B+木の付加情報を用いた近似問合せ手法の検討

湯浅　拓樹† 　合田　和生†† 　喜連川　優††

† 東京大学 大学院情報理工学系研究科　〒 113–8656 東京都文京区本郷 7–3–1
†† 東京大学 生産技術研究所　〒 153–8505 東京都目黒区駒場 4–6–1
E-mail: †{yuasa,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　近年の情報社会においてやり取り，蓄積されるデータの量は年を経るごとに膨大なものとなっている．大量のデータを分析する際には相応の時間やコストがかかり，そうした費用を削減する技術として近似問合せ処理に注目が集まっている．本論文では現行の多くのデータベースシステムですでに運用されている B+木の各ノードにメタ情報を付随することによって B+木を用いた近似問合せ手法を検証し，実験する．

キーワード　B+木，近似問合せ

1　は じ め に

情報が大きな重要性を持つ現代社会において，情報をデータとして貯蔵や運用を担うデータベースシステム上には膨大な量のデータが蓄積されており今も増加傾向にある．活用されるべく蓄えられるデータの量が爆発的に増大していく中，データを分析するための処理能力に関しては，向上こそすれどデータ量の増大の速度には大きく遅れを取っている．処理速度があまり大きく変わらない中で処理の対象となるデータの量のみが大きくなれば，データの解析にそれだけの時間を要してしまう．しかし，迅速なトランザクションの遂行が必要とされる OLAP [6] の文脈では処理時間の膨張というのは非常に大きな問題となる．そうしたデータベース上の大量のデータを処理する際に用いられる手法の 1 つに近似クエリ処理 (Approximate Query Processing) がある．近似クエリ処理はデータベースに蓄えられたデータの分布やランダムサンプルなどといったメタ情報をもとに，与えられた解析タスクについて推定された結果を迅速に返す．この方式を用いるとクエリの実行時間を大幅に短縮できるとともに，特にビッグデータの解析において相性が良いことから研究対象としても注目されている [11] 反面，運用されている商用データベースシステムでは活用されていないことが多く，研究の余地が十二分に存在する．

対して，B+木を用いたデータの索引の生成は現行のデータベース上で広く用いられている機能である．こうして広く普及している B+木を用いた近似問合せ処理の手法はあまり研究されておらず，その手法を検証することは実装の易さや処理能力の向上といった観点から利点が多い．

本論文では，データベース上の B+木の各ノードに，その子ノードのデータの分布に関する情報をメタデータとして付随し，それらを活用することによる近似問合せ手法を提案し，実験によりその有効性を示す．即ち，拡張された B+木を用いた分析を行う際に，記憶容量や構造生成に弄する時間のオーバーヘッドを検証する．

本論文の構成は以下の通りである．第 2 章では本論文で対象とし解決する課題について解説する．第 3 章では一般的な B+木や今回用いる拡張 B+木について解説し，その後に今回対象としている種々の探索アルゴリズムについても解説する．第 4 章では本研究の提案する拡張 B+木の生成におけるオーバーヘッドの比較実験の概要とその結果を提示する．第 5 章ではデータベースシステム上で一般に行われている近似問合せ処理について解説し，第 6 章にて本論文を総括する．

2　問合せ処理の高速化

近似問合せ処理は近年ビッグデータの分析において効果を発揮する技術として広く研究されている分野である．その仕組みはデータベースに問い合わせられるクエリの結果を，対象データすべてを使って処理するのではなくある程度の推定を許すことで実行時間を一定の精度を保ちつつ加速するというものである．本論文では B+木を利用したクエリ処理を対象として近似問合せ処理について実験している．通常 B+木をデータベース上の索引として利用する場合には，分析対象となっているデータをすべて検索し，それらをすべて実行に利用することで結果を返している．しかしその方法ではデータの検索，そして大量のデータの処理に大幅な時間がかかってしまう．その問題を解決するため，本稿ではクエリを問い合わせる際に B+木上で読み込むことができるページ数に制限をかける．こうしたクエリを用いると，検索して読み込めるデータの数も同様に制限することができ，それによりクエリの実行時間も大幅に短縮することが期待される．また本稿では，B+木に対してデータの範囲を指定した検索を行うクエリを高速化の対象とする．

