分散環境における多次元データに対する集約演算結果の推定とその評価

張　涵† 渡　佑也† 欅　惇志†,†† 宮崎　純† 中村　匡秀†††

† 東京工業大学 情報理工学院情報工学系　〒 152–8552 東京都目黒区大岡山 2-12-1
†† 国立研究開発法人科学技術振興機構，ACT-I　〒 332–0012 埼玉県川口市本町 4-1-8
††† 神戸大学大学院 システム情報学研究科　〒 657–8501 兵庫県神戸市灘区六甲台町 1-1
E-mail: †{zhang,watari,keyaki}@lsc.cs.titech.ac.jp, ††miyazaki@cs.titech.ac.jp, †††masa-n@cs.kobe-u.ac.jp

あらまし　本研究では，分散キーバリューストア (KVS) 上の集約演算結果の推定について，ヒストグラムにカーネル密度推定を導入した手法を提案し，範囲クエリの結果の効率化と推定精度の向上を目指す．分散 KVS 上での大規模多次元データに対する集約演算は，データの全スキャンが多発し非効率的である．また，このような大規模なデータベースにおける集約演算は，正確な値ではなく概数で良い場合が多い．そこで我々は，ヒストグラムとカーネル密度推定を用いて，データの集約演算結果を推定し，データの全スキャンを回避することで集約演算処理の効率化する手法を提案した．本稿では，提案手法にサーバー間のデータ転送量を考慮した改良を行ったものを分散 KVS 上に実装し，検証を行う．

キーワード　集約演算，範囲クエリ，分散キーバリューストア，データ要約，ヒストグラム，カーネル密度推定

1. はじめに

近年，インターネット・スマートフォン等の普及により，ユーザーの位置情報や利用ログといった，ビッグデータと呼ばれる大規模データが収集されるようになった．また同時にビジネス等では，このようなビッグデータを分析することにより，価値ある情報として有効活用することが不可欠になりつつある．

そのような分析の一例として集約演算を使用するものが存在し，大規模データに対する集約演算機能を提供するデータベースとして，スケールアウトが容易な分散 KVS [3] が注目されている．しかしながら多くの場合，分散 KVS は単純なインデックスしか持たない，もしくはインデックスが存在しない．そのため，大規模多次元データにアクセスする際にはデータの全スキャンが頻繁に発生し非効率的である．

分散 KVS における，大規模多次元データに対しての集約演算を効率的に処理する手法として，渡ら [13] がリレーショナルデータベース (RDB) [11] と分散 KVS を相互に利用する手法を提案をした．渡らの提案手法では，データ空間をグリッドに分割し，グリッドごとの集約演算結果を事前に計算・保持する．これによって，グリッドがクエリ範囲に完全に内包される場合には，対象グリッド内のデータの全スキャンを行うことなく集約演算結果を参照することで処理の効率化を達成した．しかし，範囲クエリに完全に内包されないグリッドについては，依然としてグリッド内のデータの全スキャンを行っている．これは，範囲クエリに完全に内包されない部分に位置するグリッドが多い場合や，グリッドあたりのデータ数が多い場合は，スキャンするデータ数が大量になることがある．

これに対し我々は，大規模なデータベースにおける集約演算では正確な値ではなく概数で良い場合が多いことを踏まえ，渡らの提案手法にヒストグラムとカーネル密度推定を用いたデータ要約を導入し，集約演算処理時にはその結果の推定値を計算する手法 [12] を提案した．この手法は，グリッドからヒストグラムを構築し，ヒストグラムの各バケットに対してカーネル密度推定を利用して得た統計情報を保持し，クエリの処理にはこの統計情報のみを用いるものである．我々は，この手法を用いてクエリ処理時のデータの全スキャンを完全に回避することで，高いクエリスループットが実現できることを示し，また推定値と真値の平均誤差が 5% 以下の推定精度を達成した．

本稿では，我々の手法 [12] を分散 KVS として HBase を用いて実装したほか，従来の HBase のデータ取得の実装に変更を加えたものを提案し，その性能について検証を行う．

2. 基礎知識

2.1 ヒストグラム

データ分布は属性値 (AttributeValue) と対応する度数 (Frequency) で構成される．ヒストグラムは階級幅 (属性値の幅) とその階級幅内に対応する度数の総和の組（この組をバケットと呼ぶ）によって構成され，データ要約・クエリ結果の推定の手法として広く使用されている方法である [1]．データ分布がどのように分割されるかによって，構築されるヒストグラムが異なる．

以下，代表的なヒストグラムへの分割法について説明する．

（1） 用語定義

図 1 と図 2 はそれぞれ 1 次元のデータ分布とヒストグラムの例である．

- v_i：データ分布での i 番目の属性値
- f_i：データ分布での i 番目の属性値に対応する度数
- s_i：データ分布での i 番目の属性値の幅 $(v_{i+1} - v_i)$
- a_i：データ分布での i 番目の属性値の面積 $(f_i * s_i)$
- V_i：i 番目バケットの定義域の最小値
- F_i：i 番目バケット内の度数の総和
- S_i：i 番目バケットの幅 $(V_{i+1} - V_i)$

