位置・キーワードに基づく kNN データモニタリングのための分散アルゴリズム

鶴岡　翔平† 　西尾　俊哉† 　天方　大地† 　原　隆浩†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1-5

E-mail: †{tsuruoka.shohei,nishio.syunya,amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし　近年，多くのアプリケーションではパブリッシュ/サブスクライブ（Pub/Sub）モデルに基づいてデータが配信されており，ユーザは自身が興味をもつデータのみを取得する．また，スマートフォンや SNS の普及により，データやクエリ (サブスクリプション) の数が増加しており，単一のワーカーで全ての処理を行うとリアルタイム性を保証することが困難になっている．本稿では，Pub/Sub モデルにおいて，複数のワーカーにクエリを分散して処理を分担することにより，各クエリに対して，クエリのキーワードを含むデータの中でクエリの指定位置に最も近い k 個のデータ（kNN データ）を効率的にモニタリングする問題に取り組む．本稿では，データの追加だけではなく，クエリの追加および削除がおきる環境を想定する．また，これらのイベントに対して各ワーカーの処理時間が均一かつ全体の処理時間が短くなるように kNN データを更新するアルゴリズムを提案する．実データを用いた実験により，提案手法が高速に kNN データを更新できることを確認した．

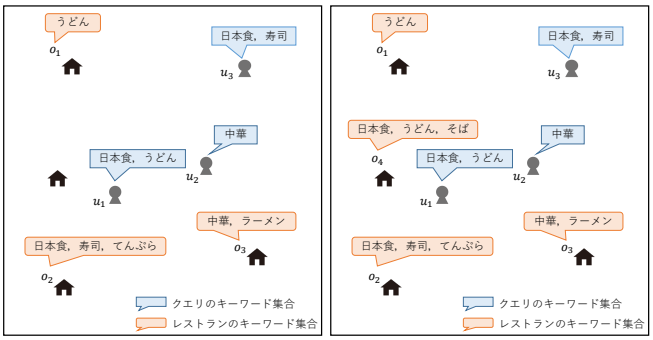
キーワード　Pub/Sub，kNN クエリ，分散処理

1　はじめに

近年，GPS を搭載した端末の普及により，位置情報やキーワードを含むデータが大量に生成されている．それらのデータを検索するアプリケーションの多くは，ユーザが自身の興味のある位置やキーワードをクエリとして事前に登録し，それらに基づいてユーザに適切なデータを配信するパブリッシュ/サブスクライブ（Pub/Sub）モデルに基づいている [1], [2], [3], [4], [5]．大量にデータが存在する環境では，クエリのキーワードを含むデータの中で，クエリに最も近い k 個のデータ (kNN データ) を検索する kNN クエリが有用である [6], [7], [8], [9]．また，Pub/Sub モデルでは頻繁にデータが生成されるため．各クエリの kNN データは頻繁に変化する．そのため，kNN データのモニタリングには多大な計算コストがかかる．

例 1. 図 1 は，Pub/Sub モデルにおいて，3 人のユーザがレストランが配信するデータをモニタリングする例を示している．各ユーザはクエリ（位置情報およびキーワード）を登録しており，クエリに合う k 個のデータをモニタリングしている．図 1a において，$k = 1$ と仮定し，ユーザ u_1 に注目する．u_1 の kNN データは，日本食またはうどんを含み，かつ u_1 に最も近いデータ o_2 である．同様に，u_2 の kNN データは o_3，u_3 の kNN データは o_2 となる．図 1b において，新たに o_4 が生成されたとする．u_1 に注目すると，o_4 は，日本食およびうどんを含み，o_2 よりも u_1 に近いため，u_1 の kNN データは o_4 に変化する．同様に，u_3 の kNN データが o_4 に変化する．

スマートフォンや SNS の普及により，生成されるデータやユーザが登録するクエリの数は増加している．データやクエリの数が増加すると，新たにデータが生成されたときのクエリの kNN データを更新するための処理量が増加するため，単一のワーカーで全ての処理を行う場合にリアルタイム性を保証することが難しくなる [10], [11], [12], [13], [14]．そこで，本稿では，クエリの追加や削除が頻繁に起こる環境において，複数のワーカーにクエリを分散して処理を分担することにより，各クエリの kNN データを効率的にモニタリングする問題に取り組む．



(a) 時刻 t　　**(b)** 時刻 $t+1$

図 1: Pub/Sub モデルにおける kNN データモニタリングの例

課題.　クエリの追加や削除が頻繁に起こる環境において，複数のワーカーにクエリを分散して各クエリの kNN データを効率的にモニタリングするための課題として，以下の 2 つが挙げられる：(i) 負荷分散が達成されるようにクエリをワーカーに割り当てること，および (ii) クエリの追加に対して効率的に処理を行うこと．(i) の課題に対する最も単純な方法として，ワーカーが持つクエリの数が均等になるようにクエリを割り当てる方法が考えられる．しかし，kNN データが頻繁に変化するクエリが特定のワーカーに集中した場合，そのワーカーの処理量

が大きくなってしまい，全体の処理時間も長くなる．(ii) の課題において，新たに追加されるクエリは kNN データを持っていないため，新たに生成されるデータが kNN データとなりうる範囲が大きくなる．そのため，事前に分散したクエリと同様に kNN データを更新すると，クエリの kNN データの更新する回数が大きくなり，処理時間が増えてしまう．

本稿では，文献 [15] において著者らが取り組んだ問題に対して，クエリの追加や削除を考慮し，各ワーカーの処理時間が均一かつ全体の処理時間が短くなるように kNN データを更新するアルゴリズムを提案する．

貢献． 以下に本稿の貢献を示す．

- クエリの追加や削除を考慮して，各ワーカーの処理時間が均一かつ全体の処理時間が短くなるように kNN データを更新するアルゴリズムを提案する．
- 実データを用いた実験により，提案アルゴリズムの有効性を確認する．

本稿の構成． 2 章で関連研究について説明し，3 章で本稿の問題を定義する．4 章でクエリを分散するアルゴリズム，5 章でクエリの kNN データを更新するアルゴリズムについて説明し，6 章で実データを用いた実験の結果を示す．最後に 7 章で本稿をまとめる．

2 関 連 研 究

本章では，クエリの分散処理に関する既存研究を紹介する．

Pub/Sub モデルにおけるクエリの分散． 様々な研究が Pub/Sub モデルにおいて，複数のワーカーでクエリを分散して管理し，クエリの解をモニタリングする問題に取り組んでいる [10], [11], [12], [13], [14].

