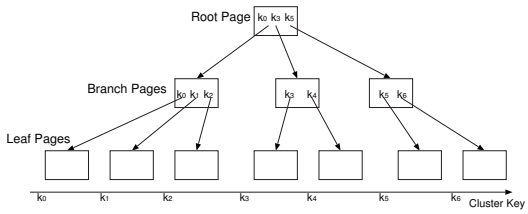

図 2　クラスタ表の B+木構造

クラスタ鍵指定によって任意のデータ行もしくは任意の範囲に高速アクセスするため，クラスタ鍵順にデータ行が格納されている．

レコード挿入，更新，削除によって，ページの分割及び結合が起きる．一般的なデータベースシステムでは，データロード時の充填率や，結合を実施するしきい値充填率などが設定可能であるが，MySQL InnoDB では，データロード時は 15/16 の充填率でデータが葉ページに格納され，レコード削除時は一定時間経過した後に，隣のページと結合可能ならば結合する戦略を取っている．レコード断片化はサポートしておらず，レコード削除，挿入で代替する．また，ページ分割は，レコード挿入，更新によってページが溢れる場合に実施され，直前のレコード挿入がページ内で最大鍵値であり，挿入中のレコードがさらに大きな鍵値であるときのみ，連続挿入であると見做して，分割前のページはそのままに，新しいページに対象レコードのみを挿入する．それ以外のケースでは，充填率が同じになるように，すなわちそれぞれほぼ充填率 0.5 で分割される．

3.2 環　　境

Linux 2.4 が動作する PC (CPU: Xeon 3.2GHz x2，Memory: 2GB) 上で，1Gbit ファイバチャネル経由でディスクドライブ 4 台 (16GB, 10000rpm) をソフトウェア RAID0(チャンクサイズ 64KB) として raw デバイス経由で先頭から 4GB の表空間を確保した．データベースシステムとして，MySQL 5.0 InnoDB を用いているが，評価のために，表，索引毎に IOPS(注3) を出力する実装を追加した．

3.3 設　　定

OLTP ベンチマークとして TPC-C Rev. 5.6 を用いた．主鍵を全てクラスタ鍵としたクラスタ表を用いてデータベーススキーマを構成した．可変長文字列は全て固定長文字列として扱い，レコードが固定長となるようにした．ウェアハウス数 16 で初期データをロードした．トランザクション比率は，`new-order`: 45%，`payment`: 45%，`order-status`: 2%，`delivery`: 6%，`stock-level`: 2%とし，ウェアハウスあたり 5 並列，計 80 並列に思考時間 0 で 100,000 トランザクションを投入し，トランザクション応答時間の変化を調べた．また，トランザクション投入後の充填率の分布を調べた．

3.4 結　　果

この実験におけるトランザクションスループットの平均値は 1138 tpm(注4) であった．まず，図 3 にトランザクション応答時間の変化を示した．`new-order`，`payment`，`order-status`，`delivery` トランザクションの応答時間が約 2,000 秒経過後から約 2,500 秒経過後まで増加していることが確認できる．

(注3)：IO per second.

