分散インデックスへのシノプシス埋込みによる近似問合せ処理の検討

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　業務の改善・新たな知識発見に向けたデータ分析はクラウド環境で広く実施されている．集計計算が頻繁に実施され，正確値よりもレスポンスの早さを必要とされることがある．しかし，データ量の増加と分散に伴い，クラウド費用は大幅に増加している．本論文では，分散インデックス構造にシノプシスを埋込むことによって，一部ノードを非稼働に保ちながら付加情報を用いた近似解を高速に返す近似問合せ方式を提案し，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 [48]．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 [20]．代表的なものの一つには索引があり，中でも B^+-tree [9] 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．

近年，収集された大量のデータの管理・分析には，その管理の容易さからクラウドシステムが広く利用されている．クラウドシステムは，一般に数千台のコンピュータ，テラバイト級のデータ，数百万のユーザーから構成されており，SaaS，PaaS，または IaaS として提供されている [19]．各ユーザーはクラウドから必要に応じてリソースを割り当て，実際に使用した分に対し費用を支払う．そのため，ユーザーは，データ蓄積に必要なストレージサイズ，分析の計算量などクラウドサービスのコストを考慮し，ニーズに最適なソリューションを選択する必要があるが，データ量の増加，分散の増加に伴い，データ管理は複雑化し，コストは増大するという問題がある．

この様なクラウドコストが増大する問題に対し，我々は分散インデックス構造にシノプシスを埋込むことで，一部のデータベースサーバを非稼働のまま近似解を取得する近似問合せ方式を提案する．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という点に着目し，我々は既存の 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+) を提案してきた [42, 45]．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．これまで，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 [4] 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，検証してきた．この研究を拡張し，本論文では，分散インデックスにおいてもシノプシスを埋込むことで，クエリが与えられた時に，従来アクセスしなければいけない一部のデータベースサーバにアクセスせずともシノプシスを用いて近似解を返す方式を提案する．本方式により，アクセス不要なサーバを非稼働にすることが可能となり，これによりクラウドコストを削減可能である．

本論文の構成は，以下の通りである．第 2 章では，分散インデックスとその課題について言及し，第 3 章では，分散インデックスへの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する．第 4 章では，実装・評価を示す．第 5 章では，関連研究を提示し，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　データ解析基盤の現状

現代のビジネスや技術開発において，ビジネスプロセスの改善や新たな知見の発見のため，データ解析は競争力優位性の源泉となっている．生成 AI の実用化と普及に伴い，データ活用の重要性は新たなフェーズに突入し，IDC の予測（2024-2028）によれば、AI モデルの学習や推論に使用される非構造化データの急増により，世界のデータ生成量は従来の予測を上回る

ペースで加速し，2029 年には 527.47ZB にもなると予想している [18, 28]．これに伴い，企業のデータ解析基盤は，従来のオンプレミス環境から，GPU 等のスケーラブルなリソースを即座に調達可能なクラウドデータベース管理システム (DBMS) へとシフトし，クラウド DBMS はもはやデータ解析におけるデファクトスタンダードとして位置付けられている [24]．

しかし，クラウド環境への移行が進んだ 2026 年現在においても，リソース効率の課題は完全には解決されていない．多くの運用現場では，突発的なピーク負荷に耐えうるよう，定常的な需要を超えるリソースを事前に確保する傾向がある．特に近年，大規模言語モデル (LLM) の推論ワークロードに見られるような，極めてバースト性の高いトラフィックに対し，多くのシステムは依然としてピーク負荷を基準とした静的なリソース確保（Static Provisioning）を行っている．Microsoft Azure の大規模クラスタにおける VM ワークロードを分析した研究によると，割り当てられた VM の平均 CPU 利用率は極めて低く，多くの時間帯において 20% 40% に止まっていることが示されている [16]．これは，確保された計算リソースの大半が，実際には価値を生み出していないアイドル状態であることを示しており，物理的・経済的なリソース浪費が常態化している．

このようなリソースの非効率状態は，クラウドの従量課金モデルにおいて，致命的な経済的非効率性を生む．現在の多くのクラウドサービスでは，データ処理が実行されていないアイドル時間であっても，サーバが起動していたり，ストレージが確保されている限り課金が発生し続けるからである．Flexera のレポートによれば，組織のクラウドコストにおいて約 27% が無駄であり，コスト管理は組織の大きな課題となっている [21]．結果として，多くの組織は予算の制約から，全てのデータを解析することを諦め，データのサンプリングや解析頻度の削減を余儀なくされる．すなわち，技術的には解析可能であっても，コストの壁がデータの真の価値を引き出すことを阻害することになっているといえる．したがって，データドリブンな意思決定を実現するためには，必要な時に必要な分だけリソースを立ち上げ，アイドルコストを排除する新たな仕組みが求められている．

