DEIM2022 J23-3

既存データ構造へのシノプシス組み込みによる近似問合せ手法

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　近年インターネットやローカル環境問わず，膨大な量のデータが生成，蓄積，やり取りされている．大量データの分析には容量に応じて相応の時間やコストがかかり，その費用を低減するために様々な試行がなされている．本論文ではコストの削減法の中でも近似問合せ処理とその活用法に着目し，現行の大規模データベースシステムにおいて広く利用されている B 木をはじめとする索引構造にメタ情報としてのシノプシスを組み込むことによって B 木を用いた近似問合せ手法を提案，検証する．

キーワード　B 木，近似問合せ，AQP，シノプシス

1　はじめに

現代社会において情報というのは大きな意味を持ち活用の場は多岐にわたる．その需要に応じて膨大な量のデータがやり取りされる中，データの蓄積や維持，管理を担うデータベースシステム上には相応の量のデータが保管されている．データベース上に保管されているデータを用いて分析や活用を行う際にはこうした大量のデータに対して問合せを処理していく必要があるため，その大きさに見合った処理能力が要求される．現代ではデータの処理能力は向上してはいるものの，データ量の増大の速度に対しては大きな遅れを取っている．システムの処理能力の向上が十分でないままにデータが膨張を繰り返すと，データの解析はデータの増大に応じて遅くなっていき，かかる時間やコストはより大きなものとなっていく．データの解析は通常データベースシステム上のみで完結するものではなく，システムが出力した問合せの結果を受けた分析者が考察を深める時間も必要であるため，OLAP [6] などの文脈においても問合せの処理速度は重要な要素である．

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に近似問合せ処理 (Approximate Query Processing) がある．近似問合せ処理ではデータベース上に蓄積されたデータの分布情報や事前算出情報，さらには逐次的なサンプリングを利用することで，与えられた解析タスクの厳密な解ではなく近似的な解を高速に返す．近似問合せ処理を用いればクエリの実行時間を大幅に短縮することが可能である．クエリの実行時間は対象とするデータの容量が大きいほど遅くなり，データ容量のオーダーの増大に伴い分析タスクにおいて要求される精度も低くなっていくため，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が進められている [11]．現在運用されている商用データベースなどでは近似問合せが利用されているものは少なく，研究をさらに進める必要がある．

対して，現行のデータベースにおいてはデータの検索を高速化するための仕組みとして様々の索引構造が実装されている．既存データ構造の一例として B 木がある．B 木では木構造を用いることでデータの検索を効率化しているが，この構造を利用し近似問合せ処理と組み合わせることで，実装の易さと処理能力の大幅な向上という大きな利点を得ることができる．

本論文では，データベース内の B 木上の各ノードにそのノード以下のデータに関する概要情報であるシノプシスをメタデータとして付随し，データの分析時に埋め込まれたシノプシスを活用す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，B 木へのシノプシス埋め込みによって生じるオーバーヘッドを計測し，近似問合せの際のシノプシスによる実行時間の削減を検証する．

本論文の構成は以下の通りである．第 2 章では本論文が対象とすし解決を目指す課題について論じる．第 3 章では一般的な B 木ならびに今回提案するシノプシス埋込み型 B 木について解説し，今回対象とした B 木の探索アルゴリズムの概要について解説する．第 4 章では本研究が提案するシノプシス埋込み型 B 木の性能検証のため，生成時のオーバーヘッドやデータの探索と分析の実行時間の比較実験の概要と結果を提示する．第 5 章ではデータベースシステム上での近似問合せ処理に関する手法や研究について解説し，第 6 章より本論文を総括する．

2　問合せ処理の高速化

近似問合せ処理は近年盛んに研究が進められている分野であり，特に処理時間が膨大になることが予想されるビッグデータの分析において大きな効果を発揮することが期待される．近似問合せ処理ではデータベース対して問合わせられるクエリの結果を算出する際に，データベース上の対象データすべてを使って処理することはせず，ある程度の結果の推定を許すことで一定の精度を保ちつつ実行時間を大きく削減するという手法である．本論文ではデータベース上の既存のデータ構造として B 木を選択し，B 木を用いた近似問合せ処理について実験を行った．通常 B 木をデータ検索時に索引構造として利用する場合には，クエリの条件に合うデータを全データから検索し，当該データ

を結果の算出に利用している．しかしこの方法ではデータの検索，さらに検索後のデータの処理において大量の時間を要してしまう．この問題を解決するため，本稿ではクエリの問合せの際に B 木に含まれるすべてのノードを探索せず，より上層の探索で必要な情報を得られた際にはそれより下層のノードの探索を削減する．探索するノード数を削減することにより，全データをクエリの条件と照合する時間が大幅に短縮され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ようなクエリを主な高速化の対象として議論する．

