# 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

張　涵† 渡　佑也†† 欅　惇志†† 宮崎　純††

† 東京工業大学工学部情報工学科　〒 152–8552 東京都目黒区大岡山 2-12-1
†† 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
E-mail: †{zhang,watari,keyaki}@lsc.cs.titech.ac.jp, ††miyazaki@cs.titech.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．そこで本研究では，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を図る．

**キーワード**　集約演算結果の推定, 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 [12] 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．

本稿では，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，集約演算結果を推定し，データの全スキャンを完全に回避することで集約演算処理を効率化する手法の提案を行う．

## 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]．データ分布を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 $(V_{i+1} - V_i)$

（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 $\beta$ 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 $\beta$-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 $\beta$ 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，

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

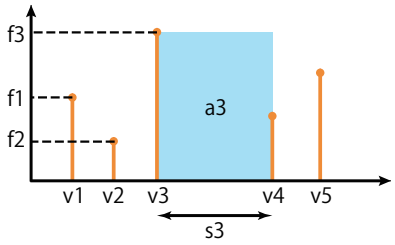



図 1　データ分布の例

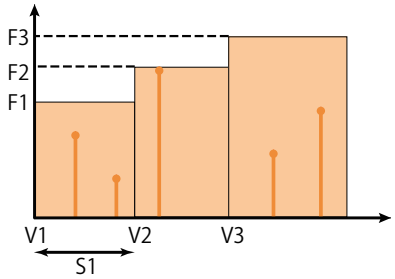



図 2　ヒストグラムの例

が最小となるように分割を行う．

（5）　MaxDiff Histogram [6]

データ分布を $\beta$ 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 $\beta$-1 件の属性値間で分割を行う．

（6）　Compressed Histogram [6]

データ分布を $\beta$ 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．Fig. 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．

