コンパクトなトライ木表現を用いた多次元部分集約手法の提案

西村　祥治†,††　　横田　治夫†

†† 日本電気株式会社　〒 216–8666 神奈川県川崎市中原区下沼部 1753
† 東京工業大学情報理工学院　〒 152–8552 東京都目黒区大岡山 2-12-1
E-mail: †s-nishimura@bk.jp.nec.com, ††yokota@cs.titech.ac.jp

あらまし　近年、データ分析の対象となるデータ量の増大のため、データアクセスのボトルネックを解消するデータ管理技術が重要になってきている。特に、大量のデータを扱う分散データベースやコールドストアは、一回当たりのデータアクセスコストが大きくなる。一方で、アクセスした結果、所望のデータが存在しない、存在したとしてもごく少数である場合、支払うコストが大きくなりすぎる。これを回避するため、ブルームフィルターやアダプティブレンジフィルターのようなデータの存在性をチェックするデータ構造が提案されてきた。本発表では多次元データに対する集約演算を対象に、多次元データ集合の部分集約値を管理するコンパクトなデータ構造を提案する。そして、初期評価結果について報告する。

キーワード　多次元部分集約, トライ木

1. はじめに

近年、IT システムや実世界からのセンサーデータなど膨大なデータを収集し、そのデータを分析することで新たな知見を見出すデータ分析が広く行われるようになってきている。このため、データの蓄積、管理を担うデータベースも扱うデータ量が増加し続けている。データ量の大規模化に伴い、データ管理の分散化および管理単位の大型化が進んでいる。例えば、大規模なデータを扱う Apache HBase は、効率的に膨大なデータを分散管理するように設計されている。さらに HBase の管理単位であるリージョンサイズのデフォルト値は、バージョンを重ねるごとに増加しており、従来 256MB が最新版では 1GB になっている。

一方で、収集されるデータのうち、一回のデータ分析で分析対象となるデータは多くの場合、ごく一部である。このため、分析に必要なデータを効率的に取得する技術が近年注目集めている。データアクセスを効率化する方法として現在大きく 3 つの方向性がある。

一つ目は、データをブロックに分割する際、データ取得時にアクセスするブロック数を最小化できるようにブロックに入れるべきデータを注意深く選ぶ方法である。例えば、Sun らはクエリパターンやデータ分布を考慮したクラスタリングアルゴリズムを提案し、近傍のデータがなるべく同一ブロックに含まれるように分割する方法を提案している [5]。また、西村らはクエリパターンを考慮し、データアクセス時にアクセスされるブロック数を最小に抑えるような空間充填曲線を用いたデータ分割手法を提案している [4]。

二つ目は、ブロックに含まれるデータの存在性を簡易に検査することでブロックアクセスを回避する手法である。データ存在性を簡易に検査するデータ構造として、Bloom Filter や Adaptive Range Filter(ARF) [1] が知られている。Bloom Filter は点クエリに対して、ARF は一次元の範囲クエリに対するデータの存在性を検査を提供する。検査の結果、False の場合は確実にデータが存在せず、True の場合データが存在する可能性がある。そして、True の場合のみブロックにアクセスする。

三つ目は、集約結果を事前に計算し、集約クエリ実行時に計算済みの結果を返す手法である。これはデータをアクセスした結果行われる典型的な演算が集約演算だからである。Small Materiarized Aggregation(SMA) [2] はデータセットを小さいブロックに分割し、各属性ごとに代表的な集約値である総数 (COUNT)、最小値 (MIN)、最大値 (MAX)、総和 (SUM) を保持した軽量インデックスを提案している。すなわち、ある集約クエリが実行されると、軽量インデックスにある属性の MIN と MAX を用いて集約範囲に完全に含まれるかを検査し、含まれる場合は問い合わされた集約値を返す。完全に含まれない場合はそのブロックにはデータがないので、集約値の既定値（例えば、総数なら 0）を返す。一部しか含まれない場合は、該当するものだけ再計算する。このアイデアを ARF による範囲管理と組み合わせたものとして、APA 木 (Adaptive Partial Aggregation 木) が提案されている [6]。

