大規模軌跡クラスタリングの類似度演算回数均一化による MapReduce モデルの分散処理効率化

鳥越　貴智†

† トヨタ自動車株式会社　〒 100-0004 東京都千代田区大手町 1-6-1 大手町ビル 6F
E-mail: †takatomo_torigoe@mail.toyota.co.jp

あらまし　人や車などの移動体について大規模な軌跡データが蓄積されるなか，類似軌跡を高速に集約する手法が求められている．中心ベースの軌跡クラスタリングを用いれば軌跡数 n に対する計算量は $\mathcal{O}(n)$ となるものの，実用的な分析需要としては，軌跡数が増大しても実行時間を比例させない処理方法の開発が喫緊の課題である．本研究では軌跡クラスタリングについて，MapReduce モデルに基づく分散データフローに加え，その中で支配的な類似度算出タスク群に対して待ち合わせ時間の少ない分散処理を行うため，演算回数を均一化した CDTW 近似計算法を提案する．人流軌跡の実データを用いて，軌跡数の増大に応じた計算資源を割り当て実行時間を抑えることで，その並列性が先行実装より高いことを示した．

キーワード　空間・時空間・時系列データ処理，並列・分散処理, 可視化・データ分類・クラスタリング, 効率性・スケーラビリティ, 軌跡の類似度

1　序　　論

GPS センサーが付いた IoT 端末の普及により，人や車などの移動データを広範囲に渡って収集できるようになった．移動データの社会的な活用例として，リアルタイムな人口分布の推計 [1]，COVID-19 の影響による訪問施設カテゴリー変化分析 [2]，被災状況把握のための通行実績道路マップ [3] などが挙げられる．これらの例では人流や交通流を地域・施設・道路といった単位で集計しているが，より各人・各車の移動について時系列で着目したい場合は，どこから出発して・どこを通り・どこへ向かったのか，その一連の軌跡を捉えることが望ましい．しかし全く同一の経路を通る軌跡は一般的に稀であるから，適切な粒度で捉えるためには類似した軌跡を集約するクラスタリング手法が求められる．

軌跡クラスタリング結果の活用先としては，たとえば観光地周遊経路の分析や推薦，幹線道路開通による交通流変化の分析や検知，などが想定される．また移動データはプライバシー性が高いため個別の軌跡は充分な保護を要するが，軌跡クラスタリングにより時系列性を保った統計処理を行えることも利点である．

なお移動データは年々増加しており，2021 年の日本においては携帯電話契約数が 2 億台 [4]，コネクティッドカーの新車販売台数が推定 370 万台 [5] といった規模である．そのため大規模に蓄積された移動データの分析結果を素早く社会に活用するためには，入力データが増大しても実行時間をなるべく抑えられるような計算手法が求められる．とくに分析の基礎的な前処理として全数処理の需要が高い軌跡クラスタリングについては，有用な計算手法の開発が喫緊の課題である．

以上の背景のもと，本研究では大規模な軌跡データのクラスタリングについて実行時間を低減する計算手法を提案する．本研究の貢献は次の 2 点に要約できる．1 点目は，軌跡クラスタリングについて MapReduce モデルに基づく分散データフローを示したことである．2 点目は，その分散データフローにおいて待ち合わせ時間を少なくするため，演算回数が均一な軌跡類似度の近似計算を示したことである．そして最大 357 万の人流軌跡を用いた実験によって，軌跡数が増大しても応じた計算資源を割り当てることで，実行時間の増加を抑えられたことを示す．

2　先 行 研 究

本研究は，軌跡クラスタリング手法の (k,l)-clustering，分散計算モデルの MapReduce モデル，軌跡類似度の CDTW に関連するため各々の先行研究について紹介する．

なお本研究では軌跡を，計測点を線分で繋いだ折れ線として定義する．すなわち d 次元の軌跡 P は，$\mathbb{R}^d$ の点列 $(p_1,\ldots,p_l)$ として捉えられる．隣接点間の距離の総和を軌跡長と呼び，点列の要素数を軌跡サイズと呼ぶ．軌跡 P について，媒介変数表示する場合は $P(t)$ として表記する．ここで媒介変数 t は区間 $[0,1]$ を渡り，$P(t)$ は始点 p_1 から終点 p_l まで軌跡上を等速に動くものとする．

2.1　(k,l)-clustering

汎用的なクラスタリング手法として，階層的クラスタリングや密度ベースクラスタリングがよく知られている．軌跡のような時系列データについても，それぞれの手法を基にしたものが多く提案されている [6]．これらは一般的にクラスタリングの前処理として，軌跡数 n について n 行 n 列の類似度行列を求める必要があり，その計算量は時間・空間ともに $\mathcal{O}(n^2)$ となる．

軌跡の表現としては道路 ID のような記号の列を扱うものと緯度経度のような点の列を扱うものに大きく分けられるが，点列については計算量に優れた手法として (k,l)-clustering [7] がある．これは，点のクラスタリング手法として代表的な k-means における中心点の代わりに，中心軌跡を用いる軌跡クラスタリング手法である．類似度についてはクラスタごとに求める中心軌跡と入力軌跡の間で測るため，計算量を $\mathcal{O}(kn)$ で抑えることができる．ここで k はクラスタ数，すなわち中心軌跡の数である．

(k,l)-clustering は，コスト関数の最小化問題として定式化できる．軌跡の集合 $\mathcal{P} = \{P_1, \ldots, P_n\}$ が与えられた時，次式のようなコスト関数の定義により，(k,l)-center と (k,l)-median という 2 種類の問題を設定できる．

$$\mathrm{cost}_{\mathrm{center}}(\mathcal{P}, \mathcal{C}) = \max_{i \in \{1,\ldots,n\}} \min_{j \in \{1,\ldots,k\}} \mathrm{similarity}(P_i, C_j)$$

$$\mathrm{cost}_{\mathrm{median}}(\mathcal{P}, \mathcal{C}) = \sum_{i=1}^{n} \min_{j \in \{1,\ldots,k\}} \mathrm{similarity}(P_i, C_j) \ / \ n$$

