単一表クエリを対象とするシノプシス埋込みによる近似問合せ処理の評価

高田　実佳†　　合田　和生†

† 東京大学情報理工学系研究科　〒 153–8503 東京都目黒区駒場 4-6-1
E-mail: †{mtakata,kgoda}@tkl.iis.u-tokyo.ac.jp

あらまし　業務データを収集し，蓄積したビッグデータを分析することによる業務の改善・新たな知識発見への期待が高まっている．そのような分析では集計が頻繁に実施される．集計には，必ずしも従来データベースが求める正確な値は必要ではなく，それよりもレスポンスの早さが求められることがある．本論文では、索引構造に付加情報を埋込むことによって高速に近似問合せ処理を実行する方式を既存 RDBMS で実装し，検証する．

キーワード　データベース，索引，近似問合せ

1　はじめに

多くの企業や組織では，日々様々な業務データが生成され，そして収集・蓄積された大量データを分析することによって業務の改善や新たな知識発見に期待が高まっている．業務データの蓄積・管理には構造データベースが広く利用されており，様々な合計値や最大値，最小値など集計値の問合せ処理が分析には必要とされる．例えば，小売業では，日々の売り上げが期待できる製品，製品数や人員を適切に調整する為，製品の購買履歴を用いて，一日の売り上げ総数，在庫数，来客数等の定期的な集計が必要とされている [1]．この様な分析はインタラクティブに実施されることが多く，その為にはデータベースに蓄積されたデータに対し，高速に様々な集計値計算が実行されることが求められる．

高速な集計値計算に向けて，正確な値を高速に求めるデータベースの検索技術はこれまで多数提案されてきた [2]．代表的なものの一つには索引があり，中でも B+Tree [3] は多くの関係データベースで利用されている． B+Tree は範囲検索を効率的に行えるメリットがあり集計計算に必要な範囲のデータを高速に検索する為に有効である．しかし，日々業務データが生成・蓄積されることで，そうして増大した業務データを対象とした集計計算の場合，索引のサイズが大きくなり，最下層のリーフページまでノードを辿ると検索時間が増大するという問題が生じている．

インタラクティブな分析における高速な集計値計算を考慮した時に，正確性は必ずしも必要ではない．こうした観点に着目し，[4] らは，既存の B+Tree など索引にシノプシスを埋込むことで似問合せの高速化手法 (Synopsis-aware search; SAS, Synopsis-aware search plus inter-attribute correlation; SAS+) を提案している．SAS は，B+Tree の中間ノードに下位ノードの最大値，最小値，合計値，総数といった統計情報をシノプシスとして持たせておく事で B+Tree のリーフページまで辿らずとも，近似値を求めることができるという近似問い合せ手法であり，SAS+は SAS を発展させ，複数カラム間の相関を考慮してシノプシスの利用可否を決定する近似問合せ手法である．本論文では，[4] の研究を発展させ，業務データの蓄積・管理に広く利用されている関係データベースシステムである PostgreSQL [5] にシノプシスを埋込んだ索引による近似問合せ処理 (SAS, SAS+) を組込み，その試作による評価結果を示す．

本論文の構成は，以下の通りである．第 2 章では，既存の近似問合せとその課題について言及する．第 3 章では，シノプシス埋込み索引による近似問合せ処理について説明し，第 4 章では，その評価を示す．第 5 章では，関連文献について言及し，第 6 章では，今後に向けた課題と将来展望について纏める．

