# 論文 / 著書情報
# Article / Book Information

| | |
|---|---|
| 論題(和文) | Fat-Btreeを用いたPostgreSQL分散化におけるページ分割手法の評価 |
| Title(English) | An Evaluation of Page-Splitting Methods in a Fat-Btree for Distributing PostgreSQL |
| 著者(和文) | 並木悠太, 神戸康多, 横田治夫 |
| Authors(English) | Yuta Namiki, Kota Kanbe, Haruo Yokota |
| 掲載誌(和文) | データ工学ワークショップ2008 論文集 |
| Citation(English) | Proceedings of Data Engineering Workshop 2008 |
| Vol, no, pages | , , |
| 発行日 / Pub. date | 2008, 3 |
| URL | http://search.ieice.org/ |
| 権利情報 / Copyright | 本著作物の著作権は電子情報通信学会に帰属します。<br>Copyright (c) 2008 Institute of Electronics, Information and Communication Engineers. |

DEWS2008 D2-4

# Fat-Btreeを用いたPostgreSQL分散化におけるページ分割手法の評価

並木　悠太†　神戸　康多††　横田　治夫†,†††

† 東京工業大学大学院情報理工学研究科計算工学専攻
〒 152-8552 東京都目黒区大岡山 2-12-1
†† フューチャーアーキテクト株式会社
〒 141-0032 東京都品川区大崎 1-2-2 アートヴィレッジ大崎セントラルタワー
††† 東京工業大学学術国際情報センター
〒 152-8550 東京都目黒区大岡山 2-12-1
E-mail: †namiki@de.cs.titech.ac.jp, ††kanbe.kota@future.co.jp, †††yokota@cs.titech.ac.jp

**あらまし**　我々は異なる PE 間でデータの配置を動的に変更可能な並列 B-tree 構造である Fat-Btree のリーフページに DBMS のデータページを格納することで，無共有型の並列 DBMS を構築する方法を検討している．この構造をとるとき，Fat-Btree において発生するタプル挿入によるデータページの分割処理を考える必要がある．この問題に対し我々は DBMS における追記型 MVCC による同時実行制御を考慮した手法として，ページ満杯時に分割を行う手法とページ利用率が閾値を超えた時点で分割を行う先行分割型の手法を提案してきた．本稿では追記型 MVCC を利用する PostgreSQL を Fat-Btree を用いて分散化したシステムを実装し，そこに両ページ分割手法を適用して評価を行う．実験の結果，高更新環境において先行分割型の手法がより高いスループットをもたらすことを確認した．

**キーワード**　分散 DB，Fat-Btree，MVCC，PostgreSQL，事前分割

# An Evaluation of Page-Splitting Methods in a Fat-Btree for Distributing PostgreSQL

Yuta NAMIKI†, Kota KANBE††, and Haruo YOKOTA†,†††

† Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8552 Japan
†† Future Architect, Inc.
Art Village Osaki Central Tower, Osaki 1-2-2, Shinagawa-ku, Tokyo, 141-0032 Japan
††† Global Scientific Information and Computing Center, Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8550 Japan
E-mail: †namiki@de.cs.titech.ac.jp, ††kanbe.kota@future.co.jp, †††yokota@cs.titech.ac.jp

**Abstract**　We are trying to construct a shared-nothing parallel DBMS by storing data in a Fat-Btree, a parallel B-tree structure capable of dynamically changing data distribution. For page splits caused by inserting tuples in the Fat-Btree, we have proposed two types of methods to cooperate with the MVCC of PostgreSQL: One splits a page when it becomes full, while the other preparatory does when its occupation rate exceeds a threshold. In this paper, we implement the system which distribute PostgreSQL by using the Fat-Btree, and evaluate two page-splitting methods. Experimental results show that the preparatory-split method provides higher throughput than the other in frequent update requests.

**Key words**　Distributed DB, Fat-Btree, MVCC, PostgreSQL, Preparatory Split

## 1. まえがき

データベースに格納される情報量は近年急激に増大するとともに，その処理性能の向上が求められている [1]．この要求に対応するため，複数のプロセッサを用いてデータと処理を分散することでシステムの性能向上を図る並列データベースシステムが存在する．

並列データベースシステムは構成によりメモリ共有型，ディスク共有型，そして無共有型に分類される [2]．無共有型ではメモリやディスクへのアクセスが局所的に行われ，入出力処理を分散させるとともに，ネットワークには必要な処理が行われた容量の小さなデータのみを流すことができるため，スケーラビリティに優れているとされる．また，低価格の一般的なハードウェアを利用し，必要に応じてシステムの規模を柔軟に変更することが可能である．

システムを構成する PE（Processing Element）にデータを配分する際の方式としてラウンドロビン方式，ハッシュ方式，値域分割方式が挙げられる [3]．値域分割方式は PE ごとに格納するタプルの主キーの値域を定めておき，これに従ってタプルを分配する．本方式は完全一致問い合わせ，範囲問い合わせへの対応に有効だが，タプルの挿入・削除が繰り返されるとデータ量に偏りが生じ，これにより負荷にも偏りを引き起こす可能性があるという問題点がある．

ここで B-tree 構造を複数の PE で管理する並列 B-tree 構造を考える．並列 B-tree 構造の一つにデータ配置を動的に変更可能な Fat-Btree [4] が提案されている．Fat-Btree は従来の並列 B-tree 構造と比較して完全一致問い合わせ，範囲問い合わせが並列に高速処理できることが明らかにされている [5], [6]．また，値域分割方式によるデータ配分を行った際に偏りが発生しても動的なデータの再配置を行うことで対処することが可能となる．

