DEIM Forum 2011 E2-2

範囲問合せ可能な分散インデックスの性能評価

近藤　直樹† 　羅　　敏† 　渡辺　陽介†† 　横田　治夫†

† 東京工業大学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
†† 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: †{naoki,luomin,watanabe}@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　データが爆発的に増加し、データを複数の計算機で管理するようになってきている。分散されたデータへのアクセスを効率化するためにインデックスを用いるが、インデックスを集中管理すると負荷が増大する。そこでインデックスを分散させる分散インデックスという手法が提案されている。分散インデックスで用いられるデータ構造には B-tree やハッシュテーブルなどがある。B-tree を用いた分散インデックスは完全一致問合せのほかに範囲問合せが可能で柔軟な問い合わせができる。一方、ハッシュテーブルを用いた分散インデックスは従来完全一致問合せに重点を置いていたが、近年の改良によって範囲問合せが可能となった。その結果、両手法の適用領域が近づいた。しかし、それらの分散インデックスは十分には比較はされていない。本稿では、範囲問合せ可能な分散インデックスである Fat-Btree と P-tree を比較する。

キーワード　分散インデックス, Fat-Btree, 範囲問合せ

Performance Evaluation for Range Queries on Distributed Indexes

Naoki KONDOH†, Min LUO†, Yousuke WATANABE††, and Haruo YOKOTA†

† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
2-12-1 Oookayama, Meguro-ku, Tokyo, 152-8552, JAPAN
†† Global Scientific Information and Computing Center, Tokyo Institute of Technology
2-12-1 O-okayama, Meguroku, Tokyo, 152-8550, JAPAN
E-mail: †{naoki,luomin,watanabe}@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

Abstract　Distributed indexing methods which enable to process efficiently range queries for distributed data have been proposed. The basic idea is to create an index structure as B+-tree on overlay network using distributed hash table. However, intensive comparisons between these methods and traditional distributed indexing methods have not been done. In this paper, we compare Fat-Btree, which is one of distributed B+-tree indexing methods, with distributed hash table indexing method which can process range queries.

Key words　ditributed index, Fat-Btree, range query

1. はじめに

データセンターなどでデータ量が爆発的に増加し、データを複数の計算機で管理するようになってきている。しかし、データを複数の計算機に分散させるとデータへの参照コストが増加する。データへのアクセスを効率化するためにインデックスが用いられる。しかし、インデックスを集中管理しているとそれ自体にアクセスが集中し負荷が増大する。

そこでインデックス自身を分散させて負荷分散を実現する分散インデックスの研究が行われている。分散インデックスはデータのインデックスをどのように分散させるか、分散させたインデックスをどうやって参照するかが重要である。インデックスのデータ構造にはハッシュテーブル、B-tree などが使われる。ハッシュテーブルは、特定のキーを探索するコストが低いという特徴がある。また、B-tree のような木構造はデータがソートされているという特徴がある。木構造でデータを管理することによって範囲問合せなどの複雑なクエリを処理することが可能である。

我々の研究室では分散インデックス環境におけるデータのインデックスに B+-tree 構造を用いた Fat-Btree [1] を提案している。Fat-Btree は B+-tree をベースにしているので範囲問合せが可能である。また、Fat-Btree ではデータのアクセス頻度

に応じて、データの再配置を行うことで負荷分散が可能である。

B-tree を用いた分散インデックスは範囲問合せが可能であるのに対して、ハッシュテーブルを用いた分散インデックスはキーによる完全一致問合せのみという制限があった。しかし、近年では P2P の分野で分散ハッシュテーブルの研究が盛んに行われている [2] [3] [4] [5]。分散ハッシュテーブルはハッシュ値の範囲によってハッシュテーブルを分散させ、ハッシュ値によってどこを参照すべきかのリストを管理するというのが基本的なアイディアである。P2P 環境で利用するために、問合せ要求の高度化への対応、分散度合いのスケーラビリティ、負荷分散などが求められるようになった。このような問題を解決するための仕組みを取り入れ、範囲問合せが可能な分散ハッシュテーブルが提案されてきている [6] [7] [8]。

このように改良された分散ハッシュテーブルが従来の B-tree ベースの分散インデックスと同列の機能を持つようになり、両者の適用領域が近づいた。しかし、分散ハッシュテーブルと従来の分散インデックスとの比較は十分には行われていない。