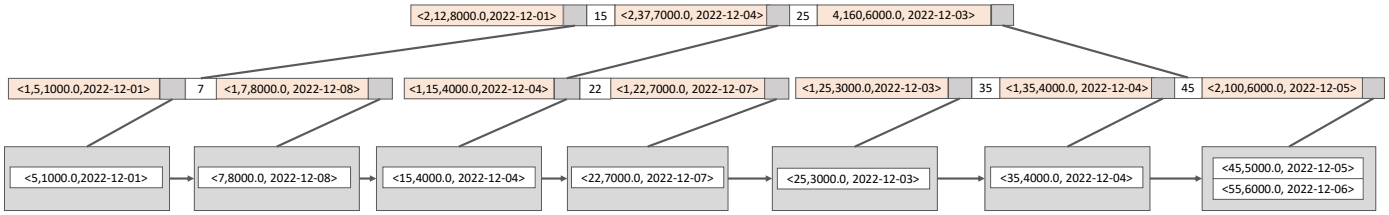
2　近似問合せ処理

本章では，近似問合せ処理へのニーズと，既存の近似問合せ処理およびその問題点について述べる．近年，ビックデータを用いた分析に期待が高まり，広くインタラクティブな集計計算が実施されている．例えば製造業では，IoT センサ等の発達により，原材料の調達，設計，製造，流通，といった製品のライフサイクルの管理のデータを計測でき収集・蓄積が進んでいる [6]．蓄積された大量のデータを利用し，日々の受注製品数，最大受注製品，売上金額なと集計値を定期的に計算し，その値に基づき，調達すべき原材料の種類，個数や，製造すべき生産量，出荷量，販売量の適切な決定に活用している [7]．このようにデータを活用した業務改善・効率化には，統計的な集計値の計算が頻繁に実施される必要がある．この様な集計計算には，必ずしも一円単位の売り上げ金額や一桁単位の生産数といった正確な値は必要ではなく，それ以上に様々な種類の集計値をインタラクティブに実行できることに価値が求められる場合がある．即ち，合計値，最大・最小値，平均値など様々な集計値の問合せに対してインタラクティブなスピードで結果を返すことが求められる．

集計値の近似値を求める近似問合せ処理はこれまで長年議論されてきた．代表的な手法は 2 通りに分類され，蓄積されたデータからサンプリングしたデータのみを用いて問合せ時に即座に集計値を計算・推定する方式であり，もう一つは事前に

ORDERKEY	PRICE	SHIPDATE
5	1000.0	2022-12-01
7	8000.0	2022-12-08
15	2000.0	2022-12-02
22	7000.0	2022-12-07
25	3000.0	2022-12-03
35	4000.0	2022-12-04
45	5000.0	2022-12-05
55	6000.0	2022-12-06

(a) テーブル例



(b) シノプシス埋込み B+Tree の例

図 1: シノプシス埋込み索引の例

データベースに蓄えられたデータを用いて集計値など付加情報を計算し保存しておくことで，問合せ時にその値を用いて集計値を返答する方式である．しかし，前者は事前準備が不要のため事前準備に時間や付加情報を保村しておくためにストレージ負荷がかからないが応答時間が長いというデメリットがあり，後者は前者に比べ応答時間が高速だが，付加情報保存のためのストレージ負荷がかかるというデメリットがある．その上，多くの既存提案は近似問合せの為に新たなデータ構造や処理を備える必要がある．

この様な問題に対し，[4] は既存のデータ構造に付加情報であるシノプシスを埋込むことによって近似問合せの高速化する手法 (Synopsis-aware search; SAS, Synopsis-aware search plus inter-attribute correlation; SAS+) を提案している．SAS は，事前にシノプシスを埋め込んだ索引を生成しておき，問合せクエリが入力されると，その索引に埋め込まれたシノプシスを利用する事で探索範囲を狭め，高速に近似解を返す検索方式であり，ストレージ負荷が約 2%程度に抑えながら最大 25% 以上近似解の応答時間を高速化を達成できる場合があることを示している．

図 1 を用いて SAS の有効性を説明する．図 1(a) は関係データベースで管理されたテーブルの例であり，そのテーブルの ORDERKEY を索引キーとし，付加情報を埋め込んだ B+Tree 索引の例を図 1(b) に示す．図 1(b) では，一般的な B+Tree に加え，その各中間ノードには，下位ノードの集計値をシノプシスと呼ばれる付加情報として埋め込まれている．図 1(b) の例では，付加情報として COUNT(ORDERKEY)， SUM(ORDERKEY)，MAX(PRICE)，MIN(SHIPDATE) を保有している．B+Tree はリーフノードと中間ノードをもつが，一般的にリーフノードにテーブルレコードを保有するためデータサイズが大きくなるが，中間ノードは軽量である．その特性を利用することで，中間ノードに下位ノードの集計値をシノプシスとして埋め込んでも，シノプシスによるストレージオーバーヘッドは軽量になると見込まれる．そして SAS による問合せ処理では．一般的な B+Tree を用いた探索同様，ルートから深さ優先探索を実施し，問合せクエリに対してマッチする集計値を中間ノードで発見した時点で，それより下位のノード検索をスキップする．これにより，問合せ検索において索引のリーフノードの探索時間を削減し，問合せ処理時間を大幅に短縮できることが検証されている．しかし，SAS は事前に準備した付加情報が利用できる場合においては非常に有効であるが，ユーザの多様な問合せニーズに対応可能や広く利用されている既存のデータベースシステムとの併用について検討が十分行われていない．本論文では，既存のデータベースシステムへの近似問合せの組込んだ高速な近似問合せ処理を提案する．