(注4)：transaction per minute

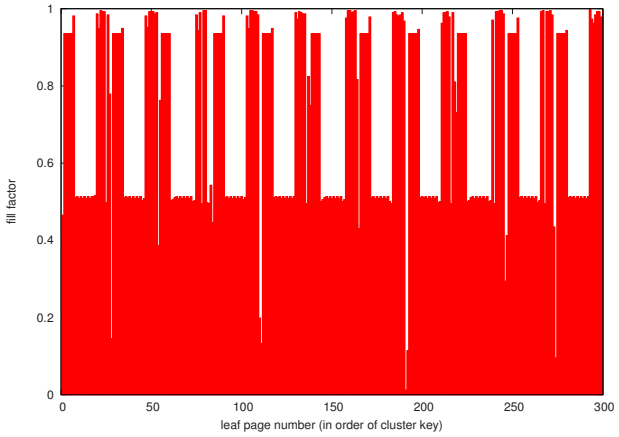

図 9　トランザクション投入後の `order` 表における充填率分布

各トランザクションの応答時間の内訳を図 4– 8 に示した．図 4 の `new-order` トランザクションにおける `Phase I` は `select warehouse`, `select customer`, `select district`, `update district` から成り，`Phase II` は `select item`, `select stock`, `update stock`, `insert order-line` から成る．また，図 5 の `payment` トランザクションにおける `Phase I` は `update warehouse`, `select warehouse`, `update district`, `select district` から成り，`Phase II` は `select customer` から成る．どちらのトランザクションにおいても `Phase I` は排他待ち時間が多くを占める．その他の内訳によると，`order` 表へのアクセス応答時間が特に増加していることが分かる．

図 9 に，`order` 表を構成する B+木における葉ページ内データレコード充填率分布を，クラスタ鍵順に示した．`order` 表のクラスタ鍵は複合鍵 (ウェアハウス番号 `w_id`, ディストリクト番号 `d_id`, 注文番号 `o_id`) から構成されており，初期ロード時には，各ウェアハウス，ディストリクト毎に `o_id` が 1 から 3000 の計 3000 レコードが充填率 15/16 でロードされる．その後，`new-order` トランザクションによって `o_id` が 3001 から順に新しい注文に対応するレコードが挿入されていく．このとき，ページ分割時に充填率が 0.5 ずつ配分され，その後前半のページにはレコードが挿入されず，レコード更新によってもレコードサイズが変化しない．また後半のページはトランザクションによってレコード挿入が続くため，後に分割される．このアクセスパターンが繰り返されると，トランザクションによって追加された領域は充填率が約 0.5 になる．

一連のトランザクションは，`order` 表のレコードに対して `o_id` 順にアクセスする．`delivery` トランザクションは，`o_id` が 2101 である注文から処理を開始する．すなわち，各ウェアハウス，ディストリクトにおいて，900 回 `delivery` トランザクションが実行されるまでは，初期ロードされた `order` 表データにアクセスし，その後は，充填率が約 0.5 の領域にアクセスする．このため，`order` 表の充填率が約半分の領域では，`delivery` トランザクションあたりに必要な IO が約 2 倍に増加することにより，応答時間が増加するものと考えられる．

次に，トランザクション比率を変更し，データを変更しない `order-status` と `stock-level` トランザクションを用いずに，`new-order`:47%, `payment`:47%, `delivery`:6%の設定で，バッファキャッシュサイズを 16MB から 128MB まで変

