複数分散インデックス共通化フレームワーク上での負荷分散処理の比較

徳田　聡介†　　渡辺　陽介††　　横田　治夫†††

† 東京工業大学 工学部 情報工学科 〒 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
††† 東京工業大学 大学院情報理学研究科 計算工学専攻 〒 152-8552 東京都目黒区大岡山 2-12-1
E-mail: {tokuda, watanabe}@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

あらまし　近年データセンター等でのデータ量が爆発的に増え、大量のデータを複数の計算機で管理する分散インデックス手法が数多く提案されている。各手法を一から実装することは開発者にとって労力を要するため、これまで各分散インデックス手法は共通環境での比較は十分にされてこなかった。そこで、我々は分散インデックス手法の実装を容易にする共通フレームワークを構築している。分散環境において、特定の計算機がボトルネックとならないようにするために負荷分散が重要であり、分散インデックス手法での対応が必要となる。本研究では、各手法の負荷分散処理のうちデータ構造に依存しない部分を共通化し、開発者の実装負担を軽減することを目的とする。実際に、複数の分散インデックス手法をフレームワーク上に実装し、負荷分散処理の性能を比較する。

キーワード　分散インデックス、負荷分散処理、フレームワーク、共通化 API

Comparison of Load Balance Processes for Multiple Distributed Indices on a Common Framework

Sosuke TOKUDA†, Yousuke WATANABE††, and Haruo YOKOTA†††

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

Abstract　A number of distributed index methods to manage data among multiple machines have been proposed. Since implementing distributed index methods is a difficult task for developers, comparison of distributed index methods in a common environment has not been sufficiently investigated. Therefore, we have been developing a framework capable of making easy to implement distributed index methods. In distributed environment, load balancing is important to avoid forcing a specific machine to become a bottleneck. Our research goal is to reduce implementation cost for load balancing by extracting common portions which are independent of data structures of distributed index methods. In this paper,we add the feature of balancing load among machines to the framework and compare performance of multiple distributed index methods in load balance processes by the developed framework.

Key words　Distributed Index, Load Balance, Framework, CommonAPI

1. はじめに

近年、データセンター等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。大規模なデータセンターになると、数千という数のサーバでデータを管理している。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てボトルネックとなってしまうため、インデックス自身を分散させて複数計算機でデータを管理する分散インデックス手法が提案されている [1] [2] [3]。データを複数の計算機で管理する場合に、特定の計算機がボトルネックとならないようにするために負荷分散が重要であり、分散インデックス手法で対応しなければならない。

また、各分散インデックス手法を公正に比較するためには共

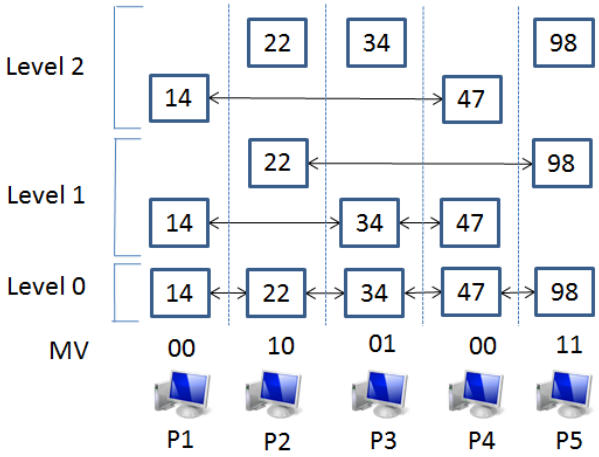

図 1　SkipGraph

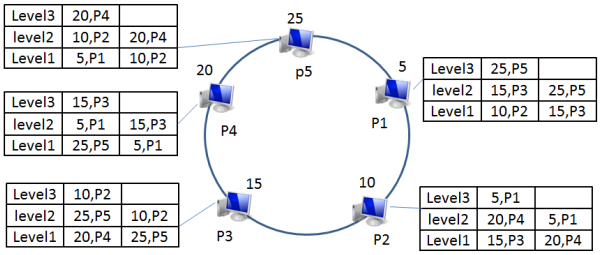

図 2　P-Ring

通の環境が必要となる。だが、各分散インデックス手法を一から実装することは開発者の労力を要するため、これまで各分散インデックス手法は共通の環境での比較は十分にされてこなかった。

そこで我々は分散インデックス手法の実装と比較を容易にする共通フレームワークを構築している [4]。共通フレームワークを用いることにより、完全一致問合せ、範囲問合せなどの分散インデックスに必要な各処理を、分散インデックス固有の箇所を実装するだけで実現可能となる。各手法のデータ構造に依存しない部分はフレームワーク実行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。これまで共通フレームワークでは完全一致問合せ、範囲問合せの共通化は行われてきたが、負荷分散処理についての共通化はなされてこなかった。

