

# 2016 年度 首藤研究室 輪講 第 4 回

青木 優介

2016 年 4 月 13 日

## 概 要

この章では、ファイルの属性 (*attribute*) や内容に基づくクエリを提供する方法を紹介する。ファイルの属性に基づくクエリを提供する方法は、多次元インデックスベース (*multi-dimensional index based*) と複数属性インデックスベース (*multi-attribute index based*) の 2 種類が存在し、前者を採用するシステムの例として VBI-Tree と SSP、後者を採用するシステムの例として Mercury を解説する。ファイルの内容に基づくクエリを提供する方法は 3 種類存在する。マッピングベース (*mapping-based*) を採用するシステムの例として CISS と ZNet、距離ベース (*distance-based*) の例として M-Chord と SIMPEER、ハッシングベース (*hashing-based*) の例として LSH Forest を解説する。

## 4.1 多次元データ共有

多くのファイル共有アプリケーションでは、属性 (*attributes*) によってファイルの特徴が表現される。この属性に基づくクエリを提供する方法は以下の 2 通りがある。

- 多次元インデックスベース (*multi-dimensional index based*): 各属性を多次元空間の次元とし、クエリは多次元空間内を進む。CAN は多次元インデックスを最初に提供した P2P システムである。CAN は kd-Tree に類似する構造のため、直接多次元データを配置することができる。CAN 以降のシステムも、多次元インデックスを提供するために R-Tree、X-Tree、M-Tree などの伝統的な多次元インデックス木構造を採用している。VBI-Tree は、木の根付近がボトルネックになることを避けるため、これらとは異なる木構造を採用している。
- 複数属性インデックスベース (*multi-attribute index based*): 属性ごとにインデックスを保持し、クエリはインデックスごとに進む。MAAN (*Multi-Attribute Addressable Network*) は全ての属性を一つの Chord リングで配置することで、複数属性インデックスを提供する。クエリは支配属性 (*dominate attribute*) に基づいて進み、他の属性はフィルターとして使われる。Mercury はルーティングハブ (*routing hub*) と呼ばれる、属性ごとの Chord リングを保持する。アイテムは全てのルーティングハブに送られる必要がある。

スカイラインクエリ (*skyline query*) とは、自身より全ての属性で優れているファイルが存在しないようなファイルを探索するクエリを指す。SkyPeer はスカイラインクエリを提供するために、効率的に返答を得ることが可能な部分空間スカイラインクエリを利用する。SkyPeer はローカルなスカイラインの計算を最適化し、不必要なデータ転送を減らすため、閾値ベースアルゴリズム (*threshold based algorithm*) を使用している。DSL は並列にスカイラインの探索を行うが、ノードは計算を開始するために、直前のノードの計算が完了するのを待つ必要があり、応答時間が遅くなる。また、必要のない部分空間スカイラインの結果が送信されることでのオーバーヘッドも起こる。SSP は、最初にローカルなスカイラインの結果が必ず最終的なスカイラインに含まれるようなノードを探索することで、これらの問題を解決している。

### 4.1.1 VBI-Tree

VBI-Tree はバイナリツリーベースの多次元インデックスのツリー構造を採用しているフレームワークである。各ピアは間順 (*in-order*) で隣り合う、葉ノードと内部ノードのペアを管理する。葉ノードは、ある多次元の領域に属するデータのインデックスを保持するノードである。内部ノードは、その子ノードが管理する領域全体を覆う領域のルーティングを管理するノードである。内部ノードは、以下の 5 種類のリンクを保持する。

- 親リンク (*Parent link*): 親ノードへのリンク
- 子リンク (*Child links*): 子ノードへのリンク
- 隣接リンク (*Adjacent links*): 間順で隣り合うノードへのリンク
- 近隣リンク (*Neighbor links*): 同じレベルで左右に  $2^i$  離れたノードへのリンク

- 上方リンク (*Upside links*): 祖先のノードへのリンク

リンクはリンク先へのポインタだけでなく、リンク先のノードの担当する多次元領域の情報も保持する。ノード  $n$  がクエリを発行又は受信すると、クエリの範囲が自身の内部ノードの担当領域に含まれるか検査する。含まれるとき、クエリの範囲を担当する子リンクにクエリを転送する。加えて、クエリの範囲を完全に覆う担当範囲を持つもっとも近い祖先のノード  $a$  にクエリを転送する。 $a$  が存在するとき、 $n$  から  $a$  までのパス上の各祖先  $a'$  について、 $n$  と反対側の子ノードに近隣リンクを通じてクエリを送信する。この方法により、根付近でのボトルネックを作らずに、クエリの範囲を含んでいる担当範囲をもつ全てのノードにクエリを送信することが可能である。