3 提 案 手 法

3.1 B 木

現行のデータベースではデータの検索を高速化するため様々な既存のデータ構造が存在する．そういった普及している索引データ構造の 1 つとして B 木がある．B 木はデータベース内のデータを探索するために生成される木構造である．データの個数を N とすると，すべてのデータを実際に訪れて探索する場合には $O(N)$ かかる探索時間を $O\log(N)$ に削減することができる．以下に B 木の主な生成方法を示す．はじめにデータベースのリレーション上のデータを属性 A に関する昇順に並べ替える．並べ替えられたデータをいくつかの区間に分割し，分割した各区間を B 木の最下層のノードとする．次に最下層のノードの 1 つ上層のノードを生成する．各最下層ノードの先頭のデータの属性 A の値とそのノードへのポインタを取得し，連続するいくつかのノードについて属性 A の値とポインタを親ノードに保持する．これをすべての最下層ノードへのポインタがいずれかの親ノードに含まれるまで行う．こうして新たに生成された層のノードのさらに親ノードを同様の手順で生成していき，1 つの親ノードに集約されればそのノードを根ノードとして B 木が生成される．B 木を用いることで属性 A に関する条件を課されたクエリの対象データを検索する際に目的のデータをより高速に収集することが可能になる．B 木の生成例を図 1 に記す．図 1 では便宜上二分木で示されているが，実際の B 木は多分木であることが主である．

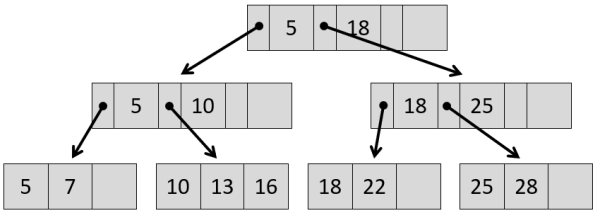

図 1 B 木

3.2 シノプシス埋込み型 B 木

本論文にて提案するシノプシス埋込み型 B 木では，B 木の最下層を除く各ノードに，当該ノードより下層のデータに関するメタ情報を付随することで近似問合せ処理に役立てる．シノプシス埋込み型 B 木内のノードにメタ情報を組み込む場合，そのノードより下位のすべての子ノードに含まれる全データに関する要約情報を算出する．ノードに埋め込むシノプシスとしては、データベースのワークロードや使用用途などによって様々のものが考えられる．各ノードの子ノードに含まれるデータについての個数，平均値，最大値，最小値などといった集計情報を付随すると，それらの情報を要求する問合せ，もしくはそれらを使って近似できる問合せが送られた際に B 木の最下層のノードまで辿らずに，結果をいち早く返すことができる．さらに，B 木を生成する際に利用した属性値とは別の属性 B の概要情報を利用することで，B 木の属性値を条件とする属性 B に関する集計を要求するようなクエリの結果を素早く返すことができ，実行時間の大幅な短縮が見込める．

このように，シノプシス埋込み型 B 木の付加情報として選ぶ候補は無数に存在し，その選び方についても研究の余地が存在する．本論文の実験においては，はじめにデータの個数に加え，B 木の属性とは別の属性の合計，最大値，最小値を B 木の生成時に算出しておき，各ノードに付随する．このシノプシス埋込み型 B 木の生成例を図 2 に示す．図 2 の 2 段目左のノード内には各子ノードへのポインタの後ろにメタ情報が追加されている．例えば最下層の最も左のノードへのポインタの後ろには，子ノードに含まれるデータである $\{5, 7\}$ についてのデータの個数，そして別の属性値 B に関する合計値，最大値，最小値の情報が含まれている．同様に最上層のノードの左側のポインタの後ろには子ノードとそれより下層のノードのデータである $\{5, 7, 10, 13, 16\}$ の個数，そして別の属性値 B に関する合計値，最大値，最小値の情報が含まれている．