ここで l は中心軌跡のサイズ，similarity$(\cdot,\cdot)$ は 2 軌跡間の類似度である．最小化の変数である $\mathcal{C} = \{C_1, \ldots, C_k\}$ は k 個の中心軌跡を表しており，それぞれサイズ l の軌跡である．このコスト関数を式変形すると，

$$\mathrm{cost}_{\mathrm{center}}(\mathcal{P}, \mathcal{C}) = \max_{i \in \{1,\ldots,n\}} \mathrm{similarity}(P_i, C_{b_i})$$

$$\mathrm{cost}_{\mathrm{median}}(\mathcal{P}, \mathcal{C}) = \sum_{i=1}^{n} \mathrm{similarity}(P_i, C_{b_i}) \ / \ n$$

$$\text{where} \qquad b_i = \operatorname*{argmin}_{j \in \{1,\ldots,k\}} \mathrm{similarity}(P_i, C_j)$$

中心軌跡 $\mathcal{C} = \{C_1, \ldots, C_k\}$ と，入力軌跡の所属クラスタ番号 $\{b_1, \ldots, b_n\}$ を求める最小化問題として捉えられる．よって k-means と同様に，中心軌跡を求めるステップと所属クラスタを求めるステップを交互に繰り返す EM アルゴリズムによって近似解を得ることができる．

このような方針のもと，(k,l)-center について実用的なアルゴリズム [8] と実装 [9] が提案されている．(k,l)-median についても同様に実用的なアルゴリズム [10] とマルチスレッドで並列処理できる実装 [11] が提案されている．

2.2 MapReduce モデル

ビッグデータ処理のため広く用いられている分散計算モデルとして MapReduce モデル [12] がある．これは分散データを並列処理する Map フェーズ，再配置する Shuffle フェーズ，集約する Reduce フェーズに分けて計算するモデルである．

点のクラスタリングにおいては k-means の亜種 k-means|| [13] が，MapReduce モデルの実行エンジン Apache Spark の MLlib にて実装 [14] されており，そのスケーラビリティが実証されている．そのため k-means と同様の EM アルゴリズムが適用できる中心ベースの軌跡クラスタリングについても，MapReduce モデルによってスケールすることが期待できる．

2.3 CDTW

軌跡クラスタリングにおいて用いられる軌跡の類似度のうち，軌跡を点列として扱うものとしては DTW (Dynamic Time Warping) やフレシェ距離がよく知られている．どちらも軌跡間の適切なマッチングを取るものであり，DTW はマッチング点同士の類似度の和として求められ，フレシェ距離は最大値として求められる．そのためフレシェ距離の方が外れ値に鋭敏な指標となる．

GPS センサーはビルの谷間など衛星通信環境の悪いところでは測位誤差が大きくなるため DTW の方が望ましいものの，DTW は離散的なマッチングを取るため測位のサンプリング間隔による影響が大きい．そこで DTW のマッチングを連続化して，和を積分とした CDTW (Continuous Dynamic Time Warping) が移動軌跡の類似度としては適切と考えられる．CDTW はその積分の重みの付け方に様々な亜種 [15] があり，文献によっては平均フレシェ距離や積分フレシェ距離と呼ばれることもある．

本研究では，軌跡 $P(t)$ と軌跡 $Q(t)$ の類似度として CDTW を次式で定義する．

$$\mathrm{CDTW}(P,Q) = \inf_{\sigma} \frac{\int_0^1 s\Big(P(t), Q(\sigma(t))\Big) \Big(L(P) + L(Q)\dot{\sigma}(t)\Big) dt}{L(P) + L(Q)}$$

ここで $s(\cdot,\cdot)$ は 2 点間の類似度，$L(\cdot)$ は軌跡長を表す．この式は 2 つの軌跡長の比で重み付けした特殊な経路積分である．そして $\sigma(t)$ は軌跡の表示パラメータを変換するマッチング関数であり，積分値の下限を取る $\sigma(t)$ を得る問題として CDTW は捉えられる．

CDTW の $(1+\epsilon)$ 近似解を得るために，計算量が $\mathcal{O}(\frac{\zeta^4 l^4}{\epsilon^2} \log \frac{\zeta l}{\epsilon})$ のアルゴリズムが提案されている [16]．ここで ζ は 2 軌跡の任意の区間長の最大比であり，l は 2 軌跡のサイズである．より実用的なアルゴリズムとして，双方向ダイクストラ法によるヒューリスティクス [10] が提案されている．

3 提案手法

3.1 (k,l)-clustering の MapReduce モデルに基づく分散データフロー

(k,l)-clustering の EM アルゴリズムは，(k,l)-center と (k,l)-median の先行実装 [9] [11] を抽象化すると，擬似コード 1 のとおり書くことができる．ここで初期クラスタリングのアルゴリズムとして Gonzalez 法 [17] を用いており，matching$(\cdot,\cdot)$ は軌跡の類似度 similarity$(\cdot,\cdot)$ と関連したマッチング関数である．

このデータフローを MapReduce モデルに基づいて分散化したものを図 1 のとおり提案する．まず初めに入力軌跡を n 行のデータとして読みこみ，これに対して $P_x \leftarrow \mathrm{simplify}(P_x, m)$ のような入力軌跡を長さ m に簡約する Map 処理を行う．MapReduce モデルは，Shuffle フェーズにて計算ノード間で通信を行いデータを再配置して，次の Shuffle フェーズまでは各計算ノー

擬似コード 1 (k,l)-clustering の EM アルゴリズム [9] [11]