文献 [10] では，位置情報およびキーワードに基づいてクエリを分散するアルゴリズムが提案されている．kd^t 木というデータ構造により，クエリを複数の部分集合に分割する．kd^t 木はデータを位置情報によって分割する kd 木にキーワードに基づく分割を加えたデータ構造である．ノードに含まれるクエリのキーワードの類似度を計算し，類似度が低ければ位置情報による分割，類似度が高ければキーワードによる分割を行う．各ノードに含まれるクエリを 1 つのワーカーで管理することで，クエリを分散する．しかし，この研究はレンジクエリを対象としており，kNN クエリを対象とした本研究とは問題が異なる．

また，文献 [11], [12], [13], [14] では，Pub/Sub モデルにおいて，キーワードに基づいてクエリを分散するアルゴリズムが提案されている．例えば，文献 [11] では，データのキーワードの出現頻度に基づいて，各ワーカーが担当するキーワードを決定し，担当するキーワードを持つクエリを管理する．これらの研究では，データおよびクエリの位置情報を考慮していない．

キーワードを用いたクエリの分散処理． 文献 [16], [17], [18] では，キーワードに基づいて複数のワーカーにデータを分散するアルゴリズムが提案されている．例えば，文献 [16] では，各ワーカーの負荷が均等になるが，クエリの実行に多くのメモリや追加の処理が必要となる分割と，各ワーカーの負荷が偏るが，クエリの処理が高速に行える分割を組み合わせ，各ワーカーの負荷の偏りを小さくしつつ，クエリの処理を高速に行える分割手法を提案している．しかし，これらの研究では，データおよびクエリの位置情報を考慮していない．

位置情報を用いたクエリの分散処理． 文献 [19], [20], [21] では，位置情報に基づいて複数のワーカーにデータを分散するアルゴリズムが提案されている．例えば，文献 [19] では，kd 木を用いてデータが含まれる領域を分割し，一つのノードに含まれるデータを一つのワーカーで処理する．ノードごとにクエリの実行にかかるコストを計算し，コストの大きいノードを分割していくことで，各ワーカーの負荷を均等にする．あるクエリを実行するとき，クエリの解となりうる領域に重複するノードのみ探索することで，高速にデータを検索する．しかし，これらの研究では，データおよびクエリのキーワードを考慮していない．また，MapReduce 環境を想定しており，ストリームデータに適用できない．

3 予 備 知 識

3.1 定　　義

生成されるストリームデータの集合を O，発行されるクエリの集合を Q，および kNN データのモニタリングを行うワーカー [1] の集合を W とする．

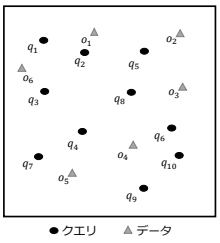
データモデル． あるデータ $o \in O$ は時間 ($o.t$)，位置 ($o.loc$)，およびキーワード集合 ($o.key$) で構成される ($o = \langle t, loc, key \rangle$)．$o.t$ は，o が生成された時間であり，$o.loc$ は，緯度と経度によって表される 2 次元平面上の点である．

クエリモデル． ある kNN クエリ $q \in Q$ は，q が登録された時間 ($q.t$)，位置 ($q.loc$)，キーワード集合 ($q.key$)，および何個のデータを受信するかを指定する変数 ($q.k$) で構成される ($q = \langle t, loc, key, k \rangle$)．$q.key$ は，key_{max} 個以下のキーワードを指定する．O が与えられたとき，q の kNN データ A は次の条件を満たす．(i) $|A| = k$，(ii)$\forall o \in A, \forall o' \in O_q - A, dist(o.loc, q.loc) \leqq dist(o'.loc, q.loc)$，(iii) $\forall o \in A$, $o.t > q.t$．ここで，O_q は O の中で $q.key$ に含まれるキーワードを一つ以上含むデータ集合，$dist(\cdot, \cdot)$ はデータとクエリとのユークリッド距離である．

例 2. 図 2 に本稿で用いるデータとクエリの例を示す．q_1 に注目し，$q.k = 1$ であるとすると，q_1 の kNN データは o_6 となる．

各ワーカー $w_i \in W$ にデータおよびクエリの送信を行うためのワーカーをメインワーカーとする，新たなデータが生成されるとき，メインワーカーはすべてのワーカーにデータを送信する．各ワーカーは，受信したデータに基づき，各クエリの kNN データを更新する．新たなクエリが生成されたとき，メインワーカーは，クエリを追加すべきワーカーを選択し，クエリを

1：ワーカーは 1 スレッド利用可能な CPU コアまたは計算機とする．



クエリ	キーワード集合
q_1	a,b
q_2	a,c,d
q_3	a,d
q_4	a,b,d
q_5	b,c
q_6	a,c
q_7	b,e
q_8	a
q_9	b,c,e
q_{10}	c,d

データ	キーワード集合
o_1	a,b,c
o_2	b,c,e
o_3	a,c
o_4	a,d
o_5	d
o_6	a,b,e

図 2: データおよびクエリの例

送信する．クエリが削除されるとき，削除するクエリを管理するワーカーを選択し，クエリの削除要請を送信する．

問題定義． O，Q および W が与えられたとき，複数のワーカー $w_i \in W$ に各クエリ $q \in Q$ を分散し，処理を分担することで，各クエリ q の kNN データをモニタリングする．[2]