本稿ではこの特徴を利用し，Fat-Btree を用いて DBMS を分散化することを考える．その構成としては，第一に Fat-Btree を複数の PE の DBMS に分散配置したタプルのインデックスとする構成，第二に Fat-Btree にタプルを格納する構成が存在する．我々は文献 [7] において，オープンソース DBMS である PostgreSQL [8] を用いて第一の構成を試作し，検索問い合わせにおいて従来の並列 B-tree 構造と比較して高いスループットが得られることを確認した．しかし，第一の構成ではタプル挿入処理を考えたとき，挿入したタプルに対応するインデックス項目の生成処理が必要がある．そのため，本稿ではこれが不要な第二の構成を考える．第二の構成では Fat-Btree のリーフページに PostgreSQL のデータページを格納することで分散化する．データページは PostgreSQL が各種記憶装置との入出力の単位とするもので，固定長の領域に複数のタプルと管理データを含むものである．

本稿で検討する第二の構成では，タプルのデータページへの格納方法に変化が生じ，新たにタプル挿入時に発生するデータページの分割処理を考える必要がある．オリジナルの PostgreSQL ではタプルの値域分割は考えられていないため，新たなタプルを格納する際に任意のデータページに格納することができる．一方，本構成では B-tree を用いることにより，タプルの主キーで格納するデータページが決定する．データページは固定長であるため，挿入の繰り返しによりタプルが単一のデータページに格納しきれなくなった場合のデータページを分割する処理を考える必要がある．

ページ分割処理に対し，我々は文献 [9] で手法の提案と見積もりを行った．提案手法は PostgreSQL の追記型 MVCC（Multi-Version Concurrency Control）[10] による同時実行制御を考慮した事前分割方式である．追記型 MVCC ではタプルの更新は新たなバージョンを生成することにより行い，一度生成されたタプルはデータページに残り続ける．事前分割方式ではこの特徴を利用し，データページが満杯になる前に PostgreSQL が同じページ内へタプルを挿入する処理を行うのと同時にページの分割処理を実行するものである．処理時間の見積もりにより，データページに対し単一の入出力要求のみが行われる理想的状況においては，事前分割方式が基本的な手法であるページ満杯時分割方式と比較してページ要求の際の待ち時間を削減可能であるとの結果を得た．

本稿ではこれらの手法の実装し，検証と評価を行う．まず Fat-Btree 単体における評価として，両手法によるページの取得・格納を行う際のレスポンスタイムを比較する．そして，PostgreSQL を分散化したシステムを構築し，スループットを比較してシステム全体の評価を行う．

本稿の構成を以下に述べる．2. において関連研究について述べ，3. で Fat-Btree を，4. で PostgreSQL における追記型 MVCC の実装を説明する．その後 5. で Fat-Btree を利用して PostgreSQL を分散化するシステムの構成を述べ，この構成において問題となる Fat-Btree におけるページ分割処理に対しこれまでに提案した手法を述べる．そして 6. で実験を行い，両手法を Fat-Btree 単体，ならびにこれを利用して PostgreSQL を分散化したシステム全体で評価する．最後に 7. においてまとめと今後の課題を述べる．

## 2. 関連研究

無共有型構成による複数 PE へのテーブルの分散配置機能の実装として，PostgreSQL に対応するものでは pgpool-II [11] や PostgresForest [12] がある．pgpool-II は PostgreSQL とデータベースを利用するクライアントアプリケーションの間に組み込まれる．そして与えられたタプルに対し格納先 PE を返すような関数を定義しておく．クライアントから分割されたテーブルに対する問い合わせを受け付けた際はこの関数により問い合わせを分割して適切な PostgreSQL サーバに並行して処理を発行する．PostgresForest はクライアントで動作する JDBC ドライバを拡張することで並列・分散機能を実装したものであり，タプルの特定の属性のハッシュ値を基に，格納先 PE を決定する．したがって，規模を拡張する際に全タプルの再配置が必要となる．分割されたテーブルの検索処理は pgpool-II と同様に行われる．これらの実装ではタプルはあらかじめ指定した基準により固定的に分割され，稼働中に動的にタプルの配置を変化させることは考慮されていない．

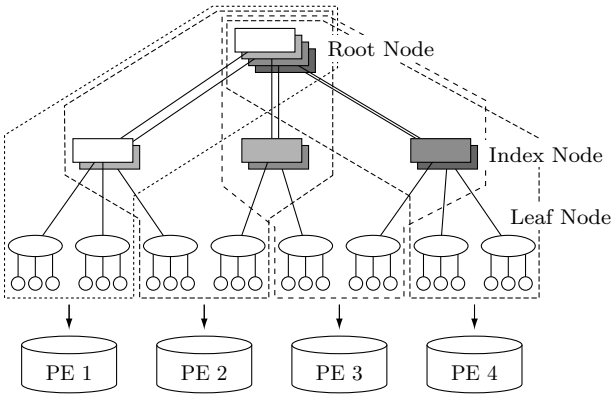



図 1　Fat-Btree の構造

分散 DBMS において並列 B-tree 構造を利用することは InfiniteDB [1] でも行われている．しかし，並列 B-tree 構造は別に分散管理されたタプルに対するインデックスとして利用されている．本研究では Fat-Btree をインデックスではなく，リーフページに DBMS のデータページを格納するために利用し，InfiniteDB とは並列 B-tree の利用法が異なる．また，InfiniteDB でもデータの動的な再配置は考慮されていない．

