社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143 (80)
2007／7／4

並列B-Tree構造Fat-Btreeを用いたPostgreSQLの分散検索の試み

並木 悠太† 神戸 康多†† 小林 大†,††† 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 W8-82
†† 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
††† 日本学術振興会特別研究員DC
†††† 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,daik}@de.cs.titech.ac.jp, ††kanbe.kota@future.co.jp, ††††yokota@cs.titech.ac.jp

あらまし データベースの規模が急激に増大する中で，処理性能を高める手段として並列データベースシステムが存在する．オープンソースDBMSとして広く用いられているPostgreSQLに対する分散化の試みとしては，データ配置を固定し，上位層を設けることで問い合わせを分割する方式等が提案されているが，データ配置を動的に変更可能な分散インデックスを用いたものはこれまでに無い．我々は分散インデックスとしてFat-Btreeを用いて複数のPEに分散したデータを検索する方法を検討し，部分的な試作を行った．実験の結果，高頻度の問い合わせ要求に対し，インデックスを単一PEで管理する方式と比較してFat-Btreeの利用が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 並列B-Tree，Fat-Btree，PostgreSQL，分散検索

An Approach of Using a Parallel B-Tree Structure, Fat-Btree, in PostgreSQL for Distributed Retrieval

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

† Department of Computer Science, 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
††† Research Fellow (DC), Japan Society for the Promotion of Science
†††† Global Scientific Information and Computing Center, Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8550 Japan
E-mail: †{namiki,daik}@de.cs.titech.ac.jp, ††kanbe.kota@future.co.jp, ††††yokota@cs.titech.ac.jp

Abstract Parallel database systems have been developed to support the performance demands on large databases. To enable distributed retrieval on multiple PostgreSQL servers with the current technology, data are distributed into PEs based on statically settled values. There has been no challenges using any distributed index which can dynamically change data distribution for retrieving data using index. We consider a retrieving function for data distributed on multiple PEs using a Fat-Btree, a parallel B-tree structure. We implement a part of it and perform experiments. The experimental result shows that data retrieval using the Fat-Btree provide higher throughput than using the non-distributed B-tree on a PE, for frequent retrieval requests.
Key words Parallel B-Tree, Fat-Btree, PostgreSQL, Distributed Retrieval

1. まえがき

近年データベースに格納される情報量は急激に増大している．こうしたデータ量の増加のなかでも処理性能を高めることが求められており，これに対応するための手段として複数のプロセッサを用いて処理を分散させ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[1]．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を行った後の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．そのため，低価格の一般的なハードウェアを利用し，必要に応じた柔軟なシステムの規模の変更に対応することが可能である．

無共有型計算機群において，計算資源の利用を分散させるために処理は対象データが格納されている各 PE（Processing Element）上で並列に実行されることが望ましい．処理が特定の PE に集中して負荷の偏りが発生すると，その PE がボトルネックとなり全体の処理性能の低下を引き起こす．したがって各 PE の負荷を均等化することは処理性能の向上につながり，そのためにデータ配分方式が重要となる [2]．

データ分配方式としては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]．ラウンドロビン方式ではデータの偏りがないが，システムの規模を拡張する際にデータ全体の再配置が必要であり，拡張性に問題がある．また，ハッシュ方式ではデータの偏りが少ないものの，範囲問い合わせへの対応や，ラウンドロビン方式と同様，拡張性に問題がある．値域分割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，データの挿入・削除が繰り返されるとデータの配置に偏りが生じ，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．

現在，多くの DBMS において無共有計算機群を用いて容量や処理量の大きいテーブルを分割して処理性能の向上を図るテーブルパーティショニングを利用することが可能であり，例えばオープンソース DBMS である PostgreSQL [4] に対しては pgpool-II [5] や，PostgresForest [6] などのミドルウェアが提案されている．これらは PostgreSQL に上位層を設けて分散したテーブルに対する問い合わせを PE に分配し，各 PE で並行処理をさせることで検索性能の向上を図るものである．しかし，データはあらかじめ指定した基準により固定的に分割され，負荷分散のための動的なデータの再配置は行われない．また，分割するテーブルに対してインデックススキャンを行う際は，まず，ミドルウェアにより問い合わせが PE 毎のものに書き換えられ，個々の PE に問い合わせが受け付けられた時点から，1 PE 内の局所的なインデックスを利用した処理が行われる．