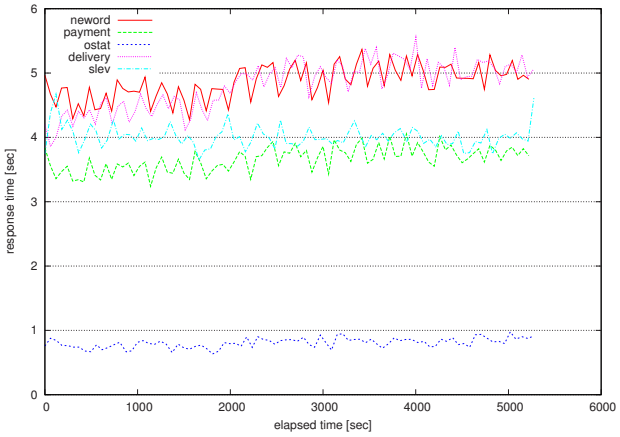

図 3　TPC-C トランザクション応答時間と内訳

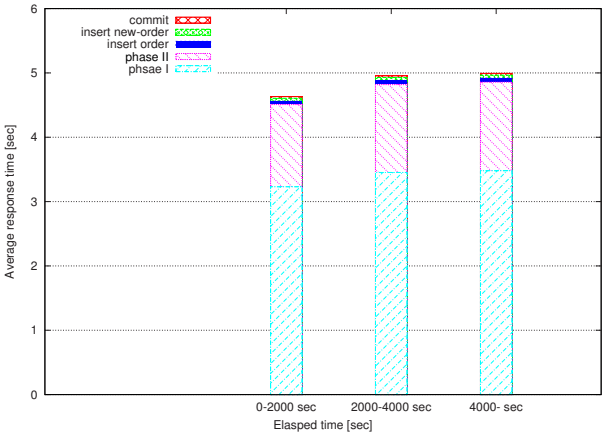

図 4　new-order トランザクション応答時間の内訳

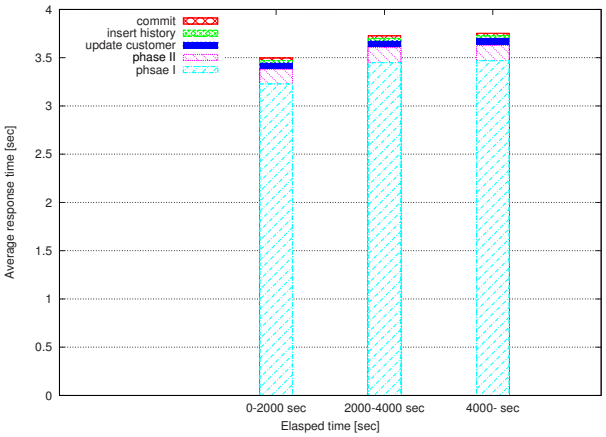

図 5　payment トランザクション応答時間の内訳

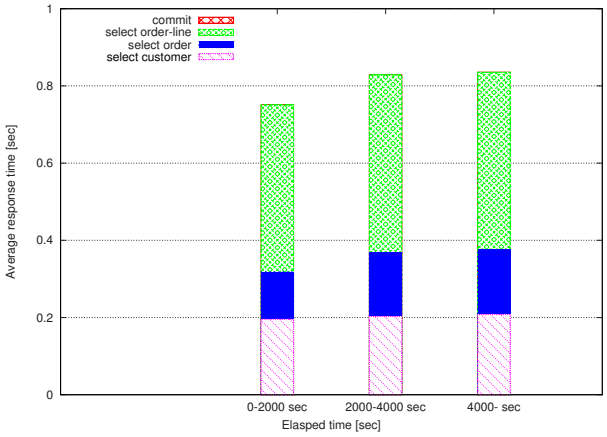

図 6　order-status トランザクション応答時間の内訳

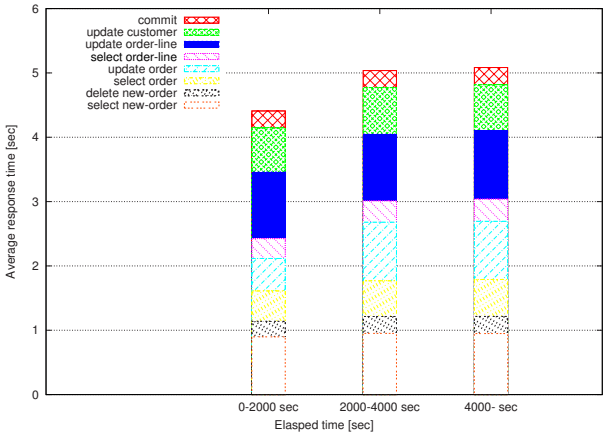

図 7　delivery トランザクション応答時間の内訳

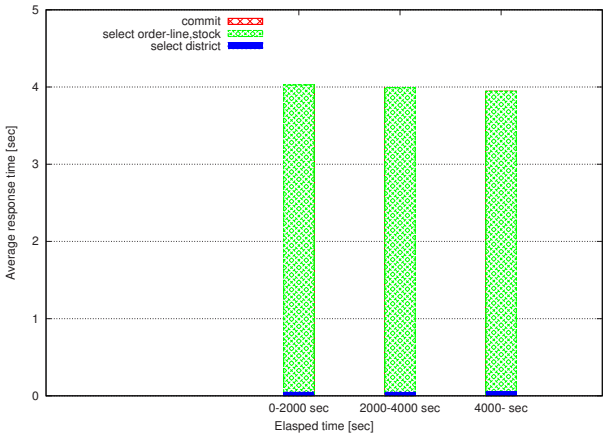

図 8　stock-level トランザクション応答時間の内訳

化させて同様のトランザクション投入実験を行った．初期ロード時の表空間はストレージ上のデータ配置も含めて同じものを使用した．図 10–13 に，トランザクション投入中の IOPS 変化を示した．このときのトランザクションスループット平均値はバッファキャッシュサイズの小さい順に 1241，1479，2018，3039 tpm であった．