3　B+木の拡張と探索

3.1　B+木と拡張 B+木

現行データベース上で広く普及しているデータの索引方式の 1 つとして B+木がある．B+木は次に示すように生成される．まずデータベース上あるリレーション上のデータをその主キーにしたがって整順する．そうして整順されたデータをいくつか

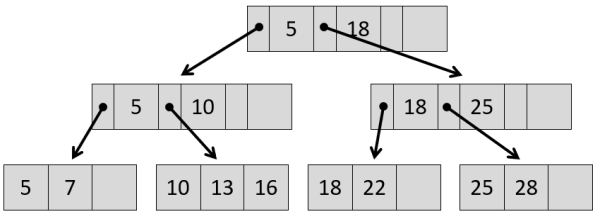

図 1　B+木

の塊ごとに分割し，分割した各ノードを B+木の最下層のノードとする．次にそれらの最下層ノードの親ノードを構成する．いくつかの連続した最下層ノードの最初のレコードの主キーの値を取得し，その値とそれが指し示すデータノードへのポインタを合わせたデータ構造を親ノードで保持する．こうして複数の親ノードを生成していき，これをすべての最下層ノードについて行う．生成された親ノードのさらに親ノードを同様の手順で生成していき，すべてのノードが 1 つのルートノードに集約されれば B+木となる．この B+木を用いることで，主キーに基づいて条件付けされた問合せ処理の際に目的とするデータにより早く到達することが可能となる．B+木の生成例を図 1 に示す．

こうした B+木の各ノードに，含まれるデータのメタ情報を付随することで近似問合せ処理に役立てる．本稿ではこうしてメタ情報を付随した B+木を「拡張 B+木」と呼称することとする．拡張 B+木上のあるノードにメタ情報を付随する場合，そのノードのすべての子ノードに含まれるデータの要約情報を付与することになる．この際付与する要約情報としては様々なものが考えられる．それぞれの子ノードに含まれる全データの個数，平均値，最大値，最小値などといった集計情報を付随した場合には，それらの情報を要求する問合せが送られて来た際に B+木を子ノードまで辿らずに結果を返すことができ，実行時間の短縮を測ることができる．また B+木の索引が張られている主キーに関する要約情報だけでなく，子ノードのデータの別の属性についての集計データを付随すれば，主キーを条件とする他の属性データに関する問合せにも対応させることができる．

そのほかにはそうした要約情報を集計した日時のデータをタイムスタンプとしてノードに付随しておけば，拡張 B+木上の付随情報が最後にいつ更新されたものかを考慮することができ，データベース上のデータの更新頻度と照らしてどの程度要約情報が信頼できるものかを測るといった利用法も考えられる．

このように拡張 B+木上の付随情報には色々なものが考えられ，その利用については十二分に研究の余地がある．本論文では最も単純な実装としてデータの個数，平均値，最大値，最小値の情報が B+木上の各ノードに付与する場合を対象として，議論を進める．その拡張 B+木の生成例を図 2 に示す．図 2 の 2 段目左のノードにはそれぞれの子ノードへのポインタの後ろにメタ情報が付随されており，例えば最下層の最も左のノードへのポインタにはそこに含まれるデータ $\{5, 7\}$ についてのデータの個数，平均値，最大値，最小値といった情報が含まれている．最上層のノードの左側のポインタの後ろにはその子ノードに含まれるデータ $\{5, 7, 10, 13, 16\}$ についてのデータの個数，平均値，最大値，最小値といった情報が含まれている．