本稿では値域分割方式により分割されたテーブルのインデックスを，データ配置を動的に変更可能な 1 つの分散インデックス構造を用いて格納し，これを利用したインデックススキャンを行う方法を検討する．分散インデックス構造には，一般的に用いられている B-tree 構造を複数の PE で管理する並列 B-tree 構造を利用する．

データ配置を動的に変更可能な並列 B-tree 構造を用いることで，値域分割方式によるデータ配置戦略を取った場合に偏りが発生しても対処することが可能となる．しかし，これまでの並列 B-tree ではインデックスの更新時にスループットの低下や，少数の PE へのアクセス集中といった問題が生じる．

これらの問題を解決するための新しい並列 B-tree 構造として Fat-Btree [7] が提案されている．Fat-Btee は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確率モデルを用いた検証 [8]，nCUBE 3 上 [9] および LAN 環境での PC クラスタ上への実装 [10] による実験により明らかにされている．

本稿では並列 B-tree 構造として，Fat-Btree を利用した PostgreSQL の分散検索を行う方法について検討を行う．そして部分的な試作を行い，インデックスを単一の PE で管理する方式と比較して，スループット向上の効果を確認する．

以下，2. において関連システムについて述べ，3. で Fat-Btree を含む並列 B-tree 構造を説明する．その後 4. で PostgreSQL において並列 B-tree 構造を利用した分散検索を行う方法を検討し，5. で実験の概要を述べ，並列 B-tree 構造として Fat-Btree を用いる方式と単一 PE で管理する方式の比較と考察を行う．最後に 6. において，まとめと今後の課題を述べる．

2. 関連システム

並列データベースで広く用いられているものにディスク共有型をとる Oracle Real Application Cluster [11] がある．ディスク共有型のため，負荷分散のためにデータ配置を変更する必要は無いが，スケーラビリティの点ではノード間のキャッシュの同期の問題から一般的には 2–4 ノード，最大でも 16 ノードが現実的な範囲であると言われている [12]．

無共有型構成によるテーブルパーティショニング機能の実装としては，PostgreSQL に対応するものでは pgpool-II や PostgresForest がある．

pgpool-II [5] 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むコネクションプーリングサーバであるが，テーブルパーティショニングの機能も持つ．データに対し格納先 PE を返すような関数を定義し，システムデータベースと呼ばれるユーザのデータを格納するものとは別のデータベースのテーブルに分割されたテーブルに関するその他の情報とともにこれを登録する．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 pgpool-II はシステムデータベースを参照し，問い合わせを分割して必要な PostgreSQL サーバに並行して処理を発行し，結果を集めてクライアントにデータを返す．

PostgresForest [6] は 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る．PostgresForest ではタプルの特定のコラムのハッシュ値を基に，格納先 PE を決定する．したがって，上で述べたように規模を拡張する際に全データ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．

これらの実装ではデータはあらかじめ指定した基準により固定的に分割され，稼働中に負荷に応じて動的にデータの配置が変化することはない．また，タプルの更新により分割の基準に従えば他 PE に移動させなければならないタプルが発生しても自動的に再配置されることはないため，このような処理が必要な場合はアプリケーション側から `DELETE`，`INSERT` を発行して PE 間の移動を手動で行う必要がある．また，これ以外のシステムにおいてもインデックスとして分散インデックス構造を用いたものはこれまでに無い．