また，B-tree 構造においてページの分割・結合操作を 1 つのページを構成するタプル数の限界に達する前に事前に行うことは，並列ではない B-tree 構造においては PO-B$^+$-tree [13] で行われており，これを Fat-Btree に適用した並行性制御方式として PO-P [14] が存在する．これらの方式は B-tree に対する操作の並行性を向上させることを目的としており，インデックスページに対しても事前処理が行われる．一方，本研究は B-tree 構造単体ではなく，Fat-Btree 構造でデータページを管理する DBMS を考え，データページに対して事前分割操作を行うことで DBMS のスループット改善を図る．また，事前分割操作が発生するのはデータページを格納しているリーフノードの階層のみである点で異なる．

## 3. Fat-Btree の構造と特徴

本節では，本研究で DBMS のデータページを格納するために用いる，並列 B-tree 構造の一種である Fat-Btree の構造と特徴を述べる．

Fat-Btree は木を構成するページのコピーの配置を工夫することで従来の並列 B-tree 構造で発生しうる更新時のスループット低下や少数 PE へのアクセス集中を防ぐ並列 B-tree 構造である．その構造を図 1 に示す．Fat-Btree は B$^+$-tree 全体をページ単位で複数の PE に分配する．また，インデックスページは各 PE に配置されているリーフページへの経路上に存在するもののみを配置する．これにより，各 PE に格納されるのは B$^+$-tree のルートページから自 PE に存在するリーフページまでの部分木となる．

B-tree 構造においてはタプルの探索はルートページから行われる一方，タプルの更新はリーフページから行われ，挿入時にリーフページに空きがない場合にはその上位のインデックスページにも更新が発生する．したがって上位のページほど参照される確率は高く，下位のページほど更新される確率が高いと言える．Fat-Btree では多くのリーフページのパスに共有されるルートページに近いインデックスページほど多数の PE にコピーされる．特にルートページはすべての PE にコピーされるため，参照時に自 PE 内で処理を完結する可能性が高く，PE 間でのリクエストの転送に伴うオーバーヘッドを削減する．また，各 PE では格納しているリーフページの探索に必要の無いインデックスページを持たないため，各 PE でインデックスページのキャッシュを行った場合にヒット率を高く保つことが可能である．一方，更新確率の高い下位のインデックスページはコピーが少ないため，更新を行う際に同期が必要な PE 数を少なくすることができ，参照・更新の両操作に適した構造を持つ．

Fat-Btree における負荷の分散はページを PE 間で移動することで行う．負荷分散の手法としては各 PE に格納されるデータ量を均等にすることでデータ量の偏りの除去を図る手法のほか，ページごとにアクセス頻度を集計してアクセス量の偏りを除去する手法も提案されている [15].

これらの特徴から，本研究においては Fat-Btree を DBMS のデータページ格納のための構造として用いる．

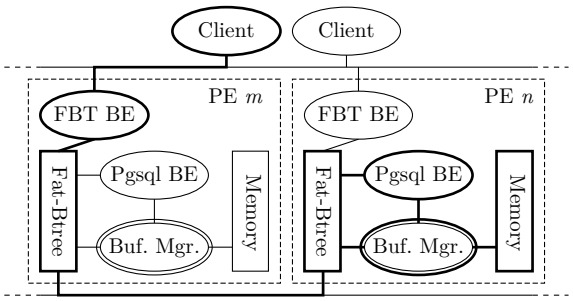
## 4. PostgreSQL における追記型 MVCC による同時実行制御

次に本研究で分散化の対象とする PostgreSQL における同時実行制御である追記型 MVCC について述べる．

DBMS では同時に複数のトランザクションが実行されてもデータの整合性を保つために同時実行制御が行われる．同時実行制御方式にはロックを用いた方式のほか，多版型同時実行制御（MVCC）方式が存在する．MVCC においてタプルの更新はそのタプルの新しいバージョンを生成することにより行う．タプルを参照する際は複数のバージョンの中から当該トランザクションに対して適切なバージョンを選択して読み出すことによりトランザクションの分離性を保証する．

MVCC の実装にあたり，タプルの過去のバージョンをタプルとしてそのまま保持する追記型と，最新バージョン以外の情報はロールバックセグメントと呼ばれる領域に格納し，旧バージョンの要求があった際はその情報から過去の内容を含むタプルを生成するロールバックセグメントを用いた方式が存在する．PostgreSQL では前者の追記型が用いられている．

以下に PostgreSQL における追記型 MVCC の実現のための機構を説明する．すべてのトランザクションには前後関係の存在するトランザクション ID（以下 XID と表記する）を割り当てる．そして各タプルにはそのタプルを生成したトランザクションの XID（以下 xmin と表記する）およびタプルを削除したトランザクションの XID（以下 xmax と表記する）を記録する．タプルの挿入時は xmin に挿入したトランザクションの XID を設定し，xmax は空とする．タプルの更新においてはタプルの内容が上書きされることは無く，更新前のタプルの論理的な削除と更新後の内容を含む新規タプルの挿入が行われる．すなわち，更新前の内容を含むタプルの xmax，および更新後の内容を含むタプルの xmin に更新を行ったトランザクション




FBT: Fat-Btree, BE: Backend
Buf. Mgr.: Buffer Manager

図 2　システムの構成

の XID を記録する．

トランザクションがあるタプルを参照したとき，自身の XID がそのタプルに記録された xmin の値以上であり，かつ xmax の値以下あるいは xmax が空であればそのタプルは基本的に可視となる．なお，実際には xmin，xmax に値を記録したトランザクションの状態（コミット，アボートあるいは同時実行中）も考慮された上で判定が行われる．