本論文では、データ管理単位の大型化が進んでいることを受け、ブロックサイズが大きい場合に対応できる多次元データに対する部分集約結果を管理するデータ構造を提案する。

2. 事前準備

2.1 Adaptive Range Filter

Adaptive Range Filter(ARF) は、範囲内のデータの存在性を検査するコンパクトなデータ構造である。ARF では、トライ木を用いて範囲を階層的に表現し、そのリーフでその範囲におけるデータの存在性を 1 ビットの占有ビット（Occupied bit）で表現する (図 1)。ARF の中間ノードは 2 つの子ノード（0 側と 1 側）を持ち、それぞれが中間ノードであるか、リーフノードであるかを 2 ビットで表現する。ARF 全体としては、

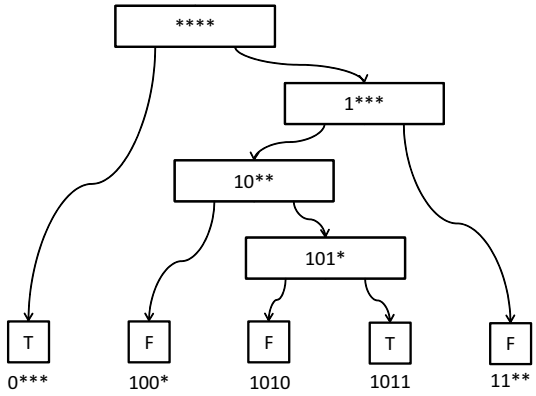


図 1　Adaptive Range Filter によるデータ存在性の範囲管理

中間ノード部とリーフノード部に分けて表現する。中間ノード部はルートノードから幅優先で各中間ノードを連結することで表現する。リーフノードは、中間ノード部でリーフノードを表すビットが出現する順序で占有ビットを連結する。このため、ARF が n ノードあれば、そのサイズは $1.5n$ となる。

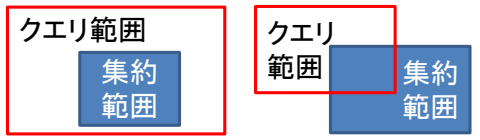
ARF の中間ノード管理する範囲は属性値をビット列表現した時の最上位側のビットパターンに対応する。すなわち、ビットを 0,1,∗(don't care) とし、属性値が 4 ビットであったとする。すると、ARF のルートノードは ∗∗∗∗、すなわち全範囲に対応する。そして、ルートノードの左側の子、すなわち 0 側の子ノードは 0∗∗∗ となり、0000 から 0111 までの範囲に対応する。このことから、ARF は値をビット列としてみた時、プリフィックスの共通性をもとに範囲を管理する手法と言える。そして、範囲を短くすることが、プリフィックスの長さを長くすること、つまり、中間ノードを深く成長させることに対応する。

ARF が範囲のデータの存在性を検査の結果 True と答える時、偽陽性を含む。このため、ARF では偽陽性となる可能性を下げるために実行されたクエリに応じて、中間ノードの深さを制御する。すなわち、ある範囲に関する検査結果が誤っているとその範囲の子ノードを成長させる。一方で、ARF が保持するトライ木が大きくなるため、表現に必要なデータサイズが増加する。そこで、ARF はデータ容量の制約を満たすために、あまりアクセスされていない範囲に対応する中間ノードを縮退させることで容量を削減する。

