複数分散インデックス手法の実装・比較のための共通フレームワークの実現

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　データセンターにおいて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数多く提案されている。また、範囲問合せをより効率良くする改良手法なども提案されている。しかし、それらの手法は共通の環境では十分には比較されていない。本研究では、分散インデックス手法の実装と比較を容易にするための共通フレームワークを構築する。範囲問合せなどの分散インデックスに必要な各処理を、手法に依存しない基本操作の組合せとして共通化できることを示す。各手法は、フレームワークのインターフェースにあわせて、データ構造に対する基本操作を実装することで実現する。本稿では、実際に提案フレームワークを用いて実装した複数の分散インデックス手法を比較する。

キーワード　分散インデックス、範囲問合せ

Development of a Common Framework for Implementation and Comparison of Distributed Indices

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 data management among multiple machines have become popular due to explosively increasing data in data centers. In such kind of management, indices are used to access distributed data efficiently. Furthermore, since centralized index mechanism is not scalable, many distributed index methods and their improvements to treat range queries are proposed. But, the costs to implement and compare them are quite expensive. Our research goal is building a common framework, which can make easy to implement distributed index methods and compare them. We show that the main procedures of distributed index procedures are common and describable by basic operations. Based on our framework, we can easily implement individual index methods. In this paper, we compare three distributed index methods on our proposed framework.

Key words　Distributed Index, Range Query

1. はじめに

データセンターなどで扱う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 [1] [2] [3] 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。

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や、ある範囲内のキーのデータを取得する範囲問合せがよく行われ、複数の計算機にデータが分散されてい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。

我々の研究グループ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 をベースにした Fat-Btree [1] を提案している。そのほかの分散インデックスとしては、P2P の分野での利用を想定した分散ハッシュテーブルの研究が盛んに行われている [4] [5] 。P2P 環境で利用するために、問合せ要求の高度化への対応が求められるようになり、分散ハッシュテーブルにおいても範囲問合せが可能な手法が提案されてきている [6] [7] [8]。Skip lists [9] をベースにした SkipGraph [2] 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。

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。比較のために分散インデックス手法を一から実装することはとても労力が必要なことであり、一から実装したもの同士の比較ではメッセージ通信やクエリのスレッド生成方法など分散インデックスのアルゴリズムとは直接関係のない部分で公平でない場合がある。

本研究では、分散インデックス手法の実装と比較を容易にするための共通フレームワークを構築する。範囲問合せなどの分散インデックスに必要な各処理を、手法に依存しない基本操作の組合せで実現できることを示す。提案フレームワークを用いることにより、分散インデックスのアルゴリズムのみを実装するだけで実現可能となり、アルゴリズムとは直接関係のない部分はフレームワークの実行環境となるシステム側が提供するので比較実験の公平さが保たれる。また、比較の際の共通のログ出力の機能もあり、ログ解析はそれぞれの手法毎に用意する必要がなく、ログ解析も容易となる。本稿では、実際に提案するフレームワークを用いて、Fat-Btree [1]、SkipGraph [10]、P-Ring [3] を実装し比較した。実験として、データの挿入、完全一致問合せ、範囲問合せの処理性能を測定し比較を行う。

以降、2. 節では分散インデックス手法の Fat-Btree、SkipGraph、P-Ring を紹介する。3. 節では提案フレームワークについて述べる。4. 節では比較評価の実験結果を述べる。5. 節では関連研究を述べる。6. 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 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対応できる。

2.1.1 Fat-Btree

特徴 B-tree をベースに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 (図 1)。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ている。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 ではキー空間を各計算機に均等に配分し、データが格納されている葉ノ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 1)。図 1 ではそれぞれの計算機の持つインデックスが色で区別してある。この木構造は、葉ノードに到達する前にノードがとぎれていれば、そのノードを持つ計算機にアクセスする必要がある。

完全一致問合せ手順 表 1 に示す。表中の問合せ転送の項目は、キーの探索中に担当範囲でない計算機から問合せを他の計算機に転送する場合に発生する。

データの挿入手順 表 2 を参照

範囲問合せ手順 表 3 を参照

2.2 SkipGraph

特徴 SkipGraph は Skip lists [9] をベースにし、要素を計算機としている (図 2)。SkipGraph では各計算機に Membership Vector 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同じリストに参加すればよいかが決定される。LEVEL により層の構造となっていて、LEVEL の値の分だけ Membership Vector の先頭ビットが同じものをリスト化する。LEVEL0 のときは全ての計算機が同じリストに属する。