本研究では、分散インデックスである Fat-Btree と範囲問合せ可能な分散インデックスを比較する。本論文では、分散インデックスの比較対象に P-tree [8] を選んだ。P-tree はハッシュテーブルの分割にキーの範囲を用いている。キーの範囲で分割することによって、通常のハッシュテーブルではコストが高い範囲問合せを行うことができる。実験として、データの挿入、範囲問合せ、完全一致問合せの処理性能を測定し比較を行う。

以降、2. 節で関連研究について述べる。3. 節では比較対象の P-tree と Fat-Btree について述べる。4. 節では比較評価の実験結果を述べる。5. 節ではまとめと今後の課題を述べる。

2. 関 連 研 究

2.1 分散インデックス

2.1.1 分散ハッシュテーブル

分散ハッシュテーブルは分散インデックスアルゴリズムの一つで、データ構造としてハッシュテーブルを用いる。ハッシュテーブルをノードごとに分割する方法はハッシュ値の範囲によって分散させる方法が一般的である。ルーティングの方法は様々である。ピアをハッシュテーブルのハッシュ関数によりハッシュ値の空間にマップし、複数回のホップで参照できるようにリンク情報を保持する。それぞれのピアが自分から出るリンクを管理することですべてのピアを参照可能にする。代表的なアルゴリズムは、Chord [2]、Kademlia [3]、Pastry [4]、Tapestry [5] などがある。

Chord [2] はフィンガーテーブルと呼ばれるリンク情報を管理することによって、$O(logN)$ 回のホップでルーティング可能である (N はピアの数)。通常、$O(logN)$ 回の通信はピアが爆発的に増えてもスケールすると言われている。しかし、ハッシュテーブルを使っているのでクエリは完全一致問合せしか使えない。

これに対して、分散ハッシュテーブルで範囲問合せを可能にするアルゴリズムが存在する [6] [8] [7]。本論文では、このうちの分散ハッシュテーブルのオーバレイ上に B+-tree を載せた P-tree [8] を比較対象にする。

同様のアプローチで Zheng らは分散ハッシュテーブル上に B-tree を構築する方法で、P2P ネットワークの状態を監視するアルゴリズム SOMO [9] というものを提案した。分散ハッシュテーブルのネットワーク上に木構造を構築し葉ノードから計算機の状態を収集する手法である。

2.1.2 並列 B-Tree

並列 B-Tree は分散インデックスアルゴリズムの一つである [10]。インデックスのデータ構造として B-Tree を用いる方法であり、範囲問合せや連続した I/O のクラスタ化などにも対応できる。B-Tree の単純な分散方法にはすべての PE にインデックス全体をコピーする方法 (CWB: copy whole btree)、ひとつの PE のみにインデックスを持たせる方法 (SIB: single index btree) がある。CWB は検索時に必要なインデックスをすべての PE が持っているのでアクセスの負荷分散になるが、インデックスの更新時にすべての PE のインデックスと同期させる必要がありコストが高くつく。また、SIB はインデックスの更新時は一つだけ更新すればよいのでコストが低いが、インデックスを持つ PE にアクセスが集中してボトルネックになる。

この問題点を解決するために、我々は Fat-Btree [1], [11], [12] を提案している。Fat-Btree の概要は 3. 2 節に記述する。

2.1.3 Skip Lists を用いた分散インデックス

Skip Lists とはハッシュテーブルや B+-tree と同様にデータ構造である。Skip Lists では数種類のリストが階層的に管理されており、全データから構成されるリストを底のレベルにして、上のレベルでは途中のデータがスキップされたリストになっている。

Skip Lists を用いた分散インデックスには Skip Graphs [13] がある。Skip Lists のレベルの層を縦に分割して、各レベルで隣のデータがあるノードへのリンクを管理する手法である。Skip Graphs は範囲問合せを効率よく処理することができる。

Skip Graphs と B-tree の利点を合わせた Skip B-Trees [14] なども提案されている。

3. 比 較 対 象

3.1 P-Tree

P-tree [8] は分散ハッシュテーブル上で範囲問合せをサポートすることを目的としたアルゴリズムである。分散ハッシュテーブルでピアの管理をするが、ピアはさらに範囲問合せに必要な B+-tree を別に持つ。

