Vol.2017-DBS-166 No.4
2017/12/22

RDBとKVSを相互に活用した大規模多次元データに対する集約演算の効率化

渡 佑也[1,a)]　欅 惇志[1,3,b)]　宮崎 純[1,c)]　中村 匡秀[2,d)]

概要：本論文では，大規模多次元データに対する集約演算を効率化する手法を提案する．近年のネットワーク技術の高度化により，大量の多次元データが収集できるようになってきた．多次元データの一例としてセンサデータが存在し，集約演算はそうしたデータを分析する上で重要な役割を果たす．リレーショナルデータベース（RDB）は，インデックスによる効率的な集約演算を実現する一方で，データ数の増加にスケールしないという問題がある．一方，分散キーバリューストア（分散 KVS）は，高いデータ挿入スループットとスケールアウト性能を示すものの，多次元データに対するクエリはしばしばデータの全スキャンを伴い非効率的である．提案手法では，RDB と分散 KVS を組み合わせて両者の利点を相互に活用する．また，データをグリッドと呼ばれる部分集合に分割し，グリッドごとに集約値を事前計算することで，データのスキャン量を削減してクエリ処理性能を向上させる．本論文では，評価実験を通して既存のデータストアと提案手法のデータ挿入およびクエリ処理の性能を比較するほか，次元数の増加が提案手法のクエリ処理性能に与える影響を調べる．

1. はじめに

近年のネットワーク技術の高度化により多くの機器がネットワークに接続され，大量の多次元データが収集できるようになってきた．多次元データの例としてセンサデータが存在し，そうしたデータを分析して新たな知見を得ることが重要である．集約演算は，その分析に有用である．

ここでは，センサデータの例として自動車の走行情報を挙げる．このデータは，自動車の位置と移動速度の情報を含んでいるものとする．ある地域内に存在する自動車の平均移動速度を計算することができれば，その地域内での渋滞状況を知ることができる．また，同じクエリをスライディングウインドウにより時刻をずらしながら行うことで，平均移動速度の時間推移をグラフにすることができる．

しかしながら，そのような集約演算を実現するには多くの問題が存在する．センサデータは継続的かつ頻繁に生成されるため，それを扱うデータストアは高い挿入スループットを提供しなければならない．また，多次元データを効率的に扱いながら，集約演算を実行する必要がある．

[1] 東京工業大学
[2] 神戸大学
[3] 国立研究開発法人科学技術振興機構，ACT-I
[a)] watari@lsc.cs.titech.ac.jp
[b)] keyaki@lsc.cs.titech.ac.jp
[c)] miyazaki@cs.titech.ac.jp
[d)] masa-n@cs.kobe-u.ac.jp

このような問題に対して，これまで様々な研究がなされてきた [1], [2], [3], [4]．Nishimura ら [5] は，MD-HBase と呼ばれる手法を提案した．MD-HBase では，空間充填曲線を用いて多次元データを一次元データに変換することで，一次元インデックスしか存在しないキーバリューストア上で多次元データを効率的に扱う．

本研究では，既存のデータストアであるリレーショナルデータベース（RDB）と分散キーバリューストア（分散 KVS）の利点を組み合わせる．両者は次に示す特徴を持つ．

- RDB [6] には，多次元データを効率的に扱うためのインデックスが存在する．しかし，RDB において高いデータ挿入スループットを実現することは，一貫性と永続性を担保するトランザクションの影響により困難である．加えて，データ数が増加した場合，RDB が大規模データにスケールしないという問題が顕在化する．
- 分散 KVS [7], [8], [9], [10] は，大規模データを効率的に扱えるよう設計されており，高いデータ挿入スループットとスケールアウト性能を持つ．しかし，ほとんどの分散 KVS には単純なインデックスしか存在せず，多次元データへのアクセスは大量のデータへの全スキャンが伴うため非効率的となる．