Input: 入力軌跡 $P_1,\ \ldots,\ P_n \in \mathcal{P}$
Input: クラスタ数 k, 中心軌跡サイズ l, ループ回数 $iter$
Output: 中心軌跡 $C_1,\ \ldots,\ C_k$ where $\forall j\,|C_j| = l$
Output: 所属クラスタ番号 $b_1,\ \ldots,\ b_n \in \{1,\ \ldots,\ k\}$

▷ 1. 初期クラスタリング
▷ 1a. 乱択した入力軌跡を簡約した中心軌跡を作成
$x \leftarrow \mathrm{sample}(\{1,\ \ldots,\ n\})$
$C_1 \leftarrow \mathrm{simplify}(P_x,\ l)$
▷ 1b. 最初のクラスタに所属を更新
for $i \leftarrow 1$ **to** n **do**
  $b_i \leftarrow 1$
  $sMin_i \leftarrow \mathrm{similarity}(P_i,\ C_1)$
for $j \leftarrow 2$ **to** k **do**
  ▷ 1c. 最も類似していない入力軌跡を簡約した中心軌跡を追加
  $x \leftarrow \mathrm{argmax}_i sMin_i$
  $C_j \leftarrow \mathrm{simplify}(P_x,\ l)$
  ▷ 1d. より類似している中心軌跡のクラスタに所属を更新
  for $i \leftarrow 1$ **to** n **do**
    $s_{i,j} \leftarrow \mathrm{similarity}(P_i,\ C_j)$
    if $s_{i,j} < sMin_i$ **then**
      $b_i \leftarrow j$
      $sMin_i \leftarrow s_{i,j}$

loop $iter$ **times**
  ▷ 2. 中心軌跡の更新
  for $i \leftarrow 1$ **to** n **do**
    ▷ 2a. 所属クラスタの中心軌跡から類似マッチング点を算出
    $M_i \leftarrow \mathrm{matching}(P_i,\ C_j)$
  for $j \leftarrow 1$ **to** k **do**
    for $o \leftarrow 1$ **to** l **do**
      ▷ 2b. 類似マッチング点を集めて中心点を算出
      $mSet_{j,o} \leftarrow \phi$
      for $i \leftarrow 1$ **to** n **do**
        if $b_i = j$ **then**
          $mSet_{j,o} \leftarrow mSet_{j,o} \cup \{m_{i,o}\}$
      $c_{j,o} \leftarrow \mathrm{center}(mSet_{j,o})$
    ▷ 2c. 中心点を並べて中心軌跡を更新
    $C_j \leftarrow (c_{j,1},\ \ldots,\ c_{j,l})$

  ▷ 3. 所属クラスタの更新
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** k **do**
      ▷ 3a. 全中心軌跡との類似度を算出
      $s_{i,j} \leftarrow \mathrm{similarity}(P_i,\ C_j)$
    ▷ 3b. 最も類似している中心軌跡のクラスタに所属を更新
    $b_i \leftarrow \mathrm{argmin}_j s_{i,j}$

ドで独立して並列実行するため，計算負荷に偏りがあるほど待ち時間が増え非効率となる．そのため軌跡クラスタリングで頻出する類似度計算や類似マッチングの計算量は均一にすることが望ましい．詳しくは次節にて述べるが，この計算量は両軌跡

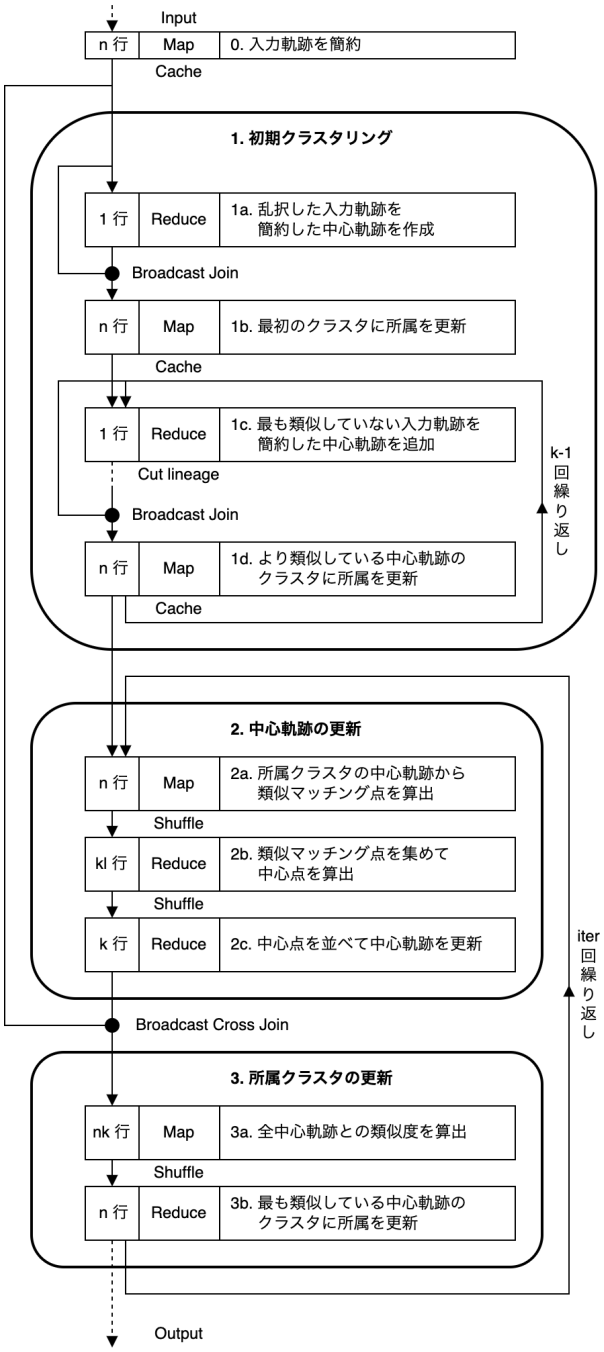

図 1　(k,l)-clustering の分散データフロー

サイズに強く依存するため，中心軌跡サイズを l に固定すると共に入力軌跡サイズを m に固定する．なお，この簡約済み入力軌跡データは以降繰り返し参照することになるためキャッシュする．