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

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．

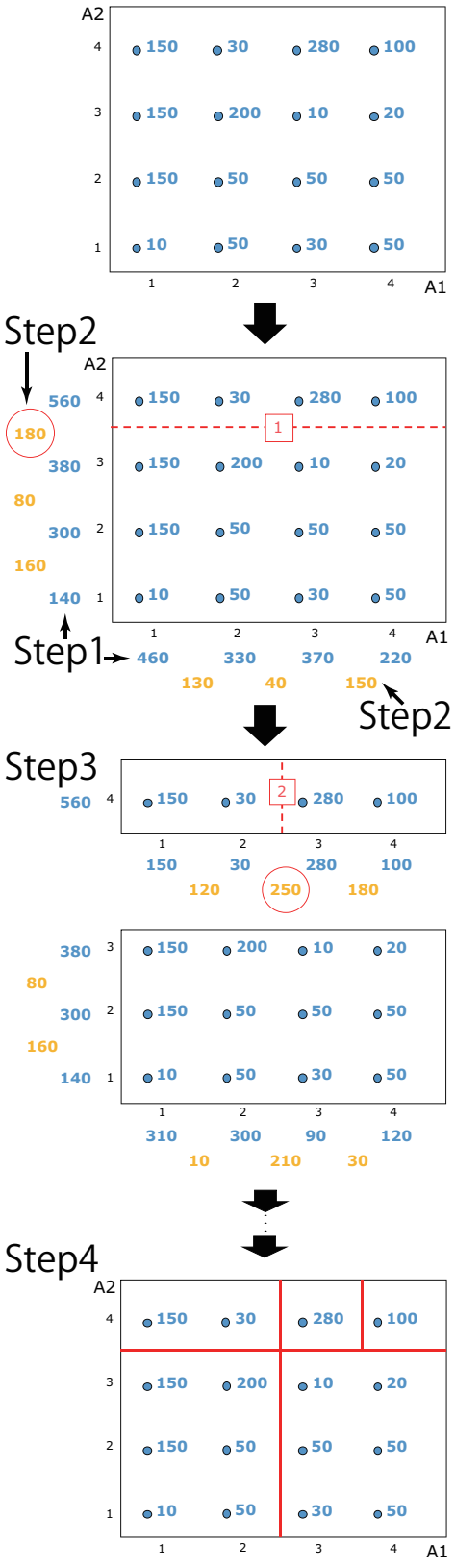



図 3　MHIST による分割の様子

## 3. 関 連 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．

小山田ら [9] はデータベースを複数のブロックに分割する際，ユーザが頻繁にアクセスするデータ領域については細かく，あまりアクセスされないデータ領域については粗く，ブロックを作成して集約値を計算し保持する手法を提案した．これによって，ユーザのアクセスするデータ領域に偏りがあった場合には，集約値の再利用性が高くなり効率的な集約演算処理を達成した．

### 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 [12] 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．

## 4. 提 案 手 法

### 4.1 概　　要

渡らの手法 [12] では，範囲クエリにおけるクエリに完全に内包されない周辺グリッドについては，周辺グリッド内を全スキャンしクエリを満たすデータを集計しなければならない．本研究では，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い，クエリ処理時にはデータ要約の結果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図る．

まず，提案手法における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)

次に，クエリが与えられたときに，データ要約の結果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，データ要約の結果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．

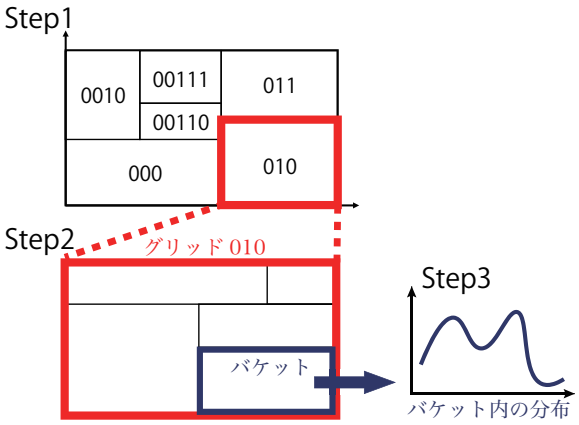



図 4　各グリッドの集約演算結果の事前計算の手順

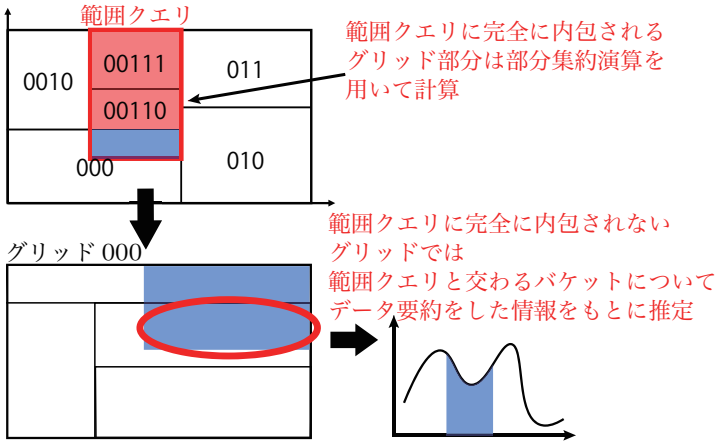



図 5　範囲クエリ結果推定の手順

### 4.2 提案手法 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手法を提案手法 0 とする．

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．

### 4.3 提案手法 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と考えられる．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．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と考えられる．

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手法を提案手法 1 とする．クエリ処理が与えられた際は，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．

### 4.4 提案手法 2: MHIST + 部分的カーネル密度推定

提案手法 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提案手法 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，提案手法 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．

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

### 4.5 データ構造

本節では，各グリッドに対し，MHIST によるバケット分割とカーネル密度推定によって得た情報を保存する際のデータ構造について説明する．

まず MHIST によって分割したバケットに関する情報は，バケットと範囲クエリの位置関係を把握するために使用する，バケット内の各次元軸の属性値の最小値 (min_pos) と最大値 (max_pos) ，そしてバケット内のデータ値の総和 (sum) を保存する．加えて，カーネル密度推定を行ったバケットについては，推定点 (eval_points) とその推定値 (estimate) を保存する．