2.2　クエリ範囲と集約範囲の関係

ある範囲の集約クエリが実行されたときのクエリの範囲（クエリ範囲）と、事前に集約値を求める範囲（集約範囲）の関係について説明する。

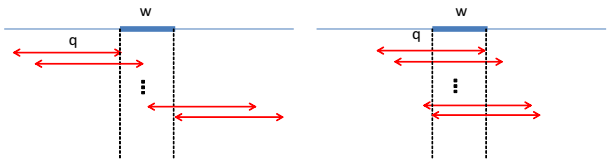
クエリ範囲と集約範囲との位置関係は、図 2 に示すように 4 種類ある。事前に求めた集約値が利用できるのは、クエリ範囲がその集約範囲を完全に包含する場合 (図 2(a)) である。この時、集約範囲の位置は固定され、クエリ範囲はクエリによって変化する。このため、集約範囲が狭いほど、より多くのクエリ範囲によって完全に包含されやすくなる。一方で、集約範囲を狭くしすぎると少数のデータを部分集約した結果しか得られず効果が低くなり、また、事前に求めた集約値を保持するための容量が増大する。ここで、事前に求めた集約値がクエリ実行時



(a) クエリ範囲が集約範囲を包含　(b) クエリ範囲と集約範囲が交差

(c) クエリ範囲が集約範囲に包含　(d) クエリ範囲と集約範囲が互いに素

図 2　クエリ範囲と集約範囲の位置関係



(a) クエリ範囲と集約範囲が重なる　(b) クエリ範囲が集約範囲を包含

図 3　ヒット率の分析

に利用されることをヒットと呼び、集約範囲の大きさとヒットのしやすさ（ヒット率）について分析をする。

まず、範囲が一次元の場合を考える。クエリ範囲の幅を q、集約範囲の幅を w とする。ある集約範囲の近傍にクエリ範囲の幅が q であるクエリが実行されたときのヒット率を求める。図 3 を参考にクエリ範囲の起点に着目すると、ある集約範囲と重なる可能性があるクエリの総数は $q+w$ に比例する。一方で、ある集約範囲を完全に包含するクエリの総数は $q-w$ に比例する。ここから、ヒット率は $\frac{q-w}{q+w}$ となる。例えば、$w=q/2$ の場合は 33%、$w=q/4$ の場合は 60%、$w=q/8$ の場合は 77%となる。

次に、範囲が多次元の場合を考える。次元数が k の場合、各次元においてクエリ範囲の幅に対する集約範囲の幅の比率が等しいとすると、一次元のヒット率の k 乗となる。ここから、クエリ範囲に対して集約範囲は相当狭くしなければ有効に機能しないことが分かる。

では、図 2 において、クエリ範囲と集約範囲が交差したり、クエリ範囲が集約範囲に包含されたりする場合を利用できる可能性はないだろうか。可能性があるのは、データの分布が極めて疎である場合である。図 4 のように集約範囲内にデータが一つしかなかったとする。この時、たとえ、クエリ範囲が集約範囲を完全に包含できなかったとしても、クエリ範囲が集約範囲内のデータを含むかを検査することで判定できる。この時、データの数が一つしかないため判定のコストは小さい。また、その判定がどちらの場合であっても、集約値を正しく回答することができる。すなわち、当該データがクエリ範囲になければデータはなしのため集約値はなしと回答できる。そして、当該データがクエリ範囲内にあるのであれば、データ自身が MIN、