3. 並列 B-tree の構成方式

3.1 並列 B-tree 構造とその構成

分散したテーブルに対するインデックスを 1 つの並列 B-tree 構造で管理するとき，その構成にはいくつかの方式を考えることができる．従来の並列 B-tree 構造としては，PE 群の中で B-tree を管理する 1 台の PE を定め，その他の PE は必要な場合にその B-tree を管理する PE に問い合わせる方式，およびすべての PE で B-tree のコピーを持つ方式がある．本稿では [7] と同様に前者を SIB（Single Index B-tree）方式，後者を CWB（Copy Whole B-tree）方式と表記する．

SIB 方式では B-tree 構造が単一の PE のみに存在する．したがってページのコピーが存在しないため，更新時に PE 間で同期を取る必要がない．しかしリクエストが単一の PE に集中するため，参照・更新操作ともに高いスループットを得ることはできない．

一方，CWB 方式では検索操作のみの環境においては PE 内のコピーを参照すればよく，PE に閉じた処理が可能なためにその他の並列 B-tree 構造と比較して高いスループットが得られる．しかし，更新が発生した場合はすべての PE で同期を取ってページの更新をする必要があり，更新操作が混在する環境においてこの同期の処理がボトルネックとなりスループットが低下する．

このほか，ルートノードのみすべての PE でコピーを持ち，任意の階層から枝単位でのマイグレーションを実現する並列 B-tree 構造として aB$^+$-tree [13] が存在するが，これは後述する Fat-Btree の特殊な形態と見なすことができる．

DBMS におけるインデックス構造として使用することを考慮すると，並列 B-tree 構造には参照操作で高いスループットが得られるだけでなく，更新操作によりスループットの低下が発生しないことが求められるが，SIB 方式，CWB 方式はこれを満たさない．

3.2 Fat-Btree

Fat-Btree はページのコピーの配置を工夫することで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 であり，その構造を図 1 に示す．Fat-Btree は B$^+$-tree 全体をページ単位で PE 間で分配する並列 B-tree の一種であり，リーフページ（データページ）を各 PE に均等に分配する．ディレクトリ部分であるリーフページ以外は，各 PE に配置されているリーフページへのアクセスパスを含むインデックスページのみを再帰的に配置する．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均等に分配されたリーフページまでの部分木となる．

B-tree 構造においてデータ項目の探索はルートページから行われる一方，データ項目の更新はリーフページから行われ，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．探索を高速に処理するためには各 PE ですべてのページのコピーを持つ CWB 方式が有利であるが，既に述べたように更新コストが非常に高い．

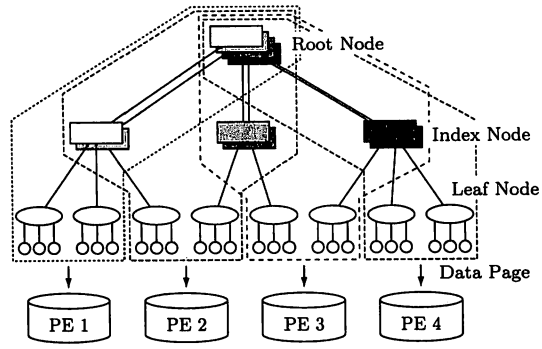

図 1 Fat-Btee の構造

Fat-Btree では多くのリーフページへ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，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，探索・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データページを PE 間で移動することによって行う．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偏りの除去を図る手法のほか，ページの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 [14]．

以上のような特徴から，並列 B-tree 構造を利用したデータベースの分散検索の実現に Fat-Btree は適しているため，本研究ではこれを分散検索の実装に用いる．

4. PostgreSQL における分散インデックス構造を用いた分散検索の実現

本節では並列 B-tree 構造を利用して複数の PostgreSQL に分散したデータを検索するための構造と実装について述べる．

なお，本実装はあくまで部分的な試作であり，1 テーブルに対し分散インデックスを利用したインデックススキャンを行い，結果のタプルに含まれる値を問い合わせを発行したクライアントのコンソールに出力する機能しか持たないことに注意されたい．