## 5. Fat-Btree を用いた DBMS の分散化とページ分割処理

本節では，本研究で構築する，Fat-Btree のリーフページに DBMS のデータページを格納することにより PostgreSQL の分散化を行うシステムの構成を述べる．そして，この構成において問題となる，タプル挿入時のデータページ分割処理を説明する．その後，この問題に対する対策としてこれまでに文献 [9] で提案し，本稿で評価を行う PostgreSQL における追記型 MVCC を考慮したページ分割手法を説明する．

### 5.1　システムの構成と問い合わせ処理の流れ

まずオリジナルの PostgreSQL と Fat-Btree を利用する本システムのデータ格納方法の違いを述べる．オリジナルの PostgreSQL においてタプルはリレーション別に固定長のデータページ単位にまとめられ，1 つのリレーションを構成する複数のデータページは 1 つのファイルとしてディスク上に保存される．一方，本システムでは図 3 にあるようにタプルの主キーの値を基に格納するデータページを決定し，それを Fat-Btree にリーフページとして格納する．この Fat-Btree を複数 PE で管理し，各 PE は自身が持つタプルに対する処理のみを担当することで PostgreSQL の分散化を実現する．

図 2 が本システムの構成である．本システムはオリジナルの PostgreSQL に対し，DBMS のデータページを格納する Fat-Btree，Fat-Btree とメモリ間でデータページの入出力を担当するバッファマネージャ（図中「Buf. Mgr.」），クライアントから SQL によるリクエストを受け付けるバックエンド（図中「FBT BE」）を追加実装したものである．

次に処理の流れを説明する．ここでは図 2 のクライアントから PE $m$ に対し，PE $n$ に格納されているデータに対する問い合わせを行う例を用いる．

1） 問い合わせの受付と Fat-Btree の探索

PE $m$ はクライアントから問い合わせを受け付け（図中「FBT BE」），`WHERE` 句で指定された主キーの値を用いて Fat-Btree の探索を開始する．探索とともに問い合わせは PE 間を転送され，リーフページ，すなわち対象タプルを含む PostgreSQL のデータページを持つ PE $n$ に到達する．

2） PostgreSQL での問い合わせ受付

PE $n$ ではクライアントからの問い合わせの処理を自ホストで動作する PostgreSQL に依頼する．問い合わせ処理のためのバックエンドプロセス（図中 PE $n$ 内の「Pgsql BE」）が生成される．

3） Fat-Btree からページを取得

バックエンドプロセスはバッファマネージャに対しメモリに対象ページを格納させる．バッファマネージャはメモリにキャッシュされたページの中から対象タプルを含むページを探索する．適切なページが無ければ Fat-Btree にアクセスしてページを取得する．

4） 処理の実行

メモリに格納されたページに対し必要な処理を行う．そして結果をクライアントに送る．

5） ページの書き戻し

メモリ内のページは変更が施されていれば定期的なチェックポイント，または他のページのキャッシュのためにメモリから追い出された時にバッファマネージャにより Fat-Btree に書き戻される．

### 5.2　Fat-Btree におけるページ分割処理

本構成では，オリジナルの PostgreSQL では必要の無かった，データページの分割処理を考える必要がある．

まず，オリジナルの PostgreSQL におけるタプルの挿入・更新操作を考える．`INSERT`，`UPDATE` によりデータページにタプルを書き込む必要が生じた場合は，タプルを挿入するリレーションを構成するデータページのうち，十分な空き容量を持つものを探索する．

一方，本システムではデータを値域分割するためにタプルの主キーの値によって格納すべきデータページが決定する．データページは固定長のため，タプル挿入の繰り返しによりページが満杯となった際は，新たなページを生成して既存のページ内のタプルを新たな値域に基づいて再分配する必要がある．

図 3 に分割の例を示す．簡単のため，ここでは 1 ページに格納可能なタプル数は 4 であるとしている．分割前において主キーの値が 1 から 3 のタプルがページ A に，主キーの値が 4 のタプルがページ D に格納されており，ページ A はこれ以上タプルを追加することは不可能な状態にある．追記型 MVCC のため，更新によりページ A における主キーの値が 2 のタプルのように複数のバージョンが存在しうる．ここで主キーの値が 3 であるタプルを更新したとする．このとき，更新後のタプルはページ A に追加されるべきだがここに空きが存在しないために，ページを分割し空き領域をつくることが必要である．この例では分割値を 3 と設定し，ページ A をページ B，C に分割している．

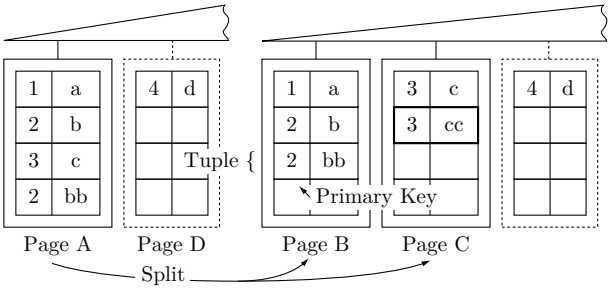



図 3　ページの分割

### 5.3　本稿で評価するページ分割方式手法

前小節では，本システムではオリジナルの PostgreSQL と異なり，主キーの値により格納すべきデータページが決定するためにデータページ分割処理を考える必要があることを述べた．

我々はデータページ分割処理に対し，PostgreSQL における追記型 MVCC を考慮した手法を提案し，見積もりを行った [9]．以下にそれらの手法を説明する．なお，以下では PostgreSQL のデータページを単にページと表記する．

**5.3.1**　ページ満杯時分割方式によるページへのタプル挿入

ページ満杯時分割方式はページの要求があった際に対象ページに空きが存在しないことが判明した時点でページの分割を開始する，本システム構成において最も基本的な手法である．本方式の手続きを図 4(a) に示す．