各バケットについて保持する情報を json 形式で表現すると以下の例のようなデータが，バケット数分だけ保持される．なお，この例では 4 次元データの場合である．

```
{
    "max_pos": [437668389, 29360, 32000, 92000],
    "min_pos": [437667562, 29210, 31000, 92000],
    "sum": 381094
    "estimate":[144.68666667, 144.68666667,
        100.09994792, 100.09994792, ......],
    "eval_points":[
        [4.37667562e+08, 2.92100000e+04,
         3.10000000e+04, 9.20000000e+04],
            ......
        [4.37668389e+08, 2.93600000e+04,
         3.20000000e+04, 9.20000000e+04]
    ]
}
```

## 5. 評価実験

### 5.1 実験環境

**CPU:** 1.3GHz Intel Core i5
**メモリ:** 8GB 1600MHz DDR3
**ストレージ:** SSD 512GB

### 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クエリ問い合わせを 50 個行い，提案手法 3 種の効果を検証する．

また，実験では，バケット分割の上限数を 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．

（1） 評価実験 1: 推定精度の測定

風速以外の四つの属性を範囲とする 4 次元範囲クエリを実行し，クエリ範囲は次の通りとした．まず，時刻の範囲は 360 日とし，その開始位置は乱数を用いて設定した．時刻以外の属性の範囲は固定とした．この範囲クエリに対し，クエリに完全に内包されない周辺グリッドについて，クエリ結果の推定精度と

グリッド一つあたりの事前計算情報のサイズを測定を行う．

推定精度を評価する方法として，次の式を採用し，この値の最大値・平均値・標準偏差を求めた．

$$\frac{|act\_value - est\_value|}{act\_value}$$

ただし $act\_value$ は正しいクエリ結果の集約値，$est\_value$ はその推定値 とする．

（2） 評価実験 2: クエリスループットの測定

5.2 節にて述べた約 1,000 万件のデータのうち，3 属性 (時刻・気温・風速)100 万件を用いて，2 次元範囲クエリを実行し，渡らの手法 [12] と提案手法 3 種のクエリスループットを測定を行う．なお，クエリ範囲の設定は評価実験 1 と同様とした．

（3） 評価実験 3: 推定精度とデータ次元数の関係の測定

提案手法を適用するデータの次元数とその際の推定精度の変化の測定を目的として，データの次元数を 2〜5 に変更して推定精度の評価実験を行った．範囲クエリは，評価実験 1 と同様のものを実行し，データの次元数に合わせて 1〜4 次元に変更した．

### 5.4 実験結果

各手法における推定精度・グリッド一つあたりの事前計算情報のサイズ・クエリスループットを表 1・表 2・表 3 に示す．

表 1 より，精度に関しては，MHIST + カーネル密度推定が最も精度の高い推定を行っており，真のクエリ結果に対して平均で 3.8% の誤差となっている．MHIST のみの手法と比較すると 6.6 倍 の精度の向上となり，カーネル密度推定によってバケット内のデータ分布をより精密に捉えられていることがわかる．

表 2 より，事前計算情報のサイズに関しては，MHIST + カーネル密度推定は MHIST のみと比較して 95 倍 大きい．これはカーネル密度推定によって得た統計情報を保持するためである．一方で，MHIST + 部分的カーネル密度推定は，次元が落ちたバケットに対してはカーネル密度推定を行わないことで，MHIST + カーネル密度推定と比較して，0.27 倍 のデータサイズでありながら，精度に関しては誤差の平均値が 1.2 倍 の増加となっており，著しく精度は低下していない．

表 3 より，クエリスループットでは渡らの手法 [12] である全スキャンと比較して，MHIST + カーネル密度推定は 3.55 倍，MHIST + 部分的カーネル密度推定は 4.74 倍 の高速化となっている．