3　分散データベースの問題

過去約 10 年間で，大規模データ解析の標準的なアーキテクチャは，Hadoop や従来の MPP データベースに代表される「シェアードナッシング（Shared-Nothing）」モデルから，クラウドネイティブな「分離型（Disaggregated）」モデルへと劇的な進化を遂げた．このパラダイムシフトを決定づけたのは，Snowflake [5] や Google BigQuery [3] などの現代的なデータウェアハウスである．これらのシステムでは，データは Amazon S3 や Azure Blob Storage といった安価で無限のスケーラビリティを持つオブジェクトストレージに永続化され，クエリ処理を行うコンピュートノード（Virtual Warehouses 等）とは物理的に分離されている．さらに，Databricks が提唱する「レイクハウス（Lakehouse）」アーキテクチャの台頭により，ACID

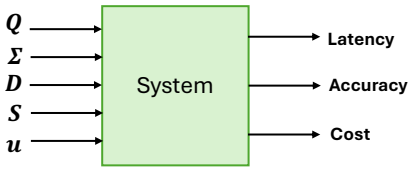

図 1: クエリ，データ，シノプシス，状態，を与えた時に値を回答するシステム

トランザクション特性を備えたオープンなテーブルフォーマット（Delta Lake 等）上で，多様な計算エンジンがデータを共有・処理することが可能となった [17]．これにより，ユーザはストレージ容量の制約から解放され，ペタバイト級のデータレイクを構築することは，技術的・コスト的に容易なものとなっている．

しかし，ストレージの課題が解決された一方で，計算リソース（Compute）の管理とコスト効率化は依然として未解決のままである．分離型アーキテクチャであっても，多くのシステムは高いクエリ性能を維持するために，計算クラスタを常時稼働させるか，あるいはウォームアップ済みのプールとして待機させておく必要がある．例えば，Snowflake の仮想ウェアハウスはクエリの有無に関わらず，稼働中は課金が継続するコストモデルを採用しており，不定期な分析ワークロードに対してはリソースの遊び（Idle time）が避けられない．また，AWS Lambda のような汎用的なサーバレス・ファンクション（FaaS）は，起動こそ高速であるものの，メモリ容量や実行時間の制限，および計算ノード間通信（Shuffle）の非効率性から，大規模な結合や集計を伴う OLAP ワークロードには不向きであることが指摘されている [26]．すなわち，現状大規データ解析において，そのデータ解析性能と完全なオンデマンド性を両立するソリューションが欠落しているといえる．

本研究では，このような問題に対し，データ問合せ処理において，ユーザのデータ解析要件に応じて，クラウドコストを効率化する仕組みを考える．上記問題を考えるため，本論文では，分散データ解析システムにおいて最も一般的かつ基本的な構成である Coordinator-Worker アーキテクチャを前提とする．このアーキテクチャは，クライアントからのリクエストを受け付け，クエリプランの作成とタスクの割り当てを行う単一のフロントノード（Coordinator）と，実際のデータ処理を並列実行する多数のバックエンドノード（Workers）から構成される．この構成は，Trino [41] や Apache Spark [46]，Google BigQuery [35] といった現代の主要な大規模インタラクティブ分析エンジンの多くで採用されている標準的なアーキテクチャである．

AWS など多くのクラウドサービスでは，サーバやストレージが稼働中に課金が発生するモデルのため，データ解析性能とコストはトレードオフの関係が成り立つ．ここで，データ解析性能を考える．従来，クラウド上での分散データ解析クラスタの構成は, アプリケーションが要求する厳格な性能要件を満たすように決定されてきた．具体的には，ユーザはワークロードの特性を事前に分析し，レイテンシー，メモリ容量，データ規