ステップ「1. 初期クラスタリング」では，中心軌跡の作成および追加を Reduce 処理として行う．この新しい中心軌跡データは 1 行と微量なため Broadcast して，入力軌跡データの分散配置はそのままに Join することで効率的に，入力軌跡と中心軌跡の類似度を測り所属クラスタを更新する Map 処理を行うことができる．この Broadcast Join の 2 回目以降は，計算グラフの系統である lineage を切るために一度外部出力したのち再読み込みすることが推奨される．なぜなら Apache Spark な

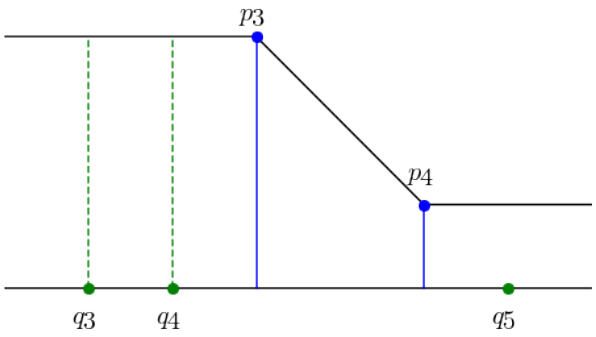

図 2　$p3$ と $[q4, q5]$ の内点までのマッチングを参照して，$p4$ と $[q4, q5]$ の内点までのマッチングを探索する場合

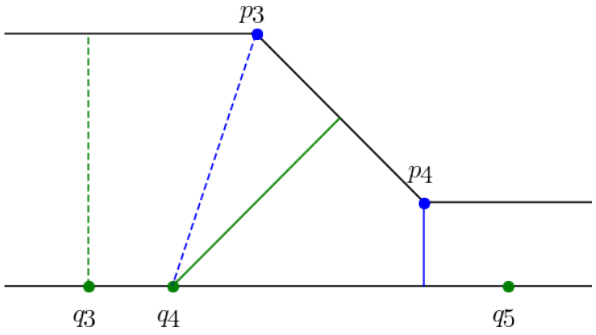

図 3　$[p3, p4]$ の内点と $q4$ までのマッチングを参照して，$p4$ と $[q4, q5]$ の内点までのマッチングを探索する場合

どの実行エンジンは，出力を求められてから lineage を辿って実行計画を立てるものの，繰り返し処理の中で複数回参照されるサブグラフが展開されてしまうと計算グラフが指数爆発してしまい，ボトルネックになってしまうからである．なお，この入力軌跡は次の繰り返しで 2 回使うためキャッシュする．

ステップ「2. 中心軌跡の更新」では，入力軌跡に Join 済みである所属クラスタの中心軌跡との類似マッチングを Map 処理として行う．それから所属クラスタ番号と中心軌跡を成す点列のインデックスによる kl 個の複合キーによって Shuffle して，それぞれの中心点を算出する Reduce 処理を行う．さらに所属クラスタ番号のみをキーに Shuffle して，中心点をインデックスによってソートして新しい中心軌跡として更新する Reduce 処理を行う．この新しい中心軌跡データは k 行であり一般的に n よりも充分に小さいため Broadcast して，入力軌跡データの分散配置はそのままに Cross Join することで効率的に，全入力軌跡と全中心軌跡の組み合わせデータを作ることができる．

ステップ「3. 所属クラスタの更新」では，全入力軌跡と全中心軌跡の類似度を算出する Map 処理を行う．それから所属クラスタ番号のみをキーに Shuffle して，最も類似している中心軌跡のクラスタに所属を更新する Reduce 処理を行う．

以上のステップ「2. 中心軌跡の更新」とステップ「3. 所属クラスタの更新」を $iter$ 回繰り返すことで，(k, l)-clustering の近似解を得る．

擬似コード 2 CDTW の動的計画法

Input: 入力軌跡 P, Q
Output: CDTW $result$

for $i \leftarrow 1$ **to** length(P) **do**
  for $j \leftarrow 1$ **to** length(Q) **do**
    if $i = 1$ and $j = 1$ **then**
      $MA_{1,1} \leftarrow ((p_1,\ q_1))$
    else if $j = 1$ **then**
      $MA_{i,1} \leftarrow \text{append}(MA_{i-1,1},\ (p_i,\ q_1))$
    else if $i = 1$ **then**
      $qq \leftarrow \text{argmin}_{q \in [q_{j-1},\ q_j]} s(p_1,\ q)$
      $MA_{1,j} \leftarrow \text{append}(MB_{1,j-1},\ (p_1,\ qq))$
    else
      $qq \leftarrow \text{argmin}_{q \in [MA_{i-1,j,i+j-1,2},\ q_j]} s(p_i,\ q)$
      $pairsA \leftarrow \text{append}(MA_{i-1,j},\ (p_i,\ qq))$
      $qq \leftarrow \text{argmin}_{q \in [q_{j-1},\ q_j]} s(p_i,\ q)$
      $pairsB \leftarrow \text{append}(MB_{i,j-1},\ (p_i,\ qq))$
      if fixedCDTW($pairsA$) < fixedCDTW($pairsB$) **then**
        $MA_{i,j} \leftarrow pairsA$
      else
        $MA_{i,j} \leftarrow pairsB$

    if $i = 1$ and $j = 1$ **then**
      $MB_{1,1} \leftarrow ((p_1,\ q_1))$
    else if $i = 1$ **then**
      $MB_{1,j} \leftarrow \text{append}(MB_{1,j-1},\ (p_1,\ q_j))$
    else if $j = 1$ **then**
      $pp \leftarrow \text{argmin}_{p \in [p_{i-1},\ p_i]} s(p,\ q_1)$
      $MA_{i,1} \leftarrow \text{append}(MA_{i-1,1},\ (pp,\ q_1))$
    else
      $pp \leftarrow \text{argmin}_{p \in [MB_{i,j-1,i+j-1,1},\ p_i]} s(p,\ q_j)$
      $pairsA \leftarrow \text{append}(MB_{i,j-1},\ (pp,\ q_j))$
      $pp \leftarrow \text{argmin}_{p \in [p_{i-1},\ p_i]} s(p,\ q_j)$
      $pairsB \leftarrow \text{append}(MA_{i-1,j},\ (pp,\ q_j))$
      if fixedCDTW($pairsA$) < fixedCDTW($pairsB$) **then**
        $MB_{i,j} \leftarrow pairsA$
      else
        $MB_{i,j} \leftarrow pairsB$