4 分散アルゴリズム

本章では，各ワーカーにクエリを分散するアルゴリズムを紹介する [15]．

4.1 最適なクエリの分散

本稿では，複数のワーカーでクエリを分散して管理することで，各クエリの kNN データを効率的にモニタリングする．新たなデータ o が生成されたとき，あるクエリ q に対して，q のキーワードに o のキーワードが含まれ，q の指定する位置に近い位置に o が生成された場合にのみクエリの kNN データを更新する．そのため，複数のワーカーでクエリを分散して管理する場合，新たなデータが生成された際の各ワーカーの処理時間は更新を行うクエリの数に依存する．各ワーカーの処理時間を均一にするために，新たなデータが生成されたときの各ワーカーの処理コストとしてワーカー w_i の負荷 $L(w_i)$ を定義する．

定義 1（ワーカーの負荷）． あるワーカー w_i の負荷 $L(w_i)$ を以下で定義する．

$$L(w_i) = \sum_{q \in Q_{w_i}} L(q) \qquad (1)$$

ここで，Q_{w_i} はワーカー w_i で管理されているクエリの集合である．$L(q)$ はクエリ q の負荷であり，あるデータが生成された際に，q の kNN データを更新する確率を表す．クエリ q が，あるワーカー w_i に割り振られたとき，$L(q)$ は，w_i に割り振られたクエリ集合および過去に発生したデータ集合の位置情報およびキーワードの分布に基づいて計算される．詳細な計算方法は後述する（3.3 節）．

次に，各ワーカーの負荷を均一にしつつ，全体の負荷を最小にするクエリの分散を決定するクエリ分散問題を定義する．

定義 2（クエリ分散問題）． O，Q，およびワーカーの数 m が与えられたとき，クエリ分散問題は，Q を以下の条件を満たす m 個の集合 Q_1，Q_2，$\cdots$，Q_m に分割する．(i) $\sum_{i=1}^{m} L(w_i)$ が最小．(ii) $\forall i \neq j$ に対して，$L(w_i) \geqq L(w_j)$，$L(w_i)/L(w_j) \leqq \sigma$．ここで，$\sigma \simeq 1$ である．

2：データの送受信にかかる負荷は考慮しない．

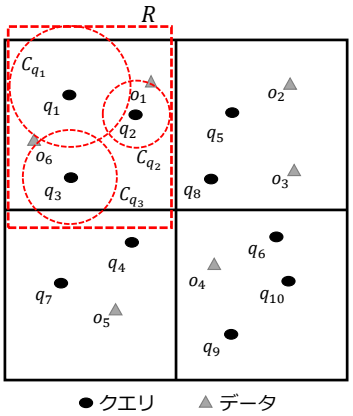


図 3: kNN データの更新が起こる領域 R

このとき，以下の定理が成り立つ [15]．

定理 1． クエリ分散問題は NP 困難である．

4.2 クエリ割り当てアルゴリズム

クエリ分散問題は NP 困難であるため，貪欲法を用いてクエリの分散を決定する．本節では，本研究で用いるワーカーへクエリを割り当てるアルゴリズムを紹介する．このアルゴリズムでは，全てのクエリの集合 Q を，集合の数が一定数以上になるまで，部分集合 Q_i に分割する．あるクエリ集合を分割するとき，クエリおよびデータの位置およびキーワードの分布に基づき，位置に基づく分割およびキーワードに基づく分割の内，負荷の小さくなる分割を選択する．そして，各ワーカーの負荷が均一になるように，分割された各クエリ集合に含まれるクエリを管理するワーカーを決定する．これにより，各ワーカーの負荷を均一にしつつ，全体の負荷をより小さくするようにクエリを分散する．

まず，あるクエリ集合 Q_i の負荷 $L(Q_i)$ を考える．$L(Q_i)$ は，新たに生成されたデータにより Q_i 内に含まれるクエリの kNN データが変化する可能性がある確率と考えることができる．ここで，クエリ q の k 番目のデータを o_k とすると，q の kNN データが変化する可能性がある場合は，$dist(o.loc, q.loc) < dist(o_k.loc, q.loc)$ となる位置に新たなデータ o が生成されたときである．つまり，$q.loc$ から $dist(o_k.loc, q.loc)$ の範囲を C_q とすると，C_q 内に生成されたデータが q の kNN データとなる可能性があり，C_q の外側に生成されたデータは q の kNN データにはならない．したがって，Q_i に含まれる全てのクエリの C_q を包含する領域を R とすると，R 内で生成されるデータが，Q_i 内に含まれるクエリの kNN データとなり得る．例えば，図 3 において，ある領域内に 3 つのクエリを含むクエリ集合が存在している．このとき，それぞれのクエリの C_q を包含する領域が R である．

あるデータが発生したときに，クエリの kNN データが変化する確率（クエリの負荷）を計算する際，文献 [1] と同様に，ク

エリと過去に発生したデータ集合の位置情報およびキーワードの分布に基づいて計算する．各キーワードの出現確率が独立であると仮定すると，新たなデータ o が生成されたとき，$o.loc$ が R 内に含まれ，かつ $o.key$ にあるキーワード key が含まれる確率 $P(R, key)$ は，以下の式で計算される．

$$P(R, key) = \frac{|O_{R,key}|}{|O|} \tag{2}$$

ここで，$O_{R,key}$ は R 内に含まれ，かつ key を含むデータの集合である．あるクエリ集合 Q_i に含まれるクエリ q の負荷 $L(q)$ について考えると，$o.loc$ が R 内に含まれ，かつ $o.key$ が $q.key$ に含まれるキーワードを一つ以上含むとき，o は q の kNN データとなり得るため，$L(q)$ は $P(R, key)$ を用いて，以下のように計算される．

