Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

Fat-Btree, P-tree, SkipGraph を用いた範囲問合せ性能の比較実験

近藤直樹[†1]　羅敏[†1]
渡辺陽介[†2]　横田治夫[†1]

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。そこで分散インデックスという手法が提案されている。また、データアクセスにおいては属性値の範囲内に入るデータを検索する範囲問合せというデータアクセスがよく行われ、複数の計算機にデータが分散されても効率よく検索できることが要求されている。そのような範囲問合せ可能な分散インデックスが新たに提案されている。しかし、それらの分散インデックスはまだ十分には比較はされていない。本研究では、同じ環境で範囲問合せ可能な分散インデックス手法を比較することを目的とする。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、P-tree と SkipGraph を比較する。

Conparison Experiment of Performance for Range Queries using Fat-Btree, P-tree, SkipGraph

Naoki Kondoh ,[†1] Min Luo ,[†1] Yousuke Watanabe [†2]
and Haruo Yokota[†1]

Due to explosive increasing of data, data is managed with multiple machines. Index is used to access to distributed data efficiently. But, centralized index often becomes a bottle neck in distributed systems. Whereat distributed indexing is widely used. And, we also need efficient range query processing on distributed data in multiple machines. There are some proposals, which can process range query efficiently. But, there is no enough comparison between these proposals. In this paper, we compare to the methods, which can do range query, on same environment. This paper covers Fat-Btree, P-tree and SkipGraph.

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ボトルネックとなってしまう。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複数の計算機でどのように分担させるか、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。

データアクセスにおいては、ある属性のキーに一致するデータを取得する完全一致問合せがある。完全一致問合せのほかには、ある範囲内のキーのデータを取得する範囲問合せもよく行われ、複数の計算機にデータが分散されても効率よく問合せできることが要求されている。効率のよい範囲問合せを行なうには、あらかじめキーによってデータをソートさせておく必要がある。基本的に、ハッシュテーブルを用いる分散インデックスは、キーをハッシュ関数に適用するので事前にソートされおらず範囲問合せできない。また、木構造を用いる分散インデックスはキーによってソートされているので効率がよいとされている。

我々の研究室では分散データベース環境におけるデータのインデックスとして B+-tree 構造をベースにした Fat-Btree[1)] を提案している。Fat-Btree はキー空間を複数計算機で範囲分割して、データを分散化する。また、各計算機でのデータ構造は 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。

そのほかの分散インデックスは、近年 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

†1 東京工業大学情報理工学研究科計算工学専攻
Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†2 東京工業大学学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

ブルを分割し、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。また、分散ハッシュテーブルにおいても範囲問合せの要求があり、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)]。

Skip lists[9)] をベースにした SkipGraph[10)] という分散インデックスも範囲問合せが可能な手法として提案されている。SkipGraph は木構造をベースにした分散インデックスと同様に範囲問合せ可能であるが、障害のある計算機の割合が高くても他の計算機へのルーティングが可能である特徴がある。

このように範囲問合せ可能な分散インデックス手法が新たに数多く提案されてきている。しかし、このような範囲問合せ可能な分散インデックスの比較はまだ十分には行われていない。本研究では、同じ計算機環境で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。範囲問合せ可能な分散インデックスを比較評価することによってどの分散インデックス手法がよいかを知ることができる。本論文では、分散インデックスの比較対象に P-tree[8)]、SkipGraph[10)] と Fat-Btree[1)] を選んだ。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。

P-tree と Fat-Btree のみで問合せにかかった時間を比較した。本論文では SkipGraph を比較対象に加え、さらに実験データとしてより大規模な実データを使用している。具体的には Wikipedia のタイトルをキーとしたテキストデータを使用し、タイトルの範囲問合せの性能を比較する。また、挿入の時間や問合せのスループットの算出だけでなくより詳しい分析を試みた。