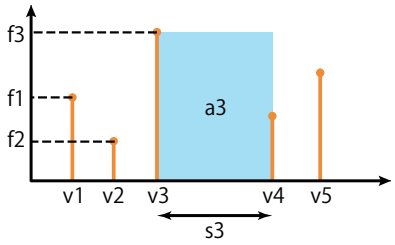

図 1　データ分布の例

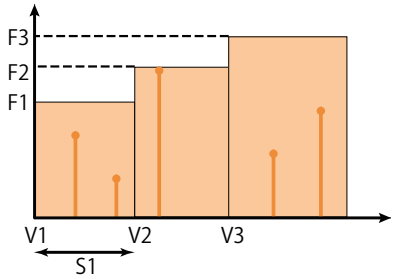

図 2　ヒストグラムの例

（2）　Equi-width Histogram

最も利用されることの多い，代表的なヒストグラムである．データ分布を β 個のバケットに分割する際，データ分布の定義域 (属性値の幅) を等幅に β-1 分割を行う．($S_{i+1} = S_i$)

（3）　Equi-depth Histogram [5]

可能な限り，分割後の各バケットの度数の総和が等しくなるように分割を行う．(可能な限り $F_{i+1} = F_i$)

（4）　V-optimal Histogram [2]

データ分布を β 個のバケットに分割する際，i 番目バケットに含まれる各属性値に対応する度数の分散を R_i とするとき，

$$\sum_{i=1}^{\beta} F_i * R_i$$

が最小となるように分割を行う．

（5）　MaxDiff Histogram [6]

データ分布を β 個のバケットに分割する際，隣接する属性値間の面積差 $(a_{i+1} - a_i)$ を計算し，上位 β-1 件の属性値間で分割を行う．

（6）　Compressed Histogram [6]

データ分布を β 個のバケットに分割する際，データ分布の度数の全総和を $SumF$ とするとき，$f_i > SumF/\beta$ を満たす属性値を単一のバケットとし，満たなさいものについては Equi-depth Histogram と同様の処理をするような分割を行う．

Poosala ら [6] の各ヒストグラムを用いた範囲クエリ結果の推定の評価実験によると，MaxDiff Histogram がもっとも精度の良い推定となることが報告されている．

また，多次元データ分布をヒストグラムへと分割する方法の一つとして MHIST [7] と呼ばれる手法がある．MHIST 分割アルゴリズムを以下に示す．

Step1

データ分布について各次元軸に対してそれぞれ周辺分布を計算し，これをもとに最も分割の必要性の高い次元軸を決定する．最も分割の必要性の高い次元軸とは，いずれの分割手法を採用するかによって判定基準が異なる．V-optimal であれば分散が最も大きい周辺分布を持つ次元軸，MaxDiff であれば周辺分布にて隣接面積差が最も大きいものを持つ次元軸，Equi-width・Equi-depth・Compressed であれば周辺分布の総和が最も大きい次元軸が選択されることになる．

Step2

Step1 で選択した次元軸を基準にデータ分布を p 分割する．

Step3

2 回目以降は，複数のバケットが存在しているので，バケットごとにデータ分布の各次元軸について周辺分布を計算し，全バケットで最も分割を行う必要性の高い次元軸を決定する．

Step4

Step3 で選択した次元軸が属するバケットを p 分割する．

Step5

分割上限バケット数に達するまで Step3 と Step4 を繰り返す．

分割方法と分割数 p によっては最終的なヒストグラムが大きく異なり，範囲のクエリ結果の推定の精度にも影響を与える．Poosala ら [7] の評価実験では，分割方法を MaxDiff Histogram，分割数を p=2 と設定すると最も良い推定精度となることが報告されている．

ここで最も良い推定精度となる，MHIST (MaxDiff Histogram, p=2) によるデータ分布の分割ついてを具体的に説明する．図 3 は 2 次元のデータ分布を五つのバケットに分割する過程を表している．各データは次元軸 A1 と次元軸 A2 の 2 軸の属性を持ち，縦軸及び横軸の値は次元軸 A1 の属性値と次元軸 A2 の属性値を表す．また，図中の点は各属性値に属するデータ群であり，データの横の数字はデータ数を表す．

まず，各次元軸に対して属性値ごとに周辺分布を計算する (図 3 の Step 1)．その後各次元軸にて隣接する属性値間の面積 (属性値の幅 * 属性値に対応する周辺分布) の差を計算する (図 3 の Step 2)．今回の例の場合，属性値は連続する整数値なので，面積差は単に隣接する周辺分布間の差となる．計算の結果，もっとも差が大きいのは次元軸 A2 の属性値 3 と 4 の間であるので，ここで 1 回目の分割が行われる．2 回目以降は，各バケットの各次元軸に対して同様の計算を行い，分割箇所を決定する (図 3 の Step 3)．この計算をバケット数が分割上限数に達するまで繰り返し行う．