本研究では、分散インデックス手法の負荷分散処理のうち各分散インデックスのデータ構造に依存しない部分をフレームワーク側で共通化する。まず、3 つの分散インデックス手法の分析を行い、負荷分散処理における共通部分、非共通部分を抽出する。そして、共通部分については我々の共通フレームワーク上の機能として提供する。負荷分散処理の一部を共通化できるように拡張された共通フレームワークを用いることにより、各手法のデータ構造に依存する処理を公正に比較実験できるようになる。実際に SkipGraph と P-Ring を実装し、比較を行った。

まず、2. 節で対象とする分散インデックス手法の SkipGraph、P-Ring、FatBtree を紹介する。3. 節で共通フレームワークについて述べる。4. 節で実験結果について述べる。5. 節で関連研究を述べる。6. 節でまとめと今後の課題を述べる。

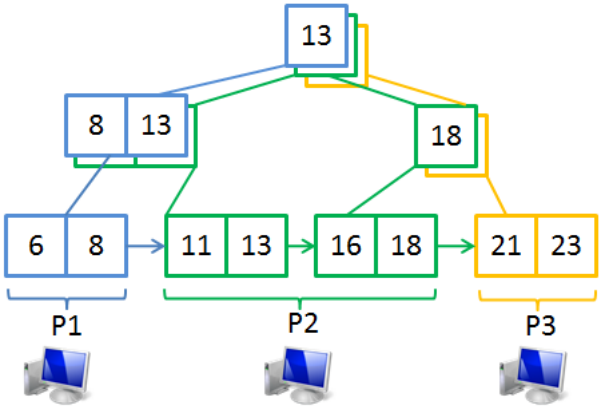

図 3　FatBtree

2.　対象とする分散インデックス手法と負荷分散

本研究では以下の分散インデックス手法を共通化の対象とする。

2.1　SkipGraph

SkipGraph [1] は Skip lists をベースにし、リストの各要素を計算機としている (図 1)。SkipGraph では各計算機に Membership Vector(MV) 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 ごとの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。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 [5]。

負荷分散処理　LEVEL0 の層の双方向ポインタを用いることで、負荷の大きい計算機から隣接する計算機へデータを転送しうる。データ移動前に、移動を行う計算機への双方向ポインタをもつ計算機へロック取得命令を送らなくてはいけない。SkipGraph 内の各計算機は他の計算機の担当範囲を保持しないので、データ移動によって担当範囲が変わっても更新情報を伝搬する必要はない（表 1)。

2.2　P-Ring

P-Ring [2] 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。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 2)。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 Hierarchical Ring（HR）と呼ばれる表を別に持つ。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 2 はオーダー 2 の例で P1 の表の最も上の行には計算機を時計回りにみて 2^{3-1} 先の P5 を指

表 1　各手法の負荷分散の非共通処理

	SkipGraph	P-Ring	FatBtree
ロック対象となる箇所	リスト上で移動対象となる計算機を指すポインタ	HR の表中の移動対象となる計算機へのポインタ	Btree 上の更新されうるノードへのポインタ
移動データ	キー情報	キー情報	移動する葉ノードとその親ノードも含めた情報
伝搬すべき情報	担当範囲の変更	担当範囲の変更	移動されたノードへのポインタおよび部分木の更新
インデックス更新	担当キーの更新	担当キーの更新	木の更新
更新情報転送先	なし	移動対象の計算機へポインタをもつ計算機	更新されたノードを共有する計算機

表 2　各手法の負荷分散の共通処理

	フレームワーク
負荷の大きい計算機の特定	各計算機の負荷を収集し、その大きさの比較によって特定する
データ移動先の決定	負荷の大きい計算機に隣接し、より負荷が小さい計算機
移動データの決定	転送するデータを選択する
ロック情報の保持	ロックに関する情報を保持する
索引のアンロック	ロックしてた要素をアンロックする

している。

負荷分散処理　P-Ring は時計回り方向にしかポインタを持っていないため、負荷分散のためにデータを転送する場合、その方向は一方向に制限される。データ移動前に、移動を行う計算機へのポインタを持つ計算機へロック取得命令を送らなくてはいけない。移動後はデータを受け取った計算機の担当範囲が変化するため、自身へのポインタを持つ計算機へ更新情報を伝搬する必要がある（表 1）。

2.3　FatBtree

FatBtree [3] は並列 B-tree の一種で、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 3)。Fat-Btree ではキー空間を各計算機に均等に配分し、自身の担当に決まった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。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。根ノードからの探索の際に次ノードが同じ計算機上になければ、そのノードを持つ計算機にクエリを転送する必要がある。