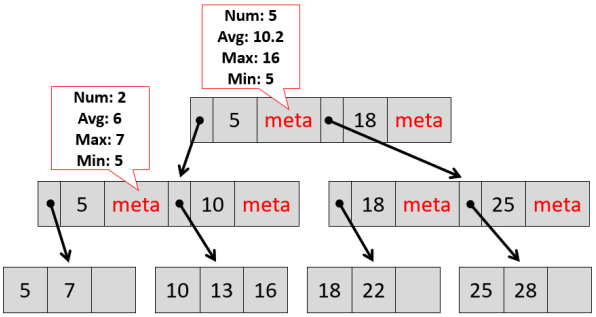

図 2　拡張 B+木の生成例

アルゴリズム 1 深さ優先探索

```
function DEPTHFIRST(b)
    FETCH(b)
    P ← P ∪ b                    ▷ 訪問済みのノードとそのデータの保存
    for each n ∈ N do            ▷ N は b の子ノードの集合
        if n ∉ P then
            DEPTHFIRST(n)
        end if
    end for
end function
```

次節からはこうして実装した拡張 B+木上で目的のデータを見つけるための探索アルゴリズムについて解説する．本論文では実行時間の短縮のためデータベースに送られる問合せが，B+木上で探索できるノードの数をある一定値以下に制限しているという状況下でのそれぞれの探索アルゴリズムの振る舞いを提示する．はじめに 2 種のベースラインとなる手法を紹介したのちに，本論文で提案する新規探索法について解説を行う．

3.2　深さ優先探索

拡張 B+木を対象とした深さ優先探索をアルゴリズム 1 に示す．b は B+木の根ノード，N は b の子ノードの集合，P は訪問済みのノードとそこに含まれるデータの集合である．深さ優先探索ではまず先に最下層のノードまで探索し，その後同じノードを親として持つノードを探索，それが終わるとそのさらに親のノードを共通の親として持つノードを探索するという順番で探索が進んでいく．

3.3　幅優先探索

拡張 B+木を対象とした幅優先探索をアルゴリズム 2 に示す．b は B+木の根ノード，N は b_0 の子ノードの集合，P は訪問済みのノードとそこに含まれるデータの集合，Q は次に訪問する予定のノードのリストである．幅優先探索では現在の層のノードをすべて順番に探索していき，すべて探索し終えたら一つ下の層のノードを同様にすべて探索していくというアルゴリズムである．

3.4　提案手法：情報密度優先探索

拡張 B+木を対象とした提案手法である情報密度優先探索を

アルゴリズム 2 幅優先探索

```
function BREADTHFIRST(b)
    FETCH(b)
    Q ← φ
    P ← P ∪ b                    ▷ 訪問済みのノードとそのデータの保存
    Q ← Q ∪ b
    while Q ≠ φ do
        b_0 ← pop(Q)             ▷ Q の先頭のノードを取り出す
        for each n ∈ N do        ▷ N は b_0 の子ノードの集合
            if n ∉ P then
                FETCH(n)
                P ← P ∪ n
                Q ← Q ∪ n
            end if
        end for
    end while
end function
```

アルゴリズム 3 情報密度優先探索

```
function DENSITYBASED(b)
    d ← 0
    Q ← φ
    FETCH(b)
    P ← P ∪ b                    ▷ 訪問済みのノードとそのデータの保存
    if N = 0 then
        finish                   ▷ N は b の子ノードの集合
    end if
    Q ← Q ∪ N
    for each n ∈ Q do
        if n ∉ P then
            FETCH(n)
            P ← P ∪ n
            d_max := n.max − n.avg
            d_min := n.avg − n.min
            d_new := (d_max − d_min) ∗ n.num
            if |d_new| > |d| then
                d ← d_new
                if d > 0 then
                    q ← n
                else
                    q ← n
                end if
            end if
        end if
    end for
    Q ← Q − q
    DENSITYBASED(q)
    for each n ∈ Q do
        DENSITYBASED(n)
    end for
end function
```