P-tree では、それぞれのピアでキーの全空間を分割し円環状に管理する。(図 1)。例えば、図 1 の p_1 は $[5, 7)$ の範囲を管理している。

範囲問合せを効率よく処理するために P-tree は分散ハッシュテーブル上にオーバーレイさせて用いるインデックスとして、B+-tree を採用している (図 1)。各ピアに配置される B+-tree には、それぞれのピアが管理する範囲の最小値からキーの全空間を時計回りにたどるような順序関係で値が格納されている。B+-tree のそれぞれの層の左端はそのピアの管理する範囲の最小値になる。B+-tree のノードはキーを管理するピアの情報を

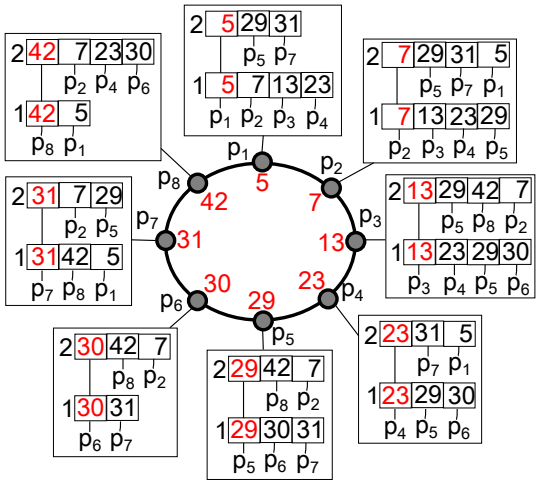

図 1　P-tree

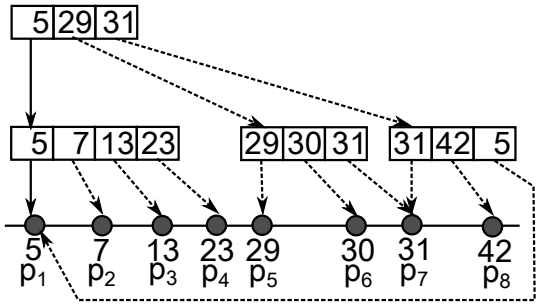

図 2　p_1 から見た P-tree のノード

格納する。例えば、図 1 の p_1 の B+-tree の最上位の 29, 31 はそれぞれ p_5, p_7 を指している。

P-tree での探索方法は、探索を要求したピアが自身の B+-tree を根から探索する。葉ノードに他のピアへの参照があれば、そのピアに通信し、続きとなる途中の B+-tree の層から探索を再開する。例えば、図 1 の p_1 から $[30, 40]$ の範囲問合せを行う場合を考える。このとき、p_1 を起点としたキーの空間は図 2 のように見える。まず、p_1 の B+-tree の根から、探索範囲の開始値である 30 を探索する。p_1 の第 2 層で 30 は p_5 にあると示されているので、問合せを p_5 に転送する。探索は p_5 の第 1 層から再開し、30 は p_5 の第 1 層で p_6 にあると示されているので、問合せを p_6 に転送する。p_6 では B+-tree の葉に到達したので p_6 が担当範囲であると分かる。p_6 で問合せ範囲内のデータを取得するが、問合せ範囲 $[30, 40]$ は p_6 の担当範囲を超えるので Chord の successor に示される隣の p_7 へ問合せを転送する。p_7 で残りのデータを取得し、範囲問合せの処理は終了する。

P-tree はピアの管理を Chord で行っているのでピアの追加、削除に対応できるようになっている。ピアの追加はそのピアが管理する範囲を持つピアから P-tree をコピーしてそれぞれの層の左端を追加したピアのものにする。削除されたピアはほかのピアが削除を探知したらそれぞれのピアの B+-tree から削除することで対応する。ピアの追加や削除によって索引に不整合が起こる可能性があるので、索引を更新するプロセスを定期的に動かしておく必要がある。