VBI-Tree は、探索のパスを保持することでクエリのループを避け、また、祖先のノードの担当領域の変化に応じた上方リンクの修正が生じることを避けている。VBI-Tree は、discrete data という新しい概念を提供している。discrete data とは、内部ノードに保持された、担当領域に該当しないデータのことである。discrete data は上方リンクの修正コストを下げるために導入されたが、データが多く挿入、離脱される動的な環境では、なおコストは高い。また、ボトムアップの探索は、高レベルのノードでのボトルネックを避けるが、探索範囲が高レベルのノードの担当する領域を横切るとき、多くの葉ノードを探索する必要がある。

#### 4.1.2 Mercury

Mercury は複数属性インデックスを提供するために、属性ごとのインデックスを保持するハブを持つ。各ノードは一つ以上のハブに参加する。ハブ内のノードは円形に配置され、ハーモニック確率密度関数 (*harmonic probability distribution function*) に基づく近隣リンクを持つ。さらに、ハブ間でのルーティングのために、各ノードは他のハブ内のノードへのリンクを持つ。

アイテムがネットワークに挿入される時、各属性  $a$  についてハブ  $H_a$  に属性値に従い配置される。クエリは、支配属性に対応するハブを進む。各ノードは他のハブ内のノードへのリンクを保持しているため、支配属性に対応するハブに 1 ステップで送信することができる。その後、Chord リングの探索アルゴリズムでクエリは転送される。支配属性での探索結果が多い場合、クエリの性能は著しく低下してしまうため、支配属性を賢く選択することは、非常に重要である。

Mercury の欠点は、データを挿入、離脱するコストが高いことである。全てのハブに対して挿入、離脱を行う必要があるため、特に属性の数が多い場合、操作するハブ数が多くなりコストが高くなる。

#### 4.1.3 SSP

Skyline Space Partitioning (SSP) は P2P ネットワーク上でスカイラインクエリを提供するための方法である。基本的なアイディアは、多次元空間を各ノードの担当する領域に分割することである。ノードは BATON として配置され、担当領域の場所を  $z$ -curve の順に並べることで、1 次元インデックスを提供できる。ルーティングのために、各ノードは BATON のリンクの情報に加えて、以下の担当範囲の情報を保持する必要がある

- 領域番号 (*Region Number*): 領域の  $z$ -curve 順での位置の識別子
- データ範囲 (*Data range*): 領域が含んでいる値の範囲
- 分割履歴 (*Split history*): 領域が生成された時点からの分割を表す、値と次元の組みのリスト
- 次の分割次元 (*Next partition dimension*): 次に領域が分割されるときに、分割される次元

SSP でのスカイラインクエリは以下の 4 ステップで行われる。

- 最初にシステムはローカルなスカイラインクエリの結果が必ず最終的なスカイラインに含まれるようなノードを探索する。このノードは、2 次元の場合では  $(0.0, 0.0)$ ,  $(1.0, 0.0)$ ,  $(0.0, 1.0)$ ,  $(1.0, 1.0)$  のような、他の全ての点よりも優れている点を含むノードを探索することで発見可能である。このノードを STARTER ノードという。
- STARTER ノードに到達すると、ローカルなスカイラインを計算し、最も大きい支配領域を持つ点  $p_{md}$  を選ぶ。スカイラインの探索空間から、 $p_{md}$  の支配領域は除かれる。
- その後、STARTER ノードは探索範囲を含んでいるノードにクエリを送信する。各ノードはローカルなスカイラインを計算し、クエリを生成したノードに結果を送信する。
- 最後に、クエリを生成したノードは、受信したローカルなスカイラインの結果から全体のスカイラインを計算する。

SSP の欠点は、クエリのスピードが STARTER ノードに大きく影響されることである。STARTER ノードに到達できない場合や、 $p_{md}$  が存在しない場合、クエリのスピードは減少してしまう。

## 4.2 高次元インデックス (*high-dimensional indexing*)

ファイルの属性だけで完全にファイルの内容を表すことは不可能である．そのため，細かい粒度で探索可能なコンテンツベース探索が望まれている．コンテンツベース探索を提供する一般的な手法は，ファイルの特徴のベクトルを作成し，それらをインデックスに従い配置する方法である．しかし，特徴のベクトルの次元は非常に大きくなり，従来の多次元インデックスでは，効率が悪い．高次元空間データをインデックスに基づき配置するための新しい方法は，以下の 3 種類に大きく分類される．