バッファキャッシュサイズ 16MB の実験結果において，図 10 中の orders:PRI が order 表への IOPS を示している．トランザクション投入直後からしばらくは IOPS が約 20 で推移するのに対し，約 1,500 秒付近で急激に増加し，以後 35 で推移することが確認された．その他の表及び索引への IO 数は減少しているものも観測されるが，これは，order 表への IOPS が増加し，トランザクション処理スループットが下がったため，時間あたりのトランザクション実行数が現象し，それに伴い IOPS も下がったものと推察される．

バッファキャッシュサイズを増やしていくと，order 表に関して，64MB 以上で IOPS が減ることが分かる．特に，バッファキャッシュサイズ 128MB の場合，全体の IOPS に関してほとんど無視できる量しか増えていないことが分かる．バッファキャッシュサイズが増えると，new-order トランザクションによって挿入されたレコードがまだバッファキャッシュ上に存在するうちに，delivery トランザクションによってアクセスされるようになり，そもそも IOPS が増加しないものと考えられる．バッファキャッシュサイズが 192MB 以上では order 表の IOPS が

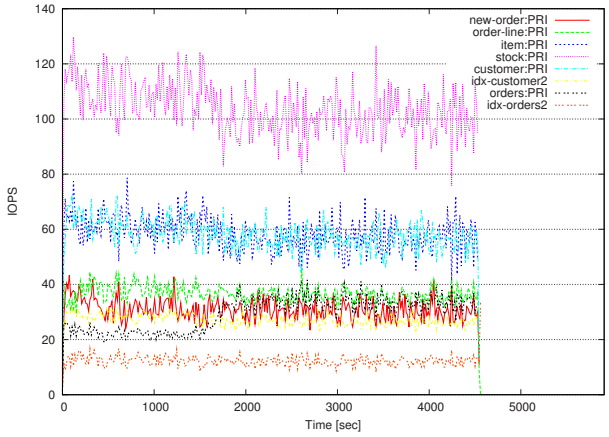

図 10　IOPS (バッファキャッシュ16MB)

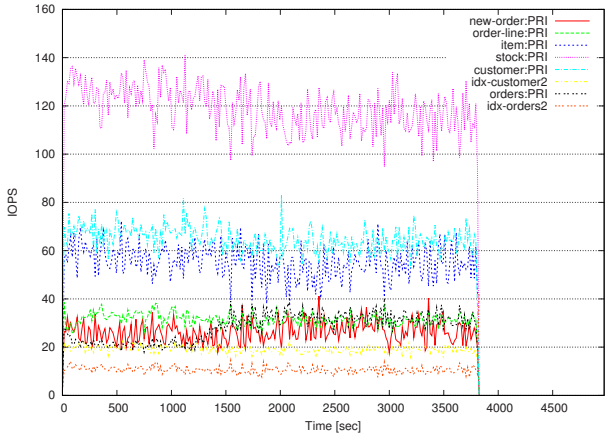

図 11　IOPS (バッファキャッシュ32MB)

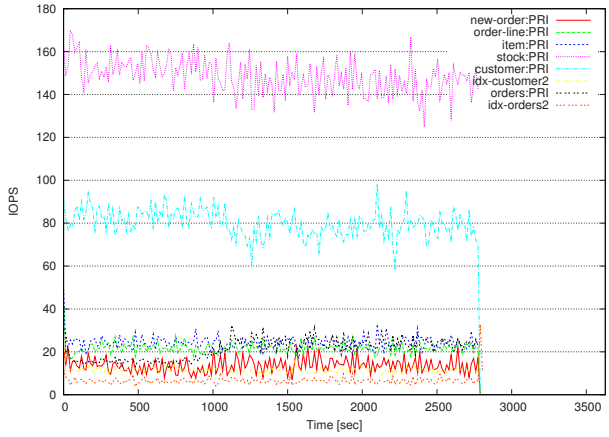

図 12　IOPS (バッファキャッシュ64MB)