$$L(q) = \sum_{key \in q.key} P(R, key) \tag{3}$$

また，Q_i の負荷 $L(Q_i)$ は，Q_i に含まれるクエリの負荷の合計と考えることができる．よって，$L(Q_i)$ は $L(q)$ を用いて，以下のように計算される．

$$L(Q_i) = \sum_{q \in Q_i} L(q) \tag{4}$$

最後に，ワーカーの負荷 $L(w_i)$ は，以下のように計算される．

$$L(w_i) = \sum_{q \in Q_{w_i}} L(q) \tag{5}$$

Q_{w_i} はあるワーカー w_i で管理されるクエリの集合である．

ここから，あるクエリを管理するワーカーを決定するアルゴリズムを紹介する．あるクエリの集合 Q_i を複数の集合に分割するとき，位置情報に基づく分割 (SpacePartition(Q_i)) およびキーワードに基づく分割 (TextPartition(Q_i)) のうち，より負荷の小さくなる方法で分割を行う．

SpacePartition(Q_i)．Q_i がある領域 r に含まれるとき，r を 4 つの領域 $r_i (1 \leqq i \leqq 4)$ に等分割し，それぞれの領域に含まれるクエリの集合 Q_{r_i} に分割する．

TextPartition(Q_i)．Q_i に含まれるクエリ q を $L(q)$ に基づいて，4 つの集合 $Q_{t_i} (1 \leqq i \leqq 4)$ に分割する．具体的には，$L(q)$ が大きいクエリから順に，各集合に分配する．あるクエリを分配するとき，その時点で分配されたクエリの負荷の合計が最も小さい集合に，クエリを分配する．この処理を Q_i に含まれる全てのクエリに対して行う．

SpacePartition(Q_i) で得られたクエリ集合の負荷の合計を C_s，TextPartition(Q_i) で得られたクエリ集合の負荷の合計を C_t としたとき，C_s が C_t より小さければ，位置情報に基づく分割を選択し，C_t が C_s より小さければ，キーワードに基づく分割を選択する．

アルゴリズム 1 は，クエリの集合 Q_i を分割するアルゴリズムを示している．キーワードに基づく分割は，Q_i に含まれるクエリを 1 つずつチェックするため，Q_i に含まれるクエリの数が多いとき，分割に多大な時間がかかる．そのため，$|Q_i| > \beta$ である場合，常に SpacePartition(Q_i) を実行する（5–6 行）．$|Q_i| \leqq \beta$ である場合，SpacePartition(Q_i) および TextPartition(Q_i) を実行し，それぞれによって得られた集合の負荷の合計 C_s および C_t を計算する（9–11 行）．$C_s < C_t$ ならば，位置情報に基づく分割を選択し，$C_s \geqq C_t$ ならば，キーワードに基づく分割を選択する（12–15 行）．分割された全ての集合を $\mathbb{Q}$ とし，負荷の降順でソートされているとすると，$|\mathbb{Q}|$ が αm 以上になるまで，負荷の最も大きい集合 Q_i を再帰的に分割する．

Algorithm 1 Queryset-Partition

Input: Q, m, α, β
Output: $\mathbb{Q}$

```
1:  ℚ ← ∅
2:  ℚ ← Q
3:  while |ℚ| < αm do
4:      Pop Q_i from ℚ
5:      if |Q_i| > β then
6:          ℚ_s ← SpacePartition(Q_i)
7:          ℚ ← ℚ ∪ ℚ_s
8:      else
9:          ℚ_s ← SpacePartition(Q_i)
10:         ℚ_t ← TextPartition(Q_i)
11:         Calculate C_t and C_s
12:         if C_s < C_t then
13:             ℚ ← ℚ ∪ ℚ_s
14:         else
15:             ℚ ← ℚ ∪ ℚ_t
```

Algorithm 2 Query-Assignment

Input: $\mathbb{Q}$

```
1: while |ℚ| > 0 do
2:     Pop Q_i from ℚ
3:     while |Q_i| > 0 do
4:         Pop q from Q_i
5:         w ← arg min_{w∈W} L(w)
6:         w.Q ← w.Q ∪ {q}
```

集合の分割後，各クエリ集合に含まれるクエリを管理するワーカーを決定する．同じクエリ集合に含まれるクエリは，位置が近く，同じような出現頻度のキーワードを持つクエリが多いため，同じデータによってクエリの更新を行う可能性が高い．そのため，1 つのクエリ集合に含まれるクエリを 1 つのワーカーで管理するのではなく，複数のワーカーに分散させることで，各ワーカーの負荷が均一になりやすい．

アルゴリズム 2 は各クエリ集合に含まれるクエリを管理するワーカーを決定するアルゴリズムを示している．$\mathbb{Q}$ は分割された全てのクエリ集合であり，負荷の降順でソートされているとし，負荷の大きい集合 Q_i から順に，管理するワーカーを決定する（2 行）．また，Q_i に含まれるクエリが負荷の降順にソートされているとする．あるクエリ q を管理するワーカーは，その時点で負荷が最も負荷が小さいワーカー w とする (5–6 行)．Q_i に含まれる全てのクエリを管理するワーカーを決定するま