もし，RDB と分散 KVS を組み合わせて両者の利点を相補的に活用することができれば，大規模データにスケールする多次元データストアを実現することができる．

また，我々は部分集約法 [11] に着目した．部分集約法では，マテリアライズドビューのようにデータの集約値を事前計算することで，クエリ処理のコストを削減する．例えば，データ集合 D が三つの部分集合 B_1，B_2，B_3 に分割されている状況を考える．このとき，D 内のデータの総和 ($\mathrm{sum}(D)$) は，$\mathrm{sum}(D) = \mathrm{sum}(B_1) + \mathrm{sum}(B_2) + \mathrm{sum}(B_3)$ のようにして，データの部分的な総和から求めることができる．もし，三つの部分和がそれぞれ事前に計算されていれば，単にそれらを足すだけで $\mathrm{sum}(D)$ が求まり，D 内のデータをスキャンするコストを大幅に削減できる．

以上の議論に基づき，本研究では，RDB と分散 KVS を組み合わせた大規模多次元データに対する効率的なデータストアを提案する．提案手法には主に二つの特徴がある．一つは，提案手法に入力された実データは分散 KVS で保存する一方，多次元インデックスは RDB で保持するという点である．前者の実データは分散 KVS に高いスループットで挿入され，後者のインデックスは複雑な構造を持つものの RDB により効率的に管理される．もう一つの特徴は，多次元空間を**グリッド**と呼ばれる部分集合に分割し，各グリッドについて総和などの集約値を事前に計算し保持する点である．クエリと共通部分を持たないグリッドのデータは無視できるほか，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集約値を再利用できるため，データのスキャンを省略することができる．以上により，集約演算のクエリ処理を効率的に実行することが可能となる．

我々は，[12] において提案手法の有用性を示したが，本論文では追加の評価実験を通して，既存の多次元データストアと性能を比較する．また，次元数の増加が提案手法のクエリ処理性能に与える影響についても明らかにする．

2. MD-HBase

MD-HBase [5] は，分散 KVS の一種である HBase [13] を多次元データを効率的に扱えるよう改良したものである．

MD-HBase では，空間充填曲線の一種である Z カーブ [14] を用いて，多次元データを一次元データに変換する．変換された値は，HBase におけるキーとして用いられる．また，多次元空間を k-d tree [15] にしたがってバケットに分割し，バケットごとのキーの最小値と最大値を保持する．

多次元の範囲クエリを処理する際，MD-HBase では，与えられたクエリのキー範囲の最小値と最大値を計算し，HBase に対して範囲スキャンを行う．HBase では，データはキーについてソートされて格納されているため，範囲スキャンを高速に行える．また，MD-HBase ではクエリ範囲と交わらない領域のスキャンを省略して効率化を図る．

MD-HBase は，HBase テーブル上に独自のインデックスを構築するが，他の分散 KVS にこの手法を適用するのは煩雑な作業を伴う．一方，提案手法では，RDB が提供するインデックスをそのまま活用するため，分散 KVS 上に新たにインデックスを構築する必要はないほか，RDB によってインデックスの一貫性が保証されるという利点も持つ．

なお，MD-HBase におけるデータのスキャン手法は，[16] においてさらに改良されているが，クエリのパターンが既知でなければならないという制約が存在する．

3. 問題設定

本研究では，データを多次元空間上に存在する点とみなす．データの定義域を**データ空間** $D\,(\subseteq \mathbb{R}^n)$ と呼ぶ．ここに n は，データの次元数である．データ空間 D は，超直方体である．すなわち，D は，$D = [s_1, e_1] \times [s_2, e_2] \times \cdots \times [s_n, e_n]$ のように直積を用いて表される．

本研究では，多次元データに対する**部分集約可能な集約演算**を扱う．これは，次のように定義される．

定義 3.1 (部分集約可能な集約演算) クエリ範囲 $Q\,(\subseteq \mathbb{R}^n)$ と集約演算 $f(Q)$ が与えられたとする．ここに $f(Q)$ は，Q 内に存在するデータの集約値を計算する．次式を成り立たせるような関数 c が存在するときに限り，f は，部分集約可能である．

$$f(Q) = c(f(G_1), f(G_2), \ldots, f(G_m))$$

ここで，Q は超直方体 $G_1, G_2, \ldots, G_m$ に分割されている．すなわち，次式が成り立つ．

$$\forall i \neq j\,(G_i \cap G_j) = \emptyset, \quad \text{かつ} \quad \bigcup_{i=1,\ldots,m} G_i = Q.$$