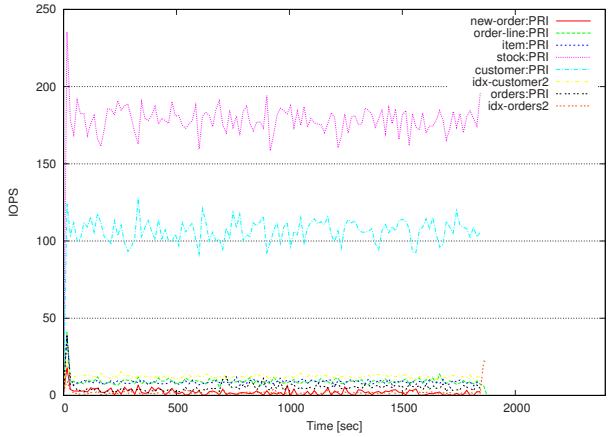

図 13　IOPS (バッファキャッシュ128MB)

0 に近いため，ほぼ全てのホットスポット領域(注5) がバッファキャッシュ上に乗っているものと考えられる．

上記から，ホットスポット領域のデータページがバッファキャッシュにほぼ乗るような状況では，充填率はトランザクションの性能低下に影響しないが，ホットスポットのサイズに比べてバッファキャッシュサイズが小さい場合は，充填率の低下が性能低下の主な要素になる．充填率が低くなることで，ホットスポットサイズが拡大するため，同じバッファサイズでデータベースを動かす場合，充填率を高めておいた方が充填率低下によるトランザクション性能低下を防ぎやすい．

`customer`，`stock`，`item` 表などでは，ランダムアクセスに近似でき，ホットスポットが大きい．キャッシュヒット率もその他の表や索引と比べると低く，本実験におけるバッファキャッシュサイズが増えるのに伴い，トランザクションスループットが増加した分に応じて IOPS が増えていることが分かる．

4. 議　　論

本章では，3. 章で評価した OLTP 負荷やそれに準ずる負荷において，アクセスパターンに適応したデータ更新戦略について考察する．

4.1 シーケンシャルレコード挿入検知によるページ分割時充填率調整

バッファキャッシュサイズが小さく充填率の低下がトランザクション性能低下に大きく影響するような場合でも，レコード挿入時に充填率を高めることで，性能低下を回避する，もしくは遅らせることが可能である．

MySQL 5.0 InnoDB の場合，クラスタ鍵順にレコードが順次挿入されていることが検知されれば，ページ分割時に，15/16:1/16 でデータを分配し，さらなる連続 (シーケンシャル) 挿入に対して充填率を高める実装がなされている．ただし，シーケンシャル挿入が発生していることを検知するための仕組みは単純である．ページに対して，前回挿入されたレコードの鍵値がページ内で最大であり，かつ今回挿入されようとしているレコードの鍵値がさらに大きい場合に，レコードがシーケンシャル挿入されていると判断する．この手法では，表の一番後ろ，すなわちクラスタ鍵が最大のレコードをシーケンシャル挿入する場合にのみ充填率が調整される．データ初期ロード時には有効化されるが，本論文における実験のようなケースにおけるシーケンシャル挿入に対しては機能しない．

本実験においては複合鍵がクラスタ鍵に指定されており，表の途中位置におけるシーケンシャル挿入であったため，現在の実装ではクラスタ鍵順にデータが挿入されていることを検知できず，1:1 でページ分割されているものと考えられる．上記の手法を改善することで，任意の位置におけるシーケンシャルレコード挿入を検知し，適切な充填率調整が以下のように可能である．