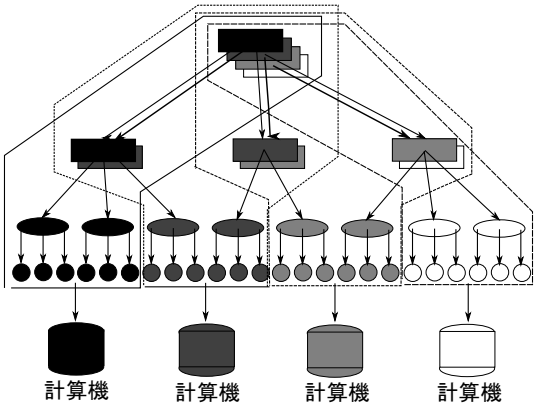

図 3　Fat-Btree

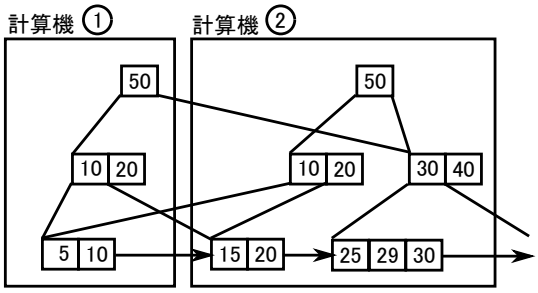

図 4　Fat-Btree の具体例

3.2　Fat-Btree

Fat-Btree ではデータのレコードを各 PE に均等に配分し、格納しているレコードから見て上位にあたるノードのみを PE に配置する (図 3)。図 3 の色分けはそれぞれの PE が持つ B+-tree の部分木を表す。並列 B-tree を用いることによって範囲問合せの対象キーを探索しやすくしている。

インデックスの更新には更新に関係するノードをすべて同期する必要があるが、更新頻度が高い下位のノードほどそのコピーを持つ PE の数が少なく同期コストを低く抑えることができる。また、根ノードはすべての PE にコピーがあるため探索の時のアクセスが集中する根ノードを分散して複数の要求に対して並列に探索処理が可能である。

Fat-Btree の探索方法は、探索を要求した PE の B+-tree をまず探索する。途中のノードで他の PE への参照があれば、その PE に通信し、木の探索を継続する。各 PE の葉ノードはデータが含まれている。範囲問合せに関しては、B+-tree の葉ノードのリンクをたどることができる。また、各 PE が管理する Fat-Btree のデータの量を隣接する PE で調節することによってアクセスの負荷分散が可能である。

範囲問合せの例として、図 4 を一部とする Fat-Btree の計算機 2 から $[5, 29]$ の範囲問合せを行う場合を考える。まず、計算機 2 内の B+-tree の根から、探索範囲の開始値である 5 を探索する。B+-tree を探索して行き、上から 2 番目の層で 10 より小さい範囲の担当は計算機 1 と示されているので、問合せを計算機 1 に転送する。計算機 1 では B+-tree の一番下の層から探索を再開し、葉ノードに到達する。計算機 1 の $\{5, 10\}$ のノードでデータを取得した後は、B+-tree のリンクをたどるこ

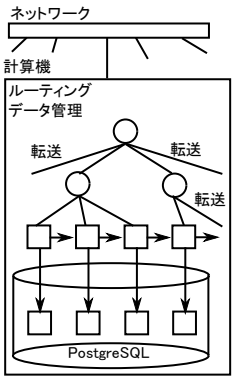

図 5　Fat-Btree の実装構成

とによって計算機 1 内の問合せ範囲のデータを取得する。問合せ範囲は計算機 1 の担当範囲を超えるので B+-tree のリンクに示されている隣の計算機 2 に問合せを転送する。計算機 2 では葉ノードのリンクをたどりつつデータを取得し、範囲問合せの処理は終了する。

4. 実　　験

Fat-Btree と P-tree のデータ挿入、範囲問合せと完全一致問合せの性能を比較し、考察する。

4.1 実験環境

以下の実験環境で実験した。実験で使用する計算機の数は 1, 2, 4, 8, 16, 32 である。

- CPU: AMD Athlon XP-M 1800+ (1.53GHz)
- Memory: PC2100 DDR SDRAM 1GB
- Network: 1000BASE-T
- HDD: TOSHIBA MK3019GAX (30GB, 5400rpm, 2.5inch)
- OS: Redhat Linux 9.0 (Kernel 2.4.20)
- Java 1.5.0_03

計算機の追加・削除の実験は行わないので、実験に用いる計算機は事前に構成しておき、アルゴリズムのルーティング情報変更による影響を極力なくす。

実験に用いるデータは、以下の通りに統一する。

- キー: 整数 (範囲 $[1, 10000]$)
- 値: 3 つの整数と一つの文字列からなるレコード

実験に用いたプログラムの概要を説明する。

4.1.1 Fat-Btree

Fat-Btree は我々の研究グループでこれまでに実装したもの [15] を使用する。実装の構成を図 5 に示す。ローカルでのデータの管理は PostgresSQL をもとにしているが、通信機能などのそのほかはすべて Java で実装されている。