今回の例の場合，最終的には図 3 の Step 4 のようになる．

2.2　カーネル密度推定

カーネル密度推定 [8] とは，有限の標本点から全体の分布を推定するノンパラメトリックな推定手法の一つである．特定の範囲に属するデータを集計するヒストグラムとは異なり，カーネル密度推定は各データ点を中心とした分布 (これをカーネル関数 K と呼び，ガウス関数が採用されることが多い.) を想定し，この重ね合わせをデータ集合の分布とするものである．

標本 $x_1, \ldots, x_n$ が与えられているとき，点 x におけるカーネル密度推定量 $f(x)$ は以下のように定義される．

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

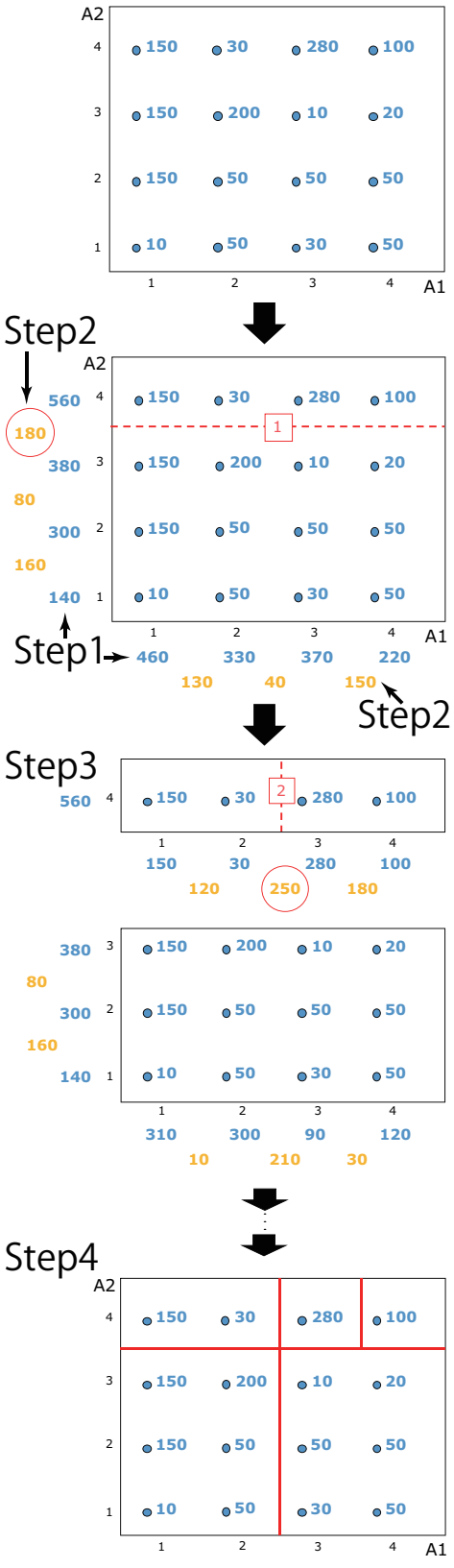

図 3　MHIST による分割の様子

このとき，$h > 0$ はバンド幅と呼ばれ，各標本がどれだけの範囲に対して影響を及ぼすかを指定する値である．

同じデータでも階級の境界の設定によって見た目が大きく変化するヒストグラムに対し，カーネル密度推定では階級の境界に依存せずに分布を捉えることが可能であり，分布の複峰性などの特徴がわかりやすい．

ただし欠点としては，標本データを全て保持しておく必要がありメモリ効率が悪いことや，新しいデータ点が追加された際に再度全ての標本点について密度計算を行う必要があることがあげられる．

3. 先 行 研 究

3.1 部分集約法

部分集約法 [10] とは，データベースを複数のブロックに分割してブロックごとに集約演算結果を事前計算した上で，集約演算のクエリを処理する際には事前計算結果を可能な限り再利用することで，データベースへのアクセスを削減して処理の効率化を図る手法である．

例として，年齢 (age) と身長 ($tall$) からなるリレーション B に対して，$age < 15$ を満たすレコードの $tall$ の総和を求める集約演算を考える．このとき，B は三つのブロック B_1, B_2, B_3 に分割されており，各ブロックについて総和が事前に計算されているとする．また、各ブロックについて以下の情報が与えられているとする．

- ブロック B_1 : age は全て 15 未満
- ブロック B_2 : age は全て 15 以上
- ブロック B_3 : age は 15 未満と 15 以上が混在

これより，目的の集約演算を処理する際には，データの全スキャンは ブロック B_3 に対してのみ行えばよく，ブロック B_1, B_2 のデータをスキャンする必要はない．

このように部分集約法は，部分集約演算結果を事前に求めておくことで，実データへのアクセスを省略して集約演算を効率化する．

3.2 部分集約法を利用したクエリ効率化

渡らは，部分集約法を利用した大規模多次元データに対する範囲クエリを効率化する手法 [13] を提案した．渡らの手法では，まず分割後の空間（これをグリッドと呼ぶ）1 つあたりに入るデータ数（これをグリッドサイズと呼ぶ）を指定して，データ空間を複数のグリッドに分割する．その後，各グリッドについて事前に計算した集約演算結果を保存し，クエリ処理時には集約演算結果を再利用することで処理の効率化を達成している．またこのとき，データサイズは小さいがクエリ処理時には複雑な検索の対象となるグリッドの分割情報はインデックスを用いることができる RDB に，各グリッドについての集約演算結果はスケールアウトを実現しやすい KVS に保存している．

