空間情報をもつストリームデータに対する効率的なOLAPシステム提案

那須　勇弥† 　塩川　浩昭‡ 　天笠　俊之‡ 　北川　博之‡

†筑波大学情報学群情報科学類 〒305-8573 茨城県つくば市天王台 1-1-1

‡筑波大学計算科学研究センター 〒305-8573 茨城県つくば市天王台 1-1-1

E-mail: †yuuya.n@kde.cs.tsukuba.ac.jp, ‡{shiokawa, amagasa, kitagawa}@cs.tsukuba.ac.jp

あらまし　本稿では位置情報を持ったストリームデータに対して効率的に多次元的なOLAP分析を行うためのシステムを提案する．センサやIoTデバイス，ソーシャルストリームなどのストリームデータには位置情報などの空間的情報を含むものが多く登場しており，時空間情報に対する効率的な OLAP 分析が重要となっている． 空間情報データに対してOLAP分析を行う場合，空間情報を地名や住所などに変換するために空間データベースへに問合せする必要がある．しかし，ストリームデータの様に継続的に到着するデータに対して同様の問合せを行う場合，性能面において大きなボトルネックが生じる．そこで本稿では，空間情報を含むストリームデータに対する効率的なOLAP分析を実現するための,空間データベースを組合せたStreamOLAPシステムのアーキテクチャを提案する．本稿では提案手法の詳細を述べるとともに，実データを用いた評価実験により提案手法の効率性を示す．

キーワード　StreamOLAP，移動体，GPS

1. はじめに

近年，センサ，IoT デバイス，ソーシャルストリーム等の技術の急激な普及に伴い，連続的かつ無制限に到来するストリームデータが増加している．特に，位置情報を含む移動体ストリームデータの活用場面が増加しており，リアルタイムに到着する位置情報や，それに付随する様々な情報を効率的に集約，集計処理する技術は重要な要素技術となっている.例えば,Google社が提供するGoogle Mapsには，デバイスの位置情報を収集・集計しリアルタイムな渋滞情報を地図上に表示する機能が備わっている．この機能を利用することで，ユーザは渋滞の少ない交通ルートを選択することができる．

このような集約・集計を効率的に行う技術として，OLAP (Online Analytical Processing) [1]が存在する．OLAP はデータウェアハウスに対し複雑な集約分析を実行する分析基盤システムである．OLAP を発展させたものとして，ストリームデータに対するOLAP分析を可能とする StreamOLAP [2]や，空間情報に対するOLAP分析を可能とするSpatial OLAP [3,4]などが研究されてきた．しかし，上述した位置情報を持った移動体ストリームデータに対するOLAP分析システムはこれまであまり研究されていない．

本研究では，StreamOLAP のアーキテクチャに空間データベースを組合せ，空間情報をもつストリームデータに対するOLAP分析(以降は空間的StreamOLAP分析と呼ぶ)を可能とする新たなシステムを提案する.空間的 StreamOLAP 分析を実現するためには，位置情報を地名や住所などに変換する必要がある．提案するシステムアーキテクチャでは，地名や住所への変換処理を空間データベースが担う．しかし，ストリームデータの様に継続的に到着するデータに対して同様の問合せを空間データベースへ行う場合，性能面において大きなボトルネックが生じる．そこで本研究では，地名や住所への変換処理を高速化するために，移動体の性質を考慮した地名・住所のキャッシング手法を導入する．本手法は，移動体の位置情報の更新時に，キャッシュされた位置情報との距離を計算し，一定距離以下であれば空間データベースへの問合せを行わず，キャッシュされた情報を使用するという方法である．キャッシュした情報を参照することで，提案システムが空間データベースへの問合せを行う回数の削減を図る．評価実験を行った結果，提案手法は StreamOLAP システムを用いたナイーブな実装と比較して90%以上の精度を維持しながら最大で80%以上高速化できることを確認した．

2. OLAP (Online Analytical Processing)

データの多次元的な集約・集計分析を行うことのできる手法としてOLAP [1]が知られている．OLAPは蓄積されたデータからデータウェアハウスを構成し，複雑な集約分析を実行する分析基盤である．OLAP ではデータウェアハウス内のデータに対して多次元的なデータ分析を行うために，スキーマ間の関係を表すスタースキーマからデータキューブと呼ばれる多次元データモデルを構築する．データキューブは分析対象となる数値を格納したメジャーとそれに関連する複数の軸で構成される．このデータキューブに対して，ユーザはより詳細な粒度の集約結果を求めるドリルダウンや，粗い粒度の集約結果を求めるロールアップなどの処理を対話的に行い，求める次元の組み合わせでのメジャーの集約結果を獲得する．

例として TPC-H ベンチマーク [15]に準じたデータを使用した OLAP 分析を，図 1，2，3 を用いて説明する．このデータはある顧客が発注した売上情報と lineitem を表すデータが保存されており，それぞれの売り上げ情報は part，supplier，customer の 3 つの軸に関連付けられている．