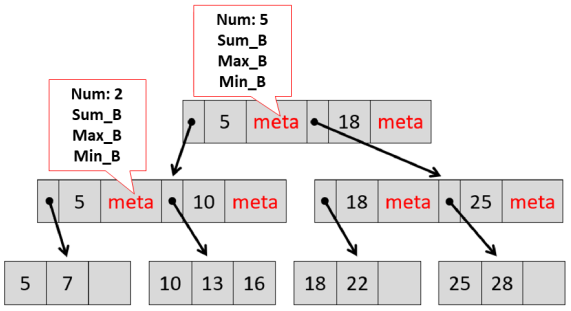

図 2 シノプシス埋込み型 B 木の生成例

さらに本稿においては複数の属性に跨る条件付けがなされている場合を考慮し，別の属性値 C との分散や相関係数を算出するためのメタ情報を付随する実験も行う．通常，JOIN のような複数の属性に跨った条件付けを必要とするクエリへの結果を B 木を用いて返そうとする場合，一方の属性に関する条件付けを基に B 木から該当データを探し，さらにそれが別の属性値の条件に合致するかを得られたデータについて検証しなければならない，2 つ目の属性に関する条件付けを検証する際には利用できる索引構造がないため実行時間が大きくなってしまう．また別の方策として，あらかじめ 2 つの属性値についての B 木を作成しておくという手法も考えられるが，検索条件が 2 次元なのに対して得られる B 木の構造は 1 次元のものを探索するた

めのものであるため，効率的な探索を行うことが難しく，ストレージの占有率に対して得られる実行時間の削減率は小さい．そのため，本稿では 1 つの属性に関する B 木にシノプシスとして別の属性値との相関情報を入れこむことで，近似的に結果を高速に得る手法を提案する．

図 3 に相関情報を付加した B 木の生成例を示す．相関情報として組み込むシノプシスは，相関係数の他に B 木の生成の際に利用して属性値 A の平均と 2 乗平均，さらには別の属性値 B の平均と 2 乗平均，そして A と B を掛け合わせた平均である．相関係数以外の情報はその上層のノードの相関係数の算出の際に利用するため導入している．相関係数は A と B それぞれの分散 V_A，V_B と互いの共分散 V_{AB} から算出することができる．またそれぞれの値は，A の平均値 μ_A，A の 2 乗平均 μ_{A^2}，B の平均値 μ_B，B の 2 乗平均 μ_{B^2}，A と B を掛け合わせた平均値 μ_{AB} を用いて，

$$V_A = {\mu_A}^2 - \mu_{A^2} \tag{1}$$

$$V_B = {\mu_B}^2 - \mu_{B^2} \tag{2}$$

$$V_{AB} = {\mu_{AB}}^2 - \mu_A * \mu_B \tag{3}$$

と表せる．それぞれの平均値はデータの個数を掛け合わせることで合計値に戻すことができ，それを別の子ノードの合計値と足し合わせて再度全体のデータの個数で徐算するとより大きい範囲の平均値を算出することが可能となり，親ノードの相関係数の算出に役立つ．

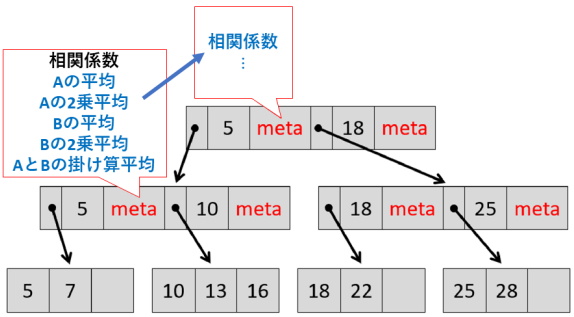

図 3　相関情報を埋め込んだ B 木の生成例

次節からはシノプシス埋込み型 B 木の探索を行う際のアルゴリズムについて解説する．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，本論文にて提案する新規探索手法について解説する．

3.3　深さ優先探索

シノプシス埋込み型 B 木を対象として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を表す．深さ優先探索では最下層のノードまで探索した後，同じノードを親として持つ他のノードを探索し，該当ノードを探索し終えると 1 つ前の親ノードに戻り同じことを繰り返す，という順で探索が進められる．

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
```

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
        b0 ← pop(Q)              ▷ Q の先頭のノードを取り出す
        for each n ∈ N do        ▷ N は b0 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
```

3.4　幅優先探索

シノプシス埋込み型 B 木の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いる層のノードの探索を先に進めていき，その層がすべて探索し終えたら 1 つ下のノードを同様の手順で探索していくという順の探索アルゴリズムである．