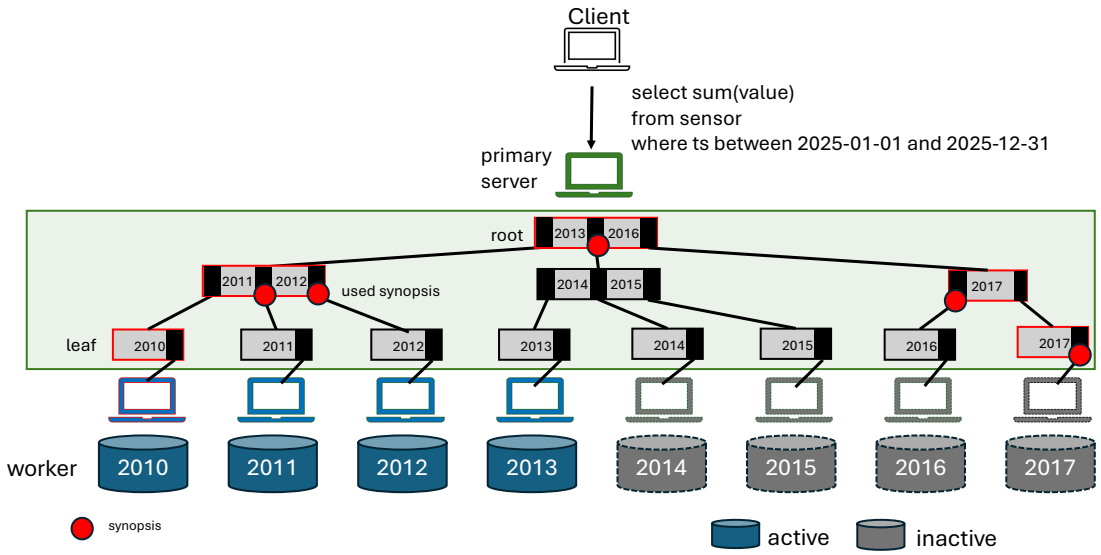

図 2: シノプシスを埋込み B$^+$-tree をもつ分散データベース例

模と I/O などの主要な指標についてインスタンスタイプとノード数を選択する．これらの要件は，正確値を前提とし，前述したとおり予期せぬ性能劣化を防ぐため，ピーク時の要件に合わせリソースを静的に確保するのが一般的である．

しかし，多くの組織において，正確値は必ずしも必要ではない．多くの企業では，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 [39]．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が，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，近似値で十分である．

厳密な値ではなく近似値を求めることを目的とした近似クエリ処理（Approximate Query Processing, AQP）は長年研究され，主にオンライン処理とオフライン処理の 2 つのカテゴリに分類される．オンライン処理は，クエリ実行時にサンプリング処理を行い，その値を用いて近似値を算出する方法であり，オフライン処理は，生データを要約した情報（シノプシス）を事前に計算・格納し，クエリ実行時に，シノプシスを利用して近似値を返す方法である [10, 31]．

ここで，入力されるクエリを $\boldsymbol{Q}$，データベースに存在する生データ $\boldsymbol{D}$，生データを要約した情報（シノプシス）を $\sum$，が与えられた時に，値を回答するシステムで考える（図 1）．このシステムを考えると，データ解析要件に応じたクラウドコストを効率化は，システムの実行に要するクラウドコスト (Cost)，レイテンシ (Latency)，問合せ回答精度 (Accuracy) のバランスを最適化するバックエンドノードの ON/OFF 状態を $\boldsymbol{S}$，生データかシノプシスのどちらを利用するかという選択を $\boldsymbol{u}$ を求めるという問題として捉えることができる．そして，求めた $\boldsymbol{S}$，$\boldsymbol{u}$ が与えられることによって，生データあるいはシノプシスを用いて回答を返すことクラウドコスト，レイテンシ，精度を最適化する実行処理を実現することができる．本論文では，クラウドコストは $\boldsymbol{S}$ と，レイテンシに応じてランニングコストは決定できることから，クラウドコストと問合せ回答精度のバランスを最適化する問題として捉える．

4 分散インデックスへのシノプシス埋込み

本研究では，第 3 章で提起した様に，システムの実行に要するクラウドコスト，レイテンシ，問合せ回答精度のバランスを最適化する $\boldsymbol{S}$，$\boldsymbol{u}$ を求め，データを返す方法を提案する．

基本的なアイデアは，シノプシスを既存索引に埋込むことによる高速かつ高精度な問合せ処理 [42] を分散インデックスに拡張し，フロントノードがシノプシスをもち，バックエンドノードに生データがある場合，シノプシスで十分な場合はシノプシスで近似解を応答し，バックエンドが稼働しており精度が必要であれば生データを参照する，という方法である．これにより，バックエンドノードが非稼働の場合でもシノプシスによって近似解を得，バックエンドノードが非稼働になることでクラウドコストを削減することができると考える．