負荷分散処理　FatBtree では B-tree の各ノードを計算機間で転送することで負荷の分散を行う。ノードを移動する前には移動されるノードへのポインタを持つ計算機へロック取得命令を送らなくてはいけない。また、移動されるノードへのポインタをもつ計算機は、ノード移動後にポインタの情報を更新しなければならない（表 1）。

2.4　負荷分散処理の共通部分の分析

各分散インデックス手法の分析を行い、負荷分散処理における非共通部分を表 1 に示し、共通部分を表 2 に示した。各手法に固有のデータ構造を扱う部分は非共通であるが、負荷分散処理の流れは手法共通である。負荷分散を行うためには、負荷の大きい計算機を特定し、負荷の大きい計算機から負荷の小さい計算機へデータを移動することになる。負荷の大きい計算機の特定はデータ構造に依存しないので共通化可能である。また、データ移動先の決定と索引のアンロックも各手法で共通化できる。

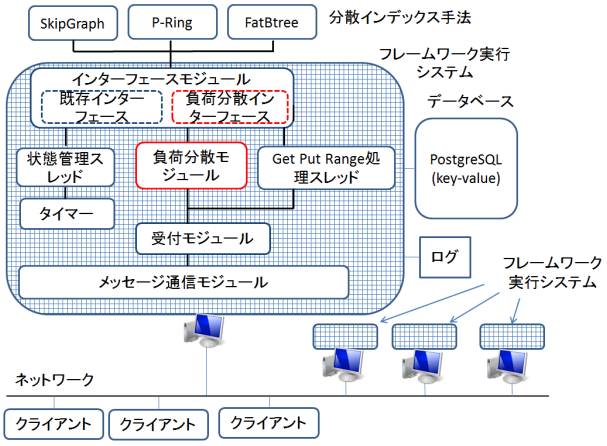

図 4　フレームワーク実行システム

フレームワーク側で共通化した具体的な処理について、3. 節で述べる。

3.　共通フレームワーク

我々の共通フレームワークにおいては、分散インデックス手法の開発者は、各手法固有のデータ構造に基づく処理のみを既定のメソッドとして実装するだけでよく、共通部分についてはフレームワーク実行システム側で事前に提供している [4]。今回はさらに負荷分散の共通化のための拡張が行われている。以降、フレームワーク実行システムのことを単に実行システムと呼ぶ。

3.1　実行システムの構成

分散環境内の各計算機で動作する、実行システムの構成を図

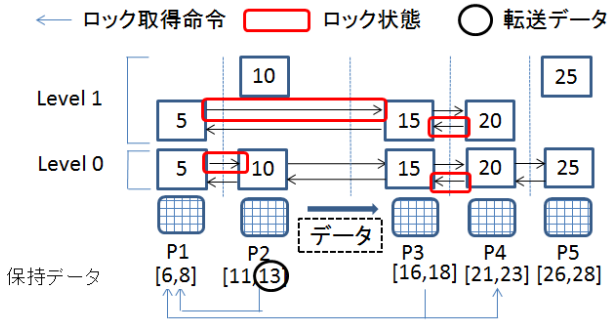

図 5　SkipGraph データ移動前 (P2 の 13 を P3 へ)

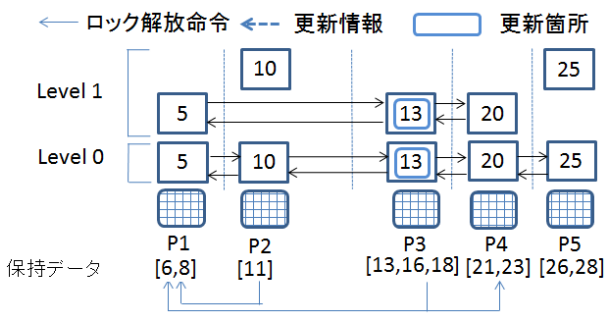

図 6　SkipGraph データ移動後 (P2 の 13 を P3 へ)

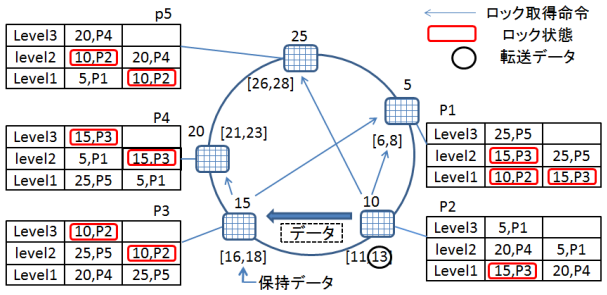

図 7　P-Ring データ移動前 (P2 の 13 を P3 へ)