3.5　提案手法：シノプシスを考慮した探索

シノプシス埋込み型 B 木を対象とした提案探索手法をアルゴリズム 3 に示す．基本的な探索手順は深さ優先探索と同じものである．深さ優先と違う点は，新しいノードを探索する際にシノプシスを考慮し，シノプシスから十分な情報が得られる場合にはそれよりも下層のノードの探索を削減しているという点である．例えば属性 A の値が 5 から 15 の間のデータの算出情報を求めるようなクエリを考えた時，次に探索するノードに含まれるデータの範囲が 6 から 10 であれば，そこに含まれるすべてのデータはクエリの条件を満たしている．したがってその範囲のデータをすべて収集し新たに値を算出せずに，そのノードに含まれるシノプシスを利用すれば厳密な解が得られるため，それより下のノードの探索を削減することができる．こうしてシノプシスを利用することで削減できる下層の無駄なノードの

アルゴリズム 3 シノプシスを考慮した探索

```
function SYNOPSISFIRST(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
            if synopsis is sufficient then
                return
            end if
            DEPTHFIRST(n)
        end if
    end for
end function
```

探索をなくすことで，実行時間の大幅な短縮が期待される．

4 評価実験

本章では提案手法の有効性を示すための評価実験の結果を示す．生成された B 木を用いて探索を行った際の性能を比較する．生成する B 木は用途によって組込むシノプシスが異なるため，各 B 木において探索手法別の結果を出力し性能を示す．

4.1 実験環境

表 1 実験環境

CPU	Intel Xeon GOld 6132
メモリ	96GB
HDD	28TB
OS	CentOS Linux release 7.7.1908 (Core)

表 2 データセット

リレーション	スケールファクタ
orders	0.1
	1
	10
lineitem	0.1
	1
	10

実験において利用した環境について説明する．実験に使用したマシンは表 1 に記す．B 木の生成や探索に関するコードはすべて C 言語を用いて記述した．

今回使用したデータセットは TPC-H である．TPC-H は購買ログデータを模して作成される汎用ベンチマークであり，データベースシステムの性能実験において利用される頻度の高い信頼性のあるデータセットであるため今回の実験に使用した．実験では TPC-H の lineitem 表と orders 表を使用し，容量を変えて実験を行った．データセットはデータ作成ツールの dbgen を用いて生成した．dbgen のバージョンは 2.18.0 である．利用した TPC-H の容量は表 2 にデータの作成時に指定するスケールファクタ (SF) とともに示す．SF は作成データの容量のオーダを指定するパラメータであり，SF が 1 では 1GB 程度である．

表 3 探索クエリの種類

クエリ名	SQL 文
Q1	SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN X, Y
Q2	SELECT MAX(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN X, Y
Q3	SELECT MAX(L_SHIPDATE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN X, Y

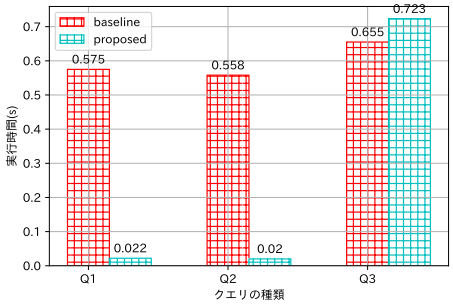

図 4 シノプシス組込み型 B 木の探索評価

B 木の作成に用いた属性は orders 表では orderkey を選択し，lineitem 表では複数属性による条件付けを行ったクエリの評価のため orderkey と shipdate それぞれについて B 木を作成した．B 木上の各ノードの容量は 4096B の固定長であり，B 木の生成時には各ノードがなるべく与えられた容量を使い切るようにデータを格納する．

4.2 シノプシス組込み型 B 木の探索評価

次に，シノプシス組込み型 B 木を用いた探索性能について比較する．探索性能の比較では SF が 10 の lineitem 表を対象として生成した B 木上でクエリの探索を行い，その結果を返すまでの実行時間を比較する．作成する 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，extendedprice の合計値，最大値，最小値を保持するもの (proposed)

の 2 種である．baseline の B 木は深さ優先探索を用いて探索を行い，proposed の B 木では提案したシノプシスを考慮した探索を行う．なお本節の実験においてはクエリの解に近似は利用しておらず，厳密な解をより素早く返す性能を比較している．

表 3 に比較に使用したクエリを示す．いずれも B 木生成に用いられた属性である orderkey の範囲を限定し，その範囲内に存在するデータの別の属性値に関する演算を行うようなクエリである．Q1 では orderkey が X から Y の間のデータに対して，その extendedprice の合計値を要請する．Q2 では orderkey が X から Y の間のデータに対して，その extendedprice の中での最大値を要請する．Q3 では orderkey が X から Y の間のデータに対して，その shipdate の最大値を要請する．Q1 と Q2 では B 木に組込んだシノプシスを利用することで上層で探索を