部分集約可能な集約演算の例としては，総和，平均，最小，最大などが挙げられる．一方，濃度は部分集約可能でない．

本研究の目的は，超直方体で表されるクエリ範囲 $Q\,(\subseteq D)$ 内に存在するデータに対して，部分集約可能な集約演算を効率的に実行することである．

4. 提案手法

本節では，**図 1** に示した概念図を用いて，提案手法の概要を説明する．提案手法は次のようにしてデータのスキャン量を削減し，集約演算を効率的に実行する．

(1) データ空間を**グリッド**と呼ばれる超直方体に分割する（図 1 の上側）．この分割は，k-d tree にしたがう．
(2) 各グリッドについて，部分集約値を事前に計算する．
(3) クエリ（図 1 では破線で示されている）が与えられたとき，クエリに完全に含まれているグリッドのスキャンを省略し，事前計算した部分集約値を参照する．これにより，データのスキャン量が削減される．

上記を実現するには，次の三種類のデータが必要となる．

- グリッドのメタデータ（グリッドの位置，大きさ，ID）
- 実データ
- 部分集約値

グリッドが極端に小さくない限り，メタデータのサイズ

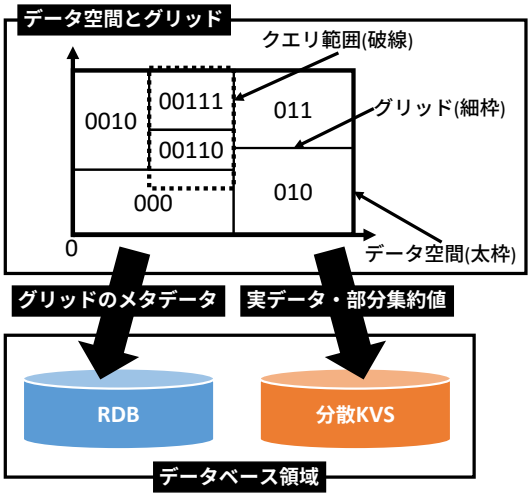

図 1 提案手法の概念

はそれほど大きなものにならない．一方で，クエリを処理するためには，クエリと交わるグリッドを列挙する必要があり，この作業は複雑である．したがって，提案手法では，インデックスにより多次元データを効率的に扱える RDB でメタデータを保持する．データ挿入の頻度と比較して，グリッド分割はそれほど頻繁に発生しないと考えられるため，メタデータはレプリケーションに適している．

一方，実データのサイズは極めて大きなものになると考えられる．また，データが継続的に生成されるため，実データと部分集約値の更新が頻繁に発生する．したがって，大規模データにスケールしやすく高いデータ挿入スループットを実現できる分散 KVS でこれらのデータを保持する．

このように，RDB と分散 KVS の利点を相補的に活用することで，大規模かつ多次元のデータを扱う際に発生する問題に対処する．本研究では，RDB，分散 KVS として PostgreSQL，HBase を用いるが，提案手法は，いかなる RDB，分散 KVS を用いても実装できることに注意する．

4.1 グリッド分割

提案手法におけるグリッド分割は k-d tree [15] にしたがう．グリッド内のデータ数が閾値 $N_{\text{threshold}}$ を超えたとき，グリッドは循環的に選択される軸について分割される．この分割は，グリッド内のデータ数が**グリッドサイズ** N_{size} 以下になるまで再帰的に行われる．なお，k-d tree における分割点は，木を平衡に保つためデータの中央値とすることが多いが，中央値の計算にはコストがかかるため，提案手法では，各軸の平均値を中央値の近似値として用いる．