データ配置方法 本来の SkipGraph は各計算機には一つのキーしか保持できない構造であるが、各計算機に複数のキーを保持できるようにした改良手法が存在する [10]。この改良手法では、各計算機にキーのリストを保持する。

完全一致問合せ手順 表 1 を参照

データの挿入手順 表 2 を参照

範囲問合せ手順 表 3 を参照

2.3 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。本来完全一致問合せしか扱えない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] [3]。本論文では、このうち P-Ring [3] を比較対象にする。

2.3.1 P-Ring

特徴 P-Ring 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とデータ配置の管理をするが、計算機はさらに範囲問合せに必要な表を別に持つ。

データ配置方法 P-Ring 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 3)。例えば、図 3 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-Ring 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、Hierarchical Ring (HR) と呼ばれるルーティング情報の表を採用している。各計算機に配置される表は、各行に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

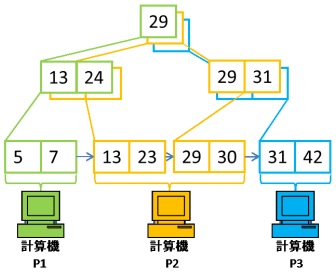

図 1　Fat-Btree

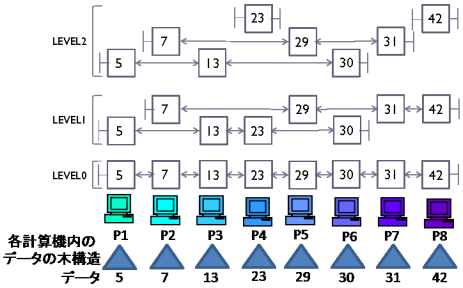

図 2　SkipGraph

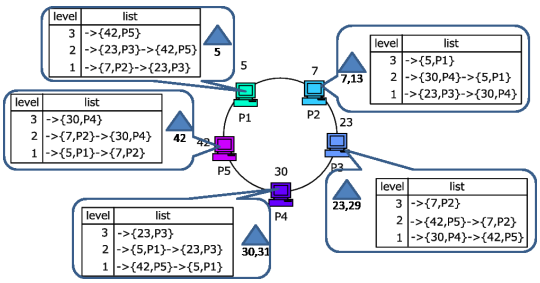

図 3　P-Ring

	Fat-Btree	SkipGraph	P-Ring
① 問合せ受付	(共通)		
② ローカルのインデックスを探索	木構造の根から 検索キーを探索	上位レベルのリストから 検索キーを探索	HR の表の上位レベルのリストから 検索キーを探索
③ 問合せ転送	木構造のノードのリンク先を 持っていない場合	より近い計算機がある場合	より近い計算機がある場合
④ 探索再開	木構造のノードのリンク先から	探索途中のレベルから	上位レベルから
⑤ データにアクセス	データを持つ葉ノードに 読み込みロックを掛る	インデックスに 読み込みロックを掛る	インデックスに 読み込みロックを掛る

表 1　問合せ処理手順: 完全一致問合せ

	Fat-Btree	SkipGraph	P-Ring
① 問合せ受付	(共通)		
② ローカルのインデックスを探索	木構造の根から 検索キーを探索	上位レベルのリストから 検索キーを探索	HR の表の上位レベルのリストから 検索キーを探索
③ 問合せ転送	木構造のノードのリンク先を 持っていない場合	より近い計算機がある場合	より近い計算機がある場合
④ 探索再開	木構造のノードのリンク先から	探索途中のレベルから	上位レベルから
⑤ データにアクセス	葉ノードと更新に影響するノードに 書き込みロックを掛る	インデックスに 書き込みロックを掛る	インデックスに 書き込みロックを掛る