範囲クエリが与えられた際は，以下のアルゴリズムによって処理が行われる．

Step1

範囲クエリと共通部分を持つグリッドを RDB のテーブルから列挙する．このとき同時に，グリッドが範囲クエリに完全に包含されるかどうかも問い合わせる．

Step2

Step1 で列挙されたグリッドのうち，範囲クエリに完全に包含されるグリッドについては，KVS から部分集約結果を取得し足し合わせる．

Step3

Step1 で列挙されたグリッドのうち，範囲クエリと一部が交わるグリッド（これを周辺グリッドと呼ぶ）に含まれるデータをすべてスキャンし，範囲クエリに含まれるデータについて集約演算を行う．

Step4

Step2 および Step3 で得られた集約演算結果を統合して最終的なクエリ結果を得る．

渡らの実験では，約 1,000 万件のデータへの 4 次元の範囲クエリに対して，提案手法はグリッド一つあたりに入るデータ数を適切に設定することで，HBase(KVS の一種) を単体で用いた場合に比べ 5.4 – 36.3 倍，PostgreSQL(RDB の一種) に対しては 2.9 – 13.8 倍のクエリスループットを実現している．

渡らの提案手法の問題点として，グリッドサイズを大きく設定すると，範囲クエリに完全に内包されるグリッドの割合が減少し，結果として実データのスキャンが比較的減少しない．また，グリッドサイズを小さく設定すると，範囲クエリに完全に内包されるグリッドの割合が増加し，実データののスキャン量は減少するもの，グリッドの数が増加するためにクエリスループットは必ずしも高くならないということがあげられる．

3.3 データ要約を利用した集約演算結果の推定

我々の手法 [12] では，渡らの手法 [13] における各グリッドに対して，ヒストグラムの作成 (バケット分割) 及び，ヒストグラムの作成とカーネル密度推定を組み合わせた方法を用いてデータ要約を行う．この結果（これを統計情報と呼ぶ）を保持しておき，そして，クエリ処理時には統計情報を利用して集約演算結果の推定を行うことで，周辺グリッドについても全スキャンを省略することによって更なる処理の効率化を目指した．

以下では，事前処理であるデータ要約の手順と，統計情報を用いて集約演算結果を推定する手順について説明する．

3.3.1 データ要約

事前処理であるデータ要約の手順を図 4 を用いて説明する．

Step1

渡らの提案手法を用いてデータ空間を複数のグリッドに分割し，部分集約演算の結果を保持する．図 4 の Step 1 では 5 回のグリッド分割の結果，6 個のグリッドに分割される．

Step2

各グリッド内のデータ分布を MHIST (MaxDiff Histogram [6]，p=2) [7] を用いて複数バケットに分割する．図 4 の Step 2 は，グリッド 010 のバケット分割の例である．

Step3

さらにカーネル密度推定を用いる場合には，構築した各バケットについてバケット内のデータ分布をもとにカーネル密度推定を行い，その結果を保持する．(図 4 の Step 3)

3.3.2 クエリ処理

クエリが与えられたときに，統計情報を用いて集約演算結果を推定する手順を説明する．

範囲クエリが与えられたとき，クエリに完全に内包されているグリッドについては事前に計算済みの部分集約演算を用いる．一方，完全に内包されていない周辺グリッドについては，事前処理でのカーネル密度推定によって得られた統計情報を使用し，集約演算結果を推定する．

図 5 にクエリ処理の具体例を示す．グリッド 00111 とグリッド 00110 はクエリに完全に内包されるため，事前計算された部分集約演算結果を用いる．グリッド 000 の一部の領域については，統計情報を用いて集約演算結果を推定する．これら 3 個のグリッドの部分集約演算結果を足し合わせ，最終的なクエリ結果を得る．

この際の統計情報を用いて集約演算結果を推定する手法について，我々は以下の 3 種を提案した．

（1） 手法 0: MHIST

集約演算の推定の基本的な手法として，データ分布から作成したヒストグラムを用いる方法 [4] がある．グリッドに 2.1 節にて説明した MHIST (MaxDiff Histogram, p=2) による分割を適用した後，範囲クエリが与えられた際は作成されたヒストグラムを用いて，範囲クエリを満たす部分の集約演算結果の推定を行う．

推定値の算出方法としては，Uniform Scheme [4] を用いた．これは，あるバケット B が範囲クエリ Q と交わるような位置関係にある場合，B 内に関して Q を満たすデータの総和 $Sum(B,Q)$ を以下の式で算出する．

$$Sum(B,Q) = F_B \cdot \frac{Size(B \cap Q)}{Size(B)}$$

ただし，F_Bは B 内の度数の総和 とする．

この手法を以降,「MHIST」と呼ぶこととする．

（2） 手法 1: MHIST + カーネル密度推定

Uniform Scheme による推定では，分割後のバケット内のデータ分布を一様分布と捉え，バケット全体に対してバケットとクエリが交わる部分の割合を用いて計算を行ってきた．しかしながら，大規模な多次元データに対しバケット内のデータ分布を一様分布として扱うことによって推定精度が悪くなる場合がある．そこで，カーネル密度推定を利用した事前計算から得た統計情報を用いることとする．より詳細にデータ分布が把握できるため，従来の一様分布を用いる手法と比較して，より精度の高い推定が行える．