B$^+$-tree による分散データベースの例を用いて概要を説明する．図 2 は，フロントノード 1 台とバックエンドノードが 8 台あり，50%のバックエンドノードは非稼働である．センサデータで時系列属性 ts をもつ sensor テーブルが存在し，各バックエンドノードにはテーブルを分割したデータを保有するデータベースを保有する．sensor データがあるとする．フロントノードには，常時稼働しており，シノプシスを埋込んだインデックスをもち，そのリーフノードは各バックエンドのデータポインタを持つ．中間ノードにはシノプシス（赤いマーカー）を保有する．バックエンドノードは，稼働・非稼働をとるとする．sensor のもつ時刻カラム ts への範囲検索クエリが与えられた時には，フロントノードはインデックスのルートから探索し，もしクエリに含まれるシノプシスが存在すればシノプシスを利

Algorithm 1 Cost-aware priority activation

Require: Q:Query; C:maximum number of ON nodes; Σ_i:synopsis for $i = 1, \ldots, N$
Ensure: S_i and u_i for all i

for each node i in N **do**
  compute the estimated hists $\hat{h}_i$ from Σ_i for all i
end for
Sort nodes by $\hat{h}_i$ descending
$n \leftarrow 0$: initialize the number of node with $s_i = 1$
for each node i in sorted order **do**
  if $n \leqq C$ **then**
    $S_i \leftarrow 1$
    $u_i \leftarrow 1$
    $n \leftarrow n + 1$
    continue
  else
    $S_i \leftarrow 0$
    $u_i \leftarrow 0$
  end if
end for

用し解を算出することができる．また，バックエンドノードが稼働かつユーザが精度を求めれば，生データにアクセスすることで厳密解を得る．これより，従来は非稼働なデータからは直接データを取得することができないが，シノプシスを活稼働状態のフロントノードにおいておき，利用することで近似解を得ることができると考える．

次に，$\boldsymbol{S}$，$\boldsymbol{u}$ の決定方法を考えると，ユーザの要件によって少なくとも次の 2 通りの方針を考えることができる．

- 精度優先: クラウドコストを制約とし，精度を最大化する方針
- クラウドコスト優先: 精度を制約とし，コストを最小化する方針

本論文では精度優先の場合での $\boldsymbol{S}$，$\boldsymbol{u}$ 決定方法について述べる．クラウドコストはバックエンドノード数の稼働状況によってきまるとして，ここでは稼働できるバックエンドノード数を制約条件としてユーザが与える場合を考える．アルゴリズム 1 に示すように，予算が許される限り，クエリ精度に対して影響が大きいバックエンドノードを優先的に稼働させ，稼働しているバックエンドノードからは厳密解を取得し，非稼働であればシノプシスによる近似解を得る．これにより，精度が最大化すると考える．クエリ精度に対して影響の大きさには，様々方法はあるが，単純には各バックエンドノードごとのクエリヒット件数とする．

最後に，$\boldsymbol{Q}$，$\boldsymbol{D}$，$\sum$，$\boldsymbol{S}$，$\boldsymbol{u}$ を用いて近似解をアルゴリズム 2 を用いて取得する．DSAS では，プライマリノードにおいて root から深さ探索を実施し，リーフノードにあるバックエンドノードへのデータポインタを辿り，与えられた $\boldsymbol{S}$，$\boldsymbol{u}$ に応じて，データをシノプシスあるいは各バックエンドノードへアクセスし取得することで，正確値あるいは近似解を算出することができる．

Algorithm 2 Distributed synopsis-alloyed search (DSAS)

Require: n_r: a reference to the root node
Ensure: R: result buffer

$R = \emptyset$
DSAS(n_r)
function DSAS(n)
  $N \leftarrow$Fetch n
  if N is an internal node **then**
    for each $e \in N$ **do**
      if e satisfies $\boldsymbol{Q}$'s constraint **then**
        if e contains synopsis $\sum_i$ sufficient for a disjoint part of query **then**
          $R \leftarrow R \cup \sum_i$
        else DSAS (e's child reference)
        end if
      end if
    end for
  else
    if $S_i = 1$ **then**
      if $u_i = 1$ **then**
        $R \leftarrow R \cup D_i$
      else
        $R \leftarrow R \cup \sum_i$
      end if
    else
      $R \leftarrow R \cup \sum_i$
    end if
  end if
end function

5 PostgreSQL を用いた分散データベース

第 3 章で述べた通り，本研究では，分散データ解析システムにおいて基本的な Coordinator-Worker アーキテクチャを前提として分散データベースにおけるクラウドコスト，レイテンシ，精度の最適化を考える．この様な分散データベースのアーキテクチャとして，フロントノードを 1 台，バックエンドノードを複数台用意し，各ノードには広く利用されているデータベースである Postgresql [4] を稼働させ，外部データラッパ (FDW) を用いてフロントノードとバックエンドノードが通信することとした．図 3 に示す様に，検証としてバックエンドノードは 4 台用意し，各バックエンドノードのデータに対するシノプシス，すなわち，シノプシスを埋込みした索引の中間ノードをフロントノードが保有する．クエリがフロントノードに与えられた時に，プランニング，実行を FDW 内で実行することができ，シノプシス埋込み索引を参照，あるいは，バックエンドノードへ参照を可能とした．