アルゴリズム 3 に示す．b は B+木の根ノード，N は b の子ノードの集合，P は訪問済みのノードとそこに含まれるデータの集合，Q 次に訪問する予定のノードのリスト，である．また，$n.max$，$n.min$，$n.avg$，$n.num$，はそれぞれ n 以下の全データの最大値，最小値，平均値，個数を表す．そして $n.higher$ と $n.lower$ はそれぞれ n の子ノードのうちデータの値が高いほうのノードと低いほうのノードを表す．情報密度優先探索はデータの最大値と平均値の差，そして平均値と最小値の差をとり，さらにそれらの差が大きくなるようなノードを優先的に探索する．このように探索を進めることで，データの分布の偏りが激しい部分を見つけ出し，より値の近いデータがたくさん偏って存在しているノードを優先的に探索することができる．そしてそれらのノードを優先的に探索することで近似に必要なデータを他のアルゴリズムよりも多く閲覧し，近似問合せの精度の向上を測ることが期待される．

4 評価実験

本章では，通常の B+木と種々の拡張 B+木のデータ容量の違いや生成に要する時間について計測し，その予測値との比較について考察を行う．

4.1 実験環境

各種の B+木の生成の比較について実験を行うにあたり，B+木を生成するデータセットは TPC-H の orders と lineitem のみを利用し，dbgen を用いてデータの作成を行った．dbgen のバージョンは 2.18.0 である．B+木を生成する各データセットの属性は orders と lineitem ともに orderkey を用いた．スケールファクタ (SF) が 0.1 ，1 ，10 それぞれの場合について各種の B+木の生成を行った．B+木上の各ノードの容量を 4096B とし，B+木生成時には各ノードがなるべく容量を使い切るようデータを格納する．

今回の実験で生成した B+木は，

（1） 付加情報のない通常のもの (baseline)

（2） 子ノードのレコード数を保持するもの (付加情報 1)

（3） 上記に加えて子ノードの orderkey の中央値，最大値，最小値を保持するもの (付加情報 2)

（4） 上記に加えて子ノードの totalprice もしくは extendedprice の中央値，最大値，最小値を保持するもの (付加情報 3)

の計 4 種ある．

実験は CPU が Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz 14core x2，メモリ容量が 96GB，ハードディスク容量が合計 29TB の実機上で行い，OS は CentOS Linux release 7.7.1908 (Core) を用いた．

4.2 拡張 B+木の容量比較

はじめに各データセットを用いて B+木を生成した際の，ノードの数を図 3 から図 5 に示す．また，通常の B+木に対する各拡張 B+木のオーバーヘッドのみを割合として算出したものを図 6 から図 8 に示す．どのデータセットの場合でも，B+木に付加情報を加えた際に B+木全体としてのノード数は大きく変わらないということが分かった．