4.2 データ構造

提案手法におけるデータ構造は，RDB と分散 KVS の双方に構築される．本節では，その詳細について述べる．

図 2 に示すように，データ構造は，大きく分けて**データベース領域**と**バッファ領域**の二つの領域で構成される．データベース領域は，本節冒頭で述べた三つのデータ（グリッドのメタデータ，実データ，部分集約値）を保持する．一方，バッファ領域は，データ挿入スループットを高めるために用意されている．データを挿入する際，グリッドの集約値を更新する必要があるほか，グリッド内のデータ数が $N_{\text{threshold}}$ を超えるとグリッド分割が発生する．複数のクライアントが同時にデータを挿入する場合でもデータの一貫性を確保するには，集約値の更新やグリッド分割で排他制御が必要となる．もし，クライアントが直接データベース領域を更新すると，高コストな排他制御によってクライアントからみたデータ挿入スループットが低下してしまう．この問題を解決するために，提案手法では，クライアントはデータを一時的にバッファ領域に挿入する．データベース領域を更新することはしないため，排他制御は生じない．また，バッファ領域は，分散 KVS 上のテーブルであるため，データ挿入スループットをスケールさせられる．

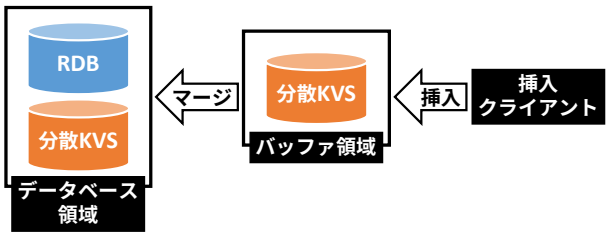

図 2 提案手法におけるデータ構造

バッファ領域内のデータに対する集約演算は，部分集約値が計算されていないため非効率的である．よって，バッファ領域内のデータは速やかにデータベース領域へ統合されなければならない．この統合作業のことを**マージ**と呼び，サーバ側で分散して行う．詳細は，4.4.1 節で述べる．

4.3 分散 KVS におけるキーのデザイン

分散 KVS におけるキーのデザインは，性能向上の観点から重要である．提案手法では，キーの先頭 1 バイトを乱数とし，HBase の Region Server 間で効率的に負荷を分散できるようにした．接頭辞は，$[0, P)$ の範囲内でランダムな値を取る．ここで P は，Region Server の台数の倍数にすることが望ましい．これは，HBase テーブルを接頭辞に基づきリージョンに分割するとき，等しい個数のリージョンを各 Region Server に割り当てられるためである．

4.4 データ挿入

データ挿入のアルゴリズムは極めてシンプルであり，4.3 節で述べた乱数の接頭辞をキーに付加し，バッファ領域の HBase テーブルに挿入するだけである．本節では，マージ処理について詳細に説明する．

4.4.1 マージ

マージ処理は，4.3 節で述べたキーの接頭辞ごとに複数台のサーバで分散して実施される．**図 3** にマージの流れを示す．マージ処理では，サーバのうち任意の一台（図 3 で

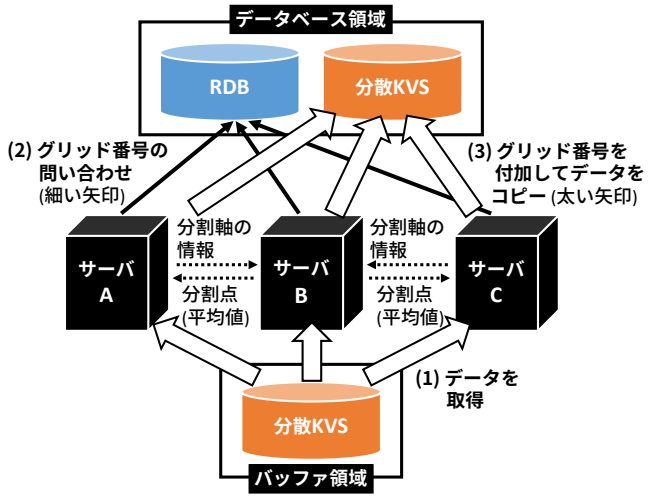

図 3 マージの流れ

はサーバ B）にマスタと呼ばれる役割を与える．

マージのアルゴリズムは，次の通りである．

アルゴリズム 1 接頭辞 p を持つデータのマージ

(1) バッファ領域から接頭辞 p を持つデータを取得する．
(2) (1) で取得したデータが属するグリッド番号を PostgreSQL に問い合わせる．
(3) (1) で取得したデータをデータベース領域の HBase テーブルにコピーする．その際，グリッド番号をキーに付加する．必要に応じて，グリッド分割を行う．
(4) (1) で取得したデータをバッファ領域から削除する．