ページの要求時に対象ページに空きがない場合，ページ内のタプルを新規に作成した 2 ページに分割してページに空きを作成した上で，適切な方のページを返す．ページを分割する際の境界値は対象ページ内に存在するタプルの主キーとなる属性（図中の pk）の一覧からタプル数を均等に近い形で 2 分割できるような値とする．

このとき，ページの要求から空きのあるページが返却されるまでの間，メモリ上にタプルの追加対象となるページが存在しないため，PostgreSQL で処理を進めることは不可能であり，待ち時間が発生する．なお，書き戻し時にはページ分割に関わる処理はない．

**5.3.2**　事前分割方式によるページへのタプル挿入手続き

ページ満杯時分割方式におけるページ取得時の待ち時間削減を図るため，事前分割方式ではページ要求の応答の際にページの利用率を確認して閾値 $r$ を超えていればバックグラウンドでページの分割処理を開始する．手続きを図 4(b) に示す．

ページの要求に対し，返却するページ内に空きがあったとしても，それが閾値以下になった時点でページを返すと同時に Fat-Btree でページの分割を開始する．これにより PostgreSQL のメモリに読み込まれたページに対してタプルの挿入を行うのと同時に，Fat-Btree では分割が発生すると予想されるページの分割処理を進行させることができる．したがってページ満杯時分割方式において空きがないページを要求した際に発生する，ページ分割終了までの待ち時間の削減が可能となる．ただし，返すページがすでに満杯でタプルの挿入が不可能なときはページ満杯時分割方式と同様に分割終了を待つものとする．こうしたページは取得時の使用率が閾値以下であったページを，満杯まで書き込みを行って書き戻したときに発生する．

ページを書き戻す際，取得時に分割操作を行ったページの場合は PostgreSQL のメモリ上で行われた変更を先行分割済みのページに反映させる処理が必要である．格納するページ中の更新が行われたタプルの更新前の内容を含むバージョン，および削除されたタプルには新たに xmax の値が設定されるため，先行分割したページに含まれる当該タプルの xmax を同様に設定する（図 4(b) の 15–16 行目）．また，更新が行われたタプルの更新後の内容を含むバージョン，および新たに挿入されたタプルは先行分割済みのページのうち適切な方に追加する（図 4(b) の 17–21 行目）．これらの処理が完了すると，分割したページは Fat-Btree に挿入される．

本方式の先行分割操作は追記型 MVCC を考慮したものである．追記型 MVCC では既存ページ上のタプルの書き換え・削除が発生しないため，取得時に先行分割したページ内のタプルは書き戻し時も必ず存在する．また，追記型 MVCC においては更新においても新たなバージョンのタプルの挿入が発生するため，更新が発生する環境においては先行分割を行うことの効果が期待できる．

なお，ページ取得時の分割処理は以下の状況を除き，PostgreSQL をブロックすること無く実行可能である．ブロックされるのは Fat-Btree 内でページの分割が行われている期間（図 4(b) の 7–13 行目）に行われたページの書き戻し処理，および，書き戻し要求に伴い，差分の分割を行っている期間（図 4(b) の 15–22 行目）に行われたページの取得要求である．

**5.3.3**　両手法による PostgreSQL の待ち時間の見積もり

我々は文献 [9] において両手法を利用した際の PostgreSQL における待ち時間の見積もりを行った．その結果，データページに対し単一の入出力要求のみが行われる理想的状況においては，事前分割方式によりページ要求の際の待ち時間を削減可能であるとの結果を得た．

しかし，見積もりにおいて使用したパラメータの値を確定することができなかったため，この差を数値で表すことが不可能であった．本稿の続く節では両方式を Fat-Btree に実装してレスポンスタイムを測定することで検証を行う．

## 6.　実　　験

両ページ分割手法を Fat-Btree に実装し，実験と評価を行った．評価は Fat-Btree 単体，およびこの Fat-Btree を利用して PostgreSQL の分散化を実現したシステム全体に対して行う．まず 6.2 で両ページ分割手法を Fat-Btree に実装し，動作パターンごとにレスポンスタイムを計測して Fat-Btree 単体での評価を行う．その後，6.3 で PostgreSQL 8.1.4 を基にしたシステムに対し，参照・更新要求の混在した問い合わせを実行してシステム全体の評価を行う．

### 6.1　実 験 環 境

実験は表 1 に示す構成のブレードサーバを複数台用いて行った．各ブレードサーバはシャーシ内蔵スイッチを経由して Gigabit Ethernet で接続されている．Fat-Btree の実装は Java により記述された自律ディスク [16], [17] における実装に対し，

```
PostgreSQL から主キーとなる属性 pk の値が k であるようなタプルの挿入のため，ページ取得要求が発生
1:  p := k をキーに探索して得た，タプルを格納すべきページ
2:  if p にタプルを納めるための空きが存在する
3:      p を返す
4:  else
5:      npl, npr := p 内のタプルの分割先とするために生成した新規ページ
6:      m := p 内のタプルを均等に近く 2 分割する pk の値
7:      for t in p 内のすべてのタプル
8:          if t の pk の値 < m
9:              npl に t を追加
10:         else
11:             npr に t を追加
12:         p を npl で上書きし，m をキーにして npr を挿入
13:     if k < m
14:         npl を返す
15:     else
16:         npr を返す
PostgreSQL から p の書き戻し要求が発生
17:  p を上書き
```