本稿の提案する B+木では，n 行のデータが含まれているデー

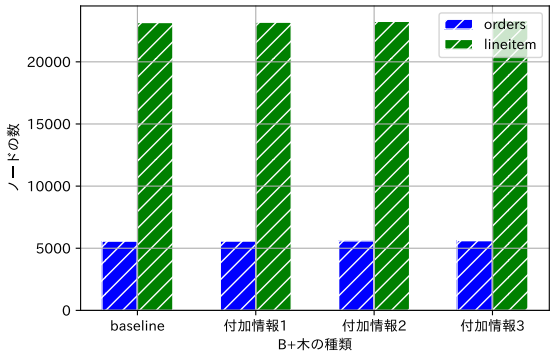

図 3　SF=0.1 の時の B+木のノードの数

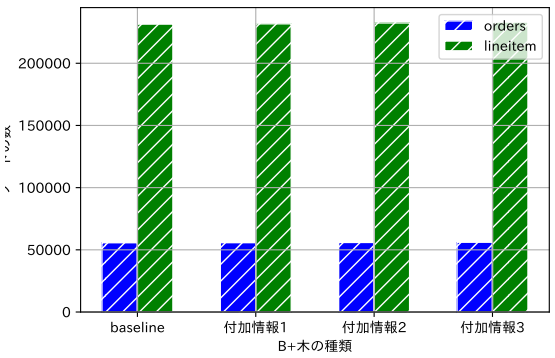

図 4　SF=1 の時の B+木のノードの数

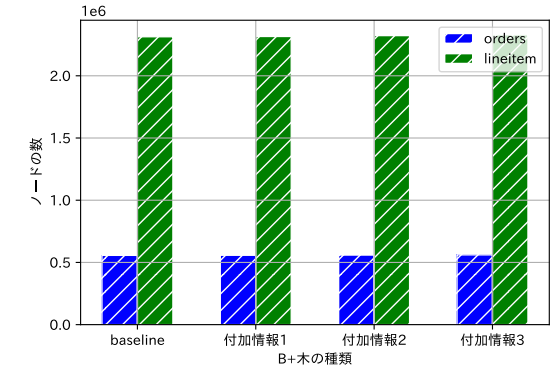

図 5　SF=10 の時の B+木のノードの数

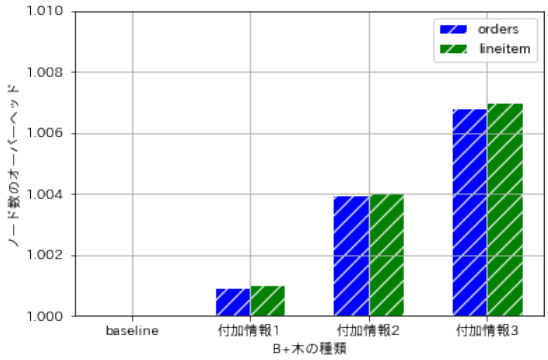

図 6　SF=0.1 の時の B+木のノード数のオーバーヘッド

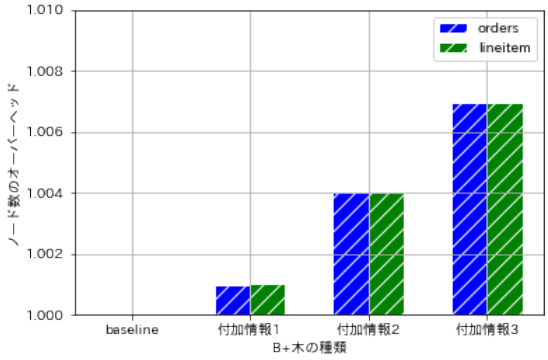

図 7　SF=1 の時の B+木のノード数のオーバーヘッド

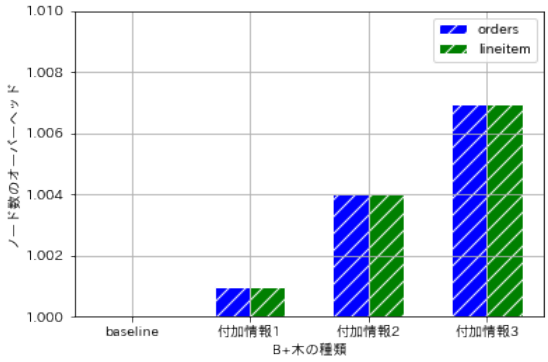

図 8　SF=10 の時の B+木のノード数のオーバーヘッド

タセットを用いて B+木を作成する際，約 $n/30$ 個の葉ノードに格納されるのに対して，その時の内部ノードの数は $n/6000$ 個以下になることが実験から分かっている．付加情報を B+木に追加する際にはその情報は内部ノードのみに格納されるため，葉ノードの数には影響しない．そのため付加情報により拡張 B+木の内部ノードの数になったとしても，全体のノード数から見ると $O(1/1000)$ の影響になると考えられ，これは実験の結果とも合致する．