$i \leftarrow$ length(P)
$j \leftarrow$ length(Q)
$pairsA \leftarrow \text{append}(MA_{i,j},\ (p_i,\ q_j))$
$pairsB \leftarrow \text{append}(MB_{i,j},\ (p_i,\ q_j))$
if fixedCDTW($pairsA$) < fixedCDTW($pairsB$) **then**
  $result \leftarrow$ fixedCDTW($pairsA$)
else
  $result \leftarrow$ fixedCDTW($pairsB$)

3.2　演算回数が均一な CDTW の動的計画法

前節で述べたとおり MapReduce モデルは，Join を含む Shuffle フェーズにて計算ノード間で通信を行いデータを再配置して，次の Shuffle フェーズまでは各計算ノードで独立して並列実行するため，計算負荷に偏りがあるほど待ち時間が増え非効

率となる．そのため図 1 で頻出する類似度計算や類似マッチングの計算量は均一にすることが望ましい．この前提において，先行研究の双方向ダイスクトラ法 [10] のような計算量に偏りがありその予測も困難であるようなアルゴリズムは適していない．対して離散的な DTW は素朴な動的計画法で解ける代表的なアルゴリズムであり，長さ l の軌跡と長さ m の軌跡について支配的である 2 点間の類似度の演算回数は lm 固定であることが保証される．

そこで CDTW でも動的計画法を用いた近似計算のアルゴリズムを，擬似コード 2 のとおり提案する．ここで入力軌跡 P,Q について，メモ行列を二つ用意する．一つ目の行列 MA は，p_1 から p_i までの部分軌跡，q_1 から $[q_{j-1}, q_j]$ の内点までの部分軌跡，とのマッチングと CDTW をメモする行列である．二つ目の行列 MB は，p_1 から $[p_{i-1}, p_i]$ の内点までの部分軌跡，q_1 から q_j までの部分軌跡，とのマッチングと CDTW をメモする行列である．この二つのメモ行列を使って軌跡間のマッチングを貪欲に探索する．探索中のマッチングを所与のものとした CDTW を fixedCDTW$(\cdot)$ とする．

動的計画法における漸化式のキーアイディアを図 2 と図 3 に示す．$p4$ と $[q4, q5]$ の内点までのマッチングを探索する場合，図 2 では $p3$ と $[q4, q5]$ までのマッチングを $M1$ から参照して，$p4$ から $[q4', q5]$ の最も類似度の小さい点へのマッチングを追加する．ここで $q4'$ は参照したマッチングにおける $[q4, q5]$ の内点である．図 3 では $[p3, p4]$ の内点と $q4$ までのマッチングを $M2$ から参照して，$p4$ から $[q4, q5]$ の最も類似度の小さい点へのマッチングを追加する．この二つのうち，より fixedCDTW$(\cdot)$ が小さいものを $p4$ と $[q4, q5]$ の内点までのマッチングを決定する．

このようにして二つのメモ行列を使う動的計画法によって，P の節点から Q の内点へのマッチング，Q の節点から P の内点へのマッチング，その $l + m$ 個の最適な順番を貪欲に探索することができる．このアルゴリズムは長さ l の軌跡と長さ m の軌跡について，2 点間の類似度の演算回数は $4lm$ で固定であることが保証される．

4 実　　験

4.1 データセット

軌跡クラスタリングの対象として，実測によるオープンデータのうち最大規模のものである Datasets from the ATC shopping center [18] を用いた．これは大阪市のショッピングセンターで延べ 92 日間に渡り 3D センサで計測された人物位置の追跡データセットであり，複数の CSV ファイルから構成される．本実験では 36 億レコードを日付ファイル名および人物 ID フィールドの複合キーで集約したのち時刻フィールドで昇順にソートすることで 357 万軌跡を得た．また位置座標は X フィールドと Y フィールドのみを利用しており，2 次元軌跡として扱った．

4.2 実　　装

提案分散データフローは Apache Spark 3.3.0 を基盤として，Spark MLlib の Estimator を継承したクラスとして Scala 言語にて実装した．提案 CDTW 計算とマッチングは Spark SQL 関数として実装して，前述の Estimator 継承クラスの内部で呼び出した．この SQL 関数は Apache Spark の表形式データ構造 Dataset/DataFrame の内部を直接操作するものであり，UDF を用いた実装と比べてデータ変換のオーバーヘッドが少ない実装となっている．これらを利用して，本実験では (k, l)-median の問題設定のなか各アルゴリズムについて，入力軌跡の簡約は配列インデックスのサンプリング，初期中心軌跡の簡約は折れ線上の等距離サンプリング，中心点は重心，2 点間の類似度は平方ユークリッド距離，として実装した．またパラメータは $k = 16$，$l = 32$，$m = 32$，$iter = 20$ を設定した．入力については，データセットの CSV ファイル群の選択によって軌跡数 n を変化させ，スケーラビリティを検証できるようにした．