表 2　問合せ処理手順: データの挿入

	Fat-Btree	SkipGraph	P-Ring
① 問合せ受付	(共通)		
② ローカルのインデックスを探索	木構造の根から 検索キーを探索	上位レベルのリストから 検索キーを探索	HR の表の上位レベルのリストから 検索キーを探索
③ 問合せ転送	木構造のノードのリンク先を 持っていない場合	より近い計算機がある場合	より近い計算機がある場合
④ 探索再開	木構造のノードのリンク先から	探索途中のレベルから	上位レベルから
⑤ 問合せ分割	B+-tree のサイドリンク	LEVEL0 のリスト	表のレベル 1 のリスト
⑤ データにアクセス	問合せ範囲内の葉ノードに 読み込みロックを掛る	インデックスに 読み込みロックを掛る	インデックスに 読み込みロックを掛る

表 3　問合せ処理手順: 範囲問合せ

されている。一つの行に格納するリストの要素の上限があり、それを Hierarchical オーダーという。オーダーを O として、レベルを l とするとその行のリストは時計回りにある計算機を O^{l-1} づつスキップするようなリストになる。例えば、図 3 はオーダー 2 の例で p_1 の表の最も上の行には計算機を時計回りに 2^{3-1} 先の 42 を指している。

完全一致問合せ手順 表 1 を参照

データの挿入手順 表 2 を参照

範囲問合せ手順 表 3 を参照

2.4　共通部分についての分析

上記 3 手法を分析すると、各手法に固有のデータ構造を扱う部分などは独立であるが、問合せ処理の流れなどはほぼ共通であることが分かる。具体的には以下のものが共通化可能である。

- ローカルのインデックスを探索
- データにアクセス
- データにアクセスするときの排他的制御
- クエリの転送
- 範囲問合せのクエリの分割

よってこれらを各手法とは別に提供できれば、手法毎の実装コストを減らすことができると考えられる。

3.　提案フレームワーク

分散インデックスの実装と比較を容易にするためのフレームワークを提案する。分散インデックス手法の開発者は、フレームワークの提供するインターフェースにあわせて、データ構造に対する操作を実装するだけでよい。

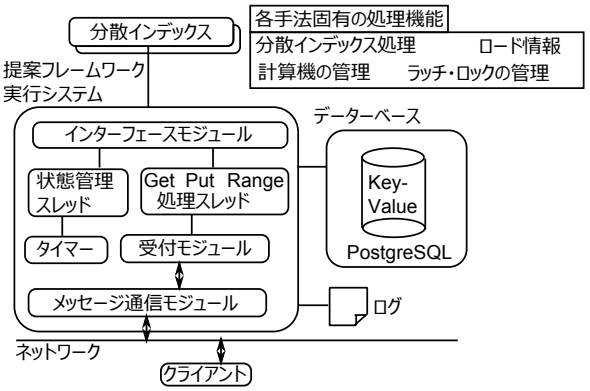

図 4　提案フレームワークの実行環境

3.1　提案フレームワークの実行環境

システムの構成図は、図 4 に示すとおりである。フレームワークは分散インデックス実装に直接関係のない部分とクエリを処理する際の共通部分を提供している。図 4 のメッセージ通信モジュールはクエリの通信や、その他の通信に使用される。受付モジュールはクエリを受けつけ、クエリを処理するスレッドをたてる。各スレッドは検索やデータ挿入を処理するためのもので、受付モジュールやタイマーモジュールから使われる。分散インデックスの操作はインターフェースモジュールを通じて、呼び出される。データのディスクへの保存はデータベースを使うことで行う。ログの出力は、それぞれのモジュールから出力される。

3.2　分散インデックス手法インターフェース

フレームワーク上で実装に必要な分散インデックスの主な処理を表 4 に示す。表中の各処理は完全一致問合せ、データの挿入、範囲問合せで必要になるものであり、開発者は各手法固有の処理をこれらの内部で実装しなければならない。表のデータノードとは、Fat-Btree におけるリーフノードのようなデータ構造を共通化したもので、データがいくつか格納されており、格納できるキー数が決められている。上限を越えるとデータノードの分割が起こり、ローカルのデータ構造に更新が発生する。

3.3　共通化されたクエリ処理アルゴリズム

ここでは、範囲問合せなどの分散インデックスに必要な処理が、手法に依存しない基本操作の組合せで実現できることを示す。

完全一致問合せ フレームワークで提供する共通な完全一致問合せのアルゴリズムを Algorithm 1 に示す。完全一致問合せはキーに対応するデータを取得する。そのため、まずキーをインデックスで探索する。探索途中でその計算機が持たないインデックスを探索する場合には、他の計算機へクエリを転送する必要がある。インデックスを探索してデータノードに到達すれば、アクセスする。インデックスを探索すると他の計算機への転送情報かデータノードに到達する。データはデータノードにある程度まとめて格納されている。データのアクセスにはラッチやロックが必要な手法が多い。