- マッピングベースアプローチ (*mapping-based approach*): 高次元空間はインデックスの前に低次元空間にマッピングされる．複数次元クエリは，低次元でのクエリに変換されて実行される．多くのシステムは，空間充填曲線 (*Space Filling Curve, SFC*) を利用している．
- 距離ベースアプローチ (*distance-based approach*): 基準点 (*reference points*) と呼ばれる，事前に定められた点からの距離に基づいて，多次元オブジェクトは直接配置される．インデックスの値は，オブジェクトから最も近い基準点への距離と，その基準点の値の合計で定められる．
- ハッシングベースアプローチ (*hashing-based approach*): インデックスのために，局所性鋭敏型ハッシュを用いる．クエリは，インデックスに対応するハッシュ値の周辺を探索する．この方法では常に正確な返答が得られるとは限らないが，結果のエラー率が事前に定義した値以下であることを保証する．

一般的に，多次元インデックスを提供するため，P2P システムは集中型システムで研究された技術を採用している．しかし，それらの技術を，分散 P2P 環境に合うように適用させる必要が有る．例えば， $k$  近傍 ( $kNN$ ) クエリの集中型システムアルゴリズムは， $k$  個のオブジェクトが含まれるまで，徐々にクエリの範囲を広げていき，最も近い  $k$  個を選択する．P2P システムにこのアルゴリズムを直接適用すると，多くのメッセージが発生し，クエリのコストが高くなる．よって， $k$  近傍クエリのアルゴリズムは P2P システムに合わせて修正する必要がある．

### 4.2.1 CISS

CISS は，高次元インデックスを提供するために，多次元空間を 1 次元空間にマッピングするヒルベルト空間充填曲線 (*Hilbert Space Filling Curve*) を利用し，1 次元データは Chord にインデックスされる．最初に，各次元の値をビットキーに変換し，多次元空間をビットキーの集合で表す．その後，ヒルベルト空間充填曲線を用いて，ビットキーの集合を一つのビットキーに変換する．最後にその一つのキーの値を Chord にインデックスする．クエリは，多次元の値から 1 次元の値に変換され Chord リング上を探索する．多次元空間の一つの領域は，1 次元空間の複数の区間に変換されてしまうため，特に次元が大きいとき，クエリの効率は悪くなる．

### 4.2.2 ZNet

ZNet は，多次元空間を 1 次元空間にマッピングするために  $z$ -curve を利用する．データ空間は四分木のような方法で再帰的に分割される．各部分空間には， $z$ -curve での位置と分割のレベルに対応した一意のアドレスが割り当てられる．各ノードは各部分空間とその空間内に位置するデータを担当する．ノードは担当する領域のアドレスの順に Skip Graph に並べられる．ノードがクエリを発行又は受信すると，探索範囲を含んでいる部分空間のアドレスを調べ，その部分空間を担当しているノードにクエリを転送する．空間の分割の情報が不足しており，アドレスが完全には知ることが不可能であるときも，接頭辞を知ることが可能である．この場合，探索範囲に近い隣接ノードにクエリを転送する．

### 4.2.3 M-Chord

M-Chord は，高次元空間での類似検索 (*similarity search*) を提供する．M-Chord は以下の 2 段階で多次元インデックスを実現する．

1. 高次元データを 1 次元データにマッピングするために，iDistance を利用する．iDistance は，全体に知られた基準点を利用してデータ空間を分割し，データを最も近い基準点からの距離によってインデックスする．
2. 1 段階目で得られた 1 次元の値を Chord リング上にインデックスする．

iDistance では，データ空間を事前に分割の集合  $S = \{P_1, P_2, \dots, P_n\}$  に分ける必要がある．各分割  $P_i$  は基準点  $O_i$  と半径  $r_i$  によって表される．各  $P_i(O_i, r_i)$  が互いに重ならない区間  $[i \cdot c, i \cdot c + r_i]$  に相当するように定数  $c$  を選ぶ ( $keyO_i = i \cdot c$  は  $O_i$  が対応する 1 次元の値)．データオブジェクト  $D$  がシステムに挿入されると，最初に，データオブジェクトから最も近い基準点  $O_j$  を調べ， $D$  のインデックスの値を  $j \cdot c + dist(D, O_j)$  と計算する．そして，1 次元インデックスの値で Chord リングに挿入する．