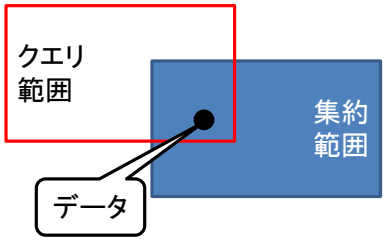


図 4　データ分布が疎である場合

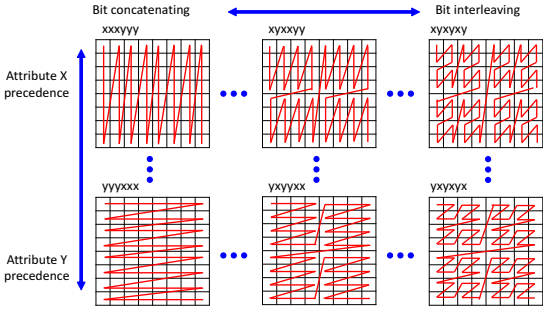


図 5　ビット混合曲線族

MAX、SUM の集約値であり、COUNT は 1 を返せばよい。

3. 提 案 手 法

本論文では上記の考察結果をもとに ARF を拡張し、多次元データ向けの部分集約構造を提案する。

3.1 空間充填曲線による多次元データの表現

まず、ARF で多次元の範囲を扱えるように拡張する。ARF は一次元のデータを対象に、プリフィックスの共通性をもとに一次元の範囲を管理している。そこで、多次元のデータを対象にプリフィックスの共通性をもとに多次元の範囲を管理できる方法として、空間充填曲線による多次元データの表現を用いる。ここでは、空間充填曲線としてビット混合曲線族 (bit-merging curve family) [4] を用いる（図 5）。この曲線は各属性のビット列を任意の順で混ぜ合わせることで構成できる曲線群である。本提案では、各属性における集約範囲の幅に相当する各属性の下位ビット列を、ビット混合順において下位に配置するような曲線を選ぶ。例えば、属性 X、属性 Y に関するクエリ範囲の幅が $2^{dx}, 2^{dy}$ である場合、図 6 で示すように、それぞれ下位 dx, dy ビットをビット混合順で下位に配置した曲線を用いる。これにより、典型的な集約パターンで事前に計算した集約値を利用できる可能性を高める。

3.2 部分集約値を管理するデータ構造

提案するデータ構造は、部分集約値を保持している範囲を管理するインデックスと集約値を管理する集約テーブルからなる。

範囲管理インデックスは ARF をベースとし、中間ノードとして保持する 2 ビットの意味付けを変更する。ARF では、中間ノードの子ノードの種類を保持していた。一方、提案手法では子ノードの有無とする。すなわち、11 は 0 側、1 側両方に子ノードがあり、10 は 0 側だけに子ノードあり、01 は 1 側だけに子ノードありの中間ノードとする。そして、00 は集約ノードであり、その範囲に対応する集約値が集約テーブルにあること

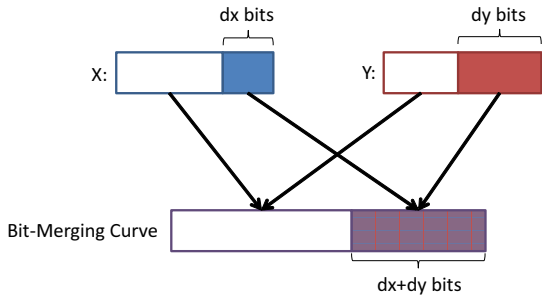


図 6　クエリ範囲の幅に関する情報がある場合のビット混合曲線の構成

を意味する。これにより 2 つの利点がある。一つは検査の効率が向上する。ARF でデータの存在性を検査する際、範囲管理インデックスをたどったあと、リーフテーブルをアクセスし、占有ビットを検査しなければならなかった。この改良により範囲管理インデックスをたどるだけで、検査が可能である。次に、範囲管理インデックスのサイズを小さくできる。本手法では多次元データを扱うため、データを表すビット長が長くなり、範囲管理インデックスの深さが大きくなりやすい。一方で、多次元データの空間の広さに対して、集約ノードの数は極めて少ない。このため、ARF の占有ビットが False の方が True に比べて圧倒的に多くなりやすい。この改良により、True の場合は 2 ビット、False の場合は 0 ビット割り当てることになるが、True の場合が False の場合より極めて小さくなるため、範囲管理インデックスをよりコンパクトにできる。

集約テーブルは集約値を保持するテーブルであり、各エントリーは範囲管理インデックスにおいて集約ノードと一対一で対応付けられる。集約テーブルのエントリーは 3 種類ある。1 つ目は集約エントリーである。集約エントリーは、それに対応する集約ノードが表す範囲における集約値（MIN,MAX,COUNT,SUM）を保持する。2 つ目はデータエントリーである。データエントリーはそれに対応する集約ノードが表す範囲内にあるデータのリストを持つ。このリストは 1 から数個とごく少数に限る。3 つ目は集約値不明エントリーである。つまり、このエントリーに対応する範囲にデータは存在するがそれらの集約値が不明であることを表す。

