Fat-Btree をインデックスに用いた PostgreSQL の分散検索

Distributed Retrieval on PostgreSQL with a Fat-Btree Index

並木 悠太 † 神戸 康多 ††
小林 大 ††† 横田 治夫 ††††

Yuta NAMIKI Kota KANBE
Dai KOBAYASHI Haruo YOKOTA

データベースシステムの性能向上の手段として並列データベースシステムがあり，オープンソース DBMS である PostgreSQL を用いたものとしては，データ配置を固定化し上位層を設けて問い合わせを分割する方式などが提案されている．本稿ではデータ配置を動的に変更し負荷分散を可能とする並列 B-tree 構造である Fat-Btree を複数の PE に分散した PostgreSQL のデータに対するインデックスとして利用し，データの検索を行う方法を検討する．また，部分的な試作を行い，インデックスを単一の PE で管理する方式とスループットおよびレスポンスタイムを比較した．実験の結果，Fat-Btree の利用がスループットの向上およびレスポンスタイムの短縮をもたらすことを確認した．

Parallel database systems have been introduced to satisfy demands on the performance. For PostgreSQL, although some trials of preparing an upper layer to distribute queries for PEs have been developed, they assume static data placement. In this paper, we use Fat-Btree, a parallel B-Tree structure capable of dynamically changing data distribution to balance access load of the parallel PostgreSQL. We partly implement a retrieval function in PostgreSQL using the Fat-Btree as a distributed index, and compare the throughput and response times for the case of using a non-distributed B-Tree index on a PE. The experimental result shows that the Fat-Btree index is effective to derive the higher throughput and shorter response times.

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を構成するにあたり，無共有型構成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 [1]．

本稿では無共有型並列データベースにおいて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，B-tree 構造を複数の PE（Processing Element）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 [2] が提案されている．Fat-Btree 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 LAN 環境での PC クラスタ上への実装による実験 [3] など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，オープンソース DBMS である PostgreSQL の分散検索を行う方法を検討する．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．

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 [4] 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 [5]．

無共有型構成において容量や処理量の大きいテーブルを複数の PE に分割して格納するテーブルパーティショニング機能の実装として，PostgreSQL に対応するものでは pgpool-II [6] や PostgresForest [7] がある．pgpool-II ではデータに対し格納先 PE を返すような関数を予め定義しておき，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ると，これを利用して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 では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，PE 数を変化させる際には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を変化させることは考えられていない．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 構造で管理するにあたり，いくつかの構成が考えられる．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 [2] 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

† 学生会員 東京工業大学大学院情報理工学研究科修士課程 namiki@de.cs.titech.ac.jp
†† 正会員 フューチャーアーキテクト株式会社 kanbe.kota@future.co.jp
††† 学生会員 東京工業大学大学院情報理工学研究科博士後期課程 daik@de.cs.titech.ac.jp
†††† 正会員 東京工業大学学術国際情報センター yokota@cs.titech.ac.jp

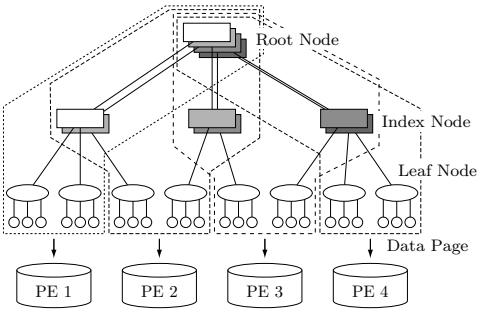

図 1　Fat-Btree の構造
Fig. 1　A structure of Fat-Btree

し，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行うため必要があり，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 [8] 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 B$^+$-tree のリーフページ（データページ）を各 PE に均等に分配するものであり，その構造を図 1 に示す．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配置する．

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．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 [9]．

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．

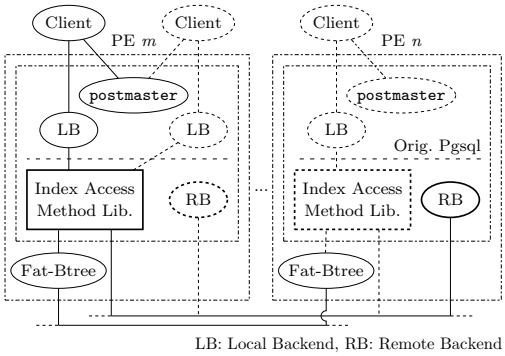

図 2　PE 内の構成
Fig. 2　A structure of a PE

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．

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．以下の段落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．

■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，本稿ではローカルバックエンドと表記する．

■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．

■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 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[1]に出力する処理を行う．