5　B+木生成の実行時間比較

次に，図 9 から図 11 に B+木を生成する際に要した実時間を示す．スケールファクタが 0.1 の時の通常の B+木の生成の際に、他の B+木よりも時間がかかっているがこれは実験マシンの初回起動時のキャッシュによる影響だと推測される．B+木の付加情報の有無が索引の生成時間に及ぼす影響はあまりないということが本実験から分かった．これは，B+木のノードの大部分が葉ノードであるため，実行時間の大部分が葉ノードの生成に使われていることによるものだと推測できる．

6　関 連 研 究

近似問合せ処理に関する手法は，クエリの実行時に逐次サンプルを発行する手法と，事前に蓄積データの要約情報を算出し

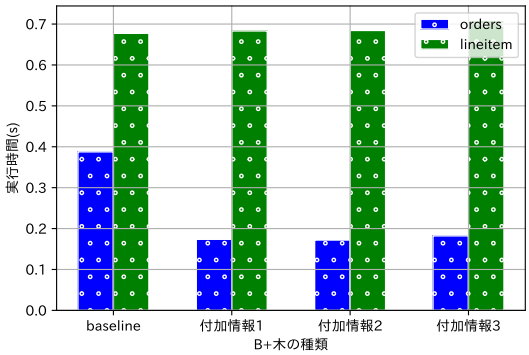

図 9　SF=0.1 の時の B+木生成にかかる実時間

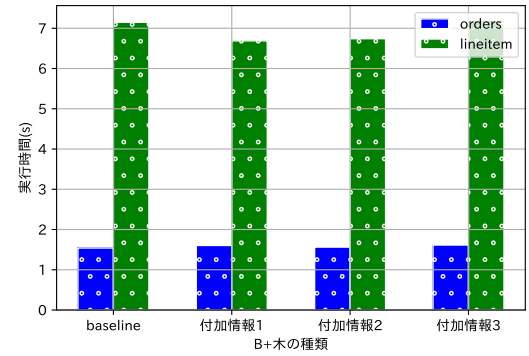

図 10　SF=1 の時の B+木生成にかかる実時間

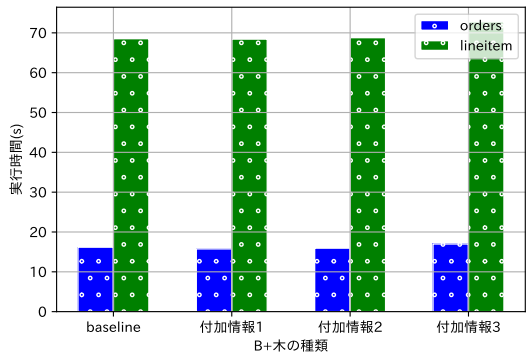

図 11　SF=10 の時の B+木生成にかかる実時間

ておき結果推定する手法の 2 つに大別される．前者の手法ではデータベース上の余分なストレージを必要としない反面，クエリの実行時に必ずサンプル生成のオーバーヘッドがかかるという欠点がある [13], [16]．それに対して後者の手法では，データベースのストレージを圧迫するかわりに，1 つの要約情報を将来の複数の問合せに対して利用することができるという利点がある．本章ではそれぞれの手法の主要な取り組みについて紹介する．

6.1　信頼区間の利用

クエリの実行時にサンプルを生成するという手法では信頼区間を利用し統計的に精度を高めるといった方式がよく取られる [9]．この方式ではサンプルデータをもとに得られた演算結果がどの程度の確率でどの程度正しい結果となっているかを知ることができる．事前に対象データの分布が知られている場合にはガウス分布や t-分布などを仮定することで信頼区間を数学的に算出することが可能となる．反対にデータの分布が知られていない時には，サンプリングを繰り返したり，サンプリングしたものからサンプリングするなどといった手法を用いることでエラー率を算出するような研究が行われている [17]．