6 実　　験

本章では，第 4 章で示したシノプシスを埋込んだ分散インデックスの効果を検証する．

本実験では，分散データベースとして，フロントノード 1 台，

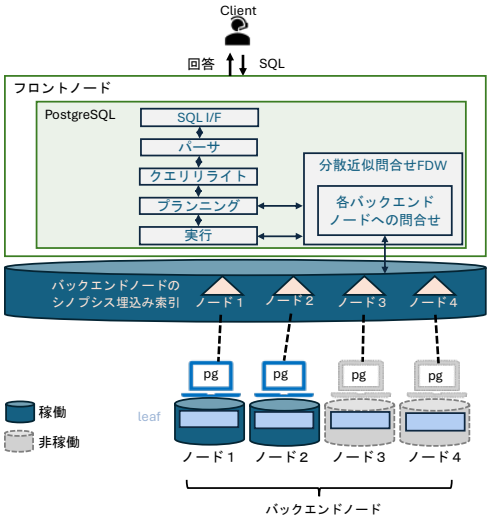

図 3: FDW を用いたアーキテクチャ

バックエンドノードは 4 台とし，各サーバには既存データベース PostgreSQL [4] を用いる．データセットとして本実験では，ベンチマークセットである TPC-H [6] の dbgen を用いて生成したデータセットを利用する．ORDERKEY を軸に LINEITEM 表と ORDERS 表をバックエンドノード数で等分しバックエンドノードに配置した．

次に，問合せ処理に用いる索引として，B^+-tree [9,20] で以下の値を索引キーとした．

- LINEITEM (SHIPDATE, ORDERDATE): LINEITEM 表 L_SHIPDATE と LINEITEM,ORDERS 表の ORDERKEY を索引キーとする索引

リーフノードには LINEITEM 表と ORDERS 表を結合した値を保有した．ノードサイズは全て 8192Byte とした．

初期評価として，TPC-H のスケールファクタ (SF) は 1 を用い，クエリには L_SHIPDATE のフィルタ条件のみをもつよう簡略化した Q6-1, TPC-H の Q6 を Q6-2 として用いた．上記のような設定において，従来のシノプシスを用いない分散インデックスを持つ場合と比べ，バックエンドノードを一部落とした時のシノプシス埋込み索引を用いた DSAS による誤差，レイテンシ，ランニングコストを，バックエンドノードを 25%-75% だけを稼働させた場合と，全ノード稼働させた場合で比較検証した．

図 4 に，誤差の結果を示す．100%すべてのバックエンドノードを用いた時の厳密解に対し，Q6-1 では，誤差が全て 1% 未満と非常に小さく，Q6-2 ではクエリの条件が増えたことで誤差が増加はしているがバックエンドノードを 50% 稼働させた状況においても誤差は 2% 未満に収まっており，データ解析への影響は無視できるほどであることが確認できる．

次に，図 5 に，レイテンシの結果を示す．非稼働なノードにアクセスする必要がある場合，シノプシスを用いて推定となるため，その分バックエンドノードを参照する必要がなく，バックエンドノードを 50% 稼働させた時，Q6-1 では 0.33 倍，Q6-2 では 0.54 倍高速化できていることが確認できる．