[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．

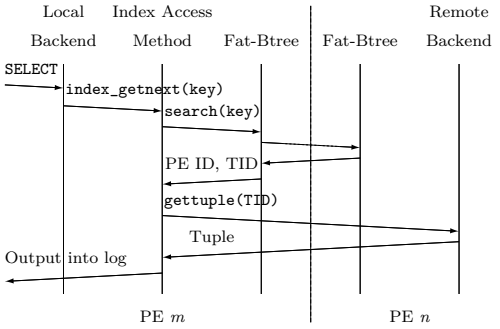

図 3 処理の流れ
Fig. 3 A processing flow

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．

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．

■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．

■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．

■TID を基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，クライアントのコンソールにタプルの各属性の値がテキストで表示されるのみである．

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 Fat-Btree と SIB 方式を利用する実装を行い，参照操作について比較を行った．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 [2, 3] から CWB 方式よりも Fat-Btree を利用することで高いスループットが得られるものと考える．

表 1 実験システムの構成
Table 1 Specifications of PCs used for experiments

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_03
Network	TCP/IP over 1000BASE-T

表 2 実験におけるパラメータ
Table 2 Parameters used for experiments

Page size in a B-tree	4,096 byte
Max no. of entries in a B-tree index node	64
Max no. of tuples in a B-tree leaf node	8
Cache size in a B-tree	65,536 pages
Concurrency control method in a Fat-Btree	INC-OPT [11]
Tuple size	114 byte

5.1 実験環境

実験には表 1 に示す構成のサーバを複数台用いた．各サーバは十分な帯域を持つネットワークで接続されている．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 [10] における実装を利用した．

5.2 実験内容

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 プログラムを必要に応じて実行する．以下，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属性の値を基に各 PE 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 方式により 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．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．

クライアントプログラムには PostgreSQL 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．

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．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 方式ならびに Fat-Btree を用い，データを分割する PE の台数，同時接続数を変化させたときのスループットの変化を図 4 に示す．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．

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

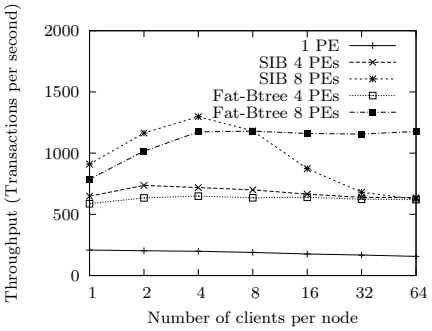

図 4　各構成でのスループットの比較

Fig. 4　A comparison of throughputs with changing indexing structure and number of PEs

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．このときレスポン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] DeWitt, D. and Gray, J.: “Parallel database systems: the future of high performance database systems”, Commun. ACM, Vol. 35, No. 6, pp. 85–98 (1992).

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

[3] 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.

[4] Oracle Corporation: “Oracle Real Application Clusters”, http://www.oracle.com/database/rac_home.html.

[5] Thomas, B.: “Solutions for highly scalable database applications: An analysis of architectures and technologies”, Performance Tuning Corporation (2006).

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

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

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

[9] 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.

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

[11] Miyazaki, J. and Yokota, H.: “Concurrency control and performance evaluation of parallel B-tree structures”, IEICE TRANSACTIONS on Information and Systems, Vol. E85-D, No. 8, pp. 1269-1283 (2002).

並木 悠太　Yuta NAMIKI
東京工業大学大学院情報理工学研究科計算工学専攻修士課程在学中．2006 芝浦工業大学工学部情報工学科卒業．日本データベース学会学生会員．

神戸 康多　Kota KANBE
フューチャーアーキテクト株式会社研究開発本部所属．2004 立命館大学大学院理工学研究科博士課程前期課程修了．日本データベース学会正会員．

小林 大　Dai KOBAYASHI
東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程在学中．2005 同大大学院情報理工学研究科計算工学専攻修士課程修了．日本データベース学会学生会員．日本学術振興会特別研究員 DC.

横田 治夫　Haruo YOKOTA
東京工業大学学術国際情報センター教授．1980 東京工業大学工学部電子物理工学科卒業．1982 同大大学院理工学研究科情報工学専攻修士課程修了．同年富士通（株）．同年 6 月（財）新世代コンピュータ開発機構研究所．1986（株）富士通研究所．1992 北陸先端科学技術大学院大学情報科学研究科助教授．1998 東京工業大学大学院情報理工学研究科計算工学専攻助教授．2001 より現職．工学博士．日本データベース学会理事．電子情報通信学会，情報処理学会，人工知能学会，IEEE，ACM 各会員．