比較対象としては，(k, l)-median の先行実装 [11] を用いた．この先行実装は C++言語によって開発されており，共有メモリ型の並列ライブラリ OpenMP によって部分的にマルチスレッド化されている．ただしハードコーディングされている部分があるため，本比較実験では main 関数のみ差し替えた．パラメータは提案手法と同様に $k = 16$，$l = 32$，$m = 32$，$iter = 20$ を設定しているが，中心軌跡の更新時にコストが改善されない場合は $iter$ 回ループせずに早期打ち切りする工夫が為されている．各アルゴリズムも提案実装とできるかぎり同様のものを設定しているが，CDTW 計算として双方向ダイスクトラ法を用いていること，初期中心軌跡の簡約として CDTW による貪欲法を用いていることが異なる．なお CDTW 計算中に C++の例外が稀に発生することがあり，その場合はフレシェ距離にフォールバックするように関数をラップした．入力については 1 日分のデータ (2012 年 10 月 14 日) に絞って，先行実験と同様に軌跡ごとのファイル群に事前変換し，その読み込み数によって n を変化させ，スケーラビリティを検証できるようにした．

4.3 実 行 環 境

実験環境は，Amazon Web Services の東京リージョンで構築した．提案実装については，Apache Spark のマネージドサービスである Amazon EMR 6.9.0 を利用した．Apache Spark クラスタを構成するインスタンスとしては，マスターノードに m6i.4xlarge (16vCPU， 64GiB memory)，コアノードに m6i.12xlarge (48vCPU， 192GiB memory) を採用し，コアノードの台数を変えることによりスケーラビリティを検証できるようにした．Apache Spark クラスタのパラメーターについては，公式ブログの推奨 [19] に従ってドライバーと各エグゼキューターに割り当てる CPU 数やメモリ数を設定した．Apache Spark Web UI から確かめられる Total uptime を実行時間として計測し，Task Time を Total Time で割ることによりタスク CPU 使用率を求めた．