これらの結果より，事前計算情報のサイズが小さいほど，推定値計算時におけるデータのロード時間は短くなるため，MHIST + 部分的カーネル密度推定の手法は，MHIST + カーネル密度推定と同程度の推定精度を達成しながら高いクエリスループットが実現できると言える．

また，図 6 より，提案手法 3 種ともに，データの次元の増加により推定精度が低下しているのがわかる．しかしながら，MHIST と比較して MHIST にカーネル密度推定を導入した 2 手法は，推定精度の低下率が抑えられている．これより，カーネル密度推定を導入することによって，高次元データへの集約演算の結果の推定も高精度に行うことができると言える．

表 1　精度の比較

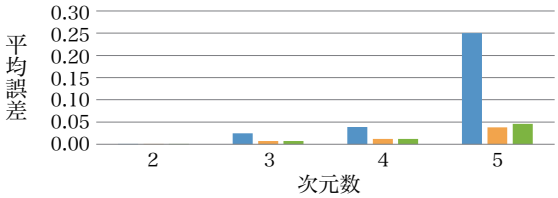
| 手法 | 最大値 | 平均値 | 標準偏差 |
|---|---|---|---|
| MHIST | 0.731 | 0.250 | 0.152 |
| MHIST + カーネル密度推定 | 0.229 | 0.038 | 0.039 |
| MHIST + 部分的カーネル密度推定 | 0.260 | 0.046 | 0.045 |

表 2　データサイズの比較

| 手法 | バイト/グリッド |
|---|---|
| MHIST | 6,708 |
| MHIST + カーネル密度推定 | 635,751 |
| MHIST + 部分的カーネル密度推定 | 173,623 |

表 3　クエリスループットの比較

| 手法 | クエリ/秒 |
|---|---|
| 全スキャン (渡ら [12] の手法) | 6.964 |
| MHIST | 41.767 |
| MHIST + カーネル密度推定 | 24.728 |
| MHIST + 部分的カーネル密度推定 | 32.966 |




| | | | | |
|---|---|---|---|---|
| MHIST | 0.00022 | 0.025 | 0.039 | 0.25 |
| MHIST + カーネル密度推定 | 0.00010 | 0.0074 | 0.012 | 0.038 |
| MHIST + 部分的カーネル密度推定 | 0.00017 | 0.0076 | 0.012 | 0.048 |

図 6　推定精度とデータ次元数の関係

## 6. まとめ

本研究では，グリッド分割された空間への範囲クエリにおいて，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，集約演算結果を推定することで，データの全スキャンを完全に回避し，集約演算処理を効率化する手法の提案を行った．グリッドをあらかじめ複数のバケットに分割し，各バケットに対してカーネル密度推定を利用して統計情報を保持し，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提案手法を実現した．

評価実験として，提案手法 3 種の推定精度とクエリスループットを測定した．その結果，MHIST + カーネル密度推定の手法では 3.8%，MHIST + 部分的カーネル密度推定の手法では 4.6% の誤差で推定を行うことができ，クエリスループットでは渡らの手法 [12] である全スキャンと比較して，MHIST + カーネル密度推定は 3.55 倍，MHIST + 部分的カーネル密度推定は 4.74 倍 の高速化を達成した．そして，MHIST + 部分的カーネル密度推定は、推定精度を維持しつつ高いクエリスループットが実現できることを示した．また，推定精度とデータ次元数の関係の測定実験では，MHIST にカーネル密度推定を導

入することによって，高次元データへの集約演算の結果の推定も高精度に行えることを確認した．

今後の課題として次のようなものが挙げられる．まず，本稿の評価実験は 1 台のマシン上にて実行したものであり，実際に分散 KVS 上に実装したものについて評価実験を行う必要があること．また，本研究では更新が行われない静的なデータを想定している．しかし実際のシステムでは，データの上書きや新しいデータの挿入が発生する．そのような状況を想定し，データをバケットへ分割する処理を評価することも課題の一つである．

## 謝　　辞

本研究の一部は，JSPS 科研費 JP26280115, JP15H02701, JP16H02908, JP15K20990 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] 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.