MHIST によるバケット分割後，各バケット内のデータ分布にカーネル密度推定を用いて，具体的なデータ点の属性値の代わりに統計情報を保持する．クエリ処理時には，統計情報を用いてクエリ結果を推定することで，データの全スキャンを回避する．グリッドと同様，範囲クエリに対して分割後のバケットは必ずしもクエリに完全に内包されるわけではない．提案手法では，バケットの各次元軸の属性値の範囲を n 等分 し，n^d(d はデータ分布の次元数) 個 の点における統計情報から算出した推定値を用いることで，部分的にクエリと交わるバケットに関する集約演算結果の推定値を求める．

以上を踏まえ，各グリッド内のデータ分布を MHIST (MaxDiff Histogram, p=2) によって複数バケットに分割し，すべてのバケットについてカーネル密度推定を行う．クエリ処理が与え

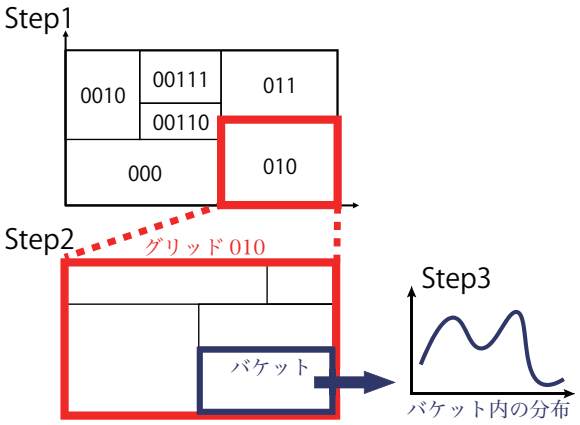

図 4　各グリッドの集約演算結果の事前計算の手順

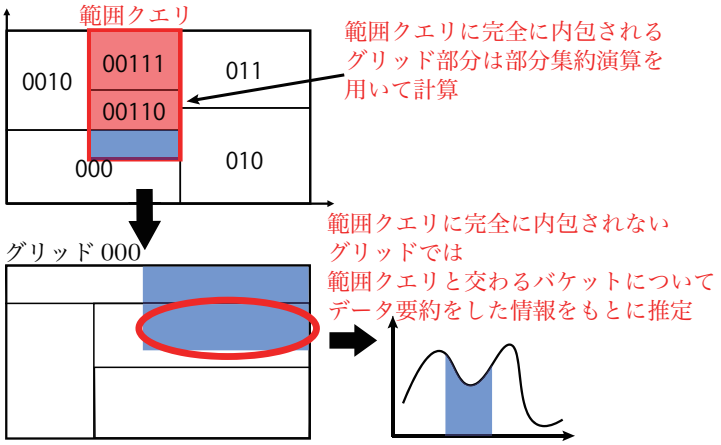

図 5　範囲クエリ結果推定の手順

られた際は，事前計算した推定点における推定値を用いて，クエリ結果を推定する．

この手法を以降,「MHIST + カーネル密度推定」と呼ぶこととする．

（3） 手法 2: MHIST + 部分的カーネル密度推定

手法 1 はすべてのバケットについてカーネル密度推定を行うのに対し，カーネル密度推定を行うバケットを限定する手法を手法 2 とする．

MHIST(MaxDiff Histogram, p=2) による分割の結果，バケットによってはある次元軸の属性値の幅がない (その次元の属性値の最小値と最大値が一致する) 場合がある．このようなバケットを，次元が落ちたバケットと呼ぶことにする．分割の結果次元が落ちたバケットについては，バケット内のデータ分布を一様分布として捉えても差し支えないと仮定し，カーネル密度推定を行なわない．

次元が落ちたバケットが範囲クエリを満たす場合は，Uniform Scheme を用いて推定値を算出する．次元が落ちのないバケットについては，手法 1 と同様にカーネル密度推定を行い，推定では事前計算した推定点における推定値を用いる．これによって，部分的にカーネル密度推定を行わず統計情報のデータサイズを削減することで，推定精度の悪化を軽減しながら，手法 1 と比較して高いクエリスループットが実現できる．

この手法を以降,「MHIST + 部分的カーネル密度推定」と呼ぶこととする．

1 台のマシンでの上記 3 種の推定手法に関する評価実験 [12] の結果，表 1 に示す通り，クエリ結果の推定精度については MHIST にカーネル密度推定を組み合わせることで，平均誤差 5% 以下を達成した．また，範囲クエリに完全に包含されるグリッドについては事前計算された部分集約演算結果を利用して正確な値を取得し，周辺グリッドに該当する部分に関してのみ統計情報を用いて集約演算結果を推定することで，従来のヒストグラムを用いた集約演算結果の推定手法よりも高い精度で推定を行えることを示した．しかしながら、分散環境におけるクエリスループットは明らかではなかったため、本稿では、二つの手法を提案し、クエリスループットの評価を行う．