- 分割対象ページに対する前回のレコード挿入が，今回挿入しようとしているレコードとページ内で隣り合うかどうか

(注5)：ここでは，未配達の注文に対応する `order` 表レコードの占めるデータページ集合．

を判定し，隣りあっている場合は当該ページにおいてはシーケンシャルレコード挿入が行われていると判断する．この場合，ページに対して最後に挿入された鍵値を覚えておく必要があるが，ページヘッダに最後に挿入されたレコードのオフセットを記録しておくだけで済む．

- シーケンシャルレコード挿入が行われていると判断されているページが一杯になって分割するときに，挿入が昇順であり，挿入レコードよりも鍵値の大きなレコードが分割前ページに存在する場合は，挿入レコードより大きなレコードを全て新ページに配置して分割する．挿入レコードが最大鍵値である場合は，任意の充填率で配分する．特に空間に対してシーケンシャルレコード挿入のみ行われていると判断できる場合は新ページに当該レコードのみを挿入すればよい．挿入が降順の場合は，適宜対応する操作を実行する．

上記の手法を用いることで，任意の位置に対するシーケンシャルレコード挿入において，再編成を必要とせず，ページ分割時に充填率を高めることが出来る．これは挿入量に対するページ分割数を減らすことを意味し，トランザクション性能を上げるだけでなく，構造劣化を遅らせる効果がある．

4.2 アクセスパターンに適応的なページ分割時充填率調整

上記の手法は，アクセスパターンとして，シーケンシャルレコード挿入とその他全てのレコードアクセスという分類しかなされていなかった．本節では，よりアクセスパターンを細かく分類し，アクセスパターンに対して適応的に構造変化量を減らし，充填率を高める方法論について検討する．具体的には，アクセスパターンはシーケンシャルレコードアクセスおよびランダムレコードアクセスの両方が空間に対して同時に起きるケースを考える．また，データレコード削除，更新に関しても包括的に扱う．ただし，ページの結合は再編成に相当するため，次節で再編成との連携について述べる．

データ更新は，以下のアクセスパターンに分類されるものとする．

- シーケンシャルレコード挿入
- シーケンシャルレコード削除
- 上記以外のレコード挿入，削除，更新

クラスタ表を構成する各葉ページは，アクセスパターンによって，それぞれのモードを保持することとする．すなわち，シーケンシャル挿入モード，シーケンシャル削除モード，ランダム更新モードの 3 つである．各ページはある時刻において，これら 3 つのいずれか一つのモードであるとする．シーケンシャル挿入が起きているページはシーケンシャル挿入モード，シーケンシャル削除が起きているページはシーケンシャル削除モード，その他の更新が起きているページはランダム更新モードになる．

初期状態では，全てのページはランダムモードとする．ページ内において鍵値連続アクセスが起きたときシーケンシャルモードに移行する．挿入の場合はシーケンシャル挿入モード，削除の場合は，シーケンシャル削除モードとする．その後，アクセスが鍵値において連続しなくなったときは，ランダム更新モードに戻る．

シーケンシャル挿入モード時においてページ分割後，新しいページをシーケンシャル挿入モードに変更し，元のページをランダム更新モードに変更する．シーケンシャル削除モードにおいては，ページ削除後に，続くページをシーケンシャル削除モードにする．

レコード挿入もしくは更新時に，ページが溢れてしまう場合，ページ分割は避けられないものとする(注6)．ページ分割時に分割元のページの充填率を X とする．ランダムアクセスを仮定する場合，B+木のページ分割は $X = 1/2$ としてページ分割を行うことで，その後のページ分割を最大限遅らせようとする．シーケンシャル挿入アクセスを仮定し，その後分割されたページにおいてランダムアクセスによるデータ量増加が少ないと期待できるならば，前節で述べたように，分割時の充填率を $X = 1/2$ よりも高くすることで，シーケンシャルデータ増加量あたりのページ分割数を削減することができる．シーケンシャル削除アクセスを仮定した場合は，その後近未来にページ内が空になることが期待されるため，ページ結合をしないことで構造変化量を削減することが出来る．