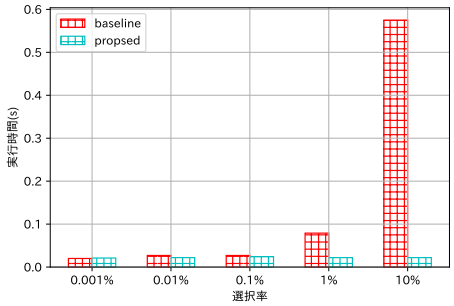

図 5　選択率による実行時間の変化

打ち切ることができるが，Q3 ではシノプシスに含まれていない shipdate のデータを必要とするためシノプシスの恩恵を受けない探索である．また，$X = 54000001$，$Y = 60000001$ としてクエリを実行することで，データの選択率を 10%程度としている．

探索評価の比較結果を図 4 に示す．シノプシスを考慮した探索では内部ノードに extendedprice の合計値や最大値の情報を含んでいるため，合計値を求める Q1 や最大値を求める Q2 のクエリに対しては探索ノード数削減の効果が大きく，実行時間の削減率が大きい．最も実行時間の削減率の大きかった Q2 では proposed の実行時間は baseline の 4.5%未満となっており，約 25 倍の実行時間短縮を実現した．Q3 ではシノプシスに含まれていない shipdate についての探索を行うため，近似なしの厳密解では実行時間の削減は見込めないが，baseline の B 木と比較しても探索にほとんど余分な時間を要しないということが確認できた．

4.3　データの選択率ごとの探索性能評価

次に，クエリが対象とするデータの選択率を変えて実験を行い，探索手法の優位性を検証する．図 5 に結果を示す．実行したクエリは Q1 であり，X，Y，の範囲が表記選択率を満たすように乱数を用いてそれぞれの変数の値を決定した．

図 5 から，クエリが対象としているデータの量が多いほど実行時間の削減効果が大きく得られ宇ということが分かった．これは対象としているデータの量が多い場合ほど baseline の手法における探索に時間を要していたため，提案手法を用いたことによる時間削減の余地が大きかったと考えられる．最も実行時間の削減が大きかった選択率 10%のクエリでは，baseline では $O(10^5)$ の数のノードを探索する必要があったのに対し，proposed では $O(10)$ 程度のノードの探索で済んでいた．

4.4　複数属性にまたがる条件付けがなされたクエリ：相関のない場合

複数の属性にまたがった条件付けがなされたクエリにおける提案手法の有効性について比較する．表 4 に実験に用いた複数条件のクエリを示す．Q4 は lineitem 表に対して，orderkey が X_1 から Y_1 の範囲内，かつ shipdate が X_2 から Y_2 の範囲内に収まるデータについて，そのデータの extendedprice の合計値を返すというものである．orderkey の選択率は 10%で固定し，

表 4　複数条件のクエリ

クエリ名	SQL 文
Q4	SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN X_1, Y_1 AND L_SHIPDATE BETWEEN X_2, Y_2

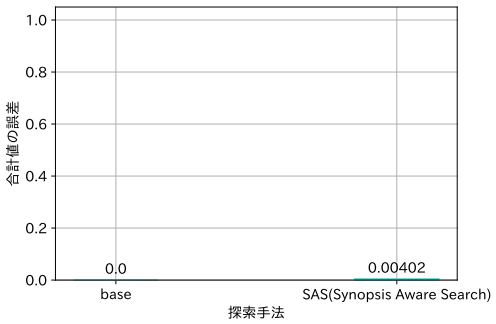

図 6　orderkey の条件範囲の位置が値範囲の 0%-10%の相関がない場合の Q4 の実行結果の誤差

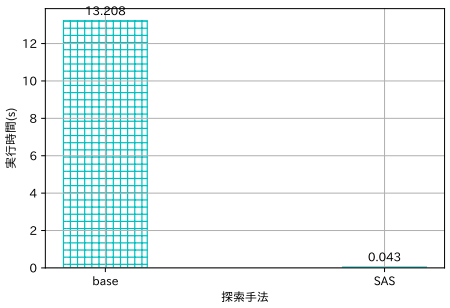

図 7　orderkey の条件範囲の位置が値範囲の 0%-10%の Q4 の相関がない場合の実行時間

選択位置を 10%ずつ移動させて実験を行った．shipdate には 7 年分の値が含まれているため，選択率を全体の 1/7 とし，選択位置を orderkey と同様に移動させた．また，本節の実験では orderkey と shipdate の間に相関がないデータセットを dbgen を用いて生成し，利用している．