手法	最大値	平均値	標準偏差
MHIST	0.731	0.250	0.152
MHIST + カーネル密度推定	0.229	0.038	0.039
MHIST + 部分的カーネル密度推定	0.260	0.046	0.045

表 1　精度の比較

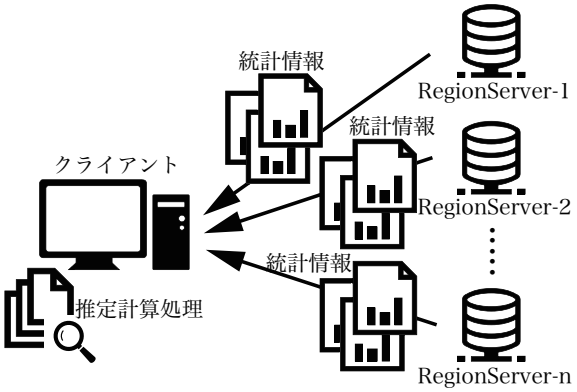

図 6　統計情報データの流れ（naïve）

4. 提 案 手 法

4.1 提案手法 1: naïve

3.3 節にて述べた我々の手法を元に，分散 KVS として HBase を用いて実装した．集約演算の対象とするデータ及び，各グリッドの統計情報を複数の Region Server と呼ばれるサーバーに分散して保存する．

周辺グリッドに該当するグリッドにおける推定値を統計情報から得る流れを図 6 に示す．クライアントは発行したクエリに対し，周辺グリッドに該当するグリッドの統計情報を対象のデータが存在する Region Server から取得し，クライアントにて各統計情報から推定計算処理を行い，これらを足し合わせることで，最終的な推定値を得る．

この手法を以降,「naïve」と呼ぶこととする．

4.2 提案手法 2: サーバー内処理

naïve では，統計情報のデータサイズが大きいとき，分散環境ではクライアント・サーバー間のデータ転送コストが増加し，クライアント側からみたクエリスループットが低下する場合が

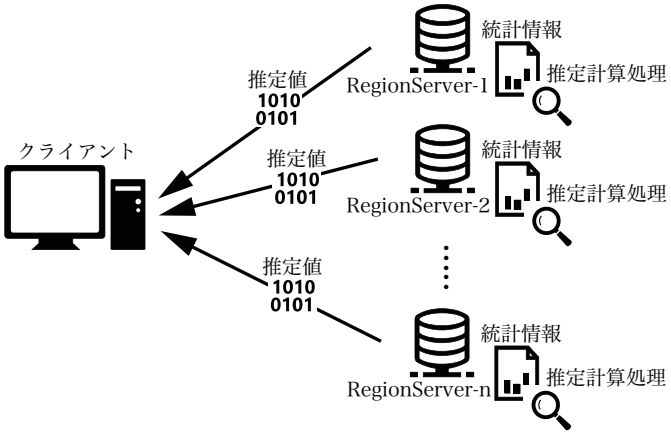

図 7　統計情報データの流れ（サーバー内処理）

あると考えられる．

これに対し，クライアントで行っていた推定計算処理を Region Server 側で行うよう HBase 内部のクエリ処理実装に変更を加えたものを提案手法 2 とする．

図 7 に提案手法 2 における，統計情報データの流れを示す．Region Server からクライアントへ転送されるのは，各統計情報から計算された推定値のみとし，複数の Region Server から転送された推定値をクライアントにて足し合わせることで，クエリ結果に対する最終的な推定値を得ることになる．

これにより，クライアント・サーバー間のデータ転送コストを低下させることが期待できる．ただし欠点として，naïve の手法とは異なり、本手法は HBase の内部実装を変更する必要がある．

この手法を以降,「サーバー内処理」と呼ぶこととする．

5. 評価実験

提案手法 2 種の性能を評価するため，渡らの手法と比較する．

5.1 実験環境

16 台のマシンから構成されるクラスタで評価実験を行った．16 台すべてに HBase をインストールし，このうち 13 台を Region Server とした．また，PostgreSQL は 8 台のマシンにインストールし，マルチスタンバイ構成のレプリケーションを構築した．用いた PostgreSQL のバージョンは 9.6.1，HBase のバージョンは 1.2.0 である．

各マシンの構成を次に示す．

OS: CentOS 6.7

CPU: Intel Core i7-3770 (3.4 GHz，4 コア/ 8 スレッド)

メモリ: 32GB

ストレージ: HDD 2TB

5.2 実験で使用したデータ

範囲クエリを実行するデータとして，室内の気象センサーより得られた時刻・気温・湿度・照度・風速の五つの属性からなる 10,446,198 件 (約 1,000 万件) データを用いた．プログラム上では，五つの属性をすべて 64 ビットの整数に変換して扱う．時刻は始点からの経過秒数に，それ以外の四つの属性は値を 1000 倍することで整数に変換した．