(3) では，グリッド内のデータ数が $N_{\text{threshold}}$ を上回ったとき，グリッド分割が発生する．グリッド分割で必要なデータは，グリッド内の実データと分割点の情報である．前者は，HBase テーブルをスキャンすることで取得できる．後者を計算する方法は次の通りである．まず，各サーバは，分割する軸についての総和とデータの個数をマスタに送信する．マスタは，その情報を基に分割点，すなわち分割する軸の平均値を計算し，各サーバに通知する．これによりグリッド分割を分散して行うことができる．なお，マスタは，PostgreSQL 上のメタデータも更新する．

分割するグリッドの数が多いとき，マスタはボトルネックとなり得る．しかし，異なるグリッドの分割処理は独立しているため，グリッドごとにマスタの役割を与えることで，この問題を回避できる．

4.5 クエリ

クエリ領域 Q ($\subseteq D$) が与えられたとき，Q 内のデータに対して集約演算を行うアルゴリズムは，次の通りである．

アルゴリズム 2 クエリ

(1) Q と交わるグリッドを PostgreSQL テーブルから列挙する．列挙されたグリッドの集合を G とする．また，各グリッドについて，Q に完全に包含されるかどうかも PostgreSQL に問い合わせる．
(2) G のグリッドのうち，Q に完全に包含されるもの（図 1 では，グリッド 00110 と 00111）について，HBase に保存されている部分集約値を取得し足し合わせる．
(3) G のグリッドのうち，Q と一部が交わるもの（図 1 では，グリッド 000）について，HBase に保存されている実データをスキャンし，Q に含まれるものについて集約値を計算する．
(4) (2) と (3) で得られた集約値を足し合わせて，最終的な集約結果を求める．

5. 評価実験

本研究では，三つの評価実験を行った．実験 1，2 では，先行研究の一つである MD-HBase [5] とデータ挿入およびクエリ処理のスループットを比較した．実験に当たり，我々は，公開されている MD-HBase の実装[*1]を一部改良して用いた．なお，この実装は二次元のデータまでしか対応していないため，実験 1，2 では二次元データを用いた．

評価実験では，17 台の PC から構成されるクラスタを用いた．各 PC は，Intel Core i7-3770 CPU (3.4 GHz)，32 GB メモリ，および 2TB の HDD で構成され，CentOS 6.7 上で HBase 1.2.0 が動作している．クラスタのうち，14 台の PC が HBase での Region Server の役割を持つほか，8 台の PC に PostgreSQL 9.6.1 をインストールし，マルチスタンバイ構成のレプリケーションを構築した．

5.1 データセット

評価実験では，二種類のデータセットを用いた．

SFB（サンフランシスコ・ベイエリア）データ

ネットワークベースのジェネレータ [17] を用いて，サンフランシスコ・ベイエリア内を移動する 22,352,824 個（約 2,200 万個）のデータを生成した．各データは，属性として緯度・経度を持つ二次元データである．我々は，このデータを **SFB データ**と呼ぶことにし，実験 1，2 で用いた．

室内気象センサデータ

我々は，2010 年 1 月 14 日から 2014 年 4 月 11 日までの間の室内に設置された気象センサのデータを 2,032,918 件（約 200 万件）収集した．各データは，時刻，温度，湿度，照度，風速などの 16 個の属性を持つ．このデータのうち 2011 年から 2013 年までの 3 年分のデータを抽出し，時刻を 3 年ずつ遅らせながら 7 倍に複製することで，2011 年から 2031 年までの擬似的な約 1,000 万件のデータを生成した．以後，このデータを**室内気象センサデータ**と呼ぶ．