先行実装については，Amazon EC2 の Ubuntu 20.04.1 LTS イメージを利用した．インスタンスとしては m6i.16xlarge

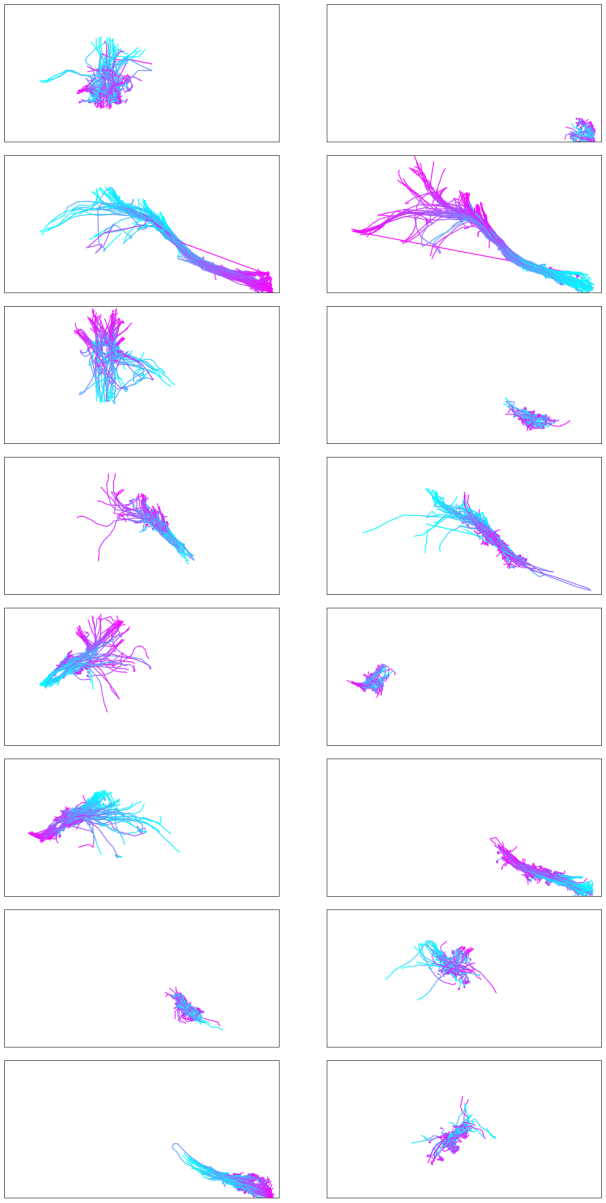

図 4　提案実装によるクラスタリング結果
$k = 16, l = 32, m = 32, iter = 20, n = 3\,578\,281$

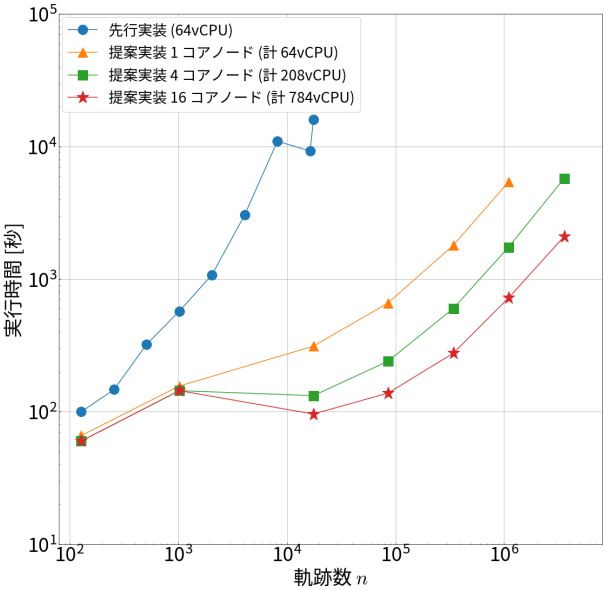

図 5　軌跡数に対する実行時間の評価
$k = 16, l = 32, m = 32, iter = 20$

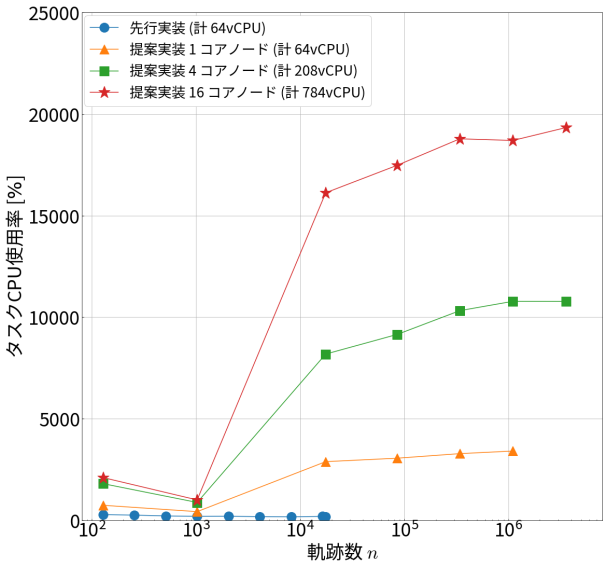

図 6　軌跡数に対するタスク CPU 使用率の評価
$k = 16, l = 32, m = 32, iter = 20$

(64vCPU, 256GiB memory) を採用し，Amazon EMR のマスターノード 1 台とコアノード 1 台を合わせた vCPU 数とメモリ量が等しくなるようにした．time コマンドによって出力される elapsed time を実行時間として計測し，user time を elapsed time で割ることでタスク CPU 使用率を求めた．

4.4　実験結果

まず提案実装の妥当性確認のため，コアノード 4 台の Apache Spark クラスタにより全軌跡 $n = 3\,578\,281$ をクラスタリングしたのち，クラスタごとに 100 軌跡をサンプリングしたものを図 4 に示す．軌跡ごとに始点から終点へ，水色から紫色へのグラデーションをかけて描画しており，始点・終点・経路の近しい軌跡群が同じクラスタにまとめられていることを確認できる．

次に軌跡数 n を変化させながら，実行時間・タスク並列度・クラスタリングコストを計測することで，そのスケーラビリティを評価した結果を図 5，6，7 に示す．提案手法については，コアノード 1 台，4 台，16 台の Apache Spark クラスタにて，軌跡数 $n = 17\,558$ (計 1 日)，$n = 85\,413$ (計 3 日)，$n = 340\,946$ (計 10 日)，$n = 1\,099\,538$ (計 31 日)，$n = 3\,578\,281$ (計 92 日) および比較のため $n = 124$，$n = 1028$ でクラスタリングをそれぞれ実行した．ただし 1 コアノードかつ $n = 3\,578\,281$ の場合のみ，実行中にエラーが発生し計測できなかった．メモリ不足によるものと推測される．先行実装については，軌跡数 $n = 128$ から $n = 16\,384$ まで 2 の累乗数および比較のため $n = 17\,558$ (計 1 日) でクラスタリングを実行した．

図 5 の実行時間については，軌跡数に関わらず提案実装の方が短い結果となった．$n = 17\,557$ で先行実装が 4.4 時間，計算資源が同じコアノード 1 台の提案実装が 5.2 分と 50 倍以上速

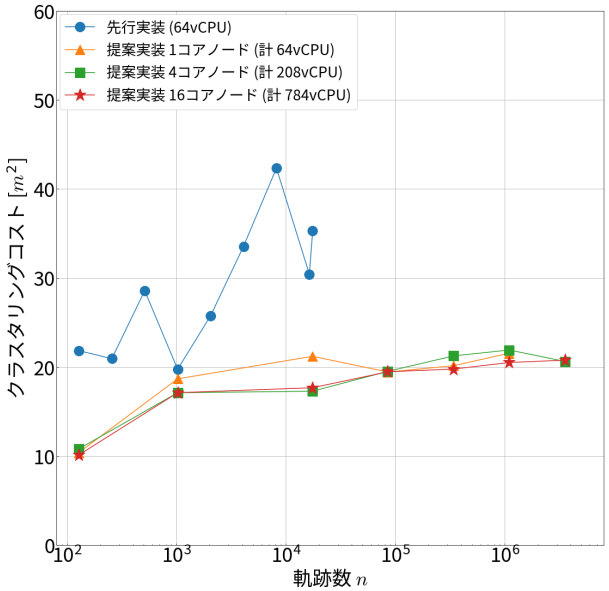

図 7　軌跡数に対するクラスタリングコストの評価
$k = 16, l = 32, m = 32, iter = 20$

かった．そしてコアノード数を増やすことにより更に高速化でき，その度合いは軌跡数が増加してもあまり変わらない．実行時間が同等の結果を比べると，コアノード 1 台で $n = 340\,946$ が 30 分，コアノード 4 台で $n = 1\,099\,538$ が 29 分，コアノード 16 台で $n = 3\,578\,281$ が 35 分となり，おおむね軌跡数が 3 倍程度増えてもコアノード数を 4 倍程度増やせば実行時間を伸ばすことなくクラスタリングできたことが分かる．なお $n = 124$，$n = 1028$ では，提案実装についてコアノード台数に関わらず実行時間がほぼ一定となっている．これは扱うレコード数が少ないため，実際には Shuffle を行わずコアノード 1 台で処理するような実行計画が立てられたためと推測される．

図 6 のタスク CPU 使用率についても，軌跡数に関わらず提案実装の方が高い結果となった．先行実装の OpenMP は 64 スレッド認識しているため理想的な上限値は 6400%となるが，$n = 64$ で 274%，$n = 17\,557$ で 169%と，あまりインスタンスのマルチコアを活かせていなかった．提案実装も n が少ない内はあまり高くないものの $n = 17\,558$ 以降は軌跡数の増加に伴ってゆっくりと上限に達していく傾向が見てとれる．実験内の最大値としては，1 コアノード (計 64vCPU) で 3406%，4 コアノード (計 208vCPU) で 10 775%，16 コアノード (計 784vCPU) で 19 337%と，クラスタサイズに応じた高い並列性を実現している．