このデータの関係を表すスタースキーマを図 1 に示す．図 1 の lineitem は売り上げ情報を示すスキーマである．lineitem はカラムとして p_key，s_key，c_key を持っており，これらはそれぞれ part, supplier, customer スキーマへの外部キーとなっている．また，sales は売り上げ金額を示すカラムである．図 2 に customer 軸の階層構造の例を示す. customer の name(Taro, Jiro, Alice, Bob)はいずれかの nation(Japan, America)に属するような階層構造を持っている.図 1 のスタースキーマから，図 3 のデータキューブを構築することができる．図 3 の左のデータキューブは part，supplier，customer の次元の粒度が name であった場合に構築されるデータキューブである．左のデータキューブの下部には customer 軸の階層構造を示している．また，右のデータキューブは part，supplier の次元の粒度が name で customer の次元の粒度が nation であった場合に構築されるデータキューブである．左のデータキューブを customer に関してロールアップすると，階層構造に基づいて customer 軸の Taro の列と Jiro の列が Japan の列に, Alice の列と Bob の列が America の列に集約され，右のデータキューブが構築される．これに対して右のデータキューブをドリルダウンすると，Japan の列での売り上げ金額が Taro の列，Jiro の列に，America での売り上げ金額が Alice 列，Bob 列に分割され，左のデータキューブが構築される．

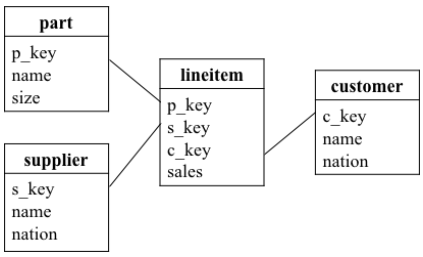

図 1 スタースキーマ

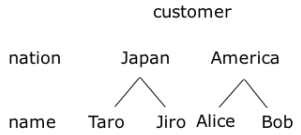

図 2 customer 軸の階層構造

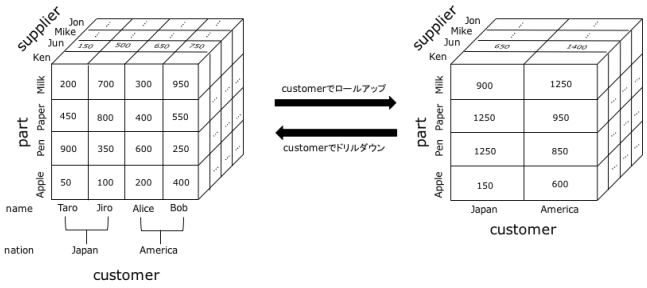

図 3 ドリルダウンおよびロールアップ

3. 関連研究

3.1. StreamOLAP

StreamOLAP [2]は，ストリームデータに対してリアルタイムに OLAP 分析を実行するための OLAP 分析システムである．既存の OLAP システム [1]は事前に蓄積したデータからデータウェアハウスを構築する必要があるため，時々刻々と変化するデータに対してリアルタイムに OLAP 分析する機能を持たない．この問題を解決するために，StreamOLAP はストリームデータに対して問合せ処理を行うことのできる SPE (Stream Processing Engine) [5,6,7]を用いたシステムアーキテクチャを提案している．

StreamOLAP では，SPE がストリームデータを集約する役割を担い，SPE が集約したデータを蓄積するバッファに対して OLAP Engine が OLAP 処理を行う．OLAP 処理において，ある次元階層の集約結果は，その次元階層より細かい粒度の次元階層における集約結果から求めることができるという性質を持っている．この性質を利用して StreamOLAP は，あらかじめ細粒度の集約結果を実体化してバッファに蓄積しておき，ユーザからの問合せを受けてからバッファ内のデータを用いてより複雑な集約処理をオンデマンドで実行することで，効率的な OLAP 分析を行うことができる．

図 4 に StreamOLAP のシステムアーキテクチャの図を示す．例として 2.1 節で用いた TPC-H のデータセットに対する StreamOLAP の OLAP 処理を説明する．なお，例では supplier，customer，lineitem の軸のみを対象としている．SPE に(Ken，Taro，100)というデータが到着したとすると，まず，SPE 内の Window と呼ばれるバッファに到着したデータが格納され，あらかじめ登録されたクエリ(ここでは[s_name, c_name, sales])に従って集約演算が実行される．その結果，Window 内に既に格納されていたデータ(Ken，Taro，200)と新たに到着したデータ (Ken, Taro, 100) が集約され，データキューブ内のデータは(Ken，Taro，300)に更新される．集約後，SPE から出力されたデータは OLAP Engine 内のバッファに蓄積される．

ユーザからのリクエストクエリが SPE に集約させた

ものと同一であった場合はOLAPエンジン内のバッファに蓄積されているデータを結果として返す．リクエストクエリが SPE に集約させたものと異なる場合(ここでは customer 軸の階層が nation に指定されている[s_name，c_nation，sales]であったとする)は，OLAPエンジンがバッファ内のデータからオンデマンドに処理を実行し(Ken，Japan，400)を出力する．