次に，比較する探索手法について解説する．探索手法は，

（1）　orderkey の B 木を用いた厳密な探索 (base)

（2）　orderkey の B 木から得る合計値と shipdate の B 木から得られた割合を用いた近似的な探索 (Synopsis Aware Search)

の 4 種を比較する．base では orderkey の B 木を利用し，orderkey に関する条件付けに該当するデータをすべて探索し，その中で shipdate の条件付けにも該当するデータをフィルタリングし，合計値を算出する．Synopsis Aware Search (SAS) ではシノプシスを積極的に利用し，orderkey の B 木から orderkey に関する条件付けを満たすデータの合計値を獲得し，それに対して shipdate の B 木から得た shipdate の条件付けを満たす該当データの割合を掛け合わせて近似的な合計値を算出する．

図 6 に各探索手法による合計値の誤差を示す．Q4 での orderkey に関する条件付けが全体のどの位置のデータに該当する

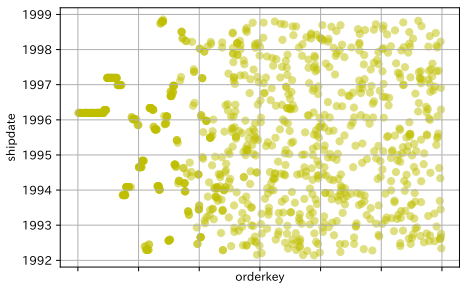

図 8　偏りのあるデータの orderkey に対する shipdate のデータ分布の概略

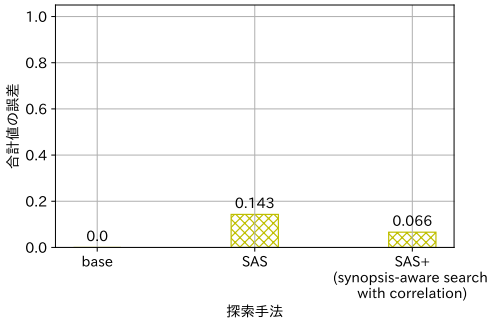

図 9　orderkey の条件範囲の位置が値範囲の 60%-70%の Q4 の Z1 での実行結果の誤差

かを変え実験をしており，6 では代表として先頭 10%を対象とした結果を示す．いずれの位置の結果についても提案手法が非常に高い精度で合計値を返すことができている．誤差が最も大きくなったのは orderkey の先頭の 10%を対象としたクエリで，その誤差は最大で約 0.4%であった．

また，図 7 に各探索手法のクエリ実行時間を示す．どちらの提案手法においても既存手法と比べて実行時間を大きく削減できていることが確認できた．クエリの実行時間は最大で約 307 倍の短縮が確認された．既存手法では 1 種の B 木の探索のみだったが，提案手法では 2 種の B 木の探索が必要であり，さらにそれぞれの B 木で得られたデータを合成するオーバヘッドがかかる．しかし，それぞれの B 木においてシノプシスを考慮し探索を大幅に削減することで実行時間を大きく削減することができたと考えられる．2 種の既存手法において実行時間が大きく異なっているが，これは先に orderkey の条件付けをもとに B 木を探索した場合にはその後のフィルタリングにかけるデータが全体の 1/10 となるのに対し，先に shipdate の条件付けをもとに探索した場合にはフィルタリングにかけるデータが全体の 1/7 となり，時間のかかる処理に対してより多くのデータが対象になったことが原因だと考えられる．

4.5　複数属性にまたがる条件付けがなされたクエリ:相関のある場合

複数の属性にまたがった条件付けがなされたクエリにおいて，属性間の相関が無視できない場合について提案手法の有効性について比較する．相関のあるデータセットを作成するため，TPC-H のデータセットを生成する際に Zipf 分布に沿った偏りのあるデータが生成されるようにした．Zipf 分布に沿うデータセットではパラメータ z の値を変化させることで，偏りの度合いを変更することができる．作成データの分布を図 8 に示す．データの分布は，データセット全体から約 1000 レコードをランダムサンプリングしたものを使って測定した．orderkey の値が若いものほどデータの偏りが強く，反対に orderkey の後ろのデータでは偏りが弱い．

また，前節の探索手法 base，SAS はそのままに，

（1） SAS の orderkey の B 木において相関を考慮する近似的な探索 (SAS+)