(a) ページ満杯時分割方式の手続き

```
PostgreSQL から主キーとなる属性 pk の値が k であるようなタプルの挿入のため，ページ取得要求が発生
1:  p := k をキーに探索して得た，タプルを格納すべきページ
2:  if p が満杯である
3:      ページ満杯時分割方式と同様にページ分割を行い，適切なページを返す
4:  else
5:      p を返す
6:      if p の利用率が r 以上である
7:          npl, npr := p 内のタプルの分割先とするために生成した新規ページ
8:          m := p 内のタプルを均等に近く 2 分割する，pk の値
9:          for t in p 内のすべてのタプル
10:             if t の pk の値 < m
11:                 npl に t を追加
12:             else
13:                 npr に t を追加
PostgreSQL から p の書き戻し要求が発生
14:  if p はページ取得時に先行分割を行った
15:     for t in クライアントにおいて削除された p 内のタプル
16:         npl または npr に存在する同タプルに削除情報を反映
17:     for t in クライアントにおいて挿入された p 内のタプル
18:         if t の pk の値 < m
19:             npl に t を追加
20:         else
21:             npr に t を追加
22:         p を npl で上書きし，m をキーにして npr を挿入
23:  else
24:     p を上書き
```

(b) 事前分割方式の手続き

図 4　ページ取得・格納時の手続き

表 1　実験システムの構成

| | |
|---|---|
| Blade server: Sun Fire B200x Blade Server | |
| CPU | Low Voltage Intel Xeon 2.0 GHz × 2 (Hyper-Threading disabled) |
| Memory | PC 2100 registered ECC SDRAM DIMM 2 GB |
| HDD | Toshiba MK3019GAXB (2.5″, 30 GB, 5,400 rpm) |
| OS | Linux 2.4.20 (Red Hat Linux 9) |
| Java VM | Java HotSpot Server VM 1.5.0_09 |
| Network | TCP/IP over 1000BASE-T |
| Chassis: Sun Fire B1600 Blade System Chassis | |
| Switch | 10 GB internal switch fabric |

表 2　実験におけるパラメータ

| | |
|---|---|
| Client program | |
| Number of client nodes | 8 |
| Number of threads per node | 8 |
| Fat-Btree | |
| Max number of entries in an index node | 64 |
| Concurrency control method | INC-OPT [18] |
| PostgreSQL | |
| Number of PEs | 1, 2, 4, 8 |
| Page size | 8,192 byte |
| Tuple size | 132 byte |
| Number of tuples in a page | 60 |
| Number of tuples | 100,000 |

PostgreSQL とのページ入出力機構を追加実装したものを利用した．また，本実験で用いた主なパラメータを表 2 に示す．

### 6.2 Fat-Btree 単体に対する実験：ページ取得・格納操作のレスポンスタイムの比較

#### 6.2.1 概　　要

本小節では両ページ分割手法を実装した Fat-Btree に対してページの取得・格納操作を行い，その際のレスポンスタイムを比較することで Fat-Btree 単体における評価を行う．

1 台のブレードサーバで Fat-Btree 構造により PostgreSQL のページを格納するプログラム，およびこれにページの取得・格納要求を行うクライアントプログラムを実行する．

以下に取得時，格納時の動作のパターンを考える．

取得時に関しては両ページ分割手法についてそれぞれページ分割の有無が考えられる．ページ取得時に分割の処理が行われるのはページ満杯時分割方式においては既に満杯のページを取得する際，事前分割方式においては Fat-Btree から取得し，クライアントに返却しようとするページの使用率が閾値を超えていたときである．

格納時に関しては事前分割方式では格納しようとするページの先行分割の有無が考えられる．先行分割が行われている場合は，取得から格納の間にクライアントで行われた変更を先行分割したページに反映させる処理が必要となる．一方，ページ満杯時分割方式では格納しようとするページの取得時における分割の有無はページ格納時の処理と無関係であるため，場合分けはない．

以上の各動作パターンについてそれぞれ 100 回操作を行い，

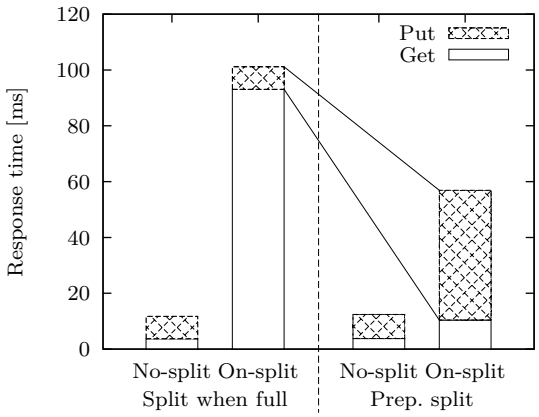



図 5　ページ取得・格納操作におけるページ満杯時分割方式（Split when full）と事前分割方式（Prep. split）のレスポンスタイム

平均レスポンスタイムを比較する．

**6.2.2**　結果と考察

実験の結果を図 5 に示す．まず両手法のページ取得時のレスポンスタイムを比較する（図中「Get」）．分割を伴わないページ取得の場合，どちらの手法でも処理は同一であるためレスポンスタイムに差はみられない．しかし分割を伴う場合を比較すると事前分割方式ではページ分割処理の完了を待つ必要がないため，ページ満杯時分割方式と比較して分割処理に要する時間に相当する分を除いた約 11 %まで削減されることが確認できた．

次にページ格納時のレスポンスタイムを比較する（図中「Put」）．すでに述べたようにページ満杯時分割方式では分割を伴う格納という動作パターンは存在しないため，図では分割を伴わないときを同じ値としている．ページ満杯時分割方式と事前分割方式で取得時に分割を伴わなかった場合，両者の処理は同一であるため，レスポンスタイムに差はみられない．しかし，事前分割方式で取得時に分割を伴った場合は，ページの取得から格納までにクライアントで行われた変更点を先行分割したページに反映させる処理が必要である．このため，ページ満杯時分割方式と比較してレスポンスタイムは悪化する．