6.2　事前算出サンプリング

問合せ処理時にサンプリング手法とは対照的に，本手法では事前に生成しておいたサンプルを用いて近似処理を行う．事前にサンプリングをしておく利点として，逐次的なサンプリングよりも時間の余裕をもって行うことが可能なため，サンプルがより実際のデータ分布を適切に表すようサンプル品質を向上させることができる．また，事前情報として DBMS 上にどういったクエリが問い合わせられるのかというワークロードを得ていれば，それをもとに各クエリに対応するようなサンプルを複数保持しておくことができ，精度の向上に貢献できる．ただしクエリごとにサンプルを保持することになるため非常に大きい容量が必要となってしまう．この問題を解決するため同じデータを利用するようなクエリをグループ分けすることで保持するサンプルを少なくする Blink DB [2] のような研究も存在する．

6.3　ヒストグラム

データセットを複数の区間に分け，その区間ごとにどのくらいのデータが存在するかを記録することでデータの分布や特徴を表した情報をヒストグラムと呼ぶ．ヒストグラムは比較的安易に作成できるという性質もあり，すでに様々の商用データベースにて生成できるように実装されている [4], [14]．作成の際にはデータを分割する区間を一定とする equi-width や 1 つの区間に含まれるデータ量を一定とする equi-depth などいくつか区間の決め方があり [5]，またその区間の決め方主眼に置いたような研究もされている [1]．ただしヒストグラムはあくまで数値として解析できるデータのみについて利用することができ，そのほかの複雑なクエリの処理には適していない．

6.4　Wavelets

ヒストグラムは元のデータのサブセットとなるデータ構造を提供するのに対し，wavelet はデータを集約し圧縮することで元のデータよりも少ない容量で元のデータの頻出情報を表現しようという手法である．Wavelet にはいくつか種類があるがそのうち最も簡単なものとしては haar-wavelet 変換が挙げられる．これについての詳しい説明は本稿では割愛する．この手法ではデータの圧縮に伴いそれが表現する情報量は少なってしまうため，どの情報が不要なものか見極めることが肝要であり，またその際の誤差を L_2 エラーなどを用いて表現する．Wavelet についても様々な研究が行われている [12], [15]．

6.5　Sketches

スケッチでは純粋なデータとして格納されているデータのう

ち，数値として登録されているものを行列として表すことで扱いやすくしたものである．例えば性別のようにある属性が 2 種類の値のみを取る場合には，その属性のデータを 0 と 1 のビット列として保持することでデータを保存しておくスペースを削減できる．行列として保持することでデータを圧縮できるだけでなく，更新が容易に実現でき，さらに並列化もしやすいという利点が存在する．スケッチには大きく分けて 2 つの種類が存在し，頻度ベースのスケッチ [3] と種類別数値のスケッチ [7] が存在する．さらにスケッチのもう一つ大きな特徴として他のシノプシスでは推定が難しいとされている count や distinct というクエリの種類についても有効であることが確認されている [5]．数値データを行列やベクトルとして保持するという性質上，bag of words などといった頻度を扱う NLP の分野のデータ解析に利用でき，また新規データの更新のしやすさという点から，金融データなどという分野でも利用が期待される．

6.6 Materialized views

DBMS にすでに実装されている機能の中で近似クエリ処理に関係のあるものとして materialized view [8], [10] というものが存在する．これはクエリの実行結果として得られたデータの束を新たに DBMS 上に保存しておくという手法で，同じ内容のクエリが何度も問い合わせされるようなシステムで効果を発揮する．こうすることで 1 度目のクエリ問い合わせにて結果を保存し，同じクエリが求められた時に同じデータを返すという処理が可能になる．本手法では 1 度目のクエリ処理では近似処理をしていないため正確なデータを常に返すことができることが大きな利点として挙げられる．ただしこうして生成されたクエリの結果を複数所持しておくのは大きくリソースを消費するため，どのクエリの結果を保持しておくかは DBMS を運用しながら慎重に決めていく必要がある．