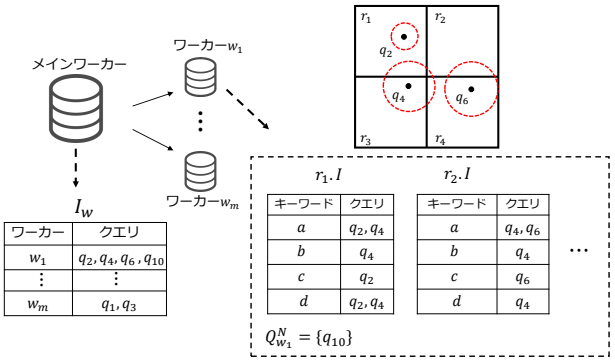


図 4: ワーカーが保持する情報の例

で，この操作を行う．この処理を，全てのクエリ集合に対して行う．

5 更新アルゴリズム

本章では，4 章のアルゴリズムで各ワーカーに割り当てたクエリの kNN データをモニタリングする手法を新たに提案する．新たに生成されたクエリ q は，kNN データを持っていないため，C_q はクエリおよびデータが存在し得る全ての領域となってしまう．そのため，q の kNN データの更新を事前に分散したクエリと同様にしてしまうと，ワーカーが保持する全ての転置ファイルに q が保持される．その結果，q が参照される回数が増えてしまい，処理時間が増えてしまう．そこで，新たに生成されたクエリの kNN データの更新と事前に分散したクエリの kNN データの更新を別々に行うことで，この問題を解決する．

データ構造． メインワーカーでは，各ワーカーが管理するクエリを転置ファイル I_w で保持する．事前に分散したクエリを管理するため，各ワーカーでは，クエリおよびデータが存在し得る領域 $\mathbb{R}^2$ を $n \times n$ の領域に等分割し，それぞれの領域 r_i に対して，ワーカーで管理するクエリ $q \in Q_{w_i}$ の C_q が r_i と重複するクエリをキーワードごとに管理する転置ファイル $r_i.I$ で管理する．また，新たに生成されたクエリは各ワーカーでリストを用いて管理する．

例 3. 図 4 に各ワーカーが保持する情報の例を示す．図 2 のクエリを各ワーカーへ分散したとする．$q_1, ..., q_9$ を事前に分散したクエリ，q_{10} を新たに生成されたクエリとする．メインワーカーは転置ファイル I_w により，各ワーカー $w_i (1 \leqq i \leqq m)$ が管理するクエリを保持している．w_1 に注目すると，w_1 が管理するクエリは，q_2，q_4，q_6，および q_{10} である．また，各ワーカーにおいて，各領域 $r_i (1 \leqq i \leqq 4)$ ごとに転置ファイル $r_i.I$ を保持している．r_1 に注目すると，r_1 と C_q が重複するのは q_2 および q_4 であるため，$r_1.I$ は q_2 および q_4 をキーワードごとに保持する．新たに生成された q_{10} は事前に分散したクエリとは別にリストで管理する．

5.1 クエリの追加・削除に対する処理

新たなクエリ q が生成されたとき，メインワーカーはランダムに決定されたワーカー w_i に q を送信し，転置ファイル I_w を更新する．w_i に送信された q は，$Q^N_{w_i}$ に追加される．

クエリを削除するとき，メインワーカーは，転置ファイル I_w から削除されるクエリ q を管理するワーカー w_i を取得して w_i に q の削除要請を送信し，転置ファイル I_w を更新する．w_i は，q を削除し，C_q が重複する領域 r_i の転置ファイル $r_i.I$ を更新する．

Algorithm 3 kNN-Update

Input: o, w_i

1: **for** $\forall r_i \in \mathbb{R}^2$ **do**
2: **if** $o.loc \in r_i$ **then**
3: Get Q_c from $r_i.I$ //Q_c is a set of querise whose keywords include $o.key$.
4: **for** $\forall q \in Q_c$ **do**
5: **if** $dist(o.loc, q.loc) < dist(o_k.loc, q.loc)$ **then**
6: Update q
7: **for** $\forall q_n \in Q^N_{w_i}$ **do**
8: **if** $q_n.key$ include $o.key$ **then**
9: Update q_n
10: **if** q_n gets $q_n.k$ data **then**
11: Pop q_n from $Q^N_{w_i}$
12: $Q_{w_i} \leftarrow Q_{w_i} \cup q_n$
13: **for** $\forall r_i \in \mathbb{R}^2$ **do**
14: **if** $C_{q_n} \cap r_i \neq \varnothing$ **then**
15: Update $r_i.I$

5.2 データの生成に対する処理

アルゴリズム 3 は，新たなデータが生成されたとき，各ワーカーで管理するクエリの kNN データを更新するアルゴリズムを示している．あるワーカー w_i に事前に分散したクエリの集合を Q_{w_i}，新しく生成され kNN データの数が k 個未満のクエリの集合を $Q^N_{w_i}$ とする．新たなデータ o が生成されたとき，メインワーカーは全てのワーカーに o を送信する．Q_{w_i} に含まれるクエリに対して，ワーカーは受信した o の $o.loc \in r_i$ となる領域の $r_i.I$ を用いて，o のキーワードを含むクエリの集合 Q_c を取得する（2–3 行）．そして，$q \in Q_c$ と o との距離 $dist(o.loc, q.loc)$ と q の k 番目の kNN データ o_k との距離 $dist(o_k.loc, q.loc)$ を比較し，$dist(o.loc, q.loc) < dist(o_k.loc, q.loc)$ の場合，q の kNN データを更新する（4–6 行）．$Q^N_{w_i}$ に含まれるクエリに対して，クエリのキーワードが，受信した $o.key$ に含まれるとき，o がクエリの kNN データとなる（8–9 行）．このとき，あるクエリ $q_n \in Q^N_{w_i}$ の kNN データの数が k 個になったとき，q_n を $Q^N_{w_i}$ から削除して Q_{w_i} に追加し，C_{q_n} が重複する領域 r_i の転置ファイル $r_i.I$ を更新する（10–15 行）．

6 性能評価

本章では，提案アルゴリズムの性能評価のために行った実験の結果を示す．

6.1 実験環境

本実験は，Windows 10 Pro，2.40GHz Intel Core i7-8700T，および 32GB RAM を搭載した 6 台の計算機で構成されるクラスタを用いた．1 台の計算機はメインワーカーとして用いた．その他 5 台の計算機は kNN モニタリングを行うワーカーとして用いた．また，CPU の 1 コアを 1 つのワーカーとした．全てのアルゴリズムは C++で実装した．