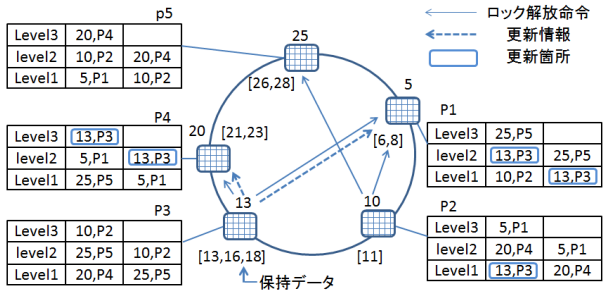

図 8　P-Ring データ移動後 (P2 の 13 を P3 へ)

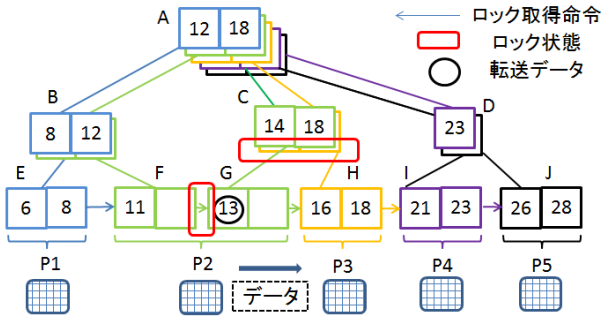

図 9　FatBtree データ移動前 (P2 の 13 を P3 へ)

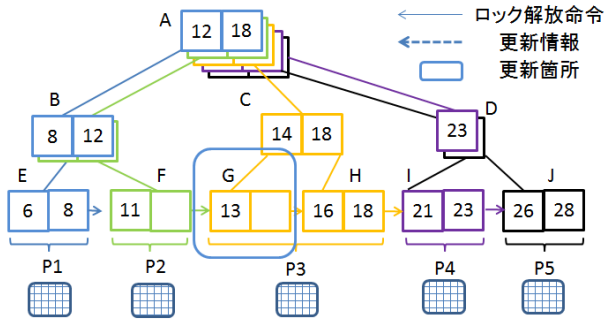

図 10　FatBtree データ移動後 (P2 の 13 を P3 へ)

4 に示す。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。受付モジュールはメッセージを受け取り、完全一致問合せ、範囲問合せ等の処理スレッドを作る。ログのレコードは、それぞれのモジュールから出力される。新たに追加した負荷分散モジュールは負荷分散における共通処理を行う。分散インデックスの固有のメソッドは共通のインターフェースモジュールを通じて呼び出される。

3.2　共通化された負荷分散処理

実行システム側で、以下の処理を共通化可能である。

- 負荷の大きい計算機の特定
- データ転送の対象となる計算機の決定
- 転送するデータの決定
- 各手法ごとのロックすべき要素の書き込みロック取得
- メッセージ通信
- ロック情報の保持
- ロック解放

実行システムは、各手法のデータ構造に依存した処理を行うために必要なメソッドを提供しており (表 3)、これらを呼び出すことで手法固有の処理を行う。分散インデックス手法の開発者はこれらのみ実装する必要がある。

負荷分散を行うにあたって、まず負荷の大きい計算機を特定する必要がある。この処理は実行システム側で完全に共通化しており、各計算機の負荷情報の収集機能を持たせ、計算機間で負荷情報を共有させることで負荷の大きい計算機を特定する。

次に、計算機間の負荷を平均化するために、最も負荷の大きい計算機から負荷の小さい計算機へとデータ移動を行う。各計算機は連続したキー空間を分割して管理しているため、データ移動の対象となる計算機は隣接している必要がある。対象となる計算機はフレームワーク実行システム側で決定する。

以下、負荷分散処理の中心となるデータ転送とインデックス更新について述べる。

（1） **データ移動**　まず、移動の対象となった計算機以上の実行システムが各手法の selectElementToLock メソッドを呼び出し、ロックする要素を取得し書き込みロックをかける。次に、

表 3　分散インデックス手法実装に必要な主な操作

名前	役割
selectElementToLock	自身からロックが必要な要素の選択
makeMoveStrategy	転送する転送データを作成
makeLockStrategy	他の計算機へ送るロック取得命令を作成
updateIndex	自分のインデックスの更新
onDataMessage	受け取った転送データからロック取得命令を作成
onLockMessage	受け取ったロック情報からロック取得命令を処理
onUpdateInfoMessage	受け取った更新情報を処理

表 4　負荷分散に関するプログラムコード行数の比較

	フレームワーク	分散インデックス手法	フレームワーク提供部分の割合
SkipGraph	797	230	78 %
P-Ring	797	351	69 %

表 5　実 験 環 境

CPU	Intel Xeon E5620 2CPU
Memory	24GB
HDD	1TB
OS	Ubuntu11.10
Java	1.6.0
PostgreSQL	9.1.7
Network	1Gb/s