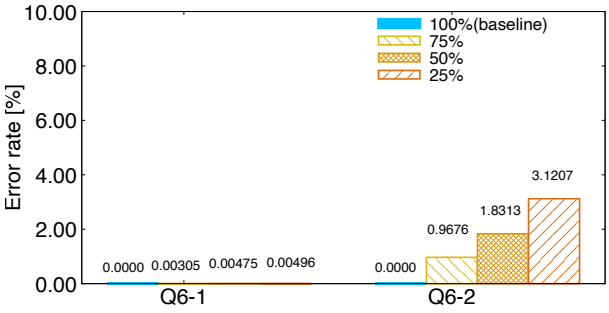

図 4: 誤差

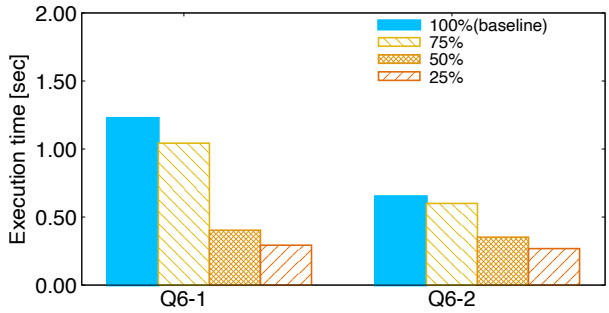

図 5: レイテンシ

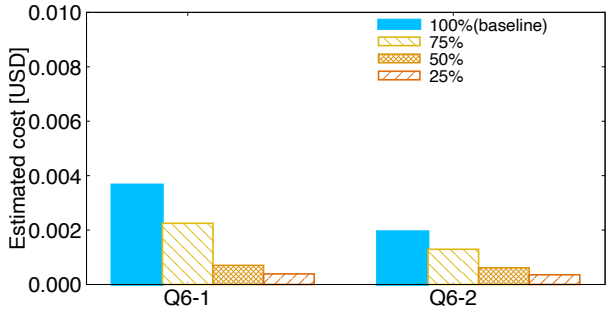

図 6: ランニングコスト

最後に，図 6 に，ランニングコストの推定結果を示す．コスト制約を増加，即ち，バックエンドの非稼働ノードを増えすにつれ，クエリ実行時のランニングコストは削減できる点と，シノプシスを用いることによる高速化により，バックエンドノードを 50% 稼働させた時では，Q6-1 では 80.8% 倍，Q6-2 では 68.4% 倍コストを削減できていることが確認できる

以上より，分散データベースにおいて，あるクエリが与えられた時に，提案方式が誤差，レイテンシ，クラウドコストのバランスを取りユーザに適した環境を提供できることが確認できた．

7 関 連 文 献

膨大な量の管理にむけ，スケーラビリティ，スループット，信頼性の確保する分散データベースが開発されてきた．大量データを小さなブロックに分割し複数ノードに分散し管理する Hadoop distributed File system (HDFS) は耐久性が高く，それを利用した Apache HBase がクラウド環境では広く利用されている [43,44]．要求されたクエリに対して複数のデータベー

スから適切なデータの検索を効率的に実現するため，分散インデックスが開発された [30]．例えば Google のようなウェブ検索などでは，文書ごと，あるいは単語ごとにデータを分割し，異なるサーバで管理することで，検索効率をあげている [13]．データ量の増加に伴い，データベースサイズ，分散数が増えることで，クラウド環境ではクラウドコストが増加するという分散データベースの問題に対し，近年では，機械学習を取り込んだインデックス設計が広がっており，分散データベースにおいても，データの更新対応や空間データへの拡張，データベースサイズ削減，メモリサイズ削減の研究が提案されている [37] ．FITting-Tree では，従来の B^+-tree [9, 20] の様に全値を格納するのではなく，各リーフノードはページの傾き，始点キー，およびデータページへのポインタを格納し，最大許容誤差を満たすようおおよそのキーの位置を推定する様設計されており，インデックスサイズ削減を実施している [23]．しかし，ユーザの許容できる予算や分析要件から分散システムを最適化する研究は十分に実施されてはいない．

一方，厳密な値ではなく近似値を求めることを目的とした近似クエリ処理（Approximate Query Processing, AQP）は，主にオンライン処理とオフライン処理の 2 つのカテゴリに分類される．オフライン処理は，保存されたデータからサンプリングされたデータを用いて，クエリ時に集計結果を計算・推定するものである [8,29,47]．問合せ時にサンプリングデータを用いて近似解を推定し，インタラクティブなクエリの繰り返しを通じて近似解の精度を向上させる技術も提案されている [8]．オンライン処理は，事前処理や事前計算値を保持するためのストレージオーバーヘッドを必要としないという利点を持つが，クエリ時に追加のサンプリング処理が必要となるため，クエリ処理時間が増加する可能性がある [25, 27, 32, 34]．一方，オフライン処理では，統計情報や要約情報などを事前に計算・格納し，クエリ時にその事前計算された情報を利用して近似解を返す方法である [10,31]．このアプローチでは，シノプシス（要約情報や追加情報）を事前に保持し，それを利用して高速に近似解を返すことが可能である．例えば，ウェーブレットアプローチとして，圧縮された要約情報を保持して検索空間を絞り込む手法が研究されている [36, 40]．また，ヒストグラムをシノプシスとして保持し，データセットの分布を示す方法も存在する [15]．このような事前処理アプローチは，クエリ時の計算コストを削減し，オンライン処理よりも高い近似精度を提供する可能性を持つが，事前計算された追加情報を保持するためのストレージ容量などオーバーヘッドを伴うという課題がある．