7 おわりに

本論文では，データベース上で広く使われている索引方式である B+木上にデータ分布に関するメタ情報を付随することで，B+木を用いた近似問合せ処理とその探索アルゴリズムを提案した．

そして，通常の B+木と各種メタ情報を付与した拡張 B+木の生成に関する比較実験を行い，付加情報のオーバーヘッドが非常に小さいことを示した．

今後は提案探索アルゴリズムを用いた B+木上の近似問合せの有用性，データの挿入や削除を行った際のより効率的なメタ情報の更新法について追求していく予定である．

文　献

[1] Jayadev Acharya, Ilias Diakonikolas, Chinmay Hegde, Jerry Li, and Ludwig Schmidt. Fast and Near Optimal Algorithms for Approximating Distributions by Histograms. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Vol. 31, pp. 249–263, 2015.

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013*, pp. 29–42, 2013.

[3] Vladimir Braverman and Rafail Ostrovsky. LNCS 8096 - Generalizing the Layering Method of Indyk and Woodruff: Recursive Sketches for Frequency-Based Vectors on Streams. Technical report, 2013.

[4] Graham Cormode, Minos Garofalakis, Antonios Deligiannakis, and Andrew McGregor. Probabilistic histograms for probabilistic data. *Proceedings of the VLDB Endowment*, Vol. 2, No. 1, pp. 526–537, 2009.

[5] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, Vol. 4, No. 1–3, pp. 1–294, 2011.

[6] Lei Duan, Tinghai Pang, Jyrki Nummenmaa, Jie Zuo, Peng Zhang, and Changjie Tang. Bus-OLAP: A Data Management Model for Non-on-Time Events Query Over Bus Journey Data. *Data Science and Engineering*, Vol. 3, No. 1, pp. 52–67, 2018.

[7] Philippe Flajolet and G Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. Technical report, 1985.

[8] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, Vol. 10, No. 4, pp. 270–294, 2001.

[9] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. Approximate selection with guarantees using proxies. *Proceedings of the VLDB Endowment*, Vol. 13, No. 11, pp. 1990–2003, 2020.

[10] Sanjay Krishnan, Jiannan Wang, Michael J Franklin, Ken Goldberg, and Tim Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. In *Proceedings of the VLDB Endowment*, Vol. 8, pp. 1370–1381. 2015.

[11] Kaiyu Li and Guoliang Li. Approximate Query Processing: What is New and Where to Go?: A Survey on Approximate Query Processing. *Data Science and Engineering*, Vol. 3, No. 4, pp. 379–397, 2018.

[12] Ioannis Mytilinis, Dimitrios Tsoumakos, and Nectarios Koziris. Distributed wavelet thresholding for maximum error metrics. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Vol. 26-June-20, pp. 663–677, 2016.

[13] Supriya Nirkhiwale, Alin Dobra, and Christopher Jermaine. A sampling algebra for aggregate estimation. Technical Report 14, 2013.

[14] Viswanath Poosala, Peter J Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, Vol. 25, No. 2, pp. 294–305, 1996.

[15] A. N. Sazish and AAmira. An efficient architecture for HWT using sparse matrix factorisation and DA principles. *IEEE Asia-Pacific Conference on Circuits and Systems, Proceedings, APCCAS*, pp. 1308–1311, 2008.

[16] Guangxuan Song, Wenwen Qu, Xiaojie Liu, and Xiaoling Wang. Approximate Calculation of Window Aggregate Functions via Global Random Sample. *Data Science and Engineering*, Vol. 3, No. 1, pp. 40–51, 2018.

[17] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. The analytical bootstrap: A new method for fast error estimation in Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 277–288, 2014.