6.1.1 データセット

本実験では，生成されるデータとして Twitter[3] および Place[4] を用いた．Twitter はアメリカの位置情報付きツイートデータ，Place はアメリカの公共スペースについて記述したキーワードと位置情報を含むデータであり，緯度が北緯 20 度から北緯 50 度，経度が西経 60 度から西経 125 度の範囲に存在するデータを使用した．図 5 に各データセットの分布，表 1 にデータセットの詳細を示す．各クエリの位置情報は，1 つのデータの位置情報をそのまま使用し，キーワードは，データに現れる単語の中から，5 個以下の単語をランダムに選んだものとした．

6.1.2 パラメータ

表 2 に，本実験で用いたパラメータを示す．太字で表されている値はデフォルトの値である．本実験では，あるパラメータを変化させる際，その他のパラメータはデフォルトの値を用いた．k は 1 から 10 の間の一様乱数とし，$\alpha = 20$ および $\beta = 100,000$ とした．これらの値は，α および β は，事前実験により最も結果が良いパラメータを選択した．実験は，文献 [2] と同様に，投稿時間が古いものからデータを発生させ，データが 1,000,000 個に到達した時点でクエリを分散する．そして，新たにデータが 1,000,000 個発生するまで行う．この際，1 度に 1,000 個のデータが発生するとし，1,000 回データの更新を行った．また，1 度データが発生し，データの更新を行った後，N_q 個のクエリが追加され，N_q 個のクエリが削除される．

6.1.3 比較手法

比較手法として，2 章で紹介した文献 [10] のアルゴリズムを，kNN クエリ向けに改良した手法を用いた．文献 [10] の研究では，レンジクエリを対象としているため，クエリの指定する位置情報は範囲となる．そのため，本研究では，クエリの C_q をクエリの指定する範囲とする．kd^t 木を用いて，クエリを複数の集合に分割する．ノードに含まれるクエリのキーワードの類似度を計算し，類似度が低ければ位置情報による分割，類似度が高ければキーワードによる分割を行う．各ノードに含まれるクエリを 1 つのワーカーで管理することで，クエリを分散する．

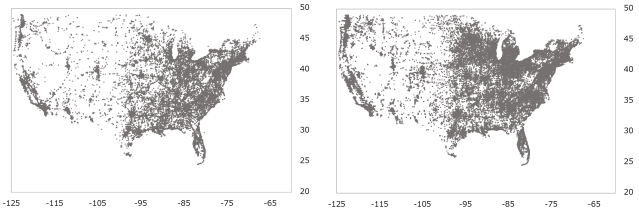
新たなデータが生成されたとき，メインワーカーはデータの位置情報およびキーワードから解となりうるクエリを持つワーカーにデータを送信する．文献 [10] の研究では，レンジクエリを対象としているため，クエリの解となりうる範囲は変化しないが，本研究では，kNN クエリを対象としているため，kNN データの変化により，クエリの解となりうる範囲が変化する．そのため，新たなデータが生成されたとき，メインワーカーは全てのワーカーにデータを送信する．各ワーカーは，受信したデータの指定する位置が，ノードの領域に重複しているノードに含まれるクエリの kNN データを更新する．本研究では，ノードの領域はノードに含まれるクエリの集合の R とする．

新たなクエリが生成されたとき，メインワーカーは，kd^t 木にクエリを挿入し，クエリが挿入されたノードを管理するワーカーにクエリを送信する．本研究では，新たに生成されるクエリは kNN データを持っておらず，クエリの範囲を指定することができない．そのため，新たなクエリが生成されたとき，メインワーカーは，管理するクエリの数が最も小さいワーカーにクエリを送信する．また，送信されたクエリは，クエリの範囲を指定できないため，新たに生成されたクエリは一つのクエリ集合 $Q^N_{w_i}$ とみなし，kNN データの更新を行う．

クエリを削除するとき，メインワーカーは削除するクエリを含むノードを管理するワーカーに削除要請を送信する．

3 : http://www.ntu.edu.sg/home/gaocong/datacode.htm

4 : https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces



(a) Twitter **(b)** Place

図 5: データの分布

表 1: データセットの詳細

	Twitter	Place
データ数	20,000,000	9,356,750
キーワード数	2,225,654	53,931
1 つのデータの平均キーワード数	5.51	2.94

表 2: パラメータの設定

パラメータ	値
ワーカー数	4, **8**, 12, 16, 20, 24
クエリの数，$\lvert Q\rvert[\times 10^6]$	**1**, 1.5, 2, 2.5, 3
クエリの更新頻度，N_q	50, **100**, 150, 200

6.2 評価結果

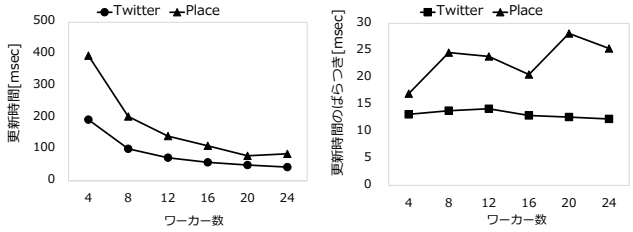
本稿では，全クエリの kNN データの更新が完了するまでの時間を評価するため，

- 平均更新時間 (1 度に発生する全てのデータに対して，kNN データの更新が完了するまでの平均時間 [msec])
- ワーカーの更新時間のばらつき (各ワーカーで管理しているクエリの kNN データの更新が完了するまでの時間の中で最大の時間と最小の時間の差 [msec])