4.1.2 Ptree

本研究では、P-tree の論文 [8] のアルゴリズムをもとに我々が独自に実装した。実装の構成を図 6 に示す。P-tree のハッシュテーブルの分散と通信機能の部分に Overlay Weaver [16] を用いた。Overlay Weaver は構造化されたオーバレイの構築ツールキットである。P-tree の分散したハッシュテーブルの環状の

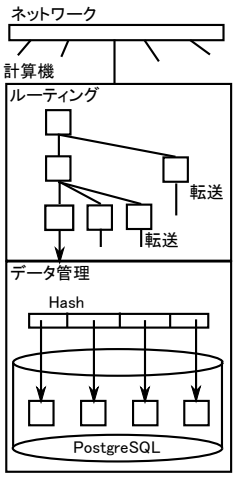

図 6　P-tree の実装構成

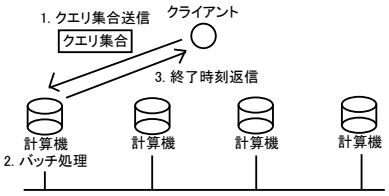

図 7　逐次実行

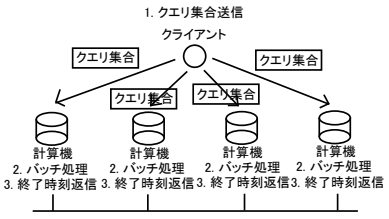

図 8　並列実行

位置情報を Chord により管理した。P-tree の木構造の管理やルーティング選択のアルゴリズムは独自に実装した。我々の実装では、キーの挿入、完全一致問合せ、範囲問合せを行うことができる。

Fat-Btree の PostgresSQL の IO 性能を合わせるため、キーに対応するデータは PostgreSQL に格納されている。ハッシュテーブルでキー値を特定した後、PostgreSQL に問合せを行う。

4.2 実験方法

問合せの実行方式は、複数ある計算機のうち一つがすべてのクエリを発行する逐次実行 (図 7) と、全部の計算機が並列してクエリを発行する並列実行 (図 8) の 2 種類を用いた。図 7 と図 8 のようにクエリの集合を計算機に送信してから同時に実行を開始する。時間計測はクエリを実行開始してから終了までの時間であり、並列実行の時は最後に終了した計算機の時間を終了時間とする。