$o$  を中心， $r$  を半径とするレンジクエリ  $QR(o, r)$  を行うとき，最初に検索範囲を含む分割を調べる．その後，検索範囲を含む各分割  $P_i(O_i, r_i)$  に対応する，1 次元レンジクエリ  $q[keyO_i + dist(O_i, o) - r, \max(keyO_i + dist(O_i, o) + r, keyO_i + r_i)]$  を Chord リング上で行う．

$o$  を中心， $k$  を探索するオブジェクトの個数とする  $k$  近傍 ( $kNN$ ) クエリは以下の 2 段階で行う．

1. 低コストな発見的手法で  $o$  から近い  $k$  個のオブジェクトを見つけ，それらのオブジェクトと  $o$  との距離の最大値  $\delta$  を求める．

2. レンジクエリ  $QR(o, \delta)$  を行い、その結果の中から、 $o$  から近い  $k$  個を選ぶ。

M-Chord の欠点として、オブジェクトを個々にインデックスする必要があるため、オブジェクトの個数が多いとき、インデックスのコストが高くなるという点が存在する。

#### 4.2.4 SIMPEER

インデックスのコストが高くなることを避けるため、SIMPEER はインデックスの前にオブジェクトを要約し、その要約のみをインデックスする。SIMPEER は、以下の 3 つのレベルのインデックス構造を利用する、スーパーピアベースのシステムである。

- 最も低いレベルでは、各ピアは自身の共有オブジェクトをインデックスする。さらに、自身の共有オブジェクトをクラスタとして、その要約を作成し、自身を担当しているスーパーピアに送信する。
- スーパーピアのレベルでは、担当しているピアから送られてきたクラスタの要約をインデックスする。また、それらのクラスタの要約でハイパークラスタを作り、他のスーパーピアに送信する
- 最も高いレベルでは、スーパーピアは自身のハイパークラスタと他のスーパーピアから受信したハイパークラスタのインデックスを作成する。

ピアはクエリを発行するとき、自身を担当しているスーパーピアにクエリを送信する。スーパーピアは、ローカルなクラスタの要約やハイパークラスタのインデックスに基づき、担当ピアや他のスーパーピアにクエリを転送する。

クラスタの要約をインデックスするために、SIMPEER は iDistance と同じように、全体に知られた基準点を利用しデータ空間を分割する。最初にクラスタの中心に最も近い基準点  $O_i$  を持つ分割  $P_i$  を調べ、クラスタの最も遠い点を基準点  $O_i$  に基づいて、1 次元のインデックスの値に変換する。レンジクエリは、探索範囲と重なっている部分空間を含む各分割  $P_i$  に対し、クエリの範囲のうち基準点  $O_i$  に最も近い点から、分割  $P_i$  の境界までを探索することで実行する。

SIMPEER には 2 つの欠点が存在する。1 つは、基準点とクエリ範囲の距離が近いとき、クエリ範囲の大きさに関わらず、広い範囲を探索する必要がある点である。もう 1 つは、分割はクラスタ全体を覆う必要があるため、探索空間が大きくなり、探索コストが高くなることである。

#### 4.2.5 LSH Forest

LSH Forest は、 $l$  個の LSH Tree を含むインデックス構造である。このシステムは、インデックスのために局所性鋭敏型ハッシュの集合  $\mathcal{H}$  を使用する。オブジェクト  $O$  は  $l$  個の LSH Tree にインデックスされる。各 LSH Tree で  $O$  には可変長  $x$  桁の識別子が割り当てられる。識別子は、 $\mathcal{H}$  から選んだ  $h_1(), h_2(), \dots, h_x()$  を利用して、 $h_1(O), h_2(O), \dots, h_x(O)$  と並べたものとなる。識別子の各桁が LSH Tree の根からのパスを表し、 $O$  を示す葉の位置が決まる。類似クエリを実行するには、クエリオブジェクトの識別子を同様に生成し、 $l$  個の LSH Tree で接頭辞が最長一致する葉をトップダウンに探す。それらの葉からボトムアップに木を走査しながら、到達したノードの子孫の葉に当たる類似のオブジェクトを集める。 $M$  個のオブジェクトを集めるまで走査を続け、 $M$  個の中から最もクエリオブジェクトに近いオブジェクトを返す。