範囲問合せ フレームワークで提供すべき共通なアルゴリズムを Algorithm 2 に示す。範囲問合せではある範囲に入るキーの値を探索する。問合せ範囲は最初と最後のキーを指定することで指定する。まず、範囲問合せは最初のキーを完全一致問合せと同様に探索する。探索により、最初のキーを担当する計算機が見つかる。しかし、問合せ範囲すべてをその計算機が満すとは限らないので担当範囲を調べ、必要ならキー空間上隣の計算機にクエリを転送する。このようにクエリは複数の計算機で処理される可能性があるため、完全一致問合せとは違い、処理した計算機の担当範囲も収集することでクエリがすべて処理されたかを判定する必要がある。

データの挿入 フレームワークで提供すべき共通なアルゴリズムを Algorithm 3 に示す。データの挿入は完全一致問合せと同様にキーを探索することから始める。担当計算機まで到達し、データノードまで判断する。その後、データノードをロックして、データを挿入する。データを挿入した際、データノードのデータ量が上限を超ている場合、各手法に知らる必要がある。

Algorithm 1 完全一致問合せ get(key, info)

入力: 検索キー key、クエリ元情報 $info$

```
node = searchKey(key, info)
if node == データノード then
    lock = searchData(node)
    データベースからキーに対応する値を取得する
    endSearchData(lock)
    結果を info に返す
else if node == 転送情報 then
    転送先へクエリを転送 get(key, info)
end if
if info == 自身の計算機 then
    結果を待つ
end if
```

Algorithm 2 範囲問合せ range(key1, key2, info)

入力: 検索キー $[key1, key2)$、クエリ元情報 $info$

```
node = searchKey(key1, info)
if node == データノード then
    r = getResponsibleRange()
    if r.end ≦ key2 then
        n = getNextMachine()
        クエリを n へ転送 range(key1, key2, info)
    end if
    nodes = 検索に関わるすべてのデータノード
    lock = seachData(nodes)
    データベースからキーの範囲に対応する値を取得する
    endSearchData(lock)
    結果を info に返す
else if node == 転送情報 then
    転送先へクエリを転送 range(key1, key2, info)
end if
if info == 自身の計算機 then
    問合せ範囲がすべて処理されるまで結果を待つ
end if
```

表 4　分散インデックス手法実装に必要な主な操作

名前	役割	入力	出力
searchKey	インデックスを検索目的で探索	探索キー	転送情報かデータノード
searchData	データノードを読み込む準備	データノード	ロック情報
endSearchData	データノードの読み込み終了の処理	ロック情報	
updateKey	インデックスを更新目的で探索	探索キー	転送情報かデータノード
updateData	データノードに書き込む準備	データノード	ロック情報
endUpdateData	データノードの書き込む終了の処理	ロック情報	
ackUpdateNode	データノードが上限に到達した場合の通知	データノード	
getResponsibleRange	計算機の担当範囲を返す		計算機の担当範囲
getNextMachine	キー空間上隣の計算機の情報を返す		IP アドレス

Algorithm 3 データの挿入 put(key, value, info)

入力: 更新データ $key, value$、クエリ元情報 $info$

```
1: node = updateKey(key, info)
2: if node == データノード then
3:     lock = updateData(node)
4:     データノードにデータを挿入する
5:     endUpdateData(lock)
6:     if データノードが上限を越える then
7:         ackUpdateNode(node)
8:     end if
9:     結果をクエリ元に返す
10: else if node == 転送情報 then
11:    転送先へクエリを転送 put(key, value, info)
12: end if
13: if info == 自身の計算機 then
14:    結果を待つ
15: end if
```

4. 実　　験

提案フーレムワーク上に実装した Fat-Btree、SkipGraph と P-Ring の性能を比較する。

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
- OS: Redhat Linux 9.0 (Kernel 2.4.20)
- Java 1.6.0_27
- PostgreSQL 8.2.21