図 7 のクラスタリングコストについても，軌跡数に関わらず提案実装の方が低く抑えられ，安定性も良い結果となった．ただし先行実装のコストは CDTW のフォールバックとしてフレシェ距離を設定した実行上の対応により，アルゴリズム本来の性能より悪くなっている可能性がある．また提案実装でコアノード数の増減によるクラスタリングコストへの影響はほぼ見られなかった．

5　結　　論

本研究では軌跡クラスタリングについて，(k, l)-clustering の MapReduce モデルに基づく分散データフローおよび，その中で支配的な類似度算出タスク群のため演算回数が均一な CDTW の動的計画法を提案した．また人流軌跡の実データを用いて，軌跡数の増大に応じた計算資源を割り当て実行時間を抑えることで，その並列性が先行実装より高いことを示した．そして 357 万軌跡という規模のクラスタリングも実用的に行えることを示した．

今後の展望としては，よりクラスタ数 k が大きい場合 k について逐次的な繰り返し処理となっている初期クラスタリングがボトルネックになることへの対処として，k-means|| [13] のような初期中心軌跡を一度に複数サンプリングするアイディアを取り入れ，より分散処理に適した軌跡クラスタリングの手法を開発することが考えられる．またリアルタイムに収集される移動データについて，事前に学習したクラスタへの所属や中心軌跡との類似度を求めることで，オンライン異常検知に応用することも期待できる．

なお提案実装は Apache Spark 拡張ライブラリとして `https://github.com/ToyotaInfoTech` にて公開予定である．

6　謝　　辞

本研究の遂行にあたり，トヨタ自動車株式会社コネクティッド先行開発部 InfoTech の吉岡顕主査，福島真太朗主幹，桑原昌広主幹には熱心なご指導を頂きました．心から感謝いたします．また，同室の園部竜也主任には日常的な議論において適切な助言をいただきました．お礼申し上げます．

文　　献

[1] 株式会社 NTT ドコモ．（お知らせ）モバイル空間統計の「国内人口分布統計（リアルタイム版）」の提供開始．`https://www.docomo.ne.jp/info/news_release/2019/12/03_00.html`, December 2019. (visited on 01/10/2023).

[2] Google LLC．感染症対策の専門家を支援。新型コロナウイルス感染症（covid-19）と戦う上で役立つデータを．`https://japan.googleblog.com/2020/04/covid-19-community-mobility-reports.html`, April 2020. (visited on 01/10/2023).

[3] トヨタ自動車株式会社．トヨタ自動車、ビッグデータを活用した新しい情報サービスの提供を開始．`https://global.toyota/jp/detail/1960877`, May 2013. (visited on 01/10/2023).

[4] 総務省．電気通信サービスの契約数及びシェアに関する四半期データの公表（令和 3 年度第 3 四半期（12 月末））．`https://www.soumu.go.jp/menu_news/s-news/01kiban04_02000205.html`, March 2022. (visited on 01/10/2023).

[5] 株式会社富士経済．コネクテッドカー (つながる車) の世界市場を調査―2035 年市場予測 (2020 年比)―．`https://www.fuji-keizai.co.jp/market/detail.html?cid=21122`, December 2021. (visited on 01/10/2023).

[6] Dimitrios Kotsakos, Goce Trajcevski, Dimitrios Gunopulos, and Charu C. Aggarwal. Time-series data clustering. In *Data Clustering: Algorithms and Applications*, chapter 15, pp. 357–380. CRC Press, August 2013.

[7] Anne Driemel, Amer Krivošija, and Christian Sohler. Clustering time series under the fréchet distance. In *Proceedings of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pp. 766–785. Society for Industrial and Applied Mathematics, December 2015.

[8] Kevin Buchin, Anne Driemel, Natasja van de L'Isle, and André Nusser. klcluster: Center-based clustering of trajectories. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '19, pp. 496–499, New York, NY, USA, November 2019. Association for Computing Machinery.

[9] Kevin Buchin, Anne Driemel, Natasja van de L'Isle, and André Nusser. Gitlab - klcluster sigspatial19. `https://gitlab.com/anusser/klcluster-sigspatial19/`, December 2019. (visited on 01/10/2023).

[10] Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov, and Sampson Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '20, pp. 99–110, New York, NY, USA, November 2020. Association for Computing Machinery.

[11] Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov, and Sampson Wong. Github - mesoptier/trajectory-clustering at v1.0. `https://github.com/Mesoptier/trajectory-clustering/tree/v1.0`, November 2020. (visited on 01/10/2023).

[12] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, Vol. 51, No. 1, pp. 107–113, January 2008.

[13] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable K-Means++. March 2012.

[14] Apache Software Foundation. Clustering - mllib - spark 1.5.0 documentation. `https://spark.apache.org/docs/1.5.0/mllib-clustering.html`, May 2015. (visited on 01/10/2023).

[15] Maike Buchin. *On the computability of the Frechet distance between triangulated surfaces*. PhD thesis, 2007.

[16] Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Approximating the integral fréchet distance. *Comput. Geom.*, Vol. 70-71, pp. 13–30, February 2018.

[17] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, Vol. 38, pp. 293–306, January 1985.

[18] Dražen Brščić, Takayuki Kanda, Tetsushi Ikeda, and Takahiro Miyashita. Person tracking in large public spaces using 3-D range sensors. *IEEE Transactions on Human-Machine Systems*, Vol. 43, No. 6, pp. 522–534, November 2013.

[19] Karunanithi Shanmugam. Best practices for successfully managing memory for apache spark applications on amazon emr. `https://aws.amazon.com/jp/blogs/big-data/best-practices-for-successfully-managing-memory-for-apache-spark-applications-on-amazon-emr/`, April 2019. (visited on 01/10/2023).