3 既存関係データベースシステムへの近似問合せ手法の組込み

本章では，ユーザの多様な問合せニーズに対応可能や広く利用されている既存のデータベースシステムとの併用に向けて，[4] らの提案した高速な近似問合せ処理方式を既存データベースに取り込む事によって，多様な問合せに対して高速に近似解を求める問合せ方式を提案する．

まず，ユーザの多様な集計値計算ニーズに応える為，広く利用されている既存データベースシステムである PostgreSQL [5] へのシノプシスを埋め込み索引を用いた近似問合せを組込む実装方式を提案する．通常，データベース管理システムは，データベース構築時に指定したデータ領域に存在しているデータにアクセスすることが一般的である．PostgreSQL はそれに加え，外部データラッパ (FDW; Foreign Data Wrapper) をサポートしており，その機能により外部サーバ (Foreign Server) やその外部サーバに存在している外部テーブル (Foreign Table) を登録しておくことで，PostgreSQL インターフェースを用いて

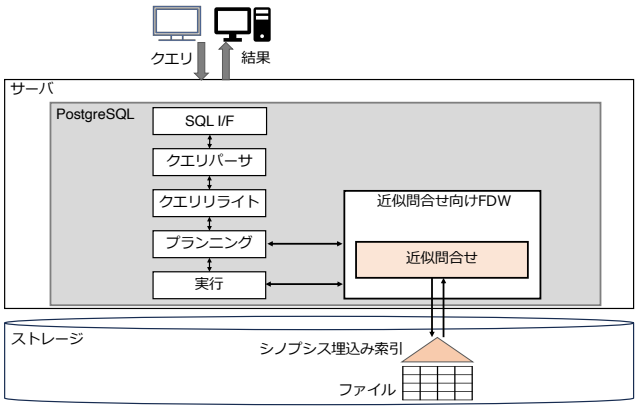


図 2: PostgreSQL 外部データラッパを用いたアーキテクチャ

外部サーバにある外部テーブルを参照できる機能である [8]. 様々な種類の外部データラッパは既に公開されており，例えば，外部のファイルにアクセスして検索する File FDW や別の PostgreSQL サーバにアクセスする Postgre FDW などが既に公開されている．その様な公開済み FDW に加え，PostgreSQL は独自の FDW を開発できるよう API を提供しており，我々はその API を利用する事で近似問合せ処理を実施する独自 FDW を開発する方式を検討に取り入れた．以下その独自 FDW を Approximate_FDW とよぶ.

PostgreSQL は FDW 作成に必要な API を提供しており，本提案ではその中から下記の 7 種類の API を用いる事とする．各 API の概要は以下の通りである.

- GetForeignRelSize: テーブルサイズを見積もる
- GetForeignPaths: アクセスパスを生成する
- GetForeignPlan: プランノードを生成する
- BeginForeignScan: スキャン開始時に呼ばれる
- IterateForeignScan: 上位ノードが 1 行必要とした時に呼ばれる
- ReSacnForeignScan: スキャン位置を先頭に戻したい時に呼ばれる
- EndForeignScan: スキャン終了時に呼ばれる

これら API は PostgreSQL に実行計画と実行処理の 2 種類に大別できる．GetForeignRelSize, GetForeignPaths, GetForeignPlan が実行計画に関連する API であり，BeginForeignScan, IterateForeignScan, ReSacnForeignScan, EndForeignScan が実行処理に関連する API である．ユーザは問合せ前に，PostgreSQL サーバに Approximate FDW，Approximate FDW を用いる外部サーバ，外部テーブルの登録を行うことで，外部テーブルへの問合せを PostgreSQL が認識した際，Approximate FDW を呼び出す事ができる．下記に例を示す様に，外部テーブルは通常テーブル定義と同様に SQL を用いて登録が可能である.