4.1 PE 内のプロセスの構成

並列 B-tree 構造として Fat-Btree を利用した際のプロセスの構成を図 2 に示す．以下の小節では図中の構成要素にそれぞれ対応して，その実装について述べる．

4.1.1 ローカルバックエンド

PostgreSQL はプロセスベースの処理を行っており，クライアントからの接続要求が `postmaster` と呼ばれるリスナープロ

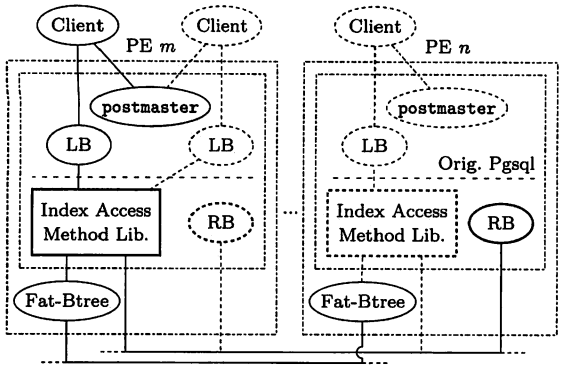

図 2　PE 内の構成

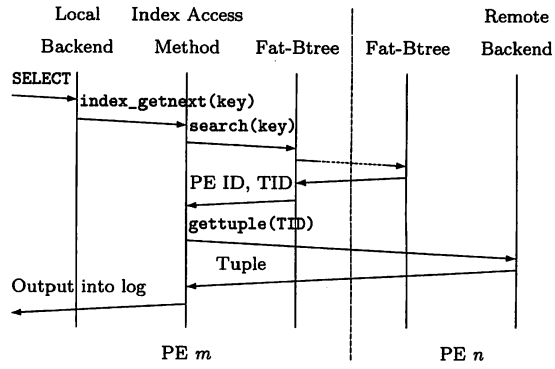

図 3　処理の流れ

セスで受け付けられると，クライアントごとに新たなプロセスが生成され，その後の通信が行われる．このプロセスはバックエンドと呼ばれるが，後述するリモートバックエンドと区別するため，ここではローカルバックエンドと表記する．

4.1.2　リモートバックエンド

分散検索を行うためには，他 PE からの問い合わせに対してデータを返すプロセスが必要である．そのためにローカルバックエンドと同様に `postmaster` からリクエストごとに分離するバックエンドプロセスを実装した．ここでは他 PE からテーブルを特定するためのリレーション ID とリレーション内のタプルを特定するための TID（Tuple ID）を受け取り，対応するタプルのデータをクライアントに返す処理を行う．以降これをリモートバックエンドと表記する．

本稿における実装ではリモートバックエンドが新たなリクエストを受けた時点で，PE にローカルなトランザクションが開始され，処理が行われる．複数の PE に関係するようなトランザクションの処理は今後の課題である．

4.1.3　インデックスアクセスメソッドライブラリ

インデックスアクセスメソッドライブラリは PostgreSQL に対し，並列 B-tree 構造へのインデックス項目の挿入・探索の機能を提供する関数を実装したライブラリである．この他，リレーション ID およびインデックスの検索により得た PE 識別子（PE ID）と TID を利用して目的のデータを持つ PE のリモートバックエンドに対してタプルを要求し，取得したタプルの値をログ(注1)に出力する処理を行う．

4.2　並列 B-tree 構造

通常の PostgreSQL においてインデックスには，インデックスを作成した属性の値をキーとして，タプルを一意に識別するための TID が格納される．複数 PE に分散したテーブルに対して 1 つのインデックスを作成する場合，TID とともにデータを格納している PE を識別する必要がある．そのため，PE ID もインデックスに格納し，データ値から PE ID と TID を利用したタプルへのアクセスを可能にする．

図では並列 B-tree 構造として Fat-Btree を用いているため，すべての PE でこのプロセスが動作する．しかし SIB 方式を用いた場合は単一の PE のみでこのプロセスが動作し，その他の PE からのアクセスが集中することになる．