3.3 集 約 戦 略

クエリに応じてデータ構造をメンテナンスは ARF のアルゴリズムをベースに拡張する。基本的な考え方は、集約テーブルにおいて集約値不明エントリーを最小化することである。以下では、初期化時、定常時に分けて説明する。

3.3.1 初期化時の集約戦略

初期化時はワークロードに関する情報はないため、ブロックにあるデータ分布の情報だけを用いて範囲管理インデックスを構築する。ここでの基本戦略は、データ分布が密の領域は集約エントリーを、疎な領域はデータエントリーを構築することである。応用によっては、事前に典型的なクエリ範囲の幅に関する情報が得られることがある。例えば、時間に関する属性は日、週、月などが典型的なクエリ範囲の幅である。このような情報が得られるのであれば、目標ヒット率をもとに集約範囲および

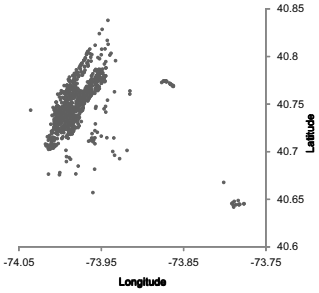


図 7　データセットのプロット。左上の大きな塊がマンハッタン、右下の小さな塊が JFK 空港。

用いるビット混合曲線を適切に選択する。

3.3.2　定常時の集約戦略

定常時は ARF のメンテナンスアルゴリズムを適用する。ただし、容量制約のため縮退する中間ノードを決定する際、集約テーブルのエントリーの種別により優先度をつける。最も積極的に選ぶべきなのは、集約値不明エントリーである。なぜなら、最も性能への影響が少ないからである。逆に最も消極的に選ぶべきなのはデータエントリーである。データエントリーは広い集約範囲をカバーしており、これ以上縮退することは集約値不明エントリーに遷移することになるためである。

4.　評　　価

提案手法の有効性を確認するため、集約データ構造のデータサイズおよび多次元集約クエリを実行時のヒット率について評価する。評価には、ニューヨーク市のタクシーのログデータ 1ヶ月分（約 1270 万件）のデータセット [3] を用いた。このデータの 1000 件をプロットしたものを図 7 に示す。アプリケーションとしては、タクシーが客を乗せた場所・時刻の時空間統計を想定した。集約単位としては、緯度、経度は 0.01 度 (13.3 ビット) と 0.1(16.6 ビット)、時刻は 1 週間 (16.9 ビット) とした。

それぞれの最小の集約単位のビット長から、それぞれ 3 ビットずつ少なくしたビット分を下位側に配置するビット混合曲線 2 種類を提案カーブとした。比較対象として、位置情報データでよく用いられる Z カーブを選択した。経度、緯度、時刻に関するビットをそれぞれ `x`、`y`、`t` とすると、比較対象の Z カーブは `(xyt)32`、提案カーブは `(xy)5(txy)27t5`、`(xy)15t12(xy)7t8(xy)10t12` と表現できるビット混合曲線である。

また、この評価では、範囲管理インデックスでは上位 60 ビットだけを保持し、データエントリーの最大エントリー数を 1 とした。今回は初期状態での評価するため、クエリに応じた適応的な範囲管理のメンテナンスは実施しない。

以下では、提案するデータ構造のノード構成、サイズ、および、クエリ実行時のヒット率に関して評価する。

4.1　ノード構成

範囲管理インデックスのノード構成について評価する。この評価では集約ノードのうち、集約エントリーに関連付けられているものは密な集約ノード、データエントリーに関連付けられているものは疎な集約ノードと表記する。評価結果を図 8 に