- Approximate_FDW(e.g., approximate_fdw) 登録

```
CREATE EXTENSION approximate_fdw;
```

- Approximate_FDW を用いる外部サーバ (e.g., approximate_server) 登録

```
CREATE SERVER approximate_server
    FOREIGN DATA WRAPPER approximate_fdw;
```

- 外部テーブル (e.g., foreign_table) 登録

```
CREATE FOREIGN TABLE foreign_table (val int)
    SERVER approximate_server;
```

この様に．FDW 機能を用いて，シノプシスを埋め込んだ索引を用いた検索を実現するアーキテクチャを図 2 に提案する．ユーザが PostgreSQL のインターフェースを介してクエリを投入すると，図 2 に示すように，クエリはパース，クエリリライトを経て，プランニング，実行を行う．事前に登録した外部テーブルへの問合せがクエリパーサ検知されると，PostgreSQL は対応する FDW(e.g., Approximate_FDW) を呼び出し，FDW 作成時に登録した実行計画，実行処理を行うので，実行処理内で，シノプシスを埋込み索引を参照し，近似問合せ処理が可能となる．これによりユーザが与えた問合せクエリに対し，独自の処理方式を用いて求めた近似解を返す事ができる．以上の仕組みを用いて，我々は問合せ時にシノプシス埋込み索引を用いて集計計算に対する近似値を求めるプロトタイプを開発し，評価を行った.

4 評　　価

本章では，[4] らの提案した近似問合せ処理（SAS）が外部データラッパ (FDW) 上で既存データベースがもつ索引と同様に実行可能か，および既存データベースがもつ索引に比べ優位性の有無を検証する.

4.1 実 験 環 境

本実験では，ベンチマークセットである TPC-H [9] の dbgen を用いて生成したデータセットを利用する．データセットのスケールファクタ (SF) は，1,10,100 を用意した.

次に，問合せ処理に用いる索引として，一般に多くのデータベースシステムで適用されている B+Tree [2, 3] を用い，以下の索引キーをもつ B+Tree 索引を用意した．全てノードサイズは 8192Byte とした.

- ORDERS (ORDERKEY): ORDERS 表の O_ORDERKEY を索引キーとしてもつ索引
- LINEITEM (ORDERKEY, LINENUMBER): LINEITEM 表の (L_ORDERKEY, L_LINENUMBER) を索引キーとする索引
- LINEITEM (SHIPDATE): LINEITEM 表の L_SHIPDATE を索引キーとする索引.

索引に埋込む付加情報であるシノプシスとしては，以下を用意した.

- **nosyn**: シノプシス埋込みなし
- **syn1**: 索引キーの集計値 (SUM, MAX, MIN, COUNT) (e.g., ORDERS (ORDERKEY) の場合 O_ORDERKEY の集計値)
- **syn2**: syn1 に加え，索引キーとは異なる属性の集計値 (SUM, MAX, MIN, COUNT)

表 1: 単一制約クエリ

Q1	SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM BETWEEN 5,4000,001 AND 6,000,0001;
Q2	SELECT MAX(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN 5,4000,001 AND 6,000,0001;
Q3	SELECT MAX(L_SHIPDATE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN 5,4000,001 AND 6,000,0001;

表 2: 複合制約クエリ

Q4	SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN 36,000,000 AND 42,000,000 AND L_SHIPDATE BETWEEN 1992-01-01 AND 1992-12-31;

- **syn3**: syn2 に加え，索引キーと索引キーとは異なる属性との相関係数および相関係数計算に必要な集計値

相関係数の計算には，分散の定理

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

および，共分散の定理

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

を用いて，以下の様に相関係数 r を計算可能である．

$$r = \frac{V_{AB}}{\sqrt{V_A} * \sqrt{V_B}} \tag{3}$$

以上より，**syn3** には，索引の各ノード以下の属性間の相関係数を計算する為，各属性の和，二乗和，属性積の和を相関係数計算に必要な集計値として保有した．

本実験では評価クエリとして，単一表の問合せに対して単一制約をもつクエリ (Q1-Q3) と複合制約をもつクエリ (Q4) を用いた．

本実験では，索引は事前に生成しファイルとして保存し，問合せ時に索引を呼び出し，索引を用いた検索は，IterateForeignScan API 内で実施した．評価クエリでは ORDERS 表および LINEITEM 表を対象としていることから，ORDERS 表および LINEITEM 表を外部表として事前に create する．これにより，外部表に対する問合せクエリを PostgreSQL インターフェース (I/F) から入力すると，近似問合せ処理を実行する FDW(Approximate_FDW) が呼び出される様にした．