4.3 実験結果

4.3.1 データの挿入

データの挿入にかかった時間を図 9、図 10 に示す。図の縦軸は、データがなにもない状態からランダムな値のレコードを

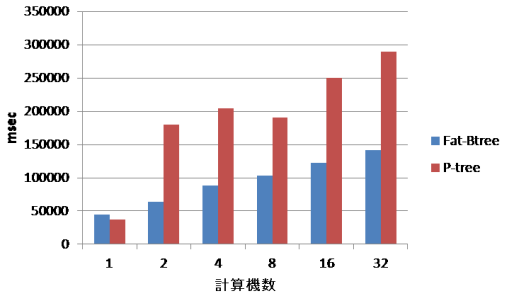

図 9　逐次データ挿入の実験

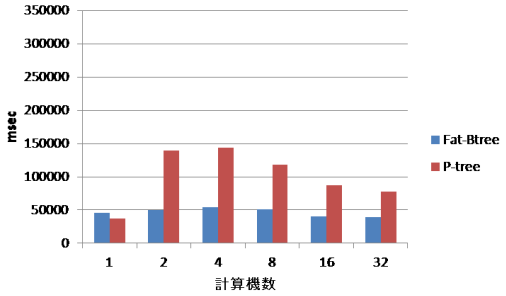

図 10　並列データ挿入の実験

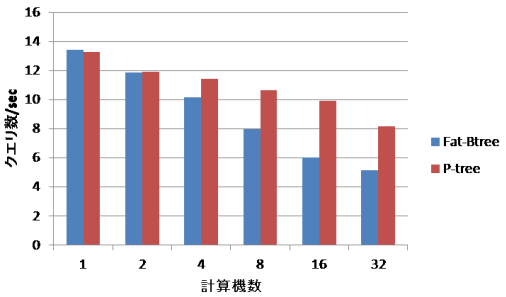

図 11　逐次範囲問合せの実験

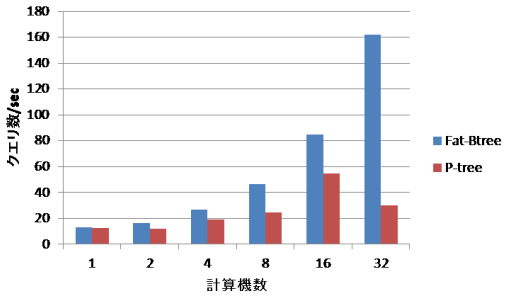

図 12　並列範囲問合せの実験

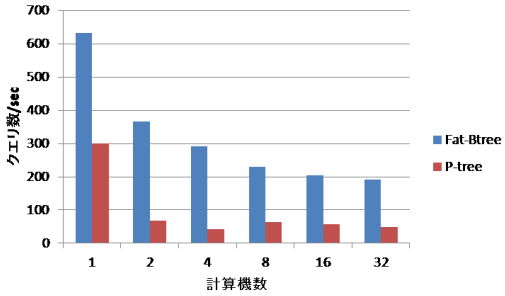

図 13　逐次完全一致問合せの実験

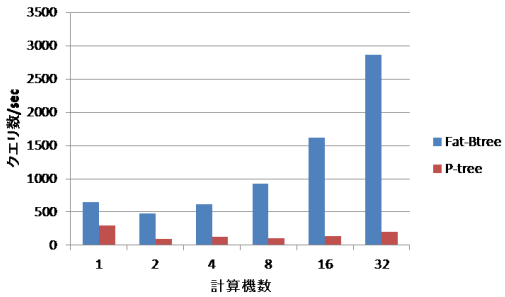

図 14　並列完全一致問合せの実験

10000 回挿入するまでの時間である。並列実行では 10000 個のクエリを計算機の台数で均等に分割して並列に実行する。

図 9 の逐次実行の実験結果から計算機数 1 以外では Fat-Btree の方が挿入が速く、計算機台数が増えるごとに Fat-Btree と P-tree の差が広がっていることがわかる。

図 10 の並列実行の実験結果から逐次実行の性能に比べて並列実行の結果が良く、それぞれ並列化の効果が出ていることがわかる。また、P-tree の方が計算機台数を増やすことによる逐次実行に対する性能向上は高かったが、性能そのものは Fat-Btree の方が良い。ただし、計算機台数を増やすごとにその差が縮まっている。

4.3.2　範囲問合せ

範囲問合せを逐次実行、並列実行した場合における 1 秒あたりの問合せ処理数 (スループット) を図 11、図 12 に示す。ここでは、全体で 10000 タプルあるデータに対して範囲問合せをした。範囲 $[n, n+100](n$ はランダム) を検索するクエリを使った。スループットの値は逐次実行ではクエリ数 1000 回の結果であり、並列実行ではそれぞれの計算機が 1000 回のクエリを実行した結果である。

図 11 より逐次実行では P-tree の方がスループットが良い。両手法とも計算機台数を増やすごとに性能が低下しているが、Fat-Btree の方が低下幅が大きく計算機台数 32 で 3 クエリ数 $/sec$ ほどの差が出ている。

図 12 より並列実行では Fat-Btree の方がスループットが良いことが分かった。Fat-Btree のスループットは計算機台数を増やすごとに急激によくなり、P-tree では計算機台数 32 で性能が低下している。

4.3.3　完全一致問合せ

完全一致問合せを逐次実行、並列実行した場合における 1 秒あたりの問合せ処理数 (スループット) を図 13、図 14 に示す。全体で 10000 タプルあるデータに対しランダムなキーで検索する。スループットの値は逐次実行ではクエリ数 10000 回の結果であり、並列実行ではそれぞれの計算機が 10000 回のクエリを実行した結果である。

図 13 の逐次実行の実験結果から、Fat-Btree の方がスループットが良いことが分かった。しかし、Fat-Btree では計算機

台数を増やすごとに性能が低下している。P-tree は計算機台数 2 で急激に性能が低下してはいるが、計算機台数 2 以上での性能低下は見られない。

図 14 の並列実行の実験結果から、Fat-Btree の方がスループットが良いことが分かった。Fat-Btree では計算機台数を増やすごとに急激に性能の向上が見られた。P-tree においても性能の向上があると考えられるが、実験結果ではスループットが伸びていないのでより詳細な分析が必要である。

4.4 実験まとめ

逐次実行の実験結果から、両手法とも計算機台数に応じて問合せごとの処理時間が増加することが分かった。計算機数の増加による通信回数の増加による時間への影響が表れたと考えられる。

