

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、ファイルの記述や内容に基づくクエリを支持する方法を紹介する。ファイルの記述に基づくクエリを支持する方法は、多次元インデックスベース手法 (*multi-dimensional index based approach*) と複数属性インデックスベース手法 (*multi-attribute index based approach*) 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支持する方法は 3 種類存在する。マッピングベース手法 (*mapping-based approach*) を採用するシステムの例として CISS と ZNet、距離ベース手法 (*distance-based approach*) の例として M-Chord と SIMPEER、ハッシングベース手法 (*hashing-based approach*) の例として LSH Forest を解説する。

4.1 多次元データ共有

既存の多くのファイル共有 P2P システムは、ファイルタイトルによるクエリは支持しているが、ファイルの記述によるクエリも支持することが望ましい。ファイル共有アプリケーションでは、よく属性 (*attribute*) によってファイルの記述が表現される。この属性に基づくクエリを支持する方法は以下の 2 通りがある。

- 多次元インデックスベース手法 (*multi-dimensional index based approach*): 多次元インデックスベース手法では、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支持した 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支持するために 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、これらとは異なる木構造を採用しているフレームワークである。
- 複数属性インデックスベース手法 (*multi-attribute index based approach*): 複数属性インデックスベース手法では、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。複数属性インデックス手法の例を 2 つ紹介する。一つは MAAN (*Multi-Attribute Addressable Network*) である。MAAN は全ての属性を一つの Chord リングに索引付けることで、複数属性インデックスを支持する。クエリは優位属性 (*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。一方で Mercury では、各属性をルーティングハブ (*routing hub*) と呼ばれる、分割された Chord リングに索引付ける。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。

データ検索の質を向上させるためには、他のユーザの評価に基づくべきである。そのような特徴は、ソーシャルネットワークやコミュニティベースシステムで一般的である。ファイル内容を記述する追加の情報の例としては、ユーザによる種々の側面の評価がある。その場合、ユーザは他に優位なファイルがないようなファイルを検索することに関心がある。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るクエリである。SkyPeer はスカイラインクエリを支持するために、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 はピア内のスカイラインクエリの計算を最適化し、不必要なデータ転送の量を減らすため、閾値ベースアルゴリズム (*threshold based algorithm*) を使用している。DSL (*Distributed Skyline Query*) 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 (*Skyline Space Partitioning*) は、最初にノード内の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ことを保証した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、二分木構造に基づいた多次元インデックスを採用したフレームワークである。各ピアは間順走査 (*in-order traversal*) で隣り合っている葉ノードと内部ノードの組を管理する。葉ノードは、特定の多次元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*): 内部ノードの親ノードへのリンクを示す．
- 子リンク (*child link*): 内部ノードの子ノードへのリンクを示す．
- 近接リンク (*adjacent link*): 間順走査で隣り合うノードへのリンクを示す．
- 隣接リンク (*neighbor link*): 同じレベルで左右に 2^i 離れたノードへのリンクを示す．
- 上方リンク (*upside link*): 内部ノードの先祖ノードへのリンクを示す．

これらのリンクはリンク先へのポインタだけでなく，リンク先のノードが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 を探索する． 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 n の隣接ノードにクエリを送信する．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，離散型データ (*discrete data*) という新しい概念を支持している．離散型データとは，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．離散型データ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支持するために，各属性のインデックスを担当するハブにノードを分配する．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*) を使用した経路表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とき，各属性について対応するハブに属性値に従い配置される．クエリは優位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，優位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，優位属性を賢く選択することは，非常に重要である．

Mercury の欠点は索引付ける属性の数が大きいとき，データを挿入，離脱するコストが高いことである．その理由は，全てのハブに対して挿入，離脱を行う必要があるからである．

4.1.3 SSP

SSP は P2P ネットワーク上でスカイラインクエリを支持するための方法である．基本的な考え方は，多次元空間を各ノードが担当する多次元領域に分割することである．担当領域の場所を *z-curve* の順に並べることで，一次元インデックスを支持できる．ノードは BATON 内に配置される．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を示す．
- データ範囲 (*data range*): 領域が含んでいる値の範囲を示す．
- 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を示す．
- 次の分割次元 (*next partition dimension*): 領域が分割されるときに，次に分割される次元を示す．

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，システムはノード内のスカイラインの計算が必ず最終的なスカイラインに含まれることを保証したノードを探索する．このノードは，他の全ての点よりも優位な点を含むノードを探索することで発見しうる．このノードを **STARTER ノード** という．
- **STARTER ノード** に到達すると，ノード内のスカイライン結果を計算し，最も大きい支配領域を持つ点 p_{md} を選ぶ．探索空間から p_{md} の支配領域は除かれ，スカイライン探索空間がその時決定される．
- 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．各ノードはノード内の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，クエリの処理スピードが **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，クエリの処理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*)