なお，インデックス構造は本来 PostgreSQL の内部に存在すべきであるが，ここでは実装上の都合により外部に単独のプロセスとして起動している．

4.3　処理の流れ

これまで述べたプロセスおよびライブラリ間の処理の流れを Fat-Btree を利用した場合を例に図 3 に示す．

1）　クライアントからの問い合わせの受付

まず，クライアントアプリケーションは適当な PE に対して `SELECT` 文を発行する．以降この問い合わせを受け付けた PE を PE m とする．このとき PE m では処理のためにローカルバックエンドプロセスが新たに生成される．

2）　インデックスからの PE ID，TID の取得

次にローカルバックエンドからインデックスアクセスメソッドライブラリ内に定義されたインデックス探索関数 `index_getnext()` を呼び出す．この関数内ではインデックスに対して検索要求 `search()` を行い，対応する PE ID と TID を取得する．分散インデックス構造に Fat-Btree を用いた場合，木の探索は PE m に存在するルートノードのコピーから始まり，インデックスノードを辿り必要ならば隣接する PE へリクエストを転送しながらリーフノードを持つ PE に到達する．以降ではリーフノードを持つ PE を PE n とする．PE n は結果を PE m に返す．なお，図では隣接した PE 間の通信となっているが，リーフノードが自 PE にある場合，あるいは 3 以上の PE を経由してリーフノードに到達することもあり得る．

3）　TID を元にリモートバックエンドからタプルの値を取得

インデックスから PE ID と TID を受け取った PE m は PE ID で表される PE のリモートバックエンドに接続し，`gettuple()` で表されるように，TID を渡して対応するタプルを取得する．ここでは部分的な試作のため，クライアントアプリケーションのコンソールにタプルの各コラムの値がテキストで表示されるのみである．

なお，インデックスに Fat-Btree を利用した場合，リーフ

(注1)：このログは動作確認を行うために用いられるものを示す．障害発生時のリカバリに用いられるものとは異なる．

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 (pgbench)	
Number of nodes	8
Number of clients per node	1, 2, 4, 8, 16, 32, 64
Fat-Btree	
Page size	4096 byte
Max number of entries in an index node	64
Max number of tuples in a leaf node	8
Concurrency control method	INC-OPT [17]
PostgreSQL	
Number of PEs	1, 4, 8
Tuple size	114 byte
Number of tuples	100,000

ノードとタプルが存在する PE は必ず同一である．したがって，現在の実装では PE m と PE n の間でインデックス探索の要求と応答，タプルの取得要求と応答の 4 回の通信を行っているが，インデックスを利用したタプル取得要求と応答の 2 回の通信に回数を削減することで処理の高速化を実現できる余地が残されている．

5. 実　　験

4. で述べた内容を PostgreSQL 8.1.4 を用いて部分的に試作し，実験を行った．並列 B-tree 構造としては Fat-Btree と SIB 方式を実装し，両者の比較を行った．

本稿では並列 B-tree 構造として，Fat-Btree の利用を中心に SIB 方式と参照操作に限定して比較を行う．この環境では更新操作を行わないため CWB 方式については評価しないが，今後更新操作を考えた際には，これまでの検証 [7], [9], [10] から CWB 方式よりも Fat-Btree の利用が高いスループットをもたらすものと考えられる．

5.1 実験環境

実験には表 1 に示す構成のブレードサーバを複数台用いた．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 ならびに SIB 方式のインデックスの実装は，Java により記述された自律ディスク [15], [16] における実装に対し，TCP/IP によるソケット通信で外部プログラムからの操作を可能とするように拡張したものを利用した．

5.2 実験内容

ブレードサーバ 1 台について PostgreSQL，並列 B-tree 構造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 および並列 B-tree 構造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場合は「ノード」と表現する．