並列実行の実験結果から、Fat-Btree は計算機台数に応じてスループットが向上することが分かった。逆に P-tree はスループットが伸びなかった。P-tree においては実装におけるロックの扱いなどで並列処理性能に差が出た可能性がある。

今回の実験構成では並列実行において範囲問合せ、完全一致問合せの処理性能は Fat-Btree の方が優れていることが確認された。しかし、通信回数や計算機内のデータ取得時間を分析することは今後の課題である。

5. ま と め

本論文では、範囲問合せの機能について比較した。比較対象は、分散ハッシュテーブルでは P-tree、従来の分散インデックスでは並列 B-tree の Fat-Btree を選択した。

今回の実験は、アルゴリズムを適用する複数の計算機が途中で追加や離脱のない安定している環境を想定した実験を行った。評価した性能は、データの挿入、範囲検索と完全一致問合せの性能を比較した。今回の実験構成では並列実行において範囲問合せ、完全一致問合せの処理性能は Fat-Btree が優れていることを確認した。

今後の課題として、通信回数や計算機内のデータ取得時間などのクエリの部分的な時間を比較するより詳細な分析をする必要がある。また、他の分散インデックスとの比較実験やデータ数、計算機数を増やした大規模な実験をする必要がある。

謝　　辞

本研究の一部は、文部科学省科学研究費補助金特定領域研究 (#21013017)、日本学術振興会科学研究費補助金基盤研究 (A)(#22240005) の助成により行われた。

文　　献

[1] H. Yokota, Y. Kanemasa, and J. Miyazaki. Fat-btree: an update-conscious parallel directory structure. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pp. 448 –457, March 1999.

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, Vol. 31, pp. 149–160, August 2001.

[3] Petar Maymounkov and David Mazie`res. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, Vol. 2429 of *Lecture Notes in Computer Science*, pp. 53–65. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-45748-8_5.

[4] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, Vol. 2218 of *Lecture Notes in Computer Science*, pp. 329–350. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-45518-3_18.

[5] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, Berkeley, CA, USA, 2001.

[6] Domenico Talia and Paolo Trunfio. Dynamic querying in structured peer-to-peer networks. In Filip De Turck, Wolfgang Kellerer, and George Kormentzas, editors, *Managing Large-Scale Service Deployment*, Vol. 5273 of *Lecture Notes in Computer Science*, pp. 28–41. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-87353-2_3.

[7] Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou. Replication, load balancing and efficient range query processing in dhts. In Yannis Ioannidis, Marc Scholl, Joachim Schmidt, Florian Matthes, Mike Hatzopoulos, Klemens Boehm, Alfons Kemper, Torsten Grust, and Christian Boehm, editors, *Advances in Database Technology - EDBT 2006*, Vol. 3896 of *Lecture Notes in Computer Science*, pp. 131–148. Springer Berlin / Heidelberg, 2006. 10.1007/11687238_11.

[8] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using p-trees. In *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*, WebDB '04, pp. 25–30, New York, NY, USA, 2004. ACM.

[9] Zheng Zhang, Shu-Ming Shi, and Jing Zhu. Somo: Self-organized metadata overlay for resource management in p2p dht. *IPTPS2003*, 2003.

[10] Bernhard Seeger and Per-Ake Larson. Multi-disk b-trees. *SIGMOD Rec.*, Vol. 20, pp. 436–445, April 1991.

[11] Jun MIYAZAKI and Haruo YOKOTA. Concurrency control and performance evaluation of parallel b-tree structures. *IEICE TRANSACTIONS on Information and Systems*, 2002.

[12] 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. *DEWS2001*, 2001.

[13] James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, Vol. 3, , November 2007.

[14] Ittai Abraham, James Aspnes, and Jian Yuan. Skip b-trees. In James Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, *Principles of Distributed Systems*, Vol. 3974 of *Lecture Notes in Computer Science*, pp. 366–380. Springer Berlin / Heidelberg, 2006. 10.1007/11795490_28.

[15] Min Luo and Haruo Yokota. Comparing hadoop and fat-btree based access method for small file i/o applications. In *Proceedings of the 11th international conference on Web-age information management*, WAIM'10, pp. 182–193, 2010.

[16] 首藤一幸, 田中良夫, 関口智嗣. オーバレイ構築ツールキット overlay weaver. *SPA2006*, 2006.