StreamOLAP は文字列や数値で構成されるストリームデータに対して効率的にOLAP分析を実行することが可能である．しかしながら，本研究で対象とする空間情報に対してOLAP分析を実行する機能を持たない．ゆえに，空間情報を持ったストリームデータに対するOLAP分析を実行することができない．

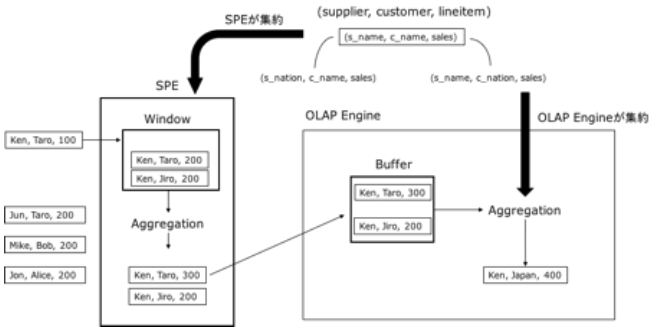

図 4 StreamOLAP のシステムアーキテクチャ

3.2. SpatialOLAP

Spatial OLAP [3,4]は，空間データベース [13]を利用して位置情報を持つデータに対して特定の領域内に含まれたデータに対するOLAP分析を可能とするOLAP分析システムである．Spatial OLAPは分析の軸として幾何学データを含む空間次元を持ち，幾何学データの領域内に含まれる位置情報を持つデータを空間検索[14]することで集約処理する．

図5に日本の行政区画を空間次元とした場合のロールアップの実行例を示す．日本の行政区画は都道府県の下に市区町村が存在するという構造になっている．これを空間次元として市区町村の領域内に含まれる位置情報を持つデータの件数を集約したとする．このとき,左の地形において3つの市区町村に対して,4件(地形の上部)，6 件(地形の左下部)，3 件(地形の右下部)という集約結果が得られている．都道府県レベルの集約結果は市区町村レベルをロールアップすることで得ることができるため，市区町村レベルの集約によって得られた結果を合計し 13 件という集約結果を得ることができる．

本研究では，ストリームデータを分析することのできる StreamOLAP と空間情報を分析することのできるSpatial OLAP を統合し，空間的 Stream OLAP 分析を可能とするシステムを提案する．

図 5 日本の行政区画でのロールアップの例

4. システムアーキテクチャ

本研究で提案する空間的ストリームOLAP分析を可能とするシステムのアーキテクチャを示す．アーキテクチャが分析対象とするデータには緯度・経度からなる位置情報が含まれているものとする．

3.1節で述べたように，StreamOLAPは数値や文字列などの単純なデータ型のみを対象としており，位置情報などの空間データを扱うためには，空間データを文字列や数値へ変換する必要がある．一般的に，Spatial OLAP においてドリルダウンやロールアップを行う場合，問合せの度に空間データベースによる空間集約が実行される．したがって，空間的ストリーム OLAP分析を行うためには，StreamOLAP の中でも空間データベースに対する問合せを行う必要がある．

そこで本研究では StreamOLAP に空間データベースを組み合わせたアーキテクチャを提案する．図6に本研究で提案するシステムアーキテクチャの概要を示す．まず，流れてきたデータを SPE が受け取る．SPE はデータに含まれる位置情報を文字列に変換するため，空間データベースへ問合せを行う．空間データベースは位置情報を受け取り，あらかじめ保存されている地形情報を用いて受け取った位置情報が含まれる空間を探すために空間検索を実行し，得られた地名・住所を返す．SPE は空間データベースへの問合せ終了後，集約を開始し結果をバッファに蓄積させる．空間データベースへの問合せ終了後の流れは従来の StreamOLAP の処理と同様である．

図7に Region，City，Prefecture の3つの階層から構成される空間次元を持つスキーマを考える．階層構造は Region が最も粒度の細かい階層で，Prefecture が最も粒度の粗い階層であるとする.図6に示したように，入力データは Lon，Lat，Dim1，Dim2，Measure の 5つの要素を持っているものとする．Lon には移動体の経度の値が，Lat には移動体の緯度の値が入るものと

する. また, SPE からの出力データは Prefecture, City, Region, Dim1, Dim2, Measure の 6 つの要素を持っている.

(lon1, lat1, a, A, 100)という値を持ったデータが到着したとすると, 空間データベースへ位置情報(lon1, lat1)を送る. 空間データベースは最も粒度の細かい階層である Region において, 受け取った位置情報を含む地形を空間検索する. 例では, region1 の geometry が lon1, lat1 の示す位置情報を含んでいる. region1 の属する City, Prefecture の値はそれぞれ city1, prefecture1 であるため, 空間データベースは SPE に空間次元の値となる prefecture1, city1, region1 を返している. 結果として, SPE から出力されるデータは(prefecture1, city1, region1, a, A, 100)となる.