5.3 実験内容

風速の平均を求めるクエリを実行することを想定し，そのうちの風速の総和を求める範囲クエリを 50 個実行し，3.3.2 節にて述べた 3 種の推定手法及び，渡らの手法を用いたときのクエリスループットを測定する．

クエリは，最大 16 台のマシンを用いて，各マシンあたり 1 から 8 スレッドまで変化させて発行し，最大 128 スレッドによる問い合わせを行った．加えて，1 スレッドでの実行時には，周辺グリッド処理時の 1 クエリあたりの平均処理時間と，1 グリッドを処理するためのあたり Region Server からクライアントへの平均データ転送量を測定した．

用いた範囲クエリは，風速以外の四つの属性に関する 4 次元の範囲クエリである．また，バケット分割の上限数は 25 とした．カーネル密度推定を行う手法では，分割後はバケットの各次元軸の属性値の範囲を 5 等分し，5^d個 の点における推定値を保持した．この理由としては，分割上限数を 25 とした場合バケット一つあたりに含まれるデータ数は 5^d個 を超えることはなく，5^d個 の点についての推定値を把握しておけば，十分に分布を捉えることが可能であると考えられるためである．

5.4 実験結果

まず，各手法におけるクエリスループットと 1 クエリあたりの平均処理時間を図 8 に，1 クエリあたりの Region Server からクライアントへの平均データ転送量を表 2 に示す．

図 8 より，全スキャンと比較して naïve の手法ではクエリスループットが悪化し，サーバー内処理の手法では 1.77 – 2.03 倍の高速化となった．これに加えて，表 2 の平均データ転送量の比較より，統計情報のデータサイズは十分大きく，クエリスループットの低下を引き起こしていると言える．

これに対して，サーバー内処理の手法ではクライアントで行っていた推定計算処理を Region Server 側で行うよう変更したことで，統計情報の転送コストを削減し naïve の手法の問題を解決している．また，サーバー内処理の手法では 3 種の推定手法間で大きなクエリスループットの差がないことから，分散環境においてクライアント・サーバー間のデータの転送コストは，クエリスループットを低下させる大きな要因であることがわかる．

次に，図 9 に実行スレッド数を変化させたときのクエリスループットの変化を示す．

図 9 より，1 スレッドで実行したときでは，3 種の推定手法間で大きなスループットの差がなかったサーバー内処理の手法において、実行スレッド数が増加するにしたがって，差が開いていることがわかる．これは，Region Server 内での 3 種の推定手法間の推定計算処理の負荷の大きさの違いによるものであると考えられ，実行スレッド数が多いほど，違いが顕著に現れている．

6. まとめ

本稿では，ヒストグラムとカーネル密度推定を用いたデータ要約を活用してクエリ結果を推定することで，クエリスループットを高速化する我々の手法 [12] を，分散 KVS として HBase

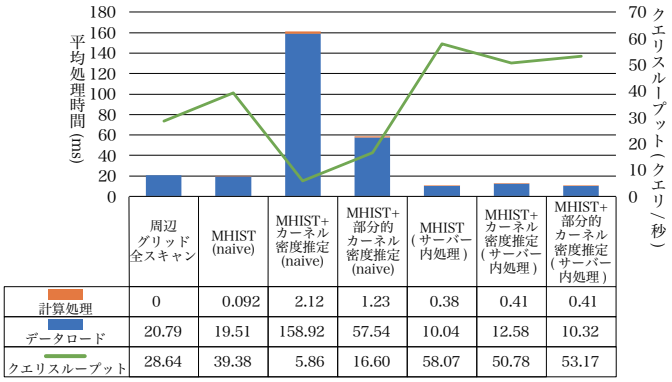

図 8　1 スレッドで実行時のクエリスループットと平均処理時間

手法	byte/グリッド
周辺グリッド全スキャン (渡らの手法)	18
MHIST(naïve)	1800（固定値）
MHIST+カーネル密度推定 (naïve)	82800（固定値）
MHIST+部分的カーネル密度推定 (naïve)	26990
MHIST(サーバー内処理)	8（固定値）
MHIST+カーネル密度推定 (サーバー内処理)	8（固定値）
MHIST+部分的カーネル密度推定 (サーバー内処理)	8（固定値）