5.2 実験 1：データ挿入スループット

提案手法と MD-HBase のデータ挿入におけるパフォーマンスを比較するために，SFB データを各手法に挿入し，そのスループットを計測した．

[*1] https://github.com/shojinishimura/Tiny-MD-HBase

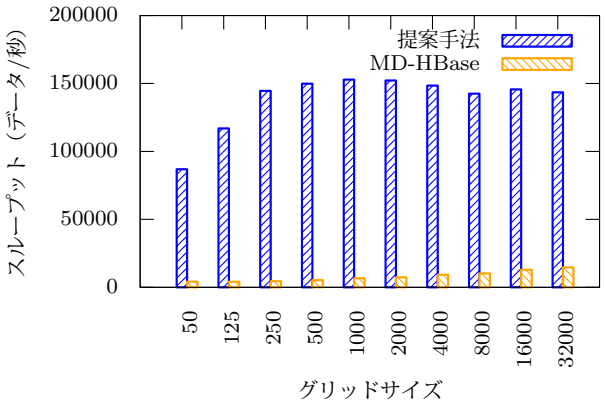

図 4 データ挿入スループット

実験では，クラスタ内の一つの PC をデータを挿入するクライアントとして動作させた．グリッドサイズ N_{size} は，$N_{\text{size}} = 50, 125, 250, 500, 1000, 2000, 4000, 8000, 16000, 32000$ の 10 通りに変化させた．なお，MD-HBase におけるグリッドサイズは，各バケット内に存在できるデータ数の上限を表す．提案手法では $N_{\text{threshold}} = N_{\text{size}} \times 10$ とした．

図 4 に実験結果を示す．図 4 から，提案手法はすべてのグリッドサイズにおいて，MD-HBase よりも高いデータ挿入スループットを実現できたことが明らかとなった．しかしながら，本実験で用いた MD-HBase の実装は，データ挿入について十分に最適化されていないため，この実験結果は，提案手法の性能を過大に評価している可能性がある．

5.3 実験 2：クエリスループット

実験 1 で挿入したデータに対して，範囲クエリの集約演算を行い，提案手法と MD-HBase のクエリ処理性能を比較する実験を行った．選択率が 0.001%，0.01%，0.1%，1%，10%になるようランダムにクエリを生成し，選択率ごとに 128 個のクライアントから同時にクエリを発行した．

図 5，**表 1** に実験結果を示す．ここで「部分集約なし」とは，事前計算した集約値を用いなかった場合の実験を示している．この実験は，純粋な範囲クエリの性能を評価するために行ったものである．提案手法と MD-HBase のいずれも，最も高いクエリスループットを実現したグリッドサイズについて図表に示した．

実験結果より，提案手法はどの選択率のクエリにおいても，MD-HBase と同等か高いクエリスループットを実現した．選択率が 10%のクエリでは，MD-HBase に対して 8.2 倍のスループットとなっている．

提案手法でクエリ処理を効率化するには，クエリに完全に含まれるグリッド数が大きい方が望ましく，その値はクエリの面積に比例する．一方，データのスキャン量は，クエリと一部が交わるグリッド数に比例するが，その値はクエリの周長に比例する．したがって，クエリ範囲が大きいほどデータのスキャン量を削減しやすくなり，選択率 10%のクエリで高いスループットが得られたと考えられる．

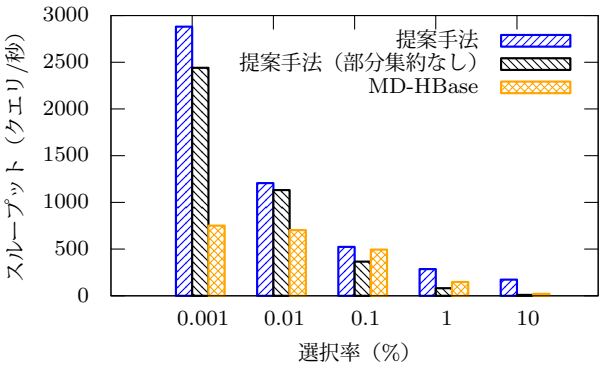

図 5 選択率を変化させたときのクエリスループット

表 1 提案手法（部分集約なし）と MD-HBase に対する提案手法のクエリスループット向上率

選択率	0.001	0.01	0.1	1	10
提案手法（部分集約なし）	1.18	1.07	1.43	3.52	17.78
MD-HBase	3.84	1.72	1.06	1.93	8.20