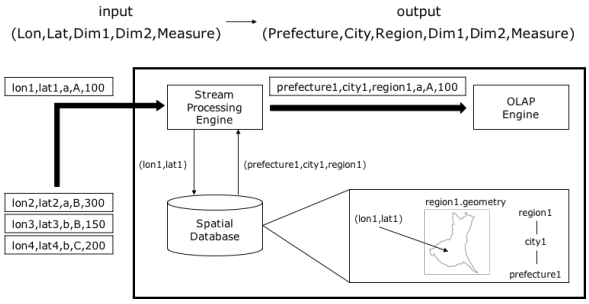

図 6 システムアーキテクチャ

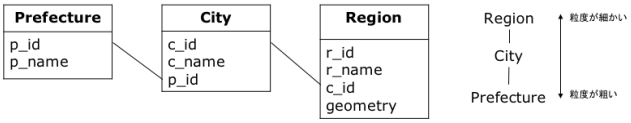

図 7 空間次元のスキーマと階層構造の例

5. キャッシュを用いた地名変換高速化

4 節で述べた提案システムアーキテクチャはストリームデータが到着する毎に, 空間データベースへの問合せが発生するため, 空間情報の地名変換に要する実行時間がボトルネックとなる可能性がある. そこで本節では, キャッシュを用いた地名変換の高速化手法を提案する.

5.1. 移動体の位置情報に関する仮説

ストリームデータとして送られてくる移動体の位置情報は短いスパンで継続的に到着する. 一般的に, そのような短い時間で移動体が移動できる距離は短くなることが予想される. ゆえに, ある移動体の位置情報の更新時に空間データベースへ問合せた地名・住所と前回更新時の地名・住所は一致する可能性が高いと考えられる. 仮に, 更新時の地名・住所が前回の結果と一致するのであれば, 毎回空間データベースへ問合せを行う必要はなく, 前回の結果をキャッシュし, そのキャッシュから結果を返すことで同様の結果を得ることができると考えられる.

5.2. 提案アルゴリズム

6.1 節で述べた仮説に基づき, 位置情報更新時の移動距離を判定基準とした地名のキャッシング手法を提案する. 図 8 に提案手法を組み込んだシステムアーキテクチャを示す. 提案手法の処理の流れを Algorithm 1 に示す. まず, ある移動体の位置情報が得られた際に, その移動体の id と位置情報, 空間データベースへ問い合わせて得た地名・住所をキャッシュに保存する. キャッシングする情報は移動体の id をキーとしたハッシュテーブルに保存される. その移動体の次回の位置更新時, まずはキャッシュに保存した位置情報と, 更新時の位置情報から移動距離を計算する. 移動距離は式(1)の Hubeny の公式 [12]を用いて計算する.

$$d = \sqrt{\left(d_y M\right)^2 + \left(d_x N \cos \mu_y\right)^2} \tag{1}$$

ただし, 式(1)で使用している M, N, および W は以下に示す式(2), (3), および(4)を用いてそれぞれ計算する. また, 式(1)から(4)で用いた各記号の意味は表 1 に示した通りである.

$$M = \frac{a(1-e^2)}{W^3} \tag{2}$$

$$N = \frac{a}{W} \tag{3}$$

$$W = \sqrt{1 - e^2 \sin^2 \mu_y} \tag{4}$$

表 1 式中の記号の意味

記号	意味
d_x	緯度の差
d_y	経度の差
μ_y	緯度の平均値
e	第一離心率
a	赤道半径

更新時の移動距離があらかじめ設定したしきい値以下であればキャッシュから地名・住所を返す. 図 8 ではこのようにキャッシュに保存されている位置情報(lon1, lat1)と更新された位置情報(lon1', lat1')との距離がしきい値以下であった場合の処理が示されている. 更新された位置情報(lon1', lat1')とともに送られた移動体の id(1)を基にハッシュテーブル内にキャッシュされている位置情報を読み取る. 図 8 では(lon1, lat1)と(lon1', lat1')との距離がしきい値以下であった

ため,キャッシュに保存されている地名を返している.更新時の移動距離があらかじめ設定した距離を超える場合，空間データベースへ問合せを行い，獲得した地名・住所を返した上でキャッシュの位置情報，地名・住所を更新する.