評価軸には，各種シノプシスを埋込み索引のリーフノード数，中間ノード数を計測し，シノプシス埋込みによるストレージオーバーヘッドを検証する．また，異なる問合せに対して，一般的な B+Tree の検索と SAS，および相関係数を考慮した SAS+による検索を比較し，エラー率 [10] および問合せに要する実行時間を用いて検証する [4]．なお，全ての実験は，OS は Amazon Linux release 2 (Karoo)，PosgreSQL はバージョン

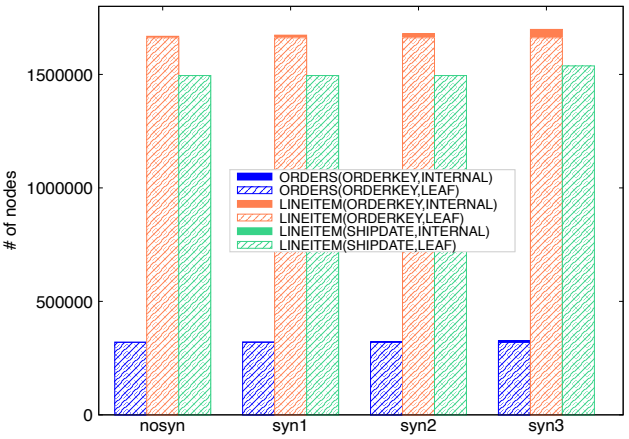


図 3: シノプシス埋込みによるストレージオーバーヘッド

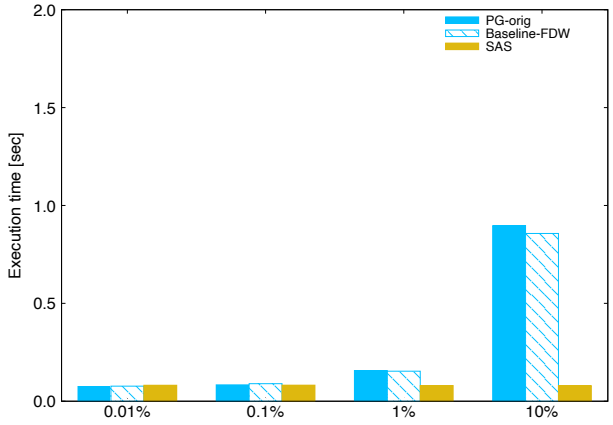


図 4: 異なる選択率に対する問合せ実行時間

14.4 を用い，2CPU コア，14GB メモリ，640GB ストレージの環境下で行なった．

4.2 実験結果

初めに，シノプシス埋込みによる，索引生成に要するオーバーヘッドを検証する．図 3 に，各索引およびシノプシスをもつデータ構造を保有することによるストレージオーバーヘッドを示す．[4] らの示した結果と同様であり，リーフノードが中間ノードに比べ圧倒的にサイズが大きいため，中間ノードはわずかに増加はしているものの，全体のノード数は，シノプシスによるストレージオーバーヘッドは **syn3** でも約 1.83% 程度であることを示している．

次に，PostgreSQL がもつ既存の B+Tree を用いた正解値を求める問合せ処理 (PG-orig) と，外部表 (FDW) に **nosyn** の場合の B+Tree を用いた正解値を検索する方法 (Baseline-FDW)，対して，B+Tree 索引に対して **syn2** のシノプシスを保持し，SAS を Approximate_FDW で実行する方式 (SAS) の実行性能を比較検証する．

まず，異なる選択率に対する問合せ処理の実行時間を比較する．図 4 にスケールファクタ 10 の ORDERS 表に対して 0.01%, 0.1%,1%,10% の選択率で ORDERKEY を取得し COUNT を計算する集計計算に対する実行時間を示す．PG-