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

2. 分散インデックス

2.1 分散ハッシュテーブル

分散ハッシュテーブルは、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。計算機間にまたがってデータを検索する際のルーティングの方法は様々である。多くの手法では計算機を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。それぞれの計算機が自分から出るリンクを管理するこ

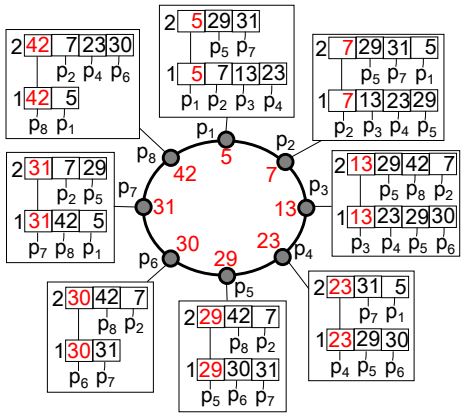

図 1 P-tree

とですべての計算機を参照可能にする。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、$O(logN)$ 回の通信は計算機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[11)] 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。

2.1.1 P-Tree

特徴 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計算機の管理をするが、計算機はさらに範囲問合せに必要な B+-tree を別に持つ。

データ配置法 P-tree では、それぞれの計算機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。各計算機に配置される B+-tree

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

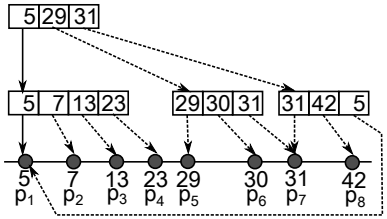

図 2 p_1 から見た P-tree のノード

には、それぞれの計算機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計算機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計算機の情報を格納する。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。

探索方法 P-tree での探索方法は、探索を要求した計算機が自身の B+-tree を根から探索する。探索中の計算機に他の計算機への参照があれば、その計算機に通信し、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 は計算機の管理を Chord で行っているので計算機の追加、削除に対応できるようになっている。計算機の追加はその計算機が管理する範囲を持つ計算機から P-tree をコピーしてそれぞれの層の左端を追加した計算機のものにする。削除された計算機はほかの計算機が削除を探知したらそれぞれの計算機の B+-tree から削除することで対応する。計算機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。

2.2 並列 B-Tree

並列 B-Tree は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計算機にインデックス全体をコピーする方法 (CWB: copy whole btree)、ひとつの計算機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計算機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計算機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計算機にアクセスが集中してボトルネックになる。

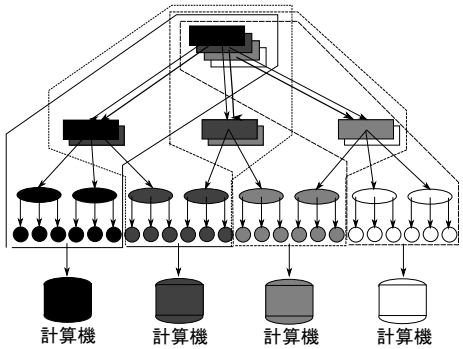

図 3 Fat-Btree

この問題点を解決するために、我々は Fat-Btree[1),12),13)] を提案している。

2.2.1 Fat-Btree

特徴並列 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計算機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計算機にコピーがあるため探索の時のアクセス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。

データ配置法 Fat-Btree ではデータのレコードを各計算機に均等に配分し、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計算機に配置する (図 3)。図 3 の色分けはそれぞれの計算機が持つ B+-tree の部分木を表す。

探索方法 Fat-Btree の探索方法は、探索を要求した計算機の B+-tree の根からまず探索する。途中のノードで他の計算機への参照があれば、その計算機に通信し、木の探索を継続する。各計算機の葉ノードはデータが含まれている。また、各計算機が管理する Fat-Btree のデータの量を隣接する計算機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。