全体で 100,000 項目のデータを用意し，プライマリキーかつインデックスを作成するコラムの値を基に各 PE で動作する PostgreSQL のテーブルに均等に分割する．これらのテーブルに対し Fat-Btree あるいは SIB により 1 つのインデックスを用意する．Fat-Btree を用いた場合はインデックス項目についてもタプルと同様に均等に分割する．すなわちすべてのタプルに対してタプルとそれに対応するインデックス項目は同じ PE に格納されることになる．データとインデックスを分散させる PE の台数は 1, 4, 8 台に変化させる．より PE 数を増やしての実験は今後の課題である．

クライアントプログラムには PostgreSQL の配布物に附属するベンチマークソフトウェアである pgbench に対し，実験環境に合わせた問い合わせ文の修正，ならびにレスポンスタイムの測定機能を加えたものを使用する．クライアントプログラムはデータを持たせる PE の台数に関わらず 8 台のノードで実行する．したがってデータを 4 PE に分割した際には 1 PE につき 2 ノードのクライアントプログラムからアクセスされることになる．各クライアントプログラムでは非同期処理機能を利用してマルチユーザ環境をシミュレートしており，ノードあたりの同時接続数を 1 から 64 まで変化させて実験を行った．したがってシステム全体で 8 ノードから最大 512 の接続を同時に受けることになる．

各クライアントプログラムはインデックスを利用してテーブル全体からランダムに 1 タプルのみが得られる SELECT 文を発行する．この問い合わせを 20 回順次発行する処理を同時接続数が 1 の時の処理とする．同時接続数を増加させたときはこの処理を複数同時に実行する．

実験で変化させるパラメータは，PostgreSQL および Fat-Btree でデータを分割する PE の台数，そして各ノードで実行する 1 クライアントプログラムにおける同時接続数となる．その他のおもなパラメータをまとめたものを表 2 に示す．

5.3 実験結果と考察

インデックス構造として SIB ならびに Fat-Btree を用い，データを分割する PE の台数，同時実行トランザクション数を変化させたときのスループットの変化を図 4 に，レスポンスタイムの変化を図 5 に示す．両グラフの横軸は各ノードのクライアントプログラムにおける同時接続数であり，この実験ではノード数が 8 であることからシステム全体で軸に与えられた値の 8 倍のトランザクションを処理していることになる．

図 4 のスループットに関して，4 PE 構成時において両インデックス構造を用いた場合の比較を行うと，Fat-Btree によるインデックス分散化の効果は見られない．クライアントノードあたりの同時接続数が 8 以下では Fat-Btree ではおよそ 10 %低いスループットしか得られていない．このとき図 5 のレスポン

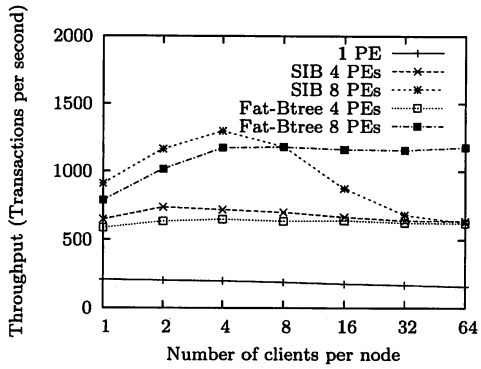

図 4　スループットの変化

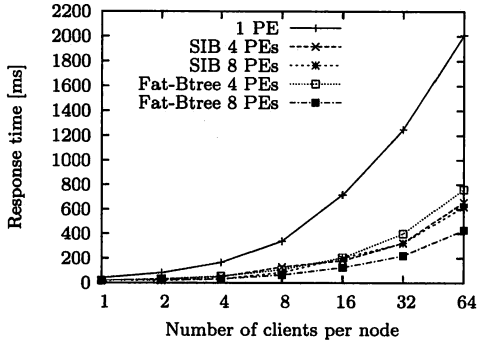

図 5　レスポンスタイムの変化

スタイムにおいても Fat-Btree ではおよそ 10 %大きい値が得られており，Fat-Btree を探索する際に PE 間でリクエストの転送を行うことによるレスポンスタイムの悪化がスループットにも現れているものであると考える．