表 2 提案手法における各種統計情報

選択率	0.001	0.01	0.1	1	10
(a)	223	2235	22349	223489	2234529
(b)	3041	11218	66813	33577	91372
(c)	0	426	7298	204146	2185014
(b)/(a)	13.61	5.02	2.99	0.15	0.04
(c)/(a)	0.00	0.19	0.33	0.91	0.98

※ (a)：クエリによって選択されたデータ数，(b)：スキャンしたデータ数，(c)：スキャンを省略したデータ数

このことは，**表 2** に示した各種統計情報からも分かる．表 2 における “(c)/(a)” を見ると，選択率 0.001%ではスキャンをほとんど省略できていないが，選択率 10%では実に 98%ものデータのスキャンを省略できている．また，“(b)/(a)” は，選択率が 0.1%かそれを下回るクエリにおいて 1 よりも大きな値となっている．これは，クエリと関係ないデータを多くスキャンしてしまったことを示している．そのため，選択率 0.1%では，提案手法と MD-HBase のスループットが拮抗したと考えられる．なお，選択率 0.01%，0.001%では，提案手法は MD-HBase より高い性能を示している．これは，クエリの大きさが MD-HBase のバケットの大きさを下回り，MD-HBase におけるスキャン量削減が限界に達したためであると考えられる．

5.4 実験 3：次元数を変化させたときのクエリスループット

提案手法において，次元数がクエリ処理性能にどのような影響を与えるかを調べる実験を行った．

実験 3 では，室内気象センサデータを提案手法に対して挿入した後，クエリスループットを計測した．その際，次元数 n は，$n = 2, 3, 4, 5, 8, 16$ に変化させ，クエリの選択率も 0.001%，0.01%，0.1%，1%，10%に変化させた．本実験

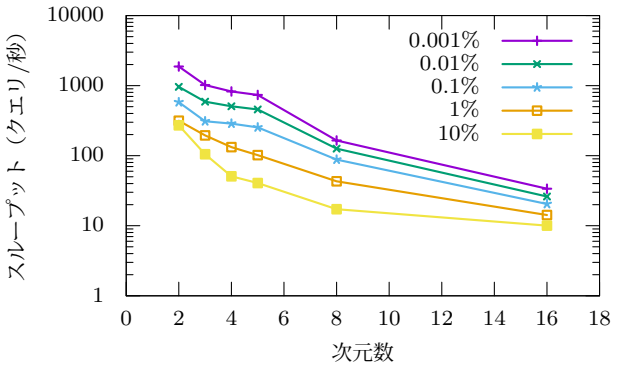

図 6 次元数を変化させたときのクエリスループット（凡例は選択率）

では，次元数が $n = k$ のとき，室内気象センサデータの最初の k 個の属性についてインデックスを構築した．

図 6 に実験結果を示す．グラフの縦軸は，クエリスループットの対数軸である．グラフは概ね直線的な減少を示していることから，提案手法は，次元数が増加するにつれ指数関数的にクエリスループットが低下することが分かった．次元数が 2 から 16 に増加した場合のスループットの低下率は，95〜98%であった．

このスループットの低下は，高次元空間において部分集約値の再利用が難しいことが原因であると考えられる．提案手法におけるデータのスキャン量は，クエリと一部が交わるグリッド数に依存し，その個数は超直方体の表面積に比例すると考えられる．簡単のため，n 次元超直方体の一辺の長さを a とすると表面積は $2na^{n-1}$ となり，これは指数関数的な増加を示す．クエリスループットは表面積に反比例すると考えられる．以上の考察は，図 6 が直線的に減少していることとよく合致している．