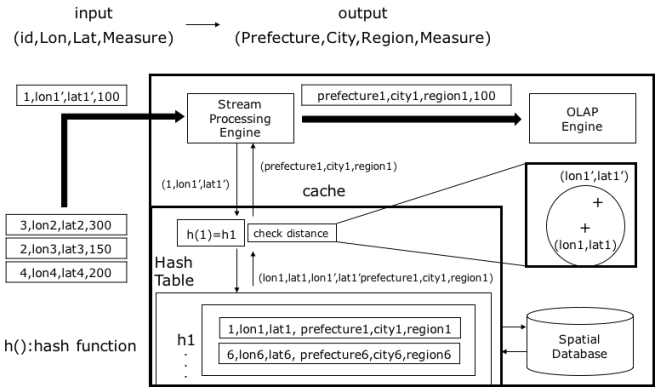

図 8 キャッシュを用いたシステムアーキテクチャ

Algorithm 1:caching algorithm

Input: *id*, the new longitude *lon'*, the new latitude *lat'*
Const: the threshold *d*
Output: *prefecture*, *city*, *region1*, *region2*
begin
  h ← *hash*(*id*)
  if exist *hashTable*[*h*][*id*] **then**
    lon ← *hashTable*[*h*][*id*][:longitude]
    lat ← *hashTable*[*h*][*id*][:latitude]
    if *calculateDistance*(*lon*, *lat*, *lon'*, *lat'*) < d **then**
      prefecture ← *hashTable*[*h*][*id*][:prefecture]
      city ← *hashTable*[*h*][*id*][:city]
      region1 ← *hashTable*[*h*][*id*][:region1]
      region2 ← *hashTable*[*h*][*id*][:region2]
    else
      prefecture,
      city,
      region1,
      region2← *searchSDB*(*lon'*, *lat'*)
      hashTable[*h*][*id*][:longitude] ← *lon'*
      hashTable[*h*][*id*][:latitude] ← *lat'*
      hashTable[*h*][*id*][:prefecture] ← *prefecture*
      hashTable[*h*][*id*][:city] ← *city*
      hashTable[*h*][*id*][:region1] ← *region1*
      hashTable[*h*][*id*][:region2] ← *region2*
    end
  else
    prefecture,
    city,
    region1,
    region2 ← *searchSDB*(*lon'*, *lat'*)
    hashTable[*h*][*id*][:longitude] ← *lon'*
    hashTable[*h*][*id*][:latitude] ← *lat'*
    hashTable[*h*][*id*][:prefecture] ← *prefecture*
    hashTable[*h*][*id*][:city] ← *city*
    hashTable[*h*][*id*][:region1] ← *region1*
    hashTable[*h*][*id*][:region2] ← *region2*
  end
end

図 9，10 にキャッシュを用いた地名，住所変換処理の例を示す．図 9 では移動距離がしきい値以下であった場合の処理の流れを示している．まずは，更新された位置情報とキャッシュに保存されている位置情報から移動距離を計算する．図 9 では移動距離がしきい値以下であるためキャッシュから直接地名を獲得することができる.図 10 では移動距離がしきい値を超えていた場合の処理の流れを示している．この場合は更新された位置情報から空間データベースへ問合せを行い，地名を獲得する．さらに，キャッシュの位置情報と地名を更新する．

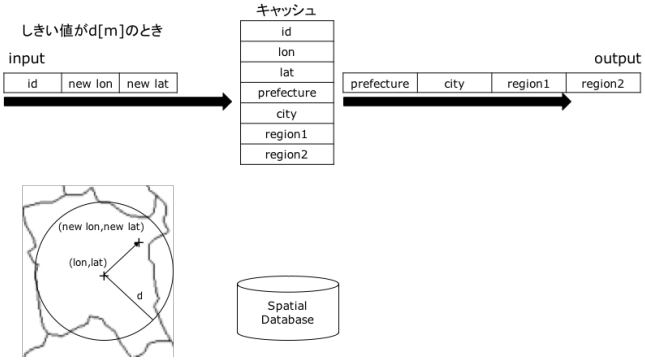

図 9 移動距離がしきい値以下の場合の変換処理

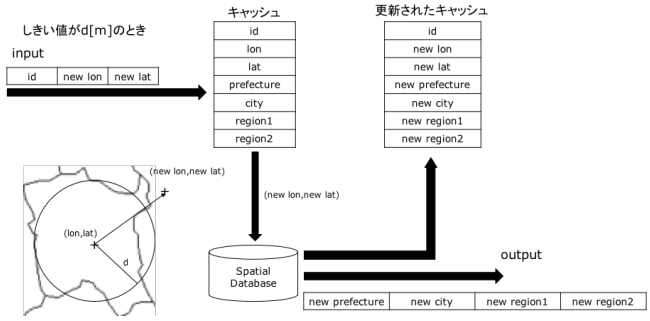

図 10 移動距離がしきい値を超える場合の変換処理

6. 評価実験

6.1. 実験概要

本節では提案システムアーキテクチャおよびキャッシュを用いた地名変換の高速化手法に関して，実行時間と精度の観点から有効性を評価する.