以上の結果をまとめると，分割を伴う場合，事前分割方式の取得時のレスポンスタイムは短縮され，格納時は悪化するという結果となる．ここで，後述する 6.3 と同様の実験により得た平均ページ分割発生率を利用して両手法で単一ページの取得の格納を行った際の平均的なレスポンスタイムを見積もる．実験により得た平均ページ分割発生率はページ満杯時分割方式で約 18 %，事前分割方式で約 31 %であった．この値を用いて単一ページの取得と格納の際の平均的なレスポンスタイムを見積もると，ページ満杯時分割方式で約 28 ms，事前分割方式で約 26 ms となり，約 6 %の削減が可能であるとの結果が得られた．

### 6.3　システム全体に対する実験：PostgreSQL 分散化時のレスポンスタイムとスループットの比較

**6.3.1**　概　　要

前小節で述べた Fat-Btree における両ページ分割手法の評価を踏まえ，PostgreSQL を分散化した実並列環境における両ページ分割手法の評価を行う．

ブレードサーバ 1 台につき，PostgreSQL，Fat-Btree およびクライアントプログラムの 3 種類のプログラムを必要に応じて実行する．以下，ブレードサーバ 1 台を PostgreSQL および Fat-Btree について述べるときは「PE」，クライアントプログラムについて述べるときは「ノード」と表記する．

初期データとして主キーに 1 から 100,000 の値を持つタプルを用意し，これを値域分割方式によりシステムを構成するすべての PE に，各ページの初期ページ使用率が 0.5 となるように均等に分割して配置する．なお，表の定義は PostgreSQL に附属するベンチマークツールである pgbench で用いられる accounts 表に倣った．また，PE の台数は 1, 2, 4, 8 台と変化させた．

クライアントプログラムはデータを持たせる PE の台数にかかわらず 8 ノードで実行する．各ノードで動作するクライアントプログラムは同時に 8 スレッドからシステムにアクセスをする．したがってシステム全体では 8 ノードから 64 の接続を同時に受けることになる．

各スレッドではテーブル全体から主キーの値をランダムに選択して 1 タプルを参照する問い合わせ，または主キーの値をランダムに選択して主キーではない属性の値を更新する問い合わせを 120 回順次システムに対して発行する．更新要求の割合は 0.0 から 1.0 の間で 0.2 間隔で設定して実験を行った．また，事前分割方式における閾値は 0.8 とした．閾値の決定方法については今後の検討が必要である．

**6.3.2**　結果と考察

まず，システムを構成する PE の台数を変化させたときのスループットを図 6 に示す．

参照のみの環境（図中「Select only」），および更新要求の割合が 0.8 のとき（図中「Upd. ratio: 0.8」）のいずれにおいても両ページ分割手法に PE 数に応じたスループットの増加が確認できた．なお，参照のみの環境において両手法のスループットに差が見られないのは，参照のみのためページ分割が必要なく両手法の処理に差がないためである．また，更新要求の割合が 0.8 の環境において 1–4 PE 構成で両手法のスループットに差は見られないのは各 PE でアクセスの集中によりページ分割処理以外の処理に大きな時間を取られ，ページ分割手法による差が見えないことによるものと考える．

以上の結果は，PE 数を $n$ として一次関数 $38n + 93$ に沿った結果となることから，本システムでは両手法ともに少なくとも 8 PE 構成までで $O(n)$ のスループットが得られるといえる．

また，8 PE 構成において更新要求の割合の変化させたときのページ満杯時分割方式および事前分割方式のスループットを 95 %信頼区間とともに図 7 に示す．信頼区間は上記の実験を 3 回繰り返した結果から得たものである．この図よりどちらの手法でも更新要求の割合の増加に伴いスループットは低下するが，更新要求の割合が高い環境においては事前分割方式により高いスループットが得られていることがわかる．更新が少ない環境において，事前分割方式は不必要な先行分割を引き起こす．これによりページ取得のためのレスポンスタイムが悪化

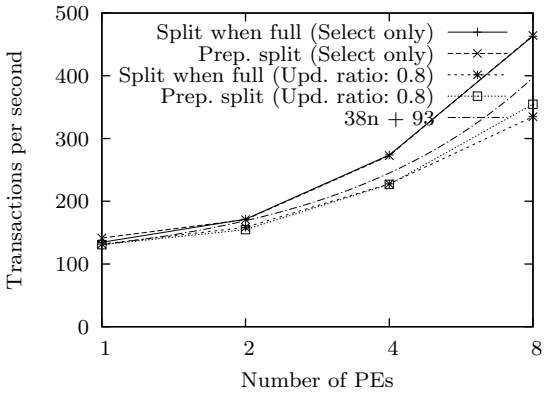



図 6　PE 数の変化とスループット

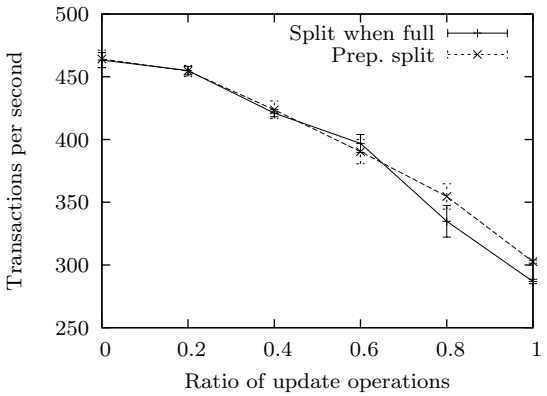



図 7　8 PE 構成時における更新要求の割合とスループットの変化

するが，その影響は少なく，ページ満杯時分割方式とほぼ同じスループットが得られている．一方，更新が多い環境においては事前分割方式によるページの先行分割が有効に利用され，図 5 で示したようにより短いレスポンスタイムで分割されたページが得られることからページ満杯時分割方式と比較して高いスループットを得ることができると考えられる．