の 2 つの指標を評価した．以下に結果を示す．

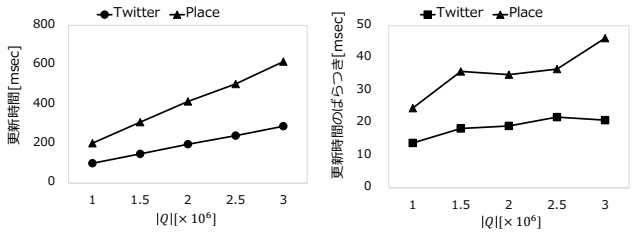
6.2.1 ワーカー数の影響

図 6 にワーカー数を変えたときの結果を示す．ワーカー数が増加すると，各ワーカーで管理するクエリの数が少なくなるため，平均更新時間は短くなる．位置とキーワードに基づく負荷



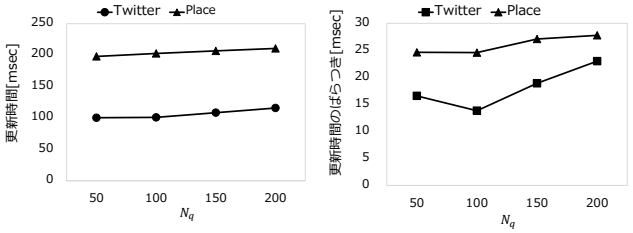
(a) 平均更新時間　　**(b)** 更新時間のばらつき

図 6: ワーカー数の影響



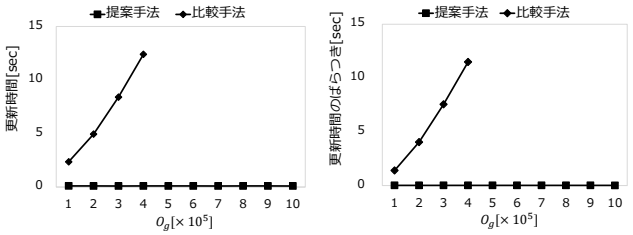
(a) 平均更新時間　　**(b)** 更新時間のばらつき

図 7: $|Q|$ の影響



(a) 平均更新時間　　**(b)** 更新時間のばらつき

図 8: N_q の影響



(a) 平均更新時間 (Twitter)　　**(b)** 更新時間のばらつき (Twitter)

図 9: 比較手法との比較

を考慮することにより，正確にクエリ集合に含まれるクエリの kNN データの更新にかかる時間を見積もることができる．また，位置に基づく分割およびキーワードに基づく分割のうち，より全体の負荷が小さくなるようにクエリ集合を分割することで，クエリの更新にかかる時間を短くする．さらに，分割の際に同じ集合に含まれたクエリは，位置が近く，同じような出現頻度のキーワードを持つクエリが多いため，同じデータによってクエリの更新を行う可能性が高い．そのため，1 つのクエリ集合に含まれるクエリを 1 つのワーカーで管理するのではなく，複数のワーカーに分散させることで，各ワーカーの更新時間のばらつきが小さくなり，平均更新時間も短くなる．

また，ワーカー数が増加すると，平均更新時間が短くなるため，平均更新時間に対する更新時間のばらつきの割合が大きくなる．

6.2.2 $|Q|$ の影響

図 7 に $|Q|$ を変えたときの結果を示す．$|Q|$ が増加すると，平均更新時間は長くなる．$|Q|$ が増加すると，1 度のデータの発生に対して，更新を行うクエリの数が増加するため，平均更新時間は長くなる．

また，$|Q|$ が増加すると，各ワーカーでクエリの更新を行う回数が大きくなり，各ワーカーでクエリの更新を行う回数の差が大きくなるため，更新時間のばらつきは大きくなる．

6.2.3 N_q の影響

図 8 に N_q を変えたときの結果を示す．新しく生成され，kNN データが k 個未満のクエリは，位置情報にかかわらず kNN データが更新されるため，頻繁に kNN データが更新される．また，kNN データが k 個集まり，$Q^N_{w_i}$ から削除され，Q_{w_i} に追加された直後のクエリは，事前に分散したクエリに比べて，C_q が大きいものが多い．そのため，新たに生成されたクエリは kNN データを更新する回数が大きくなる．N_q が増加すると，事前に分散したクエリの数が小さくなり，新たに生成されるクエリの数が大きくなるため，平均更新時間は長くなる．提案手法は，事前に分散したクエリと新たに生成されるクエリの kNN データの更新を別々に行うことで，新たなクエリが参照される回数を小さくしている．そのため，N_q が増加しても平均更新時間の増加率が小さい．

N_q が増加すると，各ワーカーでクエリの更新を行う回数が大きくなり，各ワーカーでクエリの更新を行う回数の差が大きくなるため，更新時間のばらつきは大きくなる．提案手法は新たに追加されるクエリをランダムなワーカーに割り当てることにより，各ワーカーで管理される新たなクエリの数の差を小さくしている．そのため，各ワーカー間における新たなクエリの kNN の更新にかかる時間の差を小さくできており，N_q が増加しても更新時間のばらつきの増加率は小さい．

6.2.4 比較手法との比較

図 9 に提案手法と比較手法を比較した実験の結果を示す．この実験では，新たなデータが 100,000 個生成されるごとの更新時間の平均と更新時間のばらつきの平均を比較した．O_g は新たに生成されたデータの数である．パラメータは全てデフォルトの値を用いた．

提案手法は，比較手法よりも平均更新時間が短い．比較手法では，ノードごとにクエリの kNN データの更新を行うため，実際には kNN データが変化しないクエリに対しても kNN データの更新処理を行う．そのため，提案手法と比べて，kNN データの更新を行う回数が大きくなり，平均更新時間は長くなる．また，新たに生成されるクエリは，範囲を指定することができないため，生成されたデータの位置情報から，kNN データが更新されるクエリを限定できない．そのため，$Q^N_{w_i}$ に含まれるクエリが増えるほど，kNN データの更新を行う回数が大きく