本実験では東京大学空間情報科学研究センターの「人の流れプロジェクト」[9]が提供する「2008 年東京都市圏 人の流れデータセット」を分析対象データセットとする.また,SPE として株式会社 Preferred Networks が開発した SensorBee [8]を用いる．OLAP エンジンは Go 言語を用いて実装した．さらに，位置情報の地名，住所への変換を担う空間データベースとして PostGIS

[11]を用いた．また，空間次元として日本の行政区画を用いる．行政区画の地形データは総務省統計局が提供する「平成 22 年国勢調査(小地域)」[10]のデータを PostGIS に格納して使用する．行政区画の階層構造は表 2 に示す通りになっている．3 行目の region1 は city が群または政令指定都市である場合には町村・区，それ以外の場合は空白が格納されている．city が群でも政令指定都市でもなかった場合は空白となる．このデータセットに対して,図 11 のようなスタースキーマを構築した．また，本実験で用いた実験環境は表 3 の通りである

表 2 空間次元(日本の行政区画)の階層構造

prefecture	県
city	市町村，特別区(東京)，群
region1	町村(群)，区(政令指定都市)
region2	町丁，字等

表 3 実験環境

OS	OS X EI Capitan
CPU	2.9GHz Intel Core i5
メモリ	16GB

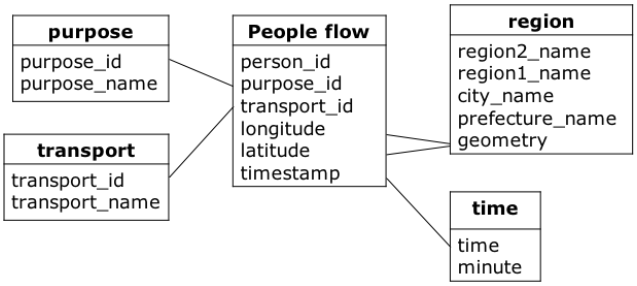

図 11 データセットのスタースキーマ

人の流れデータの中から 1 万人の 10 回分の移動データ（合計 10 万件）を SPE に送信し，1 件目のデータ入力開始時から最後のデータが SPE から出力されるまでの時間を処理時間として計測する．比較項目は提案手法の地名変換処理とキャッシュを用いない場合の地名変換処理の 2 つである．また，参考として地名変換処理を実行しない従来の StreamOLAP の処理時間も計測する．

精度については，1 つの位置情報に対して，キャッシュから得られた地名と空間データベースから得られた地名を比較した結果一致したものを正解とみなし，その正解率を精度として算出する．本実験に用いる空間次元は 4 段階に分けられている．4 段階全ての地名が一致するものを 4 段階一致，prefecture，city，region1 まで一致するものを 3 段階一致，同様に 2 段階一致，1 段階一致と定義し，それぞれの段階での精度を測る．

人の流れデータには約 60 万人を対象にそれぞれの 1 分間毎の位置情報が移動手段や移動目的などのデータとともに 24 時間分記録されている.データセット内には停滞中のデータ(ここでは更新時の位置情報が前回の位置情報と完全に一致するものとする)も含まれることから比較的人の移動の多い通勤・通学や帰宅の時間帯とそれ以外の時間帯では停滞中データが含まれる割合が変化する．そこで本実験ではデータセットを各時間帯に分割し，それぞれの時間帯のデータセットに対して評価を行う．同様に，本実験は停滞中データを含めたデータと停滞中データを除いたデータの 2 つに分けた評価についても行う．

6.2. 停滞中データを含む場合の実験結果

図 12，13 に 0~1 時の時間帯に対する処理時間と精度を示す．また，図 14，15 に 8~9 時の時間帯に対する処理時間と精度を示す.

図 12，14 はそれぞれの時間帯のデータセットを用いた場合の処理時間をしきい値毎に比較した実験結果である．それぞれのグラフからキャッシュを用いない場合と比較して最大で 80%以上処理時間を短縮できていることが明らかとなった．0~1 時の時間帯に対する処理時間(図 12)はどのしきい値を選択しても処理時間に変化が見られなかった．これは停滞中データの割合が高いことが原因であると考えられる．8~9 時の時間帯に対する処理時間(図 14)はしきい値を大きくするほど処理時間が短縮されている．しきい値 100[m]以下では処理時間を 60%程度，しきい値が 1000[m]以上では処理時間を 80%以上短縮できている．これはしきい値を大きくするほど空間データベースへの問合せ回数が減少することが原因であると考えられる.

図 13，15 はキャッシュ更新のしきい値を変化させた場合の精度を比較した実験結果である．最も精度の低い条件は図 15 に示される 8~9 時の時間帯のしきい値 10000[m]の場合での 4 段階一致の精度であるが，その場合でも 90%近い精度を維持できることが明らかとなった.

実験結果から，データセットに停滞中データが含まれる場合，90%以上精度を維持したまま処理時間を最大で 80%以上短縮できることが明らかになった.また，通勤･通学や帰宅の時間帯(8~9 時など)のように停滞中データの割合が少ない場合でも，しきい値を大きくすることによって 90%近い精度を維持したまま処理時間を 80%以上短縮できることが明らかとなった.