実験データには、Yahoo! Cloud Serving Benchmark (YCSB) [11] を用いる。YCSB とは、分散データベース用のベンチマークツールである。ベンチマークではレコードの挿入、読み書きや更新、スキャン操作が行われる。ワークロードを生成する際、この操作の数や割合を指定可能である。この実験では YCSB からテストデータを生成し、提案フレームワークに適したフォーマットに変換したものを用いる。

データの挿入実験には、テストデータの複数フィールドから一つのフィールドのみを選びデータとして用いた。完全一致問合せ実験には、使用計算機数 ×10,000 件の完全一致問合せと範囲問合せを今回の実験で用いられる最大操作数 320,000 件をあらかじめ生成した。YCSB では範囲問合せは起点となるキーと取得数を指定するが、提案フレームワークに適用するため取得数が同じになるように最初のキーと最後のキーのペアに変換しておく。

	フレームワーク	手法依存部分	フレームワークの割合
Fat-Btree	2188	1776	55%
SkipGraph		607	78%
P-Ring		535	80%

表 5　各手法のコード量

本フレームワークを用いて実際に Fat-Btree、SkipGraph、P-Ring を実装した。各手法のコード量を表 5 に示す。コード量を測定する際、コメントや空行を除きコードスタイル統一ツールを用いて、整形した後のコードの行数を測定した。実装した各手法においてはフレームワークが占める割合が 50% 以上であり、実装コストの削減が確認できた。

実験に用いた各インデックス手法のパラメータについて説明する。Fat-Btree は、今回の実験ではファンアウトは 64、一つのデータノードに格納可能なデータの上限値は 32 としている。SkipGraph は、membership vector はランダムな値を用いた。また、SkipGraph の層は最大 5 層までとした。ローカルでのデータ管理にはファンアウト 64 でデータノードの上限が 32 の B+-tree 構造を用いた。P-Ring の Hierarchical Ring のオーダーは 2 とした。ローカルでのデータ管理にはファンアウト 64 でデータノードの上限が 32 の B+-tree 構造を用いた。

4.2 実験方法

問合せの実行は、全部の計算機が並列してクエリを発行する。時間計測は各計算機で同数のクエリの実行を同時に開始してから終了するまでの時間であり、最後に終了した計算機の終了時間を全体の終了時間とする。

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データを 50,000 件挿入するまでの時間を計測する。50,000 個のクエリを計算機の台数に応じて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 5 に示す。図の縦軸は、開

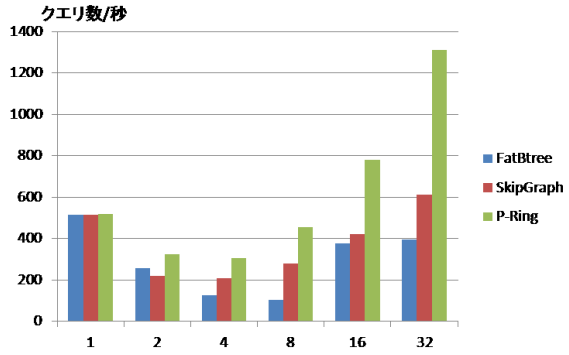

図 5　データ挿入の実験: スループット

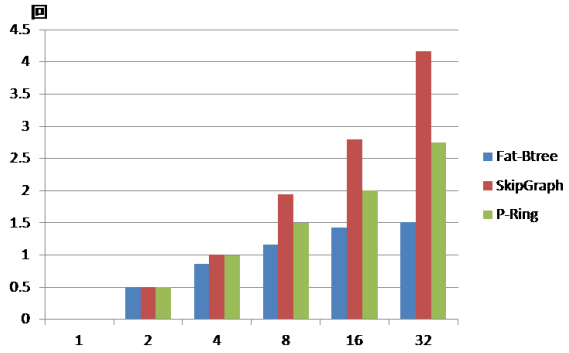

図 6　データ挿入の実験:クエリの転送回数の平均

	平均	分散	最大値
Fat-Btree	2.9	4.7	27
SkipGraph	3.0	1.8	29
P-Ring	2.8	1.0	11

表 6　計算機台数 32 台のデータ挿入実験: スレッドの数

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。

図 5 の実験結果から P-Ring の方が性能が良く、計算機台数 32 台のとき Fat-Btree と比べて 3 倍以上良かった。P-Ring と SkipGraph は台数に応じて並列に処理している効果があった。Fat-Btree は 8 台まで性能が下がり、16 台と 32 台では性能が向上した。