表 2　平均データ転送量の比較

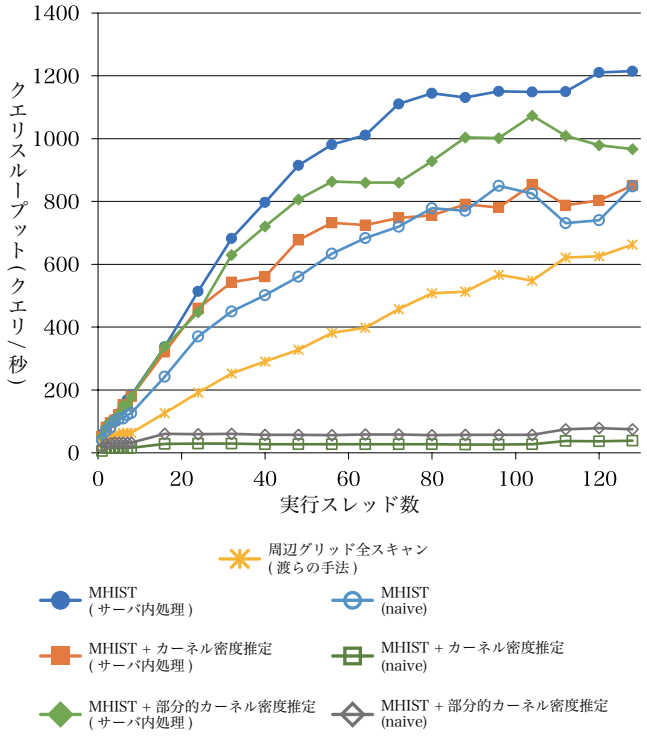

図 9　マルチスレッドで実行時のクエリスループット

を用いて実装した．さらに naïve の手法に加えて，クライアントで行っていた統計情報から推定値を得る計算処理を Region Server 上で実装し，Region Server からクライアントに転送されるのは得られた推定値のみとする手法を提案した．これによってクライアント・サーバー間のデータ転送コストを削減することで，分散環境のおいてクエリスループットが低下する問題を解決した．

評価実験として，最大 128 スレッドにて，各スレッドで 4 次元の範囲クエリを 50 個実行し，推定手法 3 種のクエリスループット，及び周辺グリッド処理時の 1 クエリあたりの平均処理時間・1 グリッドを処理するため Region Server からクライアントへの平均データ転送量を測定した．その結果，サーバー内処理の手法は全スキャンと比較して高速化を実現し，分散環境において高いクエリスループットを達成することを示した．また，クエリ結果の推定精度については MHIST にカーネル密度推定を組み合わせることで，平均誤差 5% 以下となることを確認した．

今後の課題として次のようなものが挙げられる．本稿の評価実験において Region Server として用いたマシンは，性能の良い CPU を搭載しているが，実際のシステムでは，ストレージへのデータアクセスと通信のみを目的として性能の低い CPU を搭載するマシンが Region Server として用いることが考えられる．これによって，必ずしもサーバー内処理の手法が高いクエリスループットを実現できるとは言えず，これに関する評価実験を行う必要がある．また，我々の手法は，推定の誤差が小さくしつつクエリスループットを向上させることを目標としているが，3.3 節の表 1 にて示した推定誤差は，あくまで特定のデータに対する評価実験の結果の一つであり，結果の数値に対して理論的な裏付けを伴わない．今後さらに推定誤差を低下させていくためには，バケット分割の上限数やバケットの各次元軸に関する分割数等のパラメータと，推定誤差との関係性を調査し，理論的な裏付けを考察する必要があることも課題である．

謝　　辞

本研究の一部は，JSPS 科研費 (JP15H02701，JP16H02908，JP15K20990，JP17K12684)，JST ACT-I の助成を受けたものである．ここに記して謝意を表す．

文　　献

[1] Yannis Ioannidis. The history of histograms (abridged). *VLDB '03 Proceedings of the 29th international conference on Very large data bases*, Vol. 29, pp. 19–30, 2003.

[2] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. *VLDB '98 Proceedings of the 24rd International Conference on Very Large Data Bases*, pp. 275–286, 1998.

[3] Avinash Lakshman and Prashant Malik. Cassandra - a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, pp. 35–40, 2010.

[4] M. Muralikrishna and David DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pp. 28–36, 1988.

[5] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. *SIGMOD '84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 256–276, 1984.

[6] Viswanath Poosala, Peter Haas, Yannis Ioannidis, and Eugene Shekita. Improved histograms for selectivity estimation of range predicates. *SIGMOD '96 Proceedings of the*

1996 ACM SIGMOD international conference on Management of data, pp. 294–305, 1996.

[7] Viswanath Poosala and Yannis Ioannidis. Selectivity estimation without the attribute value independence assumption. *VLDB '97 Proceedings of the 23rd International Conference on Very Large Data Bases*, pp. 486–495, 1997.

[8] Bernard. W. Silverman. *Density Estimation for Statistics and Data Analysis*. No. 26 in Monographs on Statistics and Applied Probability. CRC Press, 1986.

[9] 小山田昌史, 中台慎二. クエリへ適応的に構築される木構造によるデータ集約処理の高速化. DEIM 2016 論文集, D3-5, 2016.

[10] 小山田昌史, 陳テイ, 成田和世, 荒木拓也. Pa-proxy: Sql-on-hadoop におけるデータ集計処理を精度の劣化なく高速化するフレームワーク. DEIM 2015 論文集, E5-6, 2015.

[11] 増永良文. リレーショナルデータベース入門 [新訂版]. サイエ ンス社, 2003.

[12] 張涵, 渡佑也, 欅惇志, 宮崎純. ヒストグラムとカーネル密度推定を組み合わせた集約演算結果の推定. DEIM 2017 論文集, E5-1, 2017.

[13] 渡佑也, 欅惇志, 宮崎純. Rdb と kvs を相互に利用した多次元データに対する集約演算の効率化. DEIM 2016 論文集, D5-5, 2016.