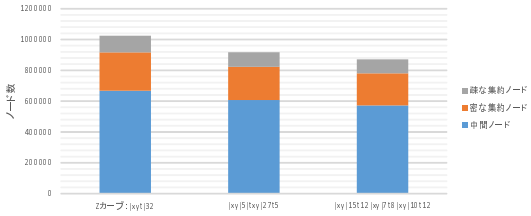


図 8　ノード種別の内訳

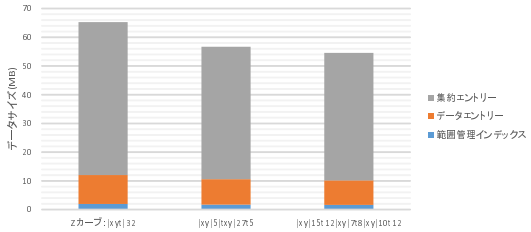


図 9　多次元部分集約データ構造のサイズの内訳

示す。いずれの曲線を用いた場合でも、集約ノードのおよそ 30%はエントリー数が 1 つしかない疎な集約ノードであった。本データセットは地理的にはニューヨーク近郊、時間的には 1ヶ月と比較的局所的にデータが密集している。また、範囲管理インデックスでは 96 ビット中上位 60 ビット分程度と大きい範囲で集約している。それでも、集約ノードの 3 割がエントリー数が 1 であるということは疎な領域を特別扱いする十分な理由があるといえる。

4.2　サ　イ　ズ

提案する多次元部分集約データ構造のサイズについて評価する。評価結果を図 9 に示す。用いた曲線によりサイズは変わるが、およそ 55-65MB 程度であった。元のデータサイズが 1.1GB であり、その 5-6%程度のサイズであることから、十分にコンパクトであるといえる。内訳をみると集約エントリーが 8 割近くを占めている。提案手法では、疎な領域を集約エントリーの代わりにデータエントリーとして扱った。この効果を試算すると、約 10MB のサイズ圧縮となった。曲線によるサイズの違いは、集約エントリーの数の差に起因している。今回は、曲線は最小となる集約単位をもとに選択したが、集約エントリー数を削減する観点での曲線を選択する方法があるかもしれない。

4.3　ヒット　率

クエリ実行時に集約結果が利用されるヒット率について評価をする。ここでは、想定する集約単位として、緯度・経度が 0.01 および 0.1 度、時間が 1 週間を想定したので、これらの組み合わせである 2 種類のクエリパターンについて評価した。

まず、緯度・経度の範囲の幅が 0.01 度、時刻の範囲の幅がが 1 週間であるクエリパターンの場合について、集約ノードへのアクセス数およびヒット率を図 10 にまとめる。まず、Z カーブのアクセス数は、提案カーブのアクセス数と比較して、5 割り増しである。これは、クエリ範囲と集約領域との交差する範囲が狭い、あるいは、不必要に交差していると言える。このことから、集約単位を考慮した空間充填曲線を選択した効果があ

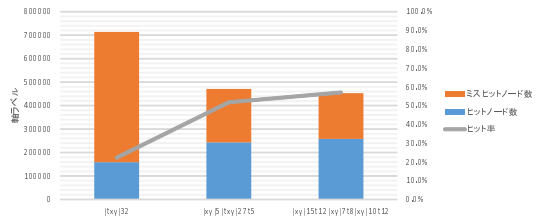


図 10　ヒット率 (緯度・経度:0.01 度単位、時間:1 週間単位)

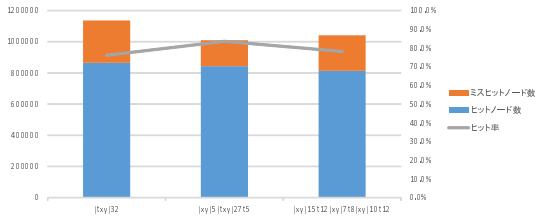


図 11　ヒット率 (緯度・経度:0.1 度単位、時間:1 週間単位)

るといえる。次に、ヒット率について調査する。ヒット率とは、クエリを実行したときにクエリ範囲と交差した集約ノード数を分母に、その集約ノードのうち、正確な集約値を求めるために元のデータをアクセスしなくて済んだ集約ノード数を分子にして求まる値である。提案カーブは、集約ノードの各次元の幅が、各次元の集約単位の幅に対して、1/8 の幅、すなわち、ビット長に関して 3 ビット分短い幅になるように設計している。このため、理論的なヒット率は、77%の 3 乗である 45.6%である。Z カーブはこれよりずっと低い 27%程度であるが、提案カーブは理論値よりも 10%高い 55%程度である。これは、疎な集約領域に対してデータエントリーを持つことで、完全にクエリ領域に内包されなくても正しく処理できる効果が出ているためと考えられる。