同様に 8 PE 構成時においては，SIB 方式の場合にクライアントノードあたりの同時接続数が 4 の時に約 1,300 tps を記録した後はスループットが低下するが，Fat-Btree を利用した場合は同時接続数 4–64 で約 1,170 tps を維持している．

すなわち，データを分散する PE の台数および同時実行トランザクション数が大きい環境において Fat-Btree が SIB 方式と比較して高いスループットをもたらすという結果が得られた．

6. まとめと今後の課題

本稿ではオープンソース DBMS の分散化のアプローチの 1 つとして，データ配置を動的に変更可能な分散インデックス構造である Fat-Btree を用い，複数の PostgreSQL に分散したデータを分散検索する方法を検討した．さらに，その最初の試みとして部分的な試作を行いインデックスを単一 PE で管理する方式とスループットの比較を行った．実験の結果，データを分割する PE 数の増加に対して Fat-Btree，SIB 方式ともにスループットが増加するが，同時実行トランザクション数の増加に対しては SIB 方式ではスループットの低下が見られるのに対し，Fat-Btree では大きな低下は見られず，分散インデックス構造を用いた分散検索を実現する際に Fat-Btree を利用することの有効性を確認した．

今後の課題として，複数の PE にわたるトランザクションの管理があげられる．現在の実装において他 PE のリモートバックエンドに対する問い合わせは，問い合わせを受けた PE 内にローカルかつその時点から始まるトランザクションとして実行される．したがって単一 PE 内のローカルなトランザクションの管理しか行っておらず，システムとしてクライアントから問い合わせを受け付けた時点と，そのリクエストがデータを持つ PE に転送され，そこで実際にトランザクションが開始される時間に差が生まれる．この間に読むべきデータを他トランザクションにより更新されてしまう可能性や，PE において問い合わせの到達順序が異なってしまう可能性があり，トランザクションの分離性が保証されない．これを防ぐためにシステム全体の規模でトランザクションを一元管理するなどの管理機構が必要である．

文　献

[1] M. Stonebraker, "The case for shared nothing," Database Engineering Bulletin, vol.9, no.1, pp.4-9, 1986.

[2] G. Copeland, W. Alexander, E. Boughter, and T. Keller, "Data placement in Bubba," Proc. of the 1988 ACM SIGMOD Int'l Conf. on Management of Data, pp.99–108, 1988.

[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] "PostgreSQL," `http://www.postgresql.org/`.

[5] "pgpool-II," `http://pgpool.projects.postgresql.org/`.

[6] NTT DATA Corporation, "PostgresForest," `http://www.nttdata.co.jp/services/postgresforest/`.

[7] 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.

[8] 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，no.416，pp.63-68，1997，DE97-77.

[9] 宮崎純，横田治夫，"無共有並列計算機向けディレクトリ構造 Fat-Btree の実装とその評価，" 情処学会研究会報告「データベースシステム」，no.61，pp.407-412，1999，1999-DBS-119.

[10] 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，2001，7A-7.

[11] Oracle Corporation, "Oracle Real Application Clusters," `http://www.oracle.com/database/rac_home.html`.

[12] B. Thomas, "Solutions for highly scalable database applications: An analysis of architectures and technologies," Performance Tuning Corporation, May 2006.

[13] M.L. Lee, M. Kitsuregawa, B.C. Ooi, K.L. Tan, and A. Mondal, "Towards self-tuning data placement in parallel database systems," Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data, pp.225-236, 2000.

[14] 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.

[15] H. Yokota, "Autonomous Disks for advanced database applications," Proc. of the 1999 Int'l Symposium on Database Applications in Non-Traditional Environments, pp.435–442, Nov. 1999.

[16] 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「データベースシステム」，vol.2001，no.70，pp.45-52，情報処理学会，July 2001.

[17] 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.