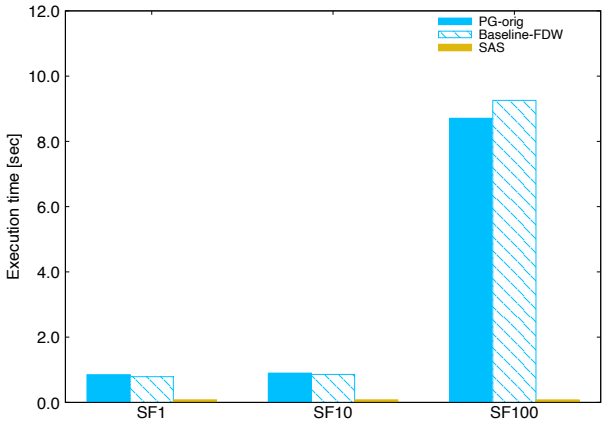


図 5: 異なるデータサイズに対する問合せ実行時間

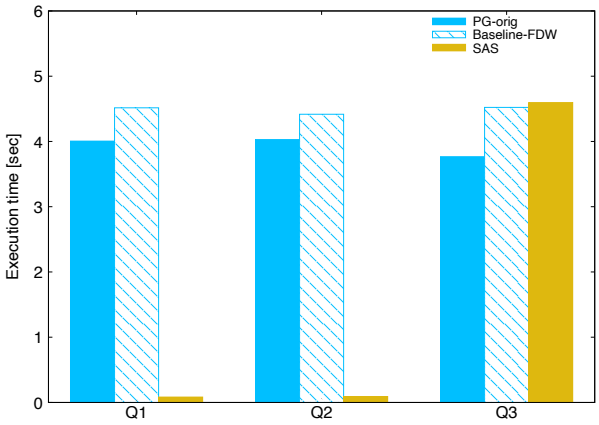
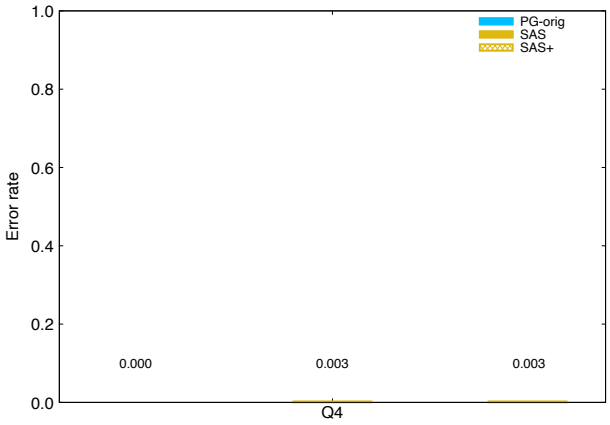
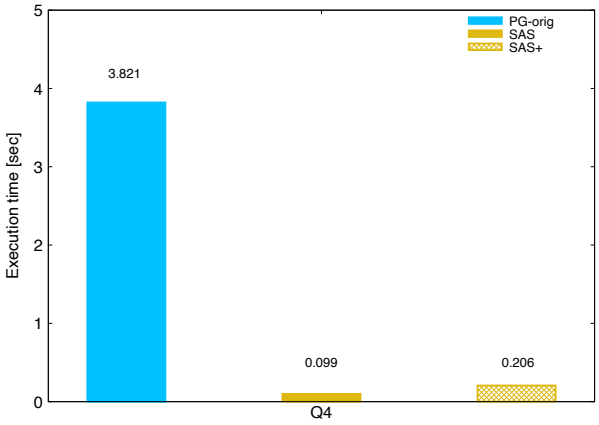


図 6: 異なるクエリにおける問合せ実行時間



(a) エラー率



(b) 実行時間

図 7: 複数制約による近似問合せ処理

orig と Baseline-FDW は選択率によらずほぼ同等性能を確認できており，FDW 上での実装が現実的であることが確認できる．また，選択率が大きくなるにつれ SAS の有効性が大きくなり，最大 91.2%PostgreSQL に比べ高速であることが言える．

続いて，異なるスケールファクタをもつデータセットに対する問合せ処理の実行時間を比較する．図 5 に，SF 1,10,10 の ORDERS 表に対して選択率 10% となる問合せを実行した時の実行時間である．選択率が変わらない場合，スケールファクタが大きい場合，PostgreSQL に比べ最大 99.1% 高速であることが確認できる．これは大きなデータサイズの方がリーフノードが多く，リーフノードの検索範囲が大きくなる為と考えられる．