上記から，シーケンシャル挿入モードのページにおいてページ分割が発生した場合は，その後のランダムアクセスによるデータ量増加量にも依存するが，$X = 1 - M(0 < M < 1/2)$ で分割することが可能である．ランダムアクセスによってデータ量が減少している場合には，$M = 0$ としてしまう制御も考えられる．また，一定時間内にページ内 1/2 以上の空間が増大するペースでのランダムデータ更新によるデータ増加が起きている場合は，シーケンシャル挿入を検知して充填率を調整することよりも，ランダム更新によりページ分割がほぼ必ず発生するので，$X = 1/2$ にするべきである．この場合，空間に対するアクセスパターンはランダム更新に近似できると言える．

ランダム更新モードのページにおいてページ分割が発生した場合は，従来通り $X = 1/2$ として分割することで，空間においてデータ量が増加している状況では充填率 1/2 以上を常に確保することができる．

シーケンシャル削除モードにおいてページ分割が発生した場合は，ランダム更新モードに変更した上で，$X = 1/2$ で分割する．シーケンシャル削除モード時は，ページ内データが直前まで削除されていたためページ分割発生頻度は低いと考えられる．

M の値を適切に制御すれば，シーケンシャルレコード挿入後に一定のランダムデータ更新が行われてデータ量が増加するアクセスパターンに対しては，ページ分割時の充填率調整のみで，空間の充填率を 1 に近づける，かつページ分割数を最小に抑えることが可能である．逆に，将来のアクセスパターンが予測できないような場合は，M を固定値にして多少の余裕を持たせるか，もしくは $M = 0$ にすることで，一定量以上のデータ量増加に対してはページ分割を許容することで，より一般的なアクセスパターンに柔軟に対応できると考えている．

また，データ更新が続いている間にページ充填率を高く保持したい場合は，適宜ページ結合や再編成を行う必要がある．それらの場合でも，本手法はシーケンシャルデータ挿入の比率が高ければ，余計なページ結合や再編成をする手間が少なくなる利点がある．

4.3 データベース再編成との連携

再編成は，主に充填率の回復 (空き領域の回収) とページ並べ替えによるクラスタ化度の回復に分類されるが，ここでは充填率の回復に注目する．ページ結合は，充填率の回復操作にあ

(注6)：オーバーフロー領域を設けたり，一般的な B+木の構造を逸脱する構造変化はしないものとする

たる．

前節の手法でページ分割数が最小化されたと仮定して，その上で，充填率を最大化するためのページ結合数を最小化するためには，ページ結合を出来るだけ遅らせることで実現できる．ただし，一般には充填率を高く維持する制約もしくは使用可能な空間の制約があることから，ここでは，アクセスパターンが与えられたとき，結合を行った後，一定時間以内に再びページ分割されることを防ぐため，想定されるデータ増加を許容できるだけのページ内空き領域を確保できるか否かを判断する方法を考える．

シーケンシャル挿入モードにおけるページ分割時の充填率調整と同様に，ページ結合後のページ空き領域が M 以上あれば，一定時間以内にランダム更新によって再び分割されないと期待できるため，結合後のページ充填率を Y として，$Y < 1 - M$ である場合は，結合してもその後再分割されないと期待できる．

また，シーケンシャル削除モードにおいては，ページ結合をしなくても近いうちにページ内の全てのレコードが削除されることが期待されるため，結合を順次行う場合でも優先順位を低くすることで，ページ結合数の削減が期待できる．

結合を実施する時期は，レコード操作の性能とレコード読み込みの性能のトレードオフで決定される．これを包括的に扱えるモデルも検討中である．また，これらの手法に加えて，実際の充填率の変化や，ページ分割，結合発生分布をフィードバックすることで，より確度の高いアクセスパターンの推定と充填率調整が出来るのではないかと考えている．