実行システムは makeLock Strategy メソッドを呼び出して分散インデックスから他の計算機上のロックすべき対象を取得し、送り先に指定された計算機へとロック取得命令を転送する。ロック取得命令を受け取った側の実行システムは、分散インデックスの onLockMessage を呼び出し、ロック取得命令を渡す。そして、移動対象の計算機上の実行システムが分散インデックスの makeMoveStrategy メソッドを呼び出し、転送データを受け取った後、データ送り手と受け手の間でメッセージ通信する。転送データを受け取った実行システムは、分散インデックスの onDataMessage を呼び出し、転送データを渡す。

図 5 で, SkipGraph 上で P2 から P3 へキー 13 を移動する前の様子を示した。。SkipGraph は P2、P3 で makeLockStrategy メソッドが呼び出された時に、双方向ポインタを持つ周囲の計算機についてのロック情報を作成する。これを P2 と P1、P3 の実行システム間でメッセージ通信し、P1、P4 の SkipGraph は P2、P3 をさすポインタをロックする。その後、データ移動が開始される。

図 7 で、P-Ring における P2 から P3 へキー 13 を移動する様子を示した。P-Ring の場合、P2、P3 は、自身へのポインタを持つ P1、P5 と P1、P4 へロック情報を送る。そして、SkipGraph と同様に、P1、P5、P4 は P2 または P3 へのポインタをロックする。

図 9 で、FatBtree における P2 から P3 へキー 13 を移動する様子を示した。図 9 の場合、移動の影響をうけるノードを持つ計算機は P2 と P3 である。P2 は葉ノード F から葉ノード G へのポインタおよび、中間ノード C から葉ノード G、中間ノード C から葉ノード H へのポインタをロックする。P3 は中間ノード C から葉ノード G、中間ノード C から葉ノード H へのポインタをロックする。

（2） **インデックス更新**　データを移動すると計算機の担当範囲が変わるので、移動されたデータの情報を持つ計算機へ、更新情報を転送する。実行システムは分散インデックスの updateIndex メソッドを呼び出し、インデックスの更新を行ったのち更新情報を取得する。実行システムは受け取った更新情報の送り先に指定された計算機へと更新情報を転送する。更新情報を受け取った側の実行システムは、分散インデックスの onUpdateMessage を呼び出し、更新情報を渡す。取得済みのロック情報は各計算機上の実行システムが保持しているため、実行システム側で、自身のロックを解放し、先ほどロック取得命令を転送した計算機へとロック解放命令を送る。

図 6 で、SkipGraph 上で P2 から P3 へキー 13 を移動した後の様子を示した。SkipGraph の場合は他の計算機の担当範囲の情報を保持していないため、更新情報を転送する必要はない。

図 8 で、P-Ring における P2 から P3 へキー 13 を移動した後の様子を示している。方式は、SkipGraph と同様であるが、P3 の担当範囲が変わり、インデックスを更新した際に P3 へのポインタをもつ P1、P4 へ更新情報を送らなければいけない。

図 10 で、FatBtree における P2 から P3 へキー 13 を移動した後の様子を示している。P2 は根ノード A-中間ノード C-葉ノード G の経路を持つ必要がなくなったため、P2 が持つこの経路を削除する。P3 上では中間ノード C、葉ノード G はそのまま保持される。そして、葉ノード F から葉ノード G、中間ノード C から葉ノード G、葉ノード G から葉ノード H へのポインタをそれぞれで更新する。

4. 実　　験

共通フレームワーク上に SkipGraph,P-Ring を実装し負荷分散処理を比較する。

4.1 実 験 環 境

実験環境を表 5 に示した。実験で使用した計算機台数は 4,8,12 である。100,000 件のデータを計算機ごとに分割してデータ挿入する。その際に中央の 2 台の計算機のデータ数を 37,500 件に固定し、残りの 25,000 件を他の計算機で均等に分割した (図 11)。データはキー・バリューを用いており、キーは連続する 0 から 99,999 の数字を用いた。負荷集計は 1.3 秒ごとに行い、負荷集計を 5 回行ったのち負荷分散処理を行うようにした。今回の実験では負荷の尺度は計算機のもつデータ容量とした。負荷分散が起こる負荷の閾値は各計算機の負荷の平均値の 1.4 倍とし、1 回の負荷分散処理で移動するデータ量は負荷が平均に達するまでの量とする。