6. まとめ

本研究では，大規模多次元データに対する集約演算を効率化する手法を提案した．提案手法では，RDB と分散 KVS を組み合わせて両者の利点を相補的に活用した．また，データの集約値を事前計算してクエリ処理時にその値を再利用することにより，データのスキャン量を削減した．

評価実験を行った結果，提案手法は，既存のデータストアである MD-HBase に対して最大で 8.2 倍の高いクエリスループットを達成したほか，MD-HBase を大幅に上回る高いデータ挿入スループットを実現した．また，提案手法のクエリスループットは，次元数の増加に伴って指数関数的に減少することが判明した．

部分集約値の再利用が困難となる高次元データでのクエリ処理性能を改善することが今後の課題である．

謝辞

本研究の一部は，JSPS 科研費 JP15H02701，JP16H02908，17K12684，及び，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．

参考文献

[1] Wang, J., Wu, S., Gao, H., Li, J. and Ooi, B. C.: Indexing multi-dimensional data in a cloud system, *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ACM, pp. 591–602 (2010).
[2] Zhang, X., Ai, J., Wang, Z., Lu, J. and Meng, X.: An efficient multi-dimensional index for cloud data management, *Proceedings of the first international workshop on Cloud data management*, ACM, pp. 17–24 (2009).
[3] Li, X., Kim, Y. J., Govindan, R. and Hong, W.: Multidimensional range queries in sensor networks, *Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM, pp. 63–75 (2003).
[4] Escriva, R., Wong, B. and Sirer, E. G.: HyperDex: A distributed, searchable key-value store, *ACM SIGCOMM Computer Communication Review*, Vol. 42, No. 4, pp. 25–36 (2012).
[5] Nishimura, S., Das, S., Agrawal, D. and El Abbadi, A.: $\mathcal{MD}$-HBase: design and implementation of an elastic data infrastructure for cloud-scale location services, *Distributed and Parallel Databases*, Vol. 31, No. 2, pp. 289–319 (2013).
[6] Codd, E. F.: A relational model of data for large shared data banks, *Communications of the ACM*, Vol. 13, No. 6, pp. 377–387 (1970).
[7] Lakshman, A. and Malik, P.: Cassandra: a decentralized structured storage system, *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, pp. 35–40 (2010).
[8] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D. and Yerneni, R.: PNUTS: Yahoo!'s hosted data serving platform, *Proceedings of the VLDB Endowment*, Vol. 1, No. 2, pp. 1277–1288 (2008).
[9] Redis, https://redis.io/.
[10] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: amazon's highly available key-value store, *ACM SIGOPS operating systems review*, Vol. 41, No. 6, pp. 205–220 (2007).
[11] 小山田昌史，　陳テイ，成田和世，荒木拓也：データの部分集約による高速かつ正確なデータ集計処理の実現，研究報告データベースシステム (DBS)，Vol. 2014, No. 19, pp. 1–7 (2014).
[12] 渡　佑也，欅　惇志，宮崎　純，中村匡秀：多次元データに対する集約演算の効率化手法におけるデータ挿入スループットの向上，DEIM2017 論文集，E5-2 (2017).
[13] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E.: Bigtable: A distributed storage system for structured data, *ACM Transactions on Computer Systems (TOCS)*, Vol. 26, No. 2, p. 4 (2008).
[14] Morton, G. M.: *A computer oriented geodetic data base and a new technique in file sequencing*, International Business Machines Company New York (1966).
[15] Bentley, J. L.: Multidimensional binary search trees used for associative searching, *Communications of the ACM*, Vol. 18, No. 9, pp. 509–517 (1975).
[16] Nishimura, S. and Yokota, H.: QUILTS: Multidimensional Data Partitioning Framework Based on Query-Aware and Skew-Tolerant Space-Filling Curves, *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, pp. 1525–1537 (2017).
[17] Brinkhoff, T.: A framework for generating network-based moving objects, *GeoInformatica*, Vol. 6, No. 2, pp. 153–180 (2002).