次に、緯度・経度の範囲を 0.1 度に広げたクエリパタンについて評価した。どの曲線を用いた場合も、アクセス数の差はほとんどなくなり、ヒット率は 75%から 80%を示すようになった。このことから、集約領域が集約範囲よりも十分大きければ、アクセス数は範囲が広くなる分、増えるがヒット率は急速に改善することが分かる。

5. 関 連 研 究

データを範囲で区切り、集約値を予め求めておく手法の研究の歴史は長い。小さなブロックに分割し、そのブロックの集約値を求めておく手法のうち、初期の研究に当たるものは、Small Materialized Aggregation(SMA) [2] が挙げられる。近年、これをクラウドデータベースを想定にプロキシ型で実現する手法として PA-Proxy [?] がある。

データが多次元の場合、このブロックをどのように分割するかが問題となる。クエリパターン、データ分布を考慮して最適なブロックに分割することは NP 困難な問題であることが知られている [5]。そこで、ボトムアップ的なアプローチとして、[5] では、クラスタリングアルゴリズムで近似的な解を求める手法を提案している。一方、トップダウン的なアプローチとして、[4] では、クエリパターンに適合する空間充填曲線を用いる方法がある。

範囲の分割方法に関しては、データアクセスに対して適応的に決める方法として Adaptive Range Filter(ARF) [1] がある。ARF はブルームフィルターのようにその範囲内のデータの存在性を検査するためのデータ構造として設計されたものである。これを部分集約値を保持できるように拡張されたものとして APA 木 [6] がある。

本提案手法は APA 木に近いが、一次元の範囲から多次元の範囲を扱うための拡張と、疎な領域を扱うための拡張を入れている。多次元の範囲を扱うために [1] では空間充填曲線を用いる方法について言及している。本研究では、一般的な空間充填曲線ではなく、[4] で提案されているクエリパターンに対して適応する空間充填曲線の設計手法を用い、想定する集約単位に対して集約ノードのアクセス数やヒット率を改善する方法を提案した。また、定性的な分析に基づき、疎な領域を特別扱いする有効性についても評価で示した。

6. お わ り に

多次元データに対する集約演算を対象に、多次元データ集合の部分集約値を管理するコンパクトなデータ構造を提案した。特徴としては、想定する集約単位に合わせた空間充填曲線を用いて多次元範囲に拡張した点と、疎な領域を特別扱いするようにした点である。これらの工夫により、データ構造のサイズとしては 10%の削減、また一般的な空間充填曲線を用いた場合に比べ、集約ノードのアクセス数を 2/3、ヒット率を 2 倍に改善した。さらに、ヒット率に関しては、疎な領域を特別扱いすることで、理論値よりも 10%ヒット率を改善した。

文　　献

[1] K. Alexiou, D. Kossmann, and P.-A. Larson. Adaptive range filters for cold data: Avoiding trips to siberia. *Proceedings of the VLDB Endowment*, 6(14):1714–1725, 2013.

[2] G. Moerkotte. Small materialized aggregates: A light weight index structure for data warehousing. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 476–487, 1998.

[3] NYC Taxi & Limousine Commission. TLC Trip Record Data. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`.

[4] N. Shoji and Y. Haruo. Quilts: Multidimensional partitioning framework based on query-aware and skew-tolerant space-filling curves. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, SIGMOD '17, 2017. (to appear).

[5] L. Sun, M. J. Franklin, S. Krishnan, and R. S. Xin. Fine-grained partitioning for aggressive data skipping. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1115–1126, New York, NY, USA, 2014. ACM.

[6] 小山田昌史 and 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. 第 8 回データ工学と情報マネジメントに関するフォーラム (DEIM), 2016.