なり，平均更新時間は長くなる．

また，提案手法は，比較手法よりも更新時間のばらつきが小さい．比較手法では，新たに生成されたクエリは管理するクエリの数が最も小さいワーカーに送信される．また，比較手法を用いてクエリを分散したとき，各ワーカーの管理するクエリの数の差が大きくなる．そのため，新たに生成されたクエリが送られるワーカーが一部のワーカーへと集中してしまい，各ワーカーでクエリの更新を行う回数の差が大きくなるため，更新時間のばらつきが大きくなる．

7 まとめ

本稿では，Pub/Sub モデルにおいて，複数のワーカーでクエリを分散管理して処理を分担することにより，各クエリの kNN データを効率的にモニタリングする問題に取り組んだ．新たに生成されたクエリは，kNN データを持っていないため，事前に分散したクエリと同様に kNN データを更新すると，クエリの kNN データの更新する回数が大きくなり，処理時間が増えてしまう．そのため，新たに生成されたクエリの kNN データの更新と事前に分散したクエリの kNN データの更新を別々に行うことで，処理時間を短くする．実データを用いた実験により，提案アルゴリズムの有効性を確認した．

謝辞．本研究の一部は，文部科学省科学研究費補助金・基盤研究 (A)(JP18H04095)，および基盤研究 (B)(T17KT0082a) の研究助成によるものである．ここに記して謝意を表す．

文　献

[1] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang, “Ap-tree: Efficiently support continuous spatial-keyword queries over stream,” Proc. IEEE Int’l Conf. on Data Engineering, pp.1107–1118, 2015.

[2] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang, “Skype: top-k spatial-keyword publish/subscribe over sliding window,” Proc. the VLDB Endowment, vol.9, no.7, pp.588–599, 2016.

[3] G. Li, Y. Wang, T. Wang, and J. Feng, “Location-aware publish/subscribe,” Proc. ACM SIGMOD Int’l Conf. on Management of Data, pp.802–810, 2013.

[4] H. Hu, Y. Liu, G. Li, J. Feng, and K.L. Tan, “A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions,” Proc. IEEE Int’l Conf. on Data Engineering, pp.711–722, 2015.

[5] M. Yu, G. Li, and J. Feng, “A cost-based method for location-aware publish/subscribe services,” Proc. ACM Conf. on Information and Knowledge Management, pp.693–702, 2015.

[6] D.W. Choi, J. Pei, and X. Lin, “Finding the minimum spatial keyword cover,” Proc. IEEE Int’l Conf. on Data Engineering, pp.685–696, 2016.

[7] G. Li, J. Xu, and J. Feng, “Keyword-based k-nearest neighbor search in spatial databases,” Proc. ACM Int’l Conf. on Information and knowledge management, pp.2144–2148, 2012.

[8] J. Lu, Y. Lu, and G. Cong, “Reverse spatial and textual k nearest neighbor search,” Proc. ACM SIGMOD Int’l Conf. on Management of Data, pp.349–360, 2011.

[9] Y. Tao, and C. Sheng, “Fast nearest neighbor search with keywords,” IEEE Trans. Knowledge and Data Engineering, vol.26, no.4, pp.878–888, 2014.

[10] Z. Chen, G. Cong, Z. Zhang, T.Z. Fuz, and L. Chen, “Distributed publish/subscribe query processing on the spatio-textual data stream,” Proc. IEEE Int’l Conf. on Data Engineering, pp.1095–1106, 2017.

[11] F. Basık, B. Gedik, H. Ferhatosmanoğlu, and M.E. Kalender, “s^3-tm : scalable streaming short text matching,” The VLDB Journal, vol.24, no.6, pp.849–866, 2015.

[12] A. Yu, P.K. Agarwal, and J. Yang, “Subscriber assignment for wide-area content-based publish/subscribe,” IEEE Trans. Knowledge and Data Engineering, vol.24, no.10, pp.1833–1847, 2012.

[13] R. Barazzutti, T. Heinze, A. Martin, E. Onica, P. Felber, C. Fetzer, Z. Jerzak, M. Pasin, and E. Rivière, “Elastic scaling of a high-throughput content-based publish/subscribe engine,” Proc. IEEE Int’l Conf. on Distributed Computing Systems, pp.567–576, 2014.

[14] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg, “Content-based publish-subscribe over structured overlay networks,” Proc. IEEE Int’l Conf. on Distributed Computing Systems, pp.437–446, 2005.

[15] 鶴岡翔平，西尾俊哉，天方大地，原隆浩，“Pub/sub 環境における knn データモニタリングの分散処理のためのクエリ割当てアルゴリズム，” 情報処理学会論文誌データベース（TOD），vol.12，no.4，pp.53–65”，2019.

[16] M.A.U. Nasir, G.D.F. Morales, D. Garcia-Soriano, N. Kourtellis, and M. Serafini, “The power of both choices: Practical load balancing for distributed stream processing engines,” Proc. IEEE Int’l Conf. on Data Engineering, pp.137–148, 2015.

[17] M.A.U. Nasir, G.D.F. Morales, N. Kourtellis, and M. Serafini, “When two choices are not enough: Balancing at scale in distributed stream processing,” Proc. IEEE Int’l Conf. on Data Engineering, pp.589–600, 2016.

[18] B. Gedik, “Partitioning functions for stateful data parallelism in stream processing,” The VLDB Journal, vol.23, no.4, pp.517–539, 2014.

[19] A.M. Aly, A.R. Mahmood, M.S. Hassan, W.G. Aref, M. Ouzzani, H. Elmeleegy, and T. Qadah, “Aqwa: adaptive query workload aware partitioning of big spatial data,” Proc. the VLDB Endowment, vol.8, no.13, pp.2062–2073, 2015.

[20] A. Eldawy, and M.F. Mokbel, “Spatialhadoop: A mapreduce framework for spatial data,” Proc. IEEE Int’l Conf. on Data Engineering, pp.1352–1363, 2015.

[21] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, “Hadoop gis: a high performance spatial data warehousing system over mapreduce,” Proc. the VLDB Endowment, vol.6, no.11, pp.1009–1020, 2013.