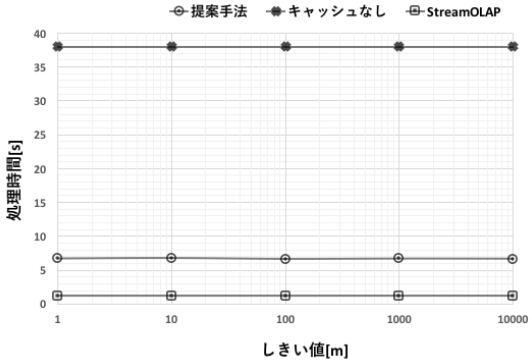

図 12 0~1 時の時間帯における処理時間の比較

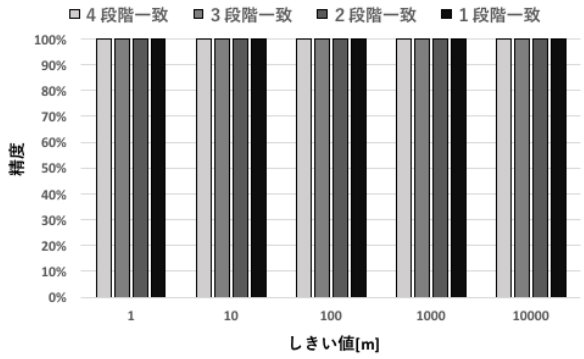

図 13 0~1 時の時間帯における精度の比較

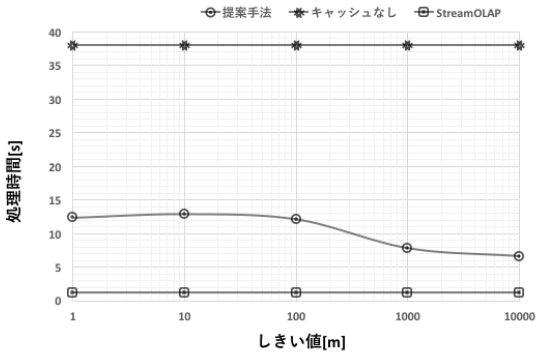

図 14 8~9 時の時間帯における処理時間の比較

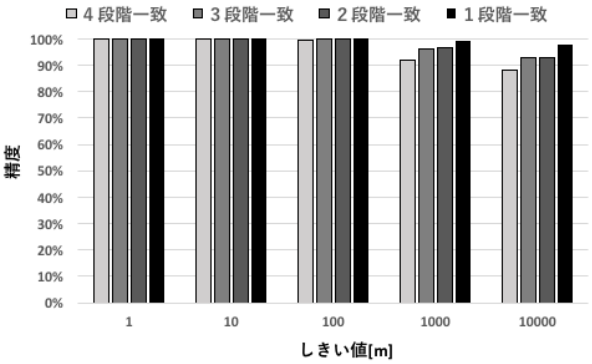

図 15 8~9 時の時間帯における精度の比較

6.3. 停滞中データを除く場合の実験結果

停滞中のデータを除くデータセットに対して処理時間と精度を評価した結果を図 16，17 に示す．

図 16 は停滞中のデータを除いた場合の処理時間を示す実験結果である．図 16 からしきい値 100[m]の場合の処理時間は 10%程度短縮されていることが明らかとなった．また，しきい値 1000[m]では 60%程度，しきい値 10000[m]では 80%以上処理時間が短縮されていることが明らかとなった．しきい値 100[m]以下の場合の処理時間が 1000[m]，10000[m]の場合と比較してあまり短縮されていないのは移動体の移動距離がしきい値を超えるものが多く，結果として空間データベースへの問合せ回数が増加することが原因であると考えられる．

図 17 は停滞中データを除くデータに対する地名変換の精度を空間次元の段階別に比較した実験結果である．図 17 からしきい値が 100[m]以下の場合は，どの段階でも 95%以上の割合で正しい地名を得られることが明らかとなった．また，しきい値 1000[m]の場合は 4 段階一致の精度は 50%程度であるが，3 段階一致以下の精度であれば 90%を超えることが明らかとなった．3 段階一致, 2 段階一致の精度にほとんど違いが見られないのは，政令指定都市や群の数が少ないことから region1 に値を持つ土地が少ないことが原因であると考えられる．

実験結果から，常に移動する移動体のみを対象とした場合，しきい値 1000[m]以上であれば 60%以上処理時間を短縮可能であることが明らかとなった．ただし，その場合の 4 段階一致の精度は 50%以下となり，実用的であるとはいえない．しかし，3 段階一致以下であれば，しきい値 1000[m]の場合でも 90%以上の精度を維持できるため，市区町村レベルの地名変換に対しては有効であるといえる．