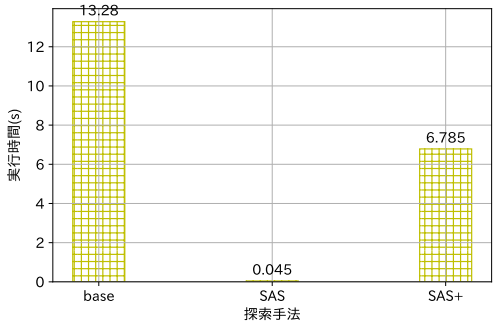

図 10　orderkey の条件範囲の位置が値範囲の 0%-10%の Q4 の Z1 での実行時間

の相関を考慮した探索を加え，計 3 種の探索手法の性能を比較する．相関を考慮する探索では，ノード内の相関係数が閾値の 0.02 を下回る場合のみシノプシスの合計値を参照する．この閾値は実験的に決定したものである．

図 9 に Z1 をデータセットとした場合の各探索手法による実行結果の誤差を示す．いずれの orderkey の選択範囲においても相関を考慮した探索手法のほうが，相関を考慮しない手法よりも高い精度で近似解を返せていることが分かる．近似解の測定誤差は最も大きい箇所では約 12.6%程であった．

図 10 に各探索手法による実行時間を示す．SAS2 では極端に高速に実行が終わる反面，非常に大きな誤差を生じる部分がある．SAS+は実行時間を削減している．これは相関係数の考慮を行う過程において，相関の弱い箇所ではシノプシスを利用するという実装が効果的に働いたことによる恩恵だと考えられる．相関の無視できない箇所では実行に時間をかけることで精度の良い結果を返し，相関の弱い箇所ではシノプシスを用いて高速に結果を返すことを実現できた．

5　関 連 研 究

近似問合せ処理における手法は以下の 2 種を用いたものに大別される．

- クエリ実行時の逐次サンプリング
- 事前算出シノプシス

前者を利用した手法ではクエリの実行時にデータ全体から一部データをサンプリングするため，近似機能を実装するための

ストレージの余分な容量を必要としないが，クエリの実行時間にサンプリングの実行が加算されるという欠点を持つ [14], [17]．それに対し後者を用いた手法では，事前算出したシノプシスをストレージに保持するためのオーバーヘッドがかかる反面，一度算出したシノプシスは将来的にデータベースに対して問合せのある複数のクエリに対して繰り返し利用することができ，実行時間は他方の手法よりも短いという利点がある．本章ではこの 2 種を踏まえた上で，近似問合せ処理の主要な手法に関して紹介する．

5.1 信頼区間

クエリ実行時にサンプルを生成する手法においては生成したサンプルを用いた近似ではどれほど精度に誤差が生じるかが重要である．サンプリングを行ったことによって生じる誤差の可能性を定量的に提示する方法として，信頼区間を利用し統計的に誤差の程度を示すものがある [9]．クエリ実行以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-分布を仮定することによって信頼区間を数学的に算出することが可能である．データの分布が未確認である場合には，分布を推定するためのサンプリングを個別に行う，もしくはサンプリングをしたデータセットから再度サンプリングを行うといった方策を用いることでエラー率を算出するような研究がされている [18]．

5.2 事前サンプリング

本章の冒頭で述べた 2 種の手法の組み合わせたような手法として，事前に生成しておいたサンプルをクエリ実行時に利用して近似処理を行うという手法がある．手法の内容としては 2 種の組み合わせを取ったものであるが，事前に生成しておいたサンプルをデータベース上に保持しておく必要があるため，性質としては事前算出シノプシスを用いたものに分類される．逐次的なサンプリングと比べ，事前にサンプルを算出しておくことでクエリの実行時間に影響を与えることがないため，より良く全体のデータを表したようなサンプルを生成することに時間を要することが可能である．また，事前情報としてデータベースに将来的にどのようなクエリが問い合わせられる可能性が高いかというワークロードの情報を得ていれば，頻出クエリに対して精度が向上するようなサンプルをあらかじめ生成しておくこともできる．ただし，各クエリの種類に対してサンプルセットを保持しておく場合にはそれに応じたストレージが要求される．この問題を解決するため，同一のデータを利用するクエリをグループ分けすることにより保持するサンプル数を少なく保つ Blink DB [2] のような研究も行われている．

5.3 ヒストグラム

データセットを一定区間ごとのサブセットに区切り，各区間内にどの程度のデータが含まれているかを算出，記録することでおおまかなデータの分布を表した情報をヒストグラムという．ヒストグラムはデータの個数を計上することで生成できるため生成が容易であり，現行の商用データベースにおいて機能として実装されているものも多い [4], [15]．作成されるヒストグラムにはデータを分割する各区間の範囲を一定とする equi-width，各区間に含まれるデータ数を一定とする equi-depth なといくつかの種類があり [5]，各区間をどのように分割するかを最適化することに主眼をおいた研究もなされている [1]．ただし複雑なヒストグラムは数値データに関してのみ生成することができ，文字列データやその他複雑なクエリ処理には適していない．