5. 関 連 研 究

データベースの構造劣化に関して，構造劣化の監視 [16], [17] と再編成による回復手法 [14], [15] がこれまで研究されてきた．本論文では，再編成による空間回収を意識した構造変化抑制手法を考察している．これは我々の知る限りこれまでにない．文献 [7] は構造変化を抑える戦略を提案している．これは，ランダムアクセスを仮定した，B+木の充填率調整のためのページ結合戦略について述べたものである．本論文では，さらにシーケンシャルアクセスによる最適化を考慮した手法を検討している．その他，B+木への平行アクセス性を高める構造変化手法 [8] なども提案されている．これにアクセスパターンに適応的な構造変化戦略手法を応用することは可能であると考えている．

データベース監視に関しては，文献 [1], [3], [10] などで研究されている．必要な情報をオンライン取得する手法は，本論文で考察している充填率調整手法において，入力パラメータとして必要である．

データベースの物理構造設計に関する研究 [2], [12], [13] もなされている．これらにおいては，近年の自律データベース管理を目指す研究において，運用負荷パターンを解析し，適切な物理構造パラメータを設定するものである．これは静的な物理構造を決定するための手法であり，本論文において考察する動的な構造変化戦略と補完的な研究である．

6. おわりに

本論文は，構造劣化の OLTP 性能低下への影響を実験を通して分析し，充填率の低下がホットスポットの小さい表にアクセスするトランザクションの性能低下を引き起こすことが分かった．これらの結果から，データ更新時に構造劣化を抑制し充填率を高める構造変化戦略を考察し，シーケンシャルレコードアクセスとランダムレコードアクセスを用いたデータベースアクセスパターンに適応的な充填率調整手法について検討した．今後，これらの手法の評価を行う予定である．

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。

文　　献

[1] Ashraf Aboulnaga, Peter J. Haas, Sam Lightstone, Guy M. Lohman, Volker Markl, Ivan Popivanov, and Vijayshankar Raman. Automated statistics collection in db2 udb. In *VLDB*, pp. 1146–1157, 2004.

[2] Sanjay Agrawal, Vivek R. Narasayya, and Beverly Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *SIGMOD Conference*, pp. 359–370, 2004.

[3] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD Conference*, pp. 287–298, 2004.

[4] TPC: Transaction Processing Performance Council. TPC BENCHMARK™ C Standard Specification. http://www.tpc.org/.

[5] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, Vol. 35, No. 6, pp. 85–98, 1992.

[6] Richard A. Hankins and Jignesh M. Patel. Effect of node size on the performance of cache-conscious b^+-trees. In *SIGMETRICS*, pp. 283–294, 2003.

[7] Theodore Johnson and Dennis Shasha. B-trees with inserts and deletes: Why free-at-empty is better than merge-at-half. *Journal of Computer and System Sciences*, Vol. 47, No. 1, pp. 45–76, 1993.

[8] David B. Lomet. Simple, robust and highly concurrent b-trees with node deletion. In *ICDE*, pp. 18–28, 2004.

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

[10] Dushyanth Narayanan, Eno Thereska, and Anastassia Ailamaki. Continuous resource monitoring for self-predicting dbms. In *MASCOTS*, pp. 239–248, 2005.

[11] Albrecht Schmidt and Michael H. Böhlen. Parameter estimation using b-trees. In *IDEAS*, pp. 325–333, 2004.

[12] Daniel C. Zilio, Sam Lightstone, and Guy M. Lohman. Trends in automating physical database design. In *INDIN*, pp. 441–445, 2003.

[13] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. Db2 design advisor: Integrated automatic physical database design. In *VLDB*, pp. 1087–1097, 2004.

[14] Chendong Zou and Betty Salzberg. On-line reorganization of sparsely-populated B+-trees. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 115–124, 1996.

[15] 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.

[16] 星野喬，合田和生，喜連川優. 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. 日本データベース学会論文誌 (DBSJ Letters), Vol. 5, No. 2, pp. 37–40, 2006.

[17] 星野喬，合田和生，喜連川優. 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. 日本データベース学会論文誌 (DBSJ Letters), Vol. 5, No. 1, pp. 101–104, 2006.