各手法間で性能に差がでている原因を確認するためクエリの転送回数を調べた (図 6)。図 6 から転送回数は SkipGraph が一番多く、Fat-Btree が一番少ないことが分かった。P-Ring と SkipGraph はアルゴリズムの性質上、データ挿入時に計算機間の転送情報の更新が発生せずデータ挿入の影響はその担当範囲の計算機のみに留まるので、これらは転送回数が性能に影響したと考えられる。Fat-Btree は転送回数が最も少ないのにもかかわらず性能が悪い。そのことから、原因は転送回数ではなく、Fat-Btree がデータ挿入時に計算機間のリンク情報の更新を必要とするためで、更新による同期が多く発生していると考えられる。同期が発生していることを確認するために 32 台での実験中のスレッドの数を表 6 に示した。これによるとスレッド数の平均はあまり変らないが、分散が Fat-Btree の方が大きいことが分かり、Fat-Btree は同期用のスレッドが同時に多く発生していることが確認できる。

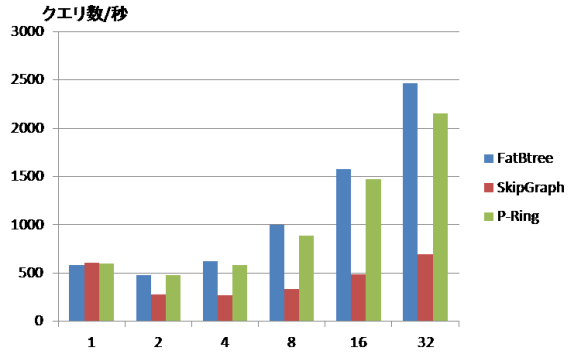

図 7　完全一致問合せの実験: スループット

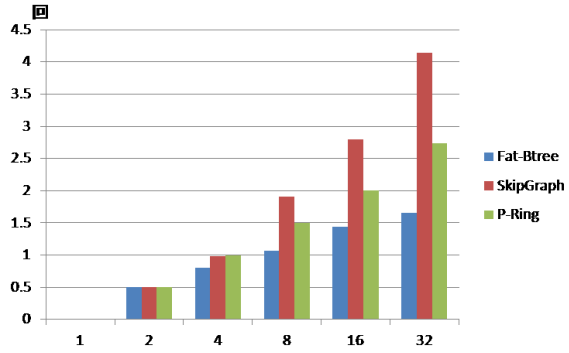

図 8　完全一致問合せの実験: クエリの転送回数

この実験ではデータ挿入時に計算機のリンク情報の更新が発生する手法は同期コストが大きく性能が良くないことが確認できた。

4.3.2　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。事前にデータが 50,000 件挿入されている状態で問合せを行う。それぞれの計算機が 10,000 個のクエリを実行する。

単位時間あたりのシステム全体の問合せ処理数 (スループット) を図 7 に示す。図 7 の実験結果から、どの手法も計算機台数が増える毎にスループットが向上しているのが分かった。最も良い性能を示したのは Fat-Btree であった。

各手法間で性能に差がでている原因を確認するためクエリの転送回数を調べた (図 8)。図 8 から、転送回数は SkipGraph が一番多く、Fat-Btree が一番少ないことが分かった。データ挿入時とは違い、検索の時は同期などが発生しないため、転送回数がどの手法の性能にも大きく影響したと考えられる。また、Fat-Btree と P-Ring との転送回数の差の原因は木構造のファンアウトの差であり、今回の実験で用いたパラメータでは、Fat-Btree の方がインデックスの木の高さが低く目的の計算機まで速く到達できたと考えられる。