オンラインおよびオフライン手法の利点と欠点を考慮し，BlinkDB [12] は，ワークロード履歴に基づくオンラインサンプル選択戦略と多次元層別サンプリングを採用することで，他の既存 AQP 手法と比較し，大量データに対する大規模並列エンジンの有効性を実証し，高精度な近似解答を取得している．[33] による提案手法は，クエリに応じて AQP にオンライン処理とサンプリングデータのいずれを使用するかを決定することで高い精度を達成した．. 分散データの連合を考慮した SAQE [14] も開発されている．しかし，近似問合せを活用することで，クラウドコストを削減する研究はみつけられていない．

最近の研究では，Spark SQL [2] や Presto/Trino [7] などの分散環境やクラウドネイティブ環境における AQP も探求されており，そこでは，インタラクティブな分析に対して，概要生成およびサンプリング技術が大規模に適用されている．Google BigQuery [3]，Amazon Redshift Spectrum [1]，Snowflake [5] などの商用システムは，大規模データセットを効率的に処理するために近似クエリ機能を組み込んでいる．Aqua [11]，IDEA [22]，VerdictDB [38] などのミドルウェアアプローチは，事前計算された要約や層別サンプリングを活用するためにクエリを書き換えることで，これらの分散エンジンを拡張している．これらのシステムは，大規模環境における応答性と正確性を強調している一方で，分散環境における AQP に関する体系的な研究は依然として比較的限られており，ほとんどのソリューションは分散実行フレームワークまたは追加のミドルウェア層のいずれかを前提としている．対して，我々の研究は，分散データベースにおいて，PostgreSQL などの広く採用されているリレーショナルデータベースシステムへの直接的かつ既存データベースシステムの大幅な改善なく，シノプシスを利用することで不要なノードを非稼働にし，クラウドコストを削減することに焦点を当てている．

8 おわりに

本論文では，クラウド環境における分散データベースにおいて，インデックスにシノプシスを埋込むことで，一部バックエンドノードを非稼働に保ちながらシノプシスを用いて近似解を応答する方式を提案し，初期検証を実施した．100% バックエンドノードを稼働させた比べ，50% のバックエンドノードだけを稼働した場合でも，誤差は 2% 未満，レイテンシは $0.33 - 0.54$ 倍，クラウドのランニングコストは $68.4 - 80.8\%$ 削減となることを確認した．今後は，大規模データや複雑なクエリを用いた評価を進める．

謝　辞

本研究の一部は，内閣府戦略的イノベーション創造プログラム（SIP）「統合型ヘルスケアシステムの構築」JPJ012425 の助成を受けたものである．

文　献

[1] Amazon Redshift Spectrum. https://docs.aws.amazon.com/redshift/. Accessed on Nov. 19, 2025.
[2] Apache Spark. https://spark.apache.org/docs/latest/api/python/index.html. Accessed on Nov. 19, 2025.
[3] Google BigQuery. https://docs.cloud.google.com/bigquery/docs. Accessed on Nov. 19, 2025.
[4] Postgressql. https://www.postgresql.org/. 2024-01-10 参照.
[5] Snowflake. https://docs.snowflake.com/. Accessed on Nov. 19, 2025.
[6] Tpc-h. https://www.tpc.org/tpch/. 2024-01-09 参照.
[7] Trino. https://trino.io/. Accessed on Nov. 19, 2025.
[8] Interactive data analysis: The control project. *Computer*,

Vol. 32, No. 8, pp. 51–59, 1999.

[9] David J Abel. A b+-tree structure for large quadtrees. *ICVGIP*, Vol. 27, No. 1, pp. 19–31, July 1984.

[10] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *Proc. SIGMOD*, pp. 574–576, 1999.

[11] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *Proc. SIGMOD*, pp. 275–286, 1999.

[12] 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 *Proc. EuroSys*, pp. 29–42, 2013.

[13] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, Vol. 23, No. 2, pp. 22–28, 2003.

[14] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. SAQE: practical privacy-preserving approximate query processing for data federations. *Proc. VLDB Endow.*, Vol. 13, No. 11, pp. 2691–2705, 2020.

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

[16] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proc. SOSP*, pp. 153–167, 2017.

[17] databricks. Data lakehouse architecture. Accessed on Feb. 7, 2026.

[18] Nedim Dedic and Clare Stanier. Towards differentiating business intelligence, big data, data analytics and knowledge discovery. In *ERP Future 2016 - Research*, Vol. 285 of *LNBIP*, pp. 114–122, 2016.