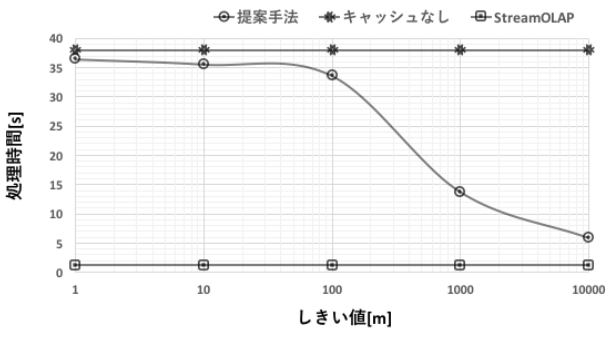

図 16 停滞中データを除いた場合の処理時間の比較

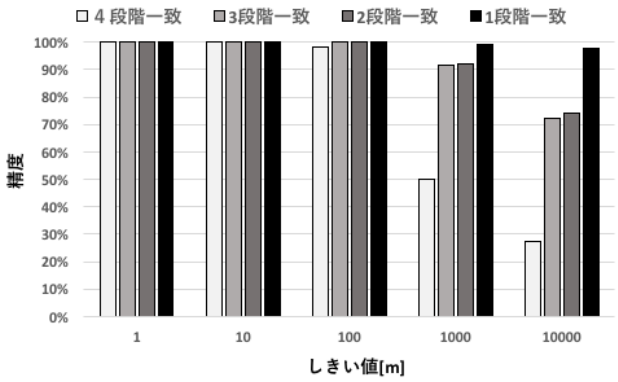

図 17 停滞中データを除いた場合の精度の比較

7. まとめと今後の課題

本研究では StreamOLAP と Spatial OLAP を組合せ，空間情報を持つストリームデータに対する OLAP 処理を可能とする空間的 StreamOLAP のシステムアーキテクチャを提案した．StreamOLAP のアーキテクチャを基に空間データベースを組合せ，位置情報を地名などの文字列に変換することで StreamOLAP の枠組みの中で空間情報を扱うことが可能となった．プロトタイプシステムのアーキテクチャは地名変換処理において処理時間に大きなボトルネックを持つ．この問題を解決するために移動体の性質を考慮したキャッシング手法を提案し,性能評価実験を行った.実験を行った結果，キャッシュを用いない場合と比較して 90%以上の精度を維持しながら最大で 80%以上処理時間を短縮できることが明らかとなった.

今後の課題として，空間的 StreamOLAP システムのための GUI の開発や，行政区画ではなく道路や線路を空間次元として扱う手法の検討に取り組む予定である.

謝　辞

本研究の一部は，理研「実社会ビッグデータ利活用のためのデータ統合･解析技術の研究開発」,ならびに，NICT 高度通信・放送研究開発委託研究「欧州との連携による公共ビッグデータの利活用基盤に関する研究開発」による.

参　考　文　献

[1] S. Chaudhuri et al, An Overview of Data Warehousing and OLAP Technology. ACM SIGMOD Record, 26 (1), pages 65-74, March 1997.

[2] Kosuke Nakabasami,Toshiyuki Amagasa,Salman Ahmed Shaikh,Franck Gass and Hiroyuki Kitagawa. An Architecture for Stream OLAP Exploiting SPE and OLAP Engine. In Proc. IEEE Big Data 2015, pp. 319-326, 2015

[3] J. Han, K. Koperski, and N. Stefanovic. GeoMiner: a system prototype for spatial data mining. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 553–556, 1997.

[4] S. Shekhar and S.Chawla. spatial Databases: A Tour. Prentice Hall, 2003.

[5] D. J. Abadi, D. Carney, U. C ¸etintemel, M. Cherniack, C. Con- vey, C. Erwin, E. F. Galvez, M. Hatoun, J.-H. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. B. Zdonik. Aurora: A Data Stream Management System. In ACM SIGMOD Conference, June 2003.

[6] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nizhizawa, J. Rosenstein, and J. Widom. STREAM: The Stanford Stream Data Manager. In ACM SIGMOD Conference, June 2003.

[7] S.Chandrasekaran,O.Cooper,A.Deshpande,M.J. Franklin,J.M.Hellerstein,W.Hong,S.Krishnamurthy, S. Madden,F.Reiss,and M.A.Shah. TelegraphCQ: Continuous Dataflow Processing. In ACM SIGMOD Conference, June 2003.

[8] SensorBee, http://docs.sensorbee.io/en/latest/preface.html/ .

[9] 人の流れプロジェクト, http://pflow.csis.u-tokyo.ac.jp/ .

[10] e-stat, http://www.e-stat.go.jp/SG1/estat/eStatTopPortal.do/ .

[11] PostGIS, http://www.postgis.net/.

[12] カシミール 3D ヒュベニの公式, http://www.kashmir3d.com/kash/manual/std_siki.htm / .

[13] Güting, R. H., An introduction to spatial database systems. The VLDB Journal—The International Journal on Very Large Data Bases, 3(4), 357-399.,1994.

[14] Samet, Hanan., The design and analysis of spatial data structures. Vol. 199. Reading, MA: Addison-Wesley, 1990.

[15] TPC-H TPC-H/Document, http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf/ .