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

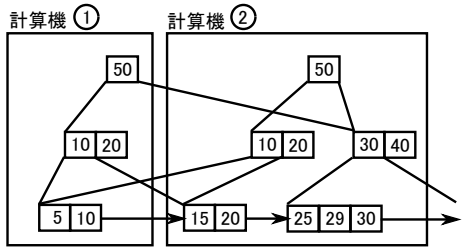

図 4 Fat-Btree の具体例

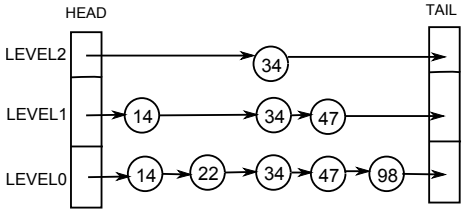

図 5 Skip list

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 の葉ノード間のリンクをたどることによって計算機 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。

2.3 Skip lists を用いた分散インデックス

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 (図 5)。単方向リストを多層にしたものであり、層を重ねていく毎に一つ下の層にあるデータが確率によって除かれていく。この Skip list は、B-tree と同等の特徴を持っている。

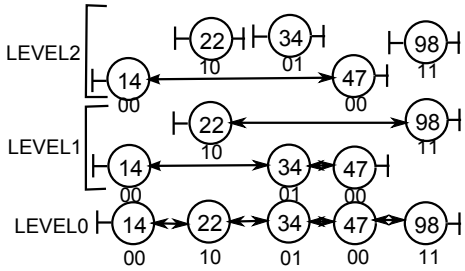

図 6 SkipGraph

Skip list で探索する場合は、一番上の層の HEAD からたどることで操作する。途中で見つからない場合、たどった要素から一つ下の層に移り探索を再開する。

Skip lists を用いた分散インデックスには SkipGraph[10] 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。SkipGraph は範囲問合せを効率よく処理することができる。通常の SkipGraph はデータ一つに対して一つのノードを割り振る。つまり、データ量に応じてノードを用意する必要がある。そこで一つのノードに複数のデータを保持するような提案がされている[14]。SkipGraph と B-tree の利点を合わせた Skip B-Trees[15] なども提案されている。

2.3.1 SkipGraph

SkigGraph は Skip list をベースにしている。SkipList との違いは上の層に行くに従って要素を除くのではなく、複数リストを持つ層にしている (図 6)。また、リストが双方向リストになっている。

SkipGraph では各ノードには MembershipVector と呼ばれるビット列が与えられ、それによりどの計算機と結合すればよいかが決定される。図 6 では、MembershipVector は計算機を表わす数字の下に示されている。LEVEL の値の分だけ MembershipVector の先頭ビットを見て、ビット列の値が同じもの同士をリスト化する。つまり、LEVEL0 のときは全ての計算機が同じリストに属することになる。

SkipGraph の探索方法は、探索を要求した計算機から一番上のリストで探索する。リストをたどる場合は、他の計算機に探索を転送することになる。SkipList と同様に探索中のリストにキーが存在しなければ、下の層に移り探索を再開する。例えば、図 6 の 14 の計算機から 98 を探索する場合を考える。まず、14 の計算機の一番上の層 LEVEL2 を探索する。14 は 47 と接続していて、目的の 98 より小さいので 47 の計算機に問合せを転送する。47

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

の計算機では次の要素がなく、層を下っていって LEVEL0 の層で 98 の計算機を発見する。SkipGraph の範囲問合せは範囲の開始点の値を持つ計算機をまず探索し、見つかった後は LEVEL0 のリストをたどることで範囲問合せをする。

SkipGraph における計算機の追加は LEVEL0 からリストに追加していき、削除では一番上の層から除いていく。

3. 実　　験

本研究では Fat-Btree、P-tree と SkipGraph 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する。

3.1 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 2, 4, 8, 16 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

今回は計算機の追加・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。

次に実験データについて述べる。Wikipedia の英語版の一部のタイトル情報を用いて実験に用いる。

- キー: 英語版 Wikipedia のタイトル
- 値: 文字列

データサイズは、アスキー文字のみを使用している Wikipedia のタイトルからランダムに抽出した 5 万件である。ただし、Fat-Btree においては実装上の制約によりタイトルの偏りを考慮した整数のキーを使用する。