ファイルの記述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支持する一般的な手法は，ファイルの特徴ベクトルを作成し，それらを索引付けする方法である．しかし，特徴ベクトルの次元は非常に大きくなるため，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索引付けされる．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計算される．
- 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支持するため，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支持するために，多次元空間を一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，一次元データは Chord に索引付けされる．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 に索引付けする．クエリは，多次元の値から一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，一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，多次元空間を一次元空間にマッピングするために z -curve を利用する．全データ空間を四分木のような方法で再帰的に分割する．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．各ノードは部分空間とその空間内に位置するデータのインデックスを担当する．ノードは担当する領域のアドレスによって 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．空間分割についての情報が不完全なために，ノードが部分空間での目的地のアドレスを完全に解決することができない場合でも，アドレスの接頭辞は知ることができる．この場合，目的の部分空間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，高次元距離空間での類似検索 (*similarity search*) を支持する．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. 高次元データを一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索引付けする．
2. 1 段階目で得られた一次元の値を Chord リング上に索引付けする．

iDistance では，データ空間を事前にパーティション (*partition*) の集合 $S = \{P_1, P_2, \dots, P_n\}$ に分割する．各パーティション 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 が対応する一次元の値)．データオブジェクト D がシステムに索引付けされるときは，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 $j \cdot c + dist(D, O_j)$ として計算する．ここで， $dist(D, O_j)$ とは D と O_j 間の距離を指す．そして，一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，最初に検索範囲と交差するパーティションを調べる．その後，検索範囲をと交差する各パーティション $P_i(O_i, r_i)$ に対応する，一次元レンジクエリ $q[keyO_i + dist(O_i, o) - r, \max(keyO_i + dist(O_i, o) + r, keyO_i + r_i)]$ を生成し Chord リングへ送る．

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 δ を求める。
2. レンジクエリ $QR(o, \delta)$ を実行し、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、オブジェクトの個数が多いとき、オブジェクトを個々に索引付けする必要があるため、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、その要約のみを索引付けする。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索引付けする。さらに、要約のまとめを生成するために、自身のオブジェクトを集め、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、各スーパーピアはクライアントピア (*client-peer*) から送られてきた要約のまとめを索引付けする。また、ハイパークラスタ (*hyper-cluster*) を作るために、これらの要約のまとめを集め、他のスーパーピアに送信する。
- 最も高いレベルでは、各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から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。スーパーピアでは、クエリはクエリ結果を保持する可能性のあるクライアントピアや、他のスーパーピアに転送されうる。

要約のまとめを索引付けするために、SIMPEER は iDistance と同じように、全データ空間をパーティションに分割することができる、全体に既知の基準点の集合を利用する。最初にクラスタの中心に最も近い基準点 O_i を持つパーティション P_i をにクラスタを割り当て、クラスタの最も遠い点を基準点 O_i に基づいて、一次元インデックスの値に変換する。レンジクエリを処理するために、基準点 O_i から形成され、クエリと交差する各パーティション P_i の間で、SIMPEER はクエリの範囲のうち基準点 O_i に最も近い点から、パーティション P_i の境界までを探索する必要がある。

SIMPEER には 2 つの欠点が存在する。1 つは、基準点とクエリの範囲のうち基準点に最も近い点の距離が近いとき、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、パーティションは各クラスタの最も遠い点を覆うために分割を大きくする必要があるため、データ分割間で大きな重複を生じさせる可能性があり、大きな探索空間やより高い探索コストを生じる点である。

4.2.5 LSH Forest

LSH Forest は、 l 個の LSH Tree の集合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。このインデックス構造では、挿入された各オブジェクト O は、 l 個の LSH Tree に l 回索引付けされる。各 LSH Tree では、 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} から選んだハッシュ関数 $h_1(), h_2(), \dots, h_x()$ を利用して、 $\{h_1(O), h_2(O), \dots, h_x(O)\}$ と連結させたものとなる。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、LSH Forest はクエリの識別子を生成するために、 \mathcal{H} から選択されたハッシュ関数を最初に使用する。その後、最大の接頭辞を持つ葉ノードがクエリの識別子と一致するのを見つけるために、1 個の LSH Tree を top-down で探す。これらの葉ノードから類似のオブジェクトを集めるために、1 段ずつボトムアップで LSH Tree を走査する。それらは、到達する内部ノードの子孫の葉ノードである。この反復処理は、少なくとも M 個のオブジェクトを集めるまで続ける。最後に、これら 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検索する。