共通フレームワークを用いて実際に SkipGraph、P-Ring を実装した際の、負荷分散処理に関するプログラムのコード行数を

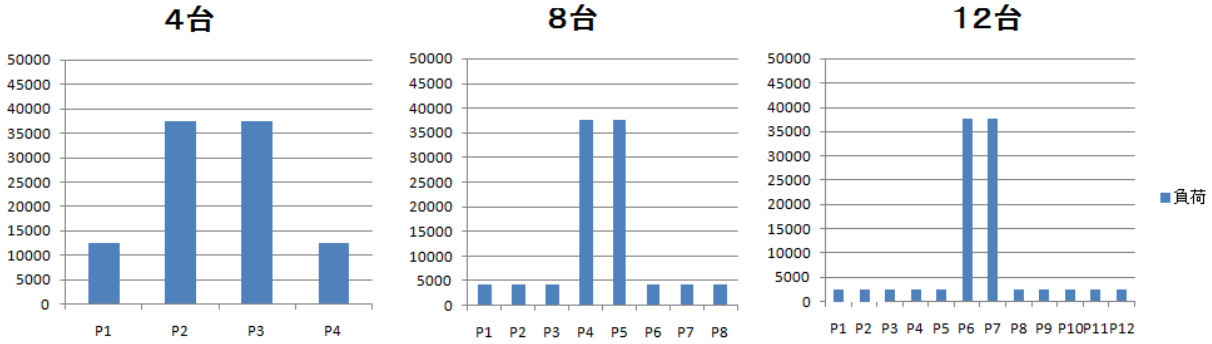

図 11　初期状態におけるデータ容量の偏り

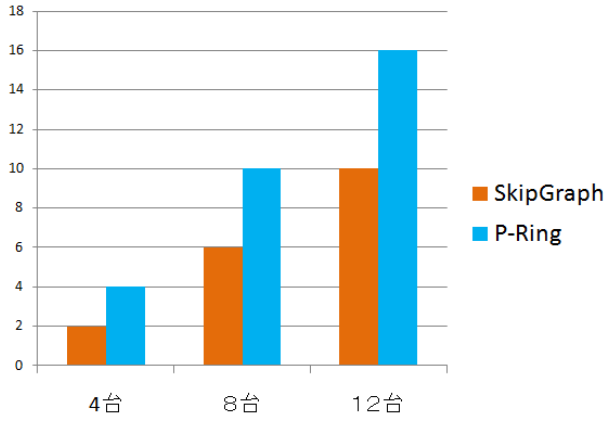

図 12　負荷分散回数の比較

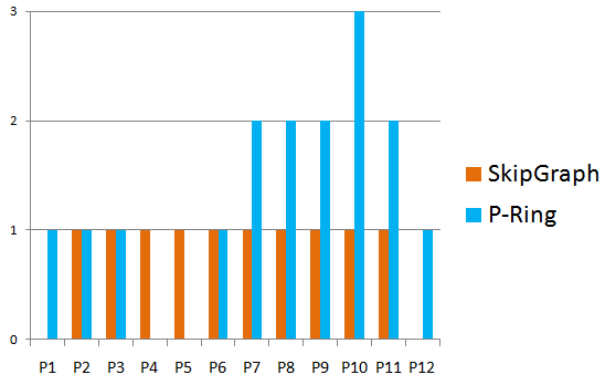

図 13　12 台の時の計算機ごとのデータ移動回数

比較した結果を表 4 に示す。コード行数の測定には eclipse のプラグイン [6] を用いた。フレームワークが占める割合は各手法で 69 %以上であり、今回の実装においてフレームワークを用いることで負荷分散処理の実装コスト削減の効果があることを確認できる。また、手法間で多くのコードを共通化することでデータ構造に起因する性能差を調べやすくなると考えている。

実験に用いた分散インデックスのパラメータを説明する。SkipGraph は、リストの層のレベルの最大の高さを 5 とし、membership vector にはランダムは 8 ビット列を用いた。P-Ring は、Hierarchical Ring のオーダーを 2 とした。ローカルのデータ管理のためにファンアウトが 64 で、データノードに格納できるデータの上限が 8 の B+-tree を用いた。

4.2　実 験 方 法

最初にすべての計算機に並列してデータ挿入のクエリを発行する。各計算機のデータ挿入完了後、各計算機で同時に負荷分散を開始する。そして、負荷分散回数および負荷分散時間を調べる。負荷分散時間は負荷分散の開始から終了までに各計算機で負荷分散が実行された時間を合計し、それらの和をとったものとする。

4.3　実 験 結 果

負荷分散が終了するまでに実行された負荷分散処理の回数を図 12 に示す。図 12 の結果から、P-Ring の方が SkipGraph より負荷分散回数が多く、4 台の時は 2 倍、8 台の時は約 1.67 倍、12 台の時は 1.6 倍の回数差となった。この原因として、SkipGraph は両隣へデータ移動できるが P-Ring は一方向にしかデータ移動できないために、負荷を計算機全体へ分散するためのデータ移動の回数が増えると考えられる。また、負荷分散にかかった時間を図 14 に示す。図 14 の結果から、P-Ring の方が SkipGraph より負荷分散にかかった時間が長く、4 台、8 台の時は約 2 倍、12 台の時は約 1.75 程度の時間差となった。負荷分散時間のうち、データベースとのやりとりに要した時間が大きな割合を占めている。つまり負荷分散のために転送したデータの合計量が負荷分散時間に大きな影響を与えている。