これらの結果より，Fat-Btree による PostgreSQL 分散化環境においてデータを分散配置する PE の台数が大きく，更新要求が多い環境においてはページ分割方式に事前分割方式を用いることで，ページ満杯時分割方式と比較して高いスループットが得られるという結果が得られた．

## 7. まとめと今後の課題

本稿では我々がこれまでに提案した Fat-Btree における追記型 MVCC による同時実行制御を考慮したページ分割手法の評価を行った．提案手法であるページ利用率が閾値を超えた時点で分割を行う事前分割方式，ならびにページ満杯時に分割を行うページ満杯時分割方式を実装し，Fat-Btree 単体，および PostgreSQL でこの Fat-Btree による分散化を行ったシステム全体に対して実験を行った．

Fat-Btree 単体の実験により，事前分割方式は分割を伴うページの取得時のレスポンスタイムを削減可能であることを確認した．また，格納時には悪化するが分割発生率を考慮すれば，事前分割方式により単一ページの取得・格納に必要なレスポンスタイムは削減可能なことを示した．そして Fat-Btree を用いて PostgreSQL を分散化したシステムを実装し，実験を行った．その結果，データを分散配置する PE の台数が大きく，問い合わせ中に更新要求が多く含まれる環境において，Fat-Btree 単体での見積もりを反映し，事前分割方式がより高いスループットをもたらすことを確認した．

今後の課題として，より大規模な環境における検証，両ページ分割方式の容量の観点からの評価が必要である．また，文献 [7] で検討を行った二次インデックスへの対応をより詳細に検討することが必要である．さらに Fat-Btree の機構を利用したタプルの動的な再配置への対応も今後の課題である．

## 文　献


[1] J. Li, H. Gao, J. Luo, S. Shi, and W. Zhang, “InfiniteDB: a PC-cluster based parallel massive database management system,” Proc. of the 2007 ACM SIGMOD Int’l Conf. on Management of Data, pp.899–909, 2007.

[2] M. Stonebraker, “The case for shared nothing,” Database Engineering Bulletin, vol.9, no.1, pp.4–9, 1986.

[3] D. DeWitt, and J. Gray, “Parallel database systems: the future of high performance database systems,” Commun. ACM, vol.35, no.6, pp.85–98, June 1992.

[4] H. Yokota, Y. Kanemasa, and J. Miyazaki, “Fat-Btree: An update-conscious parallel directory structure,” Proc. of the 15th ICDE, pp.448–457, March 1999.

[5] 金政泰彦，宮崎純，横田治夫，“並列データベースシステムにおける更新を考慮したディレクトリ構成,” 電子情報通信学会技術研究報告，vol.97，no.416，pp.63–68，1997.

[6] 風戸広史，横田治夫，“並列ディレクトリ構造 Fat-Btree におけるレンジ問い合わせの取り扱い,” Proc. of DEWS，2001.

[7] 並木悠太，神戸康多，小林大，横田治夫，“並列 B-tree 構造 Fat-Btree を用いた PostgreSQL の分散検索の試み,” 電子情報通信学会技術研究報告，vol.107，no.131，pp.473–478，2007.

[8] “PostgreSQL,” `http://www.postgresql.org/`.

[9] 並木悠太，神戸康多，小林大，横田治夫，“Fat-Btree を用いた PostgreSQL 分散化におけるページ分割手法の検討,” 電子情報通信学会技術研究報告，vol.107，no.254，pp.1-6，Oct. 2007.

[10] P.A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency control and recovery in database systems, Addison-Wesley Publishing Company, 1987.

[11] “pgpool-II,” `http://pgpool.projects.postgresql.org/`.

[12] NTT DATA Corporation, “PostgresForest,” `http://www.nttdata.co.jp/services/postgresforest/`.

[13] Y. Mond, and Y. Raz, “Concurrency control in $B^+$-trees databases using preparatory operations,” Proc. of 11th Int’l Conf. on VLDB, pp.331-334, 1985.

[14] D. Amarmend, M. Aritsugi, and Y. Kanamori, “PO-P: A concurrency control protocol for parallel B-trees,” Transactions of Information Processing Society of Japan, vol.45, no.SIG 14 (TOD 24), pp.30–38, Dec. 2004.

[15] 鈴木裕通，横田治夫，“並列ディレクトリ構造 Fat-Btree における負荷分散の手法とその実装,” Proc. of DEWS，2000，4B-4.

[16] H. Yokota, “Autonomous Disks for advanced database applications,” Proc. of the Int’l Symposium on Database Applications in Non-Traditional Environments, pp.435–442, 1999.

[17] 風戸広史，横田治夫，“自律ディスクへの Fat-Btree の実装,” 情報処理学会研究報告，vol.2001，no.70，pp.45–52，情報処理学会，July 2001，2001-DBS-125.

[18] J. Miyazaki, and H. Yokota, “Concurrency control and performance evaluation of parallel B-tree structures,” IEICE TRANSACTIONS on Information and Systems, vol.E85-D, no.8, pp.1269–1283, Aug. 2002.