4.3.3　範囲問合せ

最後に範囲問合せの性能を比較する。事前にデータが 50,000 件挿入されている状態で問合せを行う。それぞれの計算機が 10,000 個のクエリを実行する。

単位時間あたりの問合せ処理数 (スループット) を図 9 に示す。図 9 の実験結果から、どの手法も計算機台数が増える毎に向上

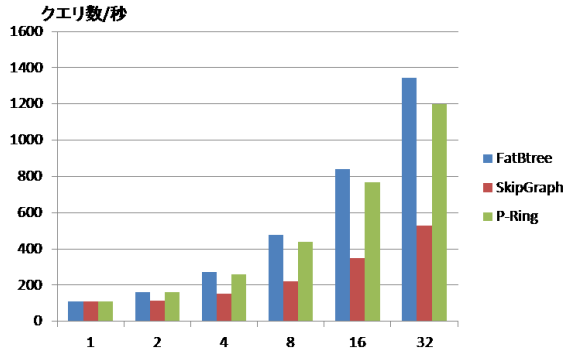

図 9　範囲問合せの実験: スループット

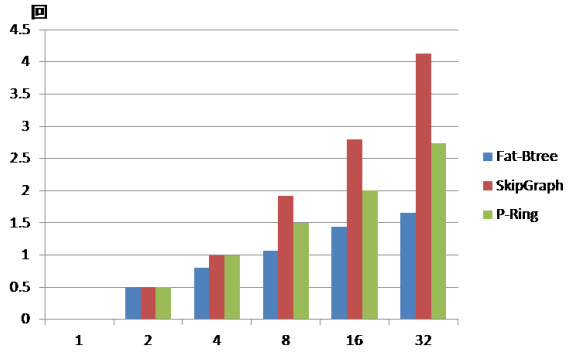

図 10　範囲問合せの実験: クエリの転送回数の平均

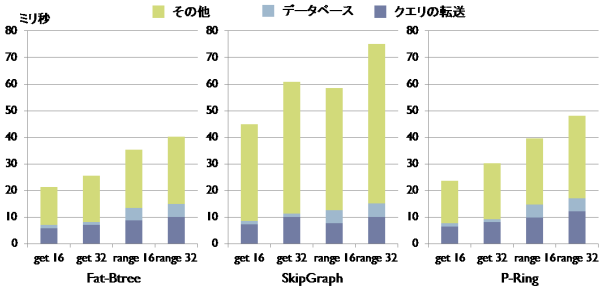

図 11　一つのクエリにかかる時間の内訳

しているのが分かった。最も良い性能を示したのは Fat-Btree であった。

各手法間で性能に差がでている原因を確認するためクエリの転送回数を調べた (図 10)。図 10 から、完全一致問合せと同様の傾向があり、転送回数は SkipGraph が一番多く、Fat-Btree が一番少ないことが分かった。完全一致問合せと同様に転送回数が性能に影響したと考えられる。

完全一致問合せの性能との違いを確認するために、16 台と 32 台の構成における完全一致問合せと範囲問合せの一つのクエリにかかる時間の内訳を確認する (図 11)。図 11 の棒グラフはクエリの転送時間、データベース操作の時間、その他の処理時間 (インデックスやクエリの転送以外の通信など) の一つのクエリにかかる時間の内訳を示している。図 11 横軸の表記の get は完全一致問合せ、range は範囲問合せを示している。図 11 から、それぞれの手法の完全一致問合せと範囲問合せの間で転送とデータベースにかかる時間が増えていることが分かる。このことから範囲問合せはデータベースから複数のデータを取得するため時間がかかり、問合せ範囲を処理するために複数の計算機で処理するためクエリの転送も増えていると考えられる。

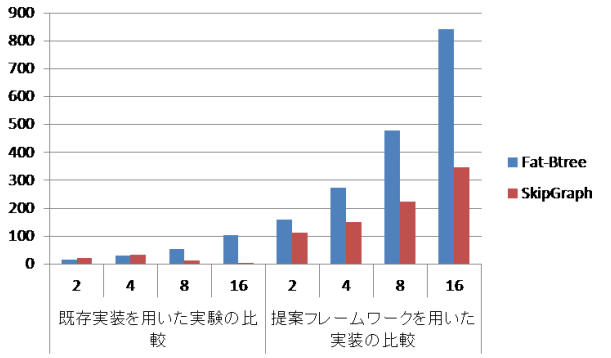

図 12　既存実装を用いた実験と提案フレームワークを用いた実験の比較 (範囲問合せの実験: スループット)

4.3.4　既存実装を用いた実験と提案フレームワークを用いた実験の比較

既存実装による実験と提案フレームワークを用いる実装ではどちらか公平な比較実験ができるかを確認する。既存実装とは、我々の研究室の既存実装である Fat-Btree [12] と既存のフレームワークである Overlay Weaver [13] を用いて我々が独自に実装した SkipGraph を指す。実験方法は、4.3.3 節と同様である。