実験に用いたプログラムの概要を説明する。それぞれの実装の構成を図 7 に示す。

- Fat-Btree は我々の研究グループでこれまでに実装したもの[16] を使用する。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。
- P-tree 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。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[17] 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。また、範囲問合せの効率化のため PostgreSQL 内部ではキーに対して B-tree のインデックスを構築した。
- SkipGraph は、SkipGraph の論文[10] のアルゴリズムをもとに我々が独自に実装した。SkipGraph の通信機能の部分のみに Overlay Weaver[17] を用いた。我々の実装では、Skip lists を一つの計算機に複数のデータを保持できるように計算機に振られた ID のインデックスとして用いた。Skip lists で計算機間のリンクをたどり、その計算機の中からデータを取得する。
  計算機にはキーの担当範囲が分るように ID がランダムに振られている。データの挿入はまずデータの担当計算機をこの SkipGraph を用いて探索しなければならない。この実装では、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。P-tree と同様にキーに対応するデータは PostgreSQL に格納されている。

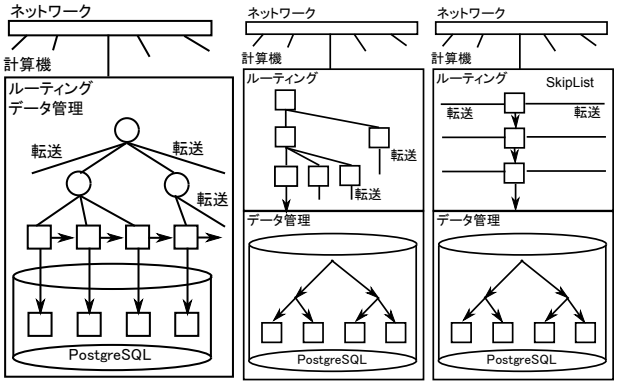

図 7 Fat-Btree, P-tree, SkipGraph の実装構成

3.2 実験方法

問合せの実行方式は、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) を用い

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

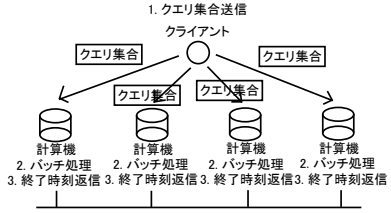

図 8 並列実行

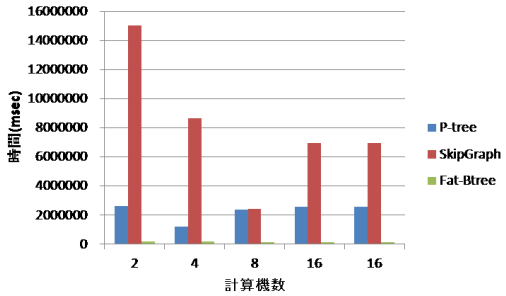

図 9 並列データ挿入の実験

た。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、最後に終了した計算機の時間を終了時間とする。

3.3 実験結果

3.3.1 データの挿入

データの挿入にかかる時間を比較する。実験方法はデータがなにもない状態から Wikipedia のタイトルをキーとするデータを 50000 件挿入するまでの時間を計測する。また、時間の内訳を把握できるようにルーティングや PostgreSQL にかかった時間も計測する。並列実行方式なので 50000 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。

データの挿入にかかった時間を図 9 に示す。図の縦軸は、開始から全部の計算機の処理が完了するまでの時間である。また、時間の内訳としてルーティングにかかる時間と PostgreSQL の操作にかかる時間の全ノードでの合計値を図 10 に示す。