次に，異なるクエリにおける近似問合せの有効性を検証する．図 6 に単一制約クエリ (Q1-Q3) における問合せ実行時間を比較する．**syn2** がシノプシスとして保持している為，Q1，Q2 では SAS が 97% 以上高速を達成している．一方，**syn2** は Q3 のクエリに対して十分なシノプシスを保持せず従来 B+Tree と同等の実行処理となっているのが確認できる．

さらに，複合制約クエリ（Q4）に対して，近似解のエラー率および実行時間を検証する．図 7(a) にエラー率，図 7(b) に実行時間を示す．PostgreSQL に比べ，エラー率は無視できる程度でありつつ，SAS では約 97.4%，属性間の相関を考慮した SAS+で 94.6% 高速であることが確認できる．本結果は，本実験で用いたデータセットが属性間にほぼ相関を持たず一様に分布している為であると考えられる．

5 関 連 文 献

大量データを収集・蓄積できるようになるに従い，蓄積した大量データから集計値を高速に計算する手法は長く研究されてきた．分析のための高速検索手法は OLAP [11] 問合せとして提案されてきた．代表的なものに索引があり，B+Tree [2,3] をはじめ検索キーを検索に効率的なデータ構造で保持することで高速検索を実現する．Data cube [12] は Group-by など集計に必要な処理をオペレータとして用意することによる処理の効率化を提案している．Materialized View [13] は過去の問合せ結果やその中間データを保存しておくという手法であり，類似する問合せを多く実行する分析系の問合せにおいて大幅な高速化が可能である．Materialized View は大規模データを対象とした検索時間を大幅に削減できるが，その効果の反面，大規模な Materialized View を保存するとストレージコストやメンテナンスコストが増加してしてしまうため，高速性とのバランス調整が求められる．

一方，正確な値ではなく近似値を求める方法として近似問合せ方式 [14,15] があり，それは 2 通りに大別される．1 つは蓄

積されたデータからサンプリングしたデータのみを用いて問合せ時に即座に集計値を計算・推定するオンライン処理型方式であり，もう 1 つは事前にデータベースに蓄えられたデータを用いて集計値など付加情報を計算し保存しておくことで，問合せ時にその値を用いて集計値を返答する事前処理型方式である．前者には，問合せ時にサンプリングしたデータを用いて近似値を推定し，問合せをインタラクティブに繰り返す毎にその近似解の精度を高めていく方式 [16] がある．この方式では事前処理が不要な為，事前準備の時間や事前に処理した結果を保持するストレージのオーバーヘッドが不要であるというメリットがあるが，問合せ時にサンプリングする処理が追加される為問合せ時間が増大するという問題がある．

後者には，事前にシノプシスと呼ばれる付加情報を計算し保持しておき，それらの情報を用いて問合せ時に近似値を高速に返す方式がある．代表的なものには，全体のデータセットを圧縮した情報を保有する wavelet [17,18] 方式がある．また，データセットの分布を示すヒストグラムを保持して近似解を推定する histogram 方式 [19] がある．この様な事前処理型方式は，オンライン処理型方式に比べ問合せ時の計算コストを抑え近似値の精度が高いというメリットがあるが，事前に付加情報を計算し保持しておくストレージオーバーヘッドが問題である．

上記の様なメリット・デメリットから，オンライン処理型方式と事前処理型方式を組み合わせた近似問合せ方式も提案されている．BlinkDB [20] はサンプリングを事前に実施することで問合せ時のサンプリングコストを軽減させながら，ワークロードを元に定期的に適切なサンプリングを実施する為，より正確な近似値を算出している．ストレージオーバーヘッドを抑制する為，問合せ処理をグルーピングし，保持するサンプルデータを低減させている．[21] らは，シノプシスとサンプリングを組み合わせて，インタラクティブな応答時間で集計値の近似値を返す AQP++を提案しており，[10] らは問合せに応じてどちらを使って近似値を求めるかを決定する方式で高い精度を達成している．この様な研究が実施されているが，特定のワークロードにおいて近似問合せの精度を重視する内容が中心であり，インタラクティブに多様な分析を実施したいユーザニーズに十分応えるものはこれまでの調査では見つかっていない．