既存実装を用いた実験と提案フレームワークを用いた実験の範囲問合せのスループットを図 12 に示す。図 12 の実験結果から、既存実装の Fat-Btree のスループットは向上し、既存フレームワークを用いた SkipGraph では 8 台から低下した。提案フレームワークを用いる場合では、それぞれスループットが向上し、Fat-Btree の方が良い性能を示した。

既存実装の Fat-Btree は提案フレームワークを用いる場合と比べて、スループットが良くない。これは、提案フレームワークでは範囲内のデータを一括してデータベースにアクセスしているが、既存実装ではアクセスが一括して行なえない実装となっていて、アクセスはデータ毎となっているためである。既存フレームワークを用いた SkipGraph の実装では、スループットが低下している。これは同時に通信できる数が制限されていて、通信部分が原因となっている。そのため同時に通信できる数の上限を解除する必要があることが分かっている。

既存実装では、それぞれ手法のアルゴリズム以外の部分が原因で性能に差がでていることが分かる。このことから完全に異なる実装同士の比較では、手法のアルゴリズム以外で差が起きてしまいアルゴリズムの公平な比較ができていないと考えられる。提案フレームワークならアルゴリズム以外の部分が共通化されているので、本質的なアルゴリズムの部分の差が実験の結果として表れ、より公平な比較実験ができる。

4.4　実験まとめ

提案フレームワークを用いることでそれぞれ独立して実装した場合に起きるアルゴリズム以外の部分の実装によってもたら

される差を減らし、より公平な比較実験を行なった。

実験結果から、どの手法も台数に応じてスループットが向上することが分かった。Fat-Btree のデータの挿入に関しては更新の同期コストが高く性能が向上しなかった。しかし、Fat-Btree は検索の性能においては一番良い性能を示した。検索問合せの完全一致問合せと範囲問合せの検索に関しては転送回数に大きく影響されることが分かった。

5. 関 連 研 究

Overlay Weaver [13] は構造化されたオーバレイの構築ツールキットであり、分散ハッシュテーブルを構築することを目的としている。P-Ring [3] の P2P フレームワークは P2P 用の分散インデックスのフレームワークである。P2P 環境のため計算機の管理に重点を置いている。SOMO [14] はネットワークのオーバーレイ上に様々なデータ構造を構築する仕組みである。SOMO の論文 [14] には、木構造を構築し、計算機の状態を葉ノードから収集してくるシステムに用いている。これらに対して、本研究の提案するフレームワークでは、分散ハッシュテーブルに限らず様々な分散インデックス手法を実装可能である。

6. ま と め

本論文では、分散インデックスの実装と比較を容易にするフレームワークを提案した。提案フレームワークでは分散インデックスの各操作に必要な処理の共通する部分を抽出できることを示し、実際に提案フレームワークを用いて実装した分散インデックスの比較を行った。比較対象は、FatBtree、SkipGraph と P-Ring を選択し、範囲問合せ、完全一致問合せの処理性能は Fat-Btree、データの挿入では P-Ring が優れていることを確認した。提案フレームワークを用いて複数の分散インデックスの実装とより対等な条件下で比較ができることを示した。

今後の課題として、データの削除やロードバランスなどの対応可能な処理の追加、通信やスレッドの生成部分などのフレームワークの改善、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。

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。

文　　献

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

[2] James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, 2007.

[3] Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke, and Jayavel Shanmugasundaram. P-ring: an efficient and robust p2p range index structure. *SIGMOD '07*, 2007.

[4] 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.*, 2001.

[5] Petar Maymounkov and David Mazie`res. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[6] Domenico Talia and Paolo Trunfio. Dynamic querying in structured peer-to-peer networks, 2008.

[7] Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou. Replication, load balancing and efficient range query processing in dhts. *Lecture Notes in Computer Science*, 2006.

[8] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using p-trees. *WebDB '04*, 2004.

[9] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 1990.

[10] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load balancing and locality in range-queriable data structures. *PODC '04*, 2004.

[11] Yahoo! cloud serving benchmark. https://github.com/brianfrankcooper/YCSB.

[12] Min Luo and Haruo Yokota. Comparing hadoop and fat-btree based access method for small file i/o applications. *WAIM'10*, 2010.

[13] 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.

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