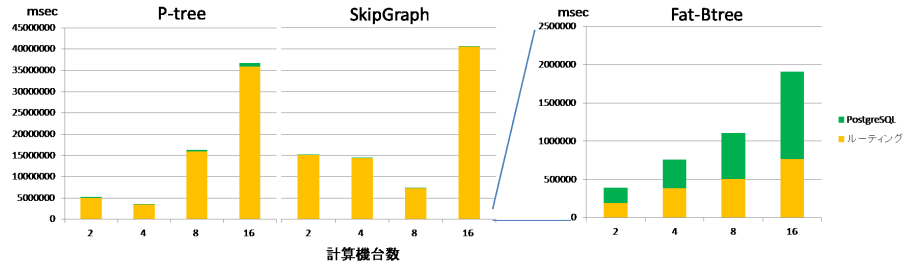

図 10 並列データ挿入の実験:全ノードでの処理時間の合計の内訳

図 9 の実験結果から Fat-Btree の方が速く、計算機台数が増えるごとに並列に処理している効果があった。また、P-tree では 4 台目、SkipGraph では 8 台目まで並列処理の効果があったが、その後はまた時間がかかる傾向になった。P-tree と SkipGraph では P-tree の方が挿入にかかる時間が短かかった。図 10 でかかった時間の内訳を見ると、P-tree や SkipGraph はルーティングの影響が非常に大きいことが分る。

3.3.2 範囲問合せ

範囲問合せの性能を比較する。比較には範囲問合せのスループットを用いる。また、時間の内訳を把握できるようにルーティングや PostgreSQL にかかった時間も計測する。事前にデータが 50000 件挿入されている状態で、クエリには結果として 100 件のデータが取得されるようなものを用いた。並列実行方式によりそれぞれの計算機が 1000 個のクエリを実行する。

範囲問合せを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 に示す。図 11 より Fat-Btree は計算機台数に応じてスループットが向上した。また、2 台目以降の P-tree と SkipGraph は 4 台目でスループットの低下が見られた。計算機台数が 8 台以降は Fat-Btree の方が性能がよかった。

時間の内訳としてルーティングにかかる時間と PostgreSQL の操作にかかる時間の結果を図 12 に示す。 P-tree と SkipGraph ではルーティングに時間がかかり、Fat-Btree では PostgreSQL の操作に多く時間がかかったことが分った。

3.3.3 完全一致問合せ

キーによる完全一致問合せの性能を比較する。比較には範囲問合せのスループットを用いる。また、時間の内訳を把握できるようにルーティングや PostgreSQL にかかった時間も計測する。事前にデータが 50000 件挿入されている状態で問合せを行なった。並列実行方