データ移動の様子を調べるために 12 台の時の計算機ごとの負荷分散回数を調べた (図 13)。図 13 の結果から、SkipGraph は計算機ごとのデータ転送回数がほぼ均等であることが分かる。それに対して、P-Ring は P7 から P11 の計算機のデータ転送回数がやや多くなっており、不均等である。P-Ring では SkipGraph と違って 2 回以上負荷分散を行う計算機が複数あるために負荷分散回数に差が生じたことがわかる。

データ転送量の変化を調べるために 12 台において計算機ごとに負荷分散が発生したときにどれだけのデータ量を転送しているかを調べた (図 15、図 16) 。図の横軸は何回目に発生した負荷分散であるかを示し、縦軸は 12 台の計算機のうち何番目の計算機であるかを示している。円の大きさは負荷分散の起きたときに転送されたデータ数を視覚的に表しており、円の中の数字は転送されたデータ数を表している。図 15、図 16 の結果から、SkipGraph の場合は負荷分散が実行されるたびにデータ転送量が減少している。それに比べて、P-Ring の場合は、負荷分

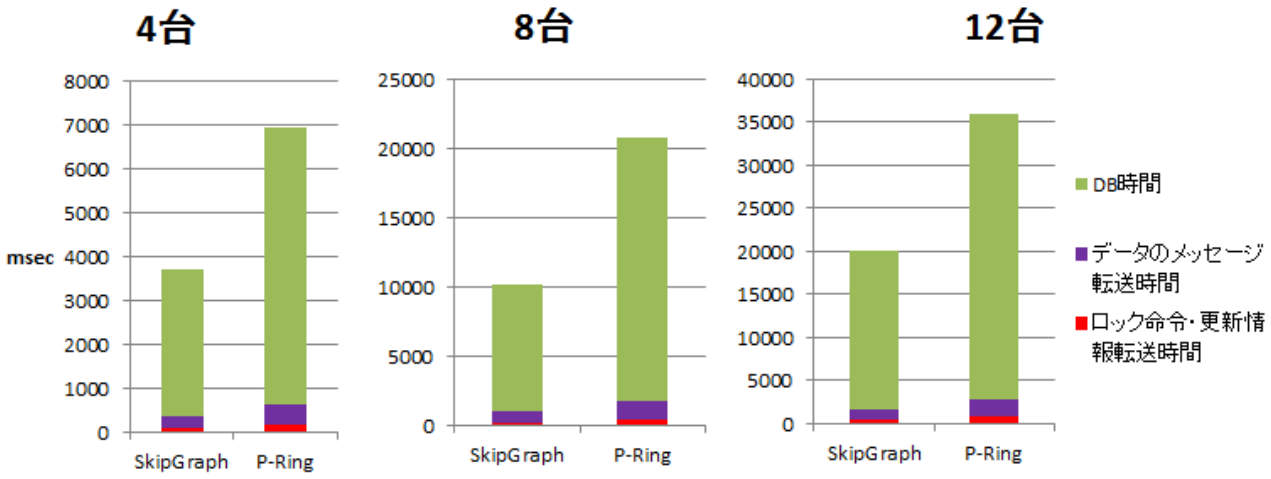

図 14　負荷分散時間の比較

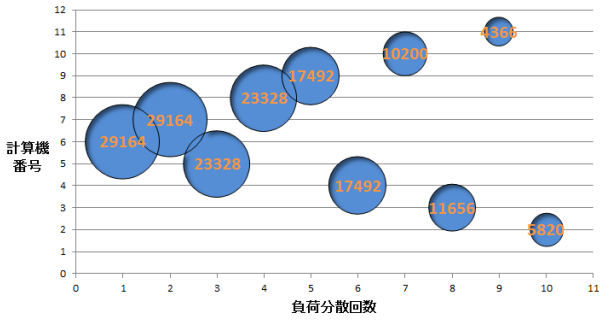

図 15　12 台の時の SkipGraph の計算機ごとのデータ転送量

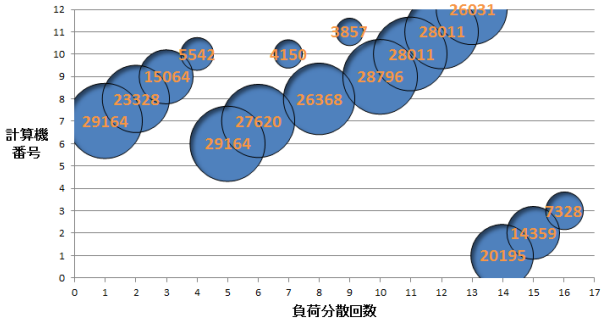

図 16　12 台の時の P-Ring の計算機ごとのデータ転送量

散の 7 回目と 9 回目を除く 5 回目から 13 回目の期間では負荷分散が行われてもデータ転送量があまり変わっていないことがわかる。そのため、P-Ring は負荷分散回数が多く、1 回のデータ転送量も減少しづらいためにデータ転送量の合計が大きくなり SkipGraph より負荷分散時間が長くなったと考えられる。次に P-Ring でデータ転送量が減少しづらい期間が発生した理由を述べる。ただし、計算機番号が x の計算機を Px で表すとする。P-Ring ではまず、初期状態で負荷の大きい P6、P7 のうち、P7 のデータが P8、P9、P10 へと分散される。そのあとに P6 の持つデータが P7、P8、P9、P10 へと順番に移動されるが、すでに負荷が平均もしくは平均近くに達しているために、移動されたデータの量はあまり変わることなく次の計算機へと渡されることになる。そのため負荷が平均より小さい計算機へそのデータが渡るまで負荷分散が繰り返され、1 回のデータ転送量もあまり小さくならない。

4.4　実験のまとめ

共通フレームワーク上に SkipGraph、P-Ring を実装し負荷分散の比較のための実験を行った。コード行数を比較することで今回の実装では、分散インデックス手法の実装コストが削減できることを確認した。実験により、SkipGraph の方が P-Ring より負荷分散回数、負荷分散時間ともに小さく性能が優れていることを示した。その原因として SkipGraph は二方向、P-Ring は一方向にデータ移動できるという違いによるものと考えられる。P-Ring は移動方向が一方向しかないので負荷が平均に達している計算機にデータ移動することがあり、移動先で再び負荷分散が必要となることが多い。その場合は負荷分散回数が増えるうえに転送データ量があまり減少しないので、P-Ring の負荷分散回数、負荷分散時間ともに大きくなった。

5.　関 連 研 究

負荷の分散方法として、トークンベースで複数計算機の負荷情報を集め、負荷の大きい計算機から負荷の小さい計算機へデータを移動する方法 [3] [7] がある。隣接する計算機間で負荷を比較する方法 [8] もある。隣接する計算機で負荷を比較すると、収束保証がなく、信頼性にかけることが言われている。本研究では、トークンベースの方法を採用し、フレームワーク側でこの処理を提供している。