5.4 Wavelet

ヒストグラムは全体のデータのサブセットであるデータ構造を提供するものであるのに対し，wavelet は全体データを集約して圧縮することで元データよりも少ない容量でデータの個数や頻度の情報を表現する手法である．データは事前の算出によってまとめられているため，クエリの実行時にはその処理にかかる時間を削減することが可能になる．Wavelet はいくつかの種類が存在するがそのうち簡単なものとして haar-wavelet 変換などがある．詳しい解説は本稿では割愛する．Wavelet を用いる手法ではデータ圧縮に伴い情報量は小さくなるため，データのどの側面の情報が重要であり，どの情報が不要なのかを見極めることが肝要である．データ削減による誤差は L_2 エラーなどを用いて表現することができる．様々なアプローチからの研究が wavelet についても行われている [13], [16]．

5.5 Sketches

Sketch はデータベースに保持されているデータのうち，数値として置き換えることができるものを行列表示にすることで問合せ処理時に扱いやすくしたものである．例えば性別のように 2 値分類されるデータでは，男性女性をそれぞれ 0 と 1 のビット列に変換し保持しておくことでデータを圧縮し容量を削減できる．行列として保持することのメリットはデータ圧縮だけでなく，データの更新を容易に反映できる点，処理の際の並列化の容易さなどが挙げられる．Sketch は大きく分けて，頻度ベースのもの [3] と種類別数値のもの [7] の 2 種が存在する．その他の Sketch の大きな利点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 のような種類のクエリの処理においても有効性が示されている [5]．数値データを行列として保持する特徴により，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可能であり，また更新の反映の容易さの観点から，金融といった分野のデータ処理においても利用が期待される．

5.6 Materialized view

現行のデータベースにおいてすでに広く利用されている機能の中で近似問合せ処理に関係のあるものとして materialized view [8], [10] がある．Materialized view はクエリの実行結果やその途中結果として得られたデータを新たにデータベース上に保存しておくという手法であり，同じ内容や似た内容のクエリが頻繁に問い合わせられるシステムにおいて大きな効果を発揮する．1 度目のクエリの問合せ時に結果を保存しておくことで，同じ内容のデータを求められた際には保存してあるデータをそのまま返すことができ，実行時間を大幅に削減できる．1 度目のクエリ処理では近似処理が行われていないため，厳密な結果を常に高速に返すことができることが大きな利点である．その

反面，すべてのクエリの結果を保持しおくことは問い合わせられるクエリの種類が多いデータベースでは容量オーバーヘッドが大きくなってしまう．また一度生成した materialized view の値を更新する際には通常の処理と同様の時間を要するため，どの結果をどの程度の期間保持し利用するかについてはデータベースを運用しながら慎重に決めていく必要がある．

5.7 シノプシスとサンプリングの複合

シノプシスとサンプリングを複合して利用することを主眼においた研究も存在する [12]．この研究では木構造を用いてシノプシスを使って厳密な値を算出できる箇所ではシノプシスを用い，精度が怪しくなるような部分ではサンプリングを使うという方策を用いることで，実行時間を最大限短縮しつつ精度を高く保つということの実現を目指している．木構造やシノプシスを用いているという部分では本稿の実験と重なる部分もあるが，クエリの条件付けが複数次元に跨っている際の対処などの観点から異なっている．

6 おわりに

本論文では，現行のデータベースにおいて広く活用されている索引構造である B 木上にデータの分布や概要に関するメタ情報を組み込みことで，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．また，複数属性に跨る条件付けをされたクエリについてもシノプシス埋込み型 B 木を用いた近似手法を提案した．

そして通常の B 木と各種シノプシスを付加したシノプシス埋込み型 B 木の生成やその探索に関する比較実験を行い，オーバーヘッドが小さいこと，実行時間の大きな短縮が可能となることを示した．

今後は複数条件下での精度の高い推定法や，データの挿入や削除を行った際の効率的なシノプシスの更新法について追究していく予定である．

謝　辞

本研究の一部は、日本学術振興会科学研究費補助金 20H04191 の助成を受けたものである。

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] Xi Liang, Stavros Sintos, Zechao Shang, and Sanjay Krishnan. Combining aggregation and sampling (nearly) optimally for approximate query processing. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pp. 1129–1141. ACM, 2021.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.