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

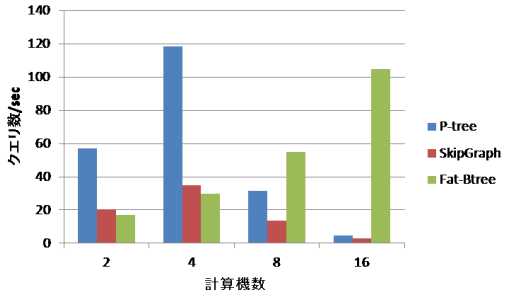

図 11 並列範囲問合せの実験

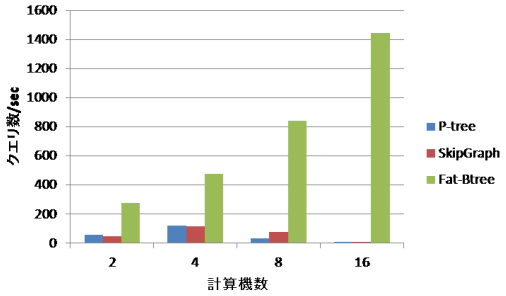

図 13 並列完全一致問合せの実験

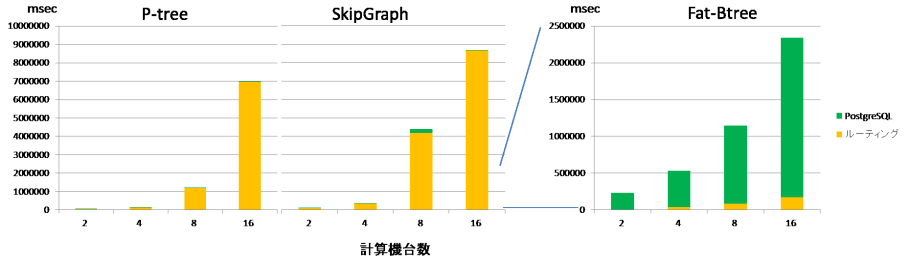

図 12 並列範囲問合せの実験:全ノードでの処理時間の合計の内訳

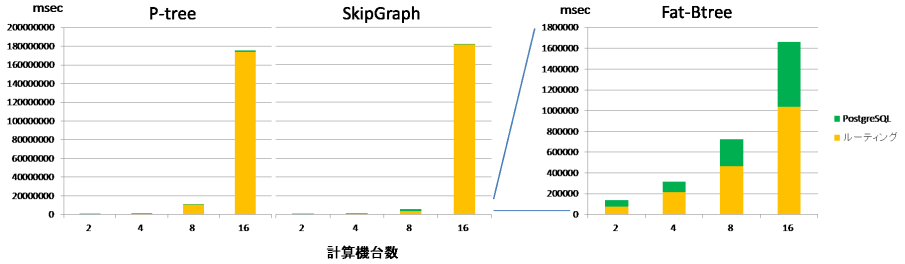

図 14 並列完全一致問合せの実験:全ノードでの処理時間の合計の内訳

式によりそれぞれの計算機が 10000 個のクエリを実行する。

完全一致問合せを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 に示す。図 13 の実験結果から、Fat-Btree の方がスループットがよく、計算機台数が増える毎に向上しているのが分かった。P-tree と SkipGraph は計算機台数が 4 台からスループットの低下が見られた。

時間の内訳としてルーティングにかかる時間と PostgreSQL の操作にかかる時間の結果を図 14 に示す。 ルーティングにかかった時間はどれも計算機台数に応じて多くなり、ほとんどがルーティングにかかっていることが分った。

3.4 実験まとめ

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 と SkipGraph はスループットが伸びなかった。時間の内訳を計測することでほとんどの時間はルーティングにかかっていることが分った。ルーティングに時間がかかる原因としてはメッセージ通信回数の差による場合と、実装方式の違いによる場合の 2 つが考えられる。これに関しては追加の実験を行う必要がある。

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。

4. 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、P-tree、SkipGraph と 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

Vol.2011-DBS-152 No.15
Vol.2011-IFAT-103 No.15
2011/8/3

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。また、問合せにかかる時間のほとんどがルーティングにかかる時間であることが判明した。

今後の課題として、より大規模な実験環境 (計算機台数やデータ数) で比較する必要がある。また、ルーティング時間とクエリの転送回数の関係やその影響をさらに詳しく分析する必要がある。

謝　　辞

本研究の一部は、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。

参 考 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) BenY. Zhao, JohnD. Kubiatowicz, and AnthonyD. 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 DeTurck, 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) William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, Vol.33, pp. 668–676, June 1990.
10) James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, Vol.3, , November 2007.
11) Zheng Zhang, Shu-Ming Shi, and Jing Zhu. Somo: Self-organized metadata overlay for resource management in p2p dht. *IPTPS2003*, 2003.
12) Jun MIYAZAKI and Haruo YOKOTA. Concurrency control and performance evaluation of parallel b-tree structures. *IEICE TRANSACTIONS on Information and Systems*, 2002.
13) 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.
14) 小西佑治, 吉田幹, 寺西裕一, 春本要, 下條真司. 単一ピアに複数キーを保持可能とする skipgraph 拡張の提案. 情報処理学会研究報告, 2007.
15) 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.
16) 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, Berlin, Heidelberg, 2010. Springer-Verlag.
17) 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.