分散インデックス手法のための他のフレームワークもいくつか提案されている。Overlay Weaver [9] 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の構築を目的としている。P-Ring [2] 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 環境のための計算機の管理に重点を置いている。SOMO [10] 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。これらに対して、本研究の共通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。特に本研究では負荷分散の一部処理の共通化を行い、フレームワークとして提供した。

6. ま と め

本論文では、我々が提案している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上に負荷分散処理に関する機能を追加した。共通化のために分析の対象とした手法は、SkipGraph、P-Ring、FatBtree の 3 つである。各手法の負荷分散方法を分析し、手法ごとのデータ構造に依存せず、共通化可能な処理をフレームワーク側で提供することで、分散インデックス手法の実装コストが削減できることを示した。実際に SkipGraph、P-Ring をフレームワーク上に実装して、負荷分散処理の比較実験を行った。SkipGraph は P-Ring より負荷分散回数、負荷分散時間がともに小さく性能が優れていることを示した。このような比較が可能になったのは共通フレームワーク上に両手法を実装し、それぞれの挙動を細かく追えるようにした成果である。

今後の課題として、パラメータ（実験機台数、実験データなど）を変更しての実験や完全一致問合せ、範囲問合せと並列に負荷分散を実行したときの性能比較、また FatBtree などの今回実装した以外の分散インデックス手法の実装追加などが挙げられる。

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] James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, Vol. 3, No. 4, p. 37, November 2007.

[2] A.Machanavajjhala J.Gehrke J.Shanmugasundaram A.Crainiceanu, P.Linga. P-ring: An efficient and robust p2p range index structure. In *SIGMOD*, 2007.

[3] H.Yokota, Y.Kanemasa, and J.Miyazaki. Fat-btree: An update-conscious parallel directory structure. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, pp. 448–457. IEEE Computer Society, 1999.

[4] 近藤直樹, 羅敏, 渡辺陽介, 横田治夫. 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. *DEIM Forrum 2012*, 2012.

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

[6] Eclipse Metrics plugin. `http://metrics.sourceforge.net/`.

[7] Hisham Feelifl and Masaru Kitsuregawa. The simulation evaluation of heat balancing strategies for b-tree index over parallel shared nothing machines, 1999.

[8] Mong Li Lee, Masaru Kitsuregawa, Beng Chin Ooi, Kian lee Tan, and Anirban Mondal. Towards self-tuning data placement in parallel database systems, 2000.

[9] 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.

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