[19] Marios D. Dikaiakos, Dimitrios Katsaros, Pankaj Mehra, George Pallis, and Athena Vakali. Cloud computing: Distributed internet computing for IT and scientific research. *IEEE Internet Comput.*, Vol. 13, No. 5, pp. 10–13, 2009.

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

[21] Flexera. State of the cloud report 2025, 2025.

[22] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Revisiting reuse for approximate query processing. *Proc. VLDB Endow.*, Vol. 10, No. 10, pp. 1142–1153, 2017.

[23] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. Fiting-tree: A data-aware index structure. pp. 1189–1206, 2019.

[24] Gartner. Market Share: Database Management Systems, Worldwide, 2024. Gartner Press Release, 2025. Accessed on Feb. 7, 2026.

[25] Rong Gu, Han Li, Haipeng Dai, Wenjie Huang, Jie Xue, Meng Li, Jiaqi Zheng, Haoran Cai, Yihua Huang, and Guihai Chen. Shadowaqp: Efficient approximate group-by and join query via attribute-oriented sample size allocation and data generation. *Proc. VLDB Endow.*, Vol. 16, No. 13, pp. 4216–4229, 2023.

[26] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back. In *Proc. CIDR*, 2019.

[27] Aaron Hurst, Daniel E. Lucani, and Qi Zhang. Pairwisehist: Fast, accurate, and space-efficient approximate query processing with data compression. *Proc. VLDB Endow.*, Vol. 17, No. 6, pp. 1432–1445, 2024.

[28] ICIS. Insight: Reshaping china, us power architecture amid ai shift, 2026. Accessed on Feb. 7, 2026.

[29] Chris Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the dbo engine. *TODS*, Vol. 33, No. 4, pp. 1–54, 2008.

[30] Theodore Johnson, Padmashree Krishna, and Adrian Colbrook. Distributed indices for accessing distributed data. In *IEEE Mass Storage Systems*, pp. 199–207, 1993.

[31] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In *Proc. ICDE*, pp. 472–483, 2014.

[32] Rundong Li, Pinghui Wang, Jiongli Zhu, Junzhou Zhao, Jia Di, Xiaofei Yang, and Kai Ye. Building fast and compact sketches for approximately multi-set multi-membership querying. In *Proc. SIGMOD*, pp. 1077–1089, 2021.

[33] Xi Liang, Stavros Sintos, Zechao Shang, and Sanjay Krishnan. Combining aggregation and sampling (nearly) optimally for approximate query processing. In *Proc. SIGMOD*, pp. 1129–1141, 2021.

[34] Qingzhi Ma and Peter Triantafillou. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proc. SIGMOD*, pp. 1553–1570, 2019.

[35] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Commun. ACM*, Vol. 54, No. 6, pp. 114–123, 2011.

[36] Ioannis Mytilinis, Dimitrios Tsoumakos, and Nectarios Koziris. Distributed wavelet thresholding for maximum error metrics. In *Proc. SIGMOD*, pp. 663–677. Association for Computing Machinery, 2016.

[37] Andreas Papadopoulos and Dimitrios Katsaros. A-tree: Distributed indexing of multidimensional data for cloud computing environments. pp. 407–414. IEEE Computer Society, 2011.

[38] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. Verdictdb: Universalizing approximate query processing. In *Proc. SIGMOD*, pp. 1461–1476, 2018.

[39] 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.

[40] Abdul Naser Sazish and Abbes Amira. An efficient architecture for HWT using sparse matrix factorisation and DA principles. In *Proc. APCCAS*, pp. 1308–1311, 2008.

[41] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. Presto: SQL on everything. In *Proc. ICDE*, pp. 1802–1813. IEEE, 2019.

[42] Mika Takata, Hiroki Yuasa, and Kazuo Goda. Synopsis-alloyed index for exact and approximate query processing. *IEEE Access*, Vol. 14, pp. 6797–6806, 2025.

[43] Le Hong Van and Atsuhiro Takasu. An efficient distributed index for geospatial databases. Vol. 9261 of *Lecture Notes in Computer Science*, pp. 28–42. Springer, 2015.

[44] Deepak Vohra. Apache hbase. In *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*, pp. 233–257. Springer, 2016.

[45] Hiroki Yuasa, Kazuo Goda, and Masaru Kitsuregawa. Exploiting embedded synopsis for exact and approximate query processing. In *Proc. DEXA*, pp. 235–240, 2022.

[46] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. USENIX*, pp. 15–28, 2012.

[47] Kai Zeng, Sameer Agarwal, Ankur Dave, Michael Armbrust,

and Ion Stoica. G-ola: Generalized on-line aggregation for interactive analysis on big data. In *Proc. SIGMOD*, pp. 913–918, 2015.

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