6 おわりに

本論文では，多様な問合せに対応可能な近似問合せとして，シノプシスと呼ばれる付加情報を埋込んだ B+Tree 索引索引を利用した問合せ処理によって集合値の問合せに対する近似解を高速に検索する方式を既存の関係データベース管理システムである PostgreSQL に組込み，検証した．実験では，PostgreSQL インターフェースを介して集合値の問合せを実行し，シノプシスを埋込んだ索引によるストレージオーバーヘッドが無視できる程度であることと，従来の B+Tree を要する PostgreSQL の検索に比べ，シノプシスを埋め込んだ索引を用いた集合値の計算処理が最大 99.1% 高速である事を確認した．今後は，複雑な問合せに対する近似問合せ処理方式について検討を進める．

文　献

[1] 川村晃一．売上分析．計測と制御, Vol. 28, No. 1, pp. 17–22, 1989.

[2] Ramez Elmasri. Fundamentals of database systems seventh edition. 2021.

[3] David J Abel. A b+-tree structure for large quadtrees. *Computer Vision, Graphics, and Image Processing*, Vol. 27, No. 1, pp. 19–31, July 1984.

[4] Hiroki Yuasa, Kazuo Goda, and Masaru Kitsuregawa. Exploiting embedded synopsis for exact and approximate query processing. In *Database and Expert Systems Applications*, pp. 235–240. Springer International Publishing, 2022.

[5] Postgressql. `https://www.postgresql.org/`. 2024-01-10 参照.

[6] Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. Digital twin-driven product design, manufacturing and service with big data. *Int. J. Adv. Manuf. Technol.*, Vol. 94, No. 9-12, pp. 3563–3576, February 2018.

[7] Gerald Rebitzer, Tomas Ekvall, Rolf Frischknecht, Davis Hunkeler, Gregory Norris, Tomas Rydberg, W-P Schmidt, Sangwon Suh, B Pennington Weidema, and David W Pennington. Life cycle assessment part 1: framework, goal and scope definition, inventory analysis, and applications. *Environ. Int.*, Vol. 30, No. 5, pp. 701–720, July 2004.

[8] Postgressql insider. `https://www.postgresql.fastware.com/postgresql-insider-fdw-ove`. 2024-01-10 参照.

[9] Tpc-h. `https://www.tpc.org/tpch/`. 2024-01-09 参照.

[10] Xi Liang, Stavros Sintos, Zechao Shang, and Sanjay Krishnan. Combining aggregation and sampling (nearly) optimally for approximate query processing. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, SIGMOD '21, pp. 1129–1141, New York, NY, USA, June 2021. Association for Computing Machinery.

[11] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, Vol. 26, No. 1, pp. 65–74, March 1997.

[12] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Min. Knowl. Discov.*, Vol. 1, No. 1, pp. 29–53, March 1997.

[13] Imene Mami and Zohra Bellahsene. A survey of view selection methods. *SIGMOD Rec.*, Vol. 41, No. 1, pp. 20–29, April 2012.

[14] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, SIGMOD '17, pp. 511–519, New York, NY, USA, May 2017. Association for Computing Machinery.

[15] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? *Data Sci. Eng.*, Vol. 3, No. 4, pp. 379–397, December 2018.

[16] Joseph M Hellerstein, Ron Avnur, Andy Chou, Christian Hidber, Chris Olston, Vijayshankar Raman, Tali Roth, and Peter J Haas. Interactive data analysis: the control project. *Computer*, Vol. 32, No. 8, pp. 51–59, August 1999.

[17] Ioannis Mytilinis, Dimitrios Tsoumakos, and Nectarios Koziris. Distributed wavelet thresholding for maximum error metrics. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, SIGMOD '16, pp. 663–677, New York, NY, USA, June 2016. Association for Computing Machinery.

[18] Abdul Naser Sazish and Abbes Amira. An efficient archi-

tecture for HWT using sparse matrix factorisation and DA principles. In *APCCAS 2008 - 2008 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 1308–1311. ieeexplore.ieee.org, November 2008.

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

[20] 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 '13, pp. 29–42, New York, NY, USA, April 2013. Association for Computing Machinery.

[21] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. AQP++: Connecting approximate query processing with aggregate precomputation for interactive analytics. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, SIGMOD '18, pp. 1477–1492, New York, NY, USA, May 2018. Association for Computing Machinery.