Title	Distributed Spatial-Keyword kNN Monitoring for Location-aware Pub/Sub
Author(s)	Tsuruoka, Shohei; Amagata, Daichi; Nishio, Shunya et al.
Citation	GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems. 2020, p. 111-114
Version Type	AM
URL	https://hdl.handle.net/11094/92859
rights	© 2020 ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Tsuruoka S., Amagata D., Nishio S., et al. Distributed Spatial-Keyword kNN Monitoring for Location-aware Pub/Sub. GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, 111 (2020); https://doi.org/10.1145/3397536.3422199.
Note	

Distributed Spatial-Keyword kNN Monitoring for Location-aware Pub/Sub

Shohei Tsuruoka
Osaka University
tsuruoka.shohei@ist.osaka-u.ac.jp

Daichi Amagata
Osaka University
amagata.daichi@ist.osaka-u.ac.jp

Shunya Nishio
Osaka University
nishio.syunya@ist.osaka-u.ac.jp

Takahiro Hara
Osaka University
hara@ist.osaka-u.ac.jp

ABSTRACT

Recent applications employ publish/subscribe (Pub/Sub) systems so that publishers can easily receive attentions of customers and subscribers can monitor useful information generated by publishers. Due to the prevalence of smart devices and social networking services, a large number of objects that contain both spatial and keyword information have been generated continuously, and the number of subscribers also continues to increase. This poses a challenge to Pub/Sub systems: they need to continuously extract useful information from massive objects for each subscriber in real time.

In this paper, we address the problem of k nearest neighbor monitoring on a spatial-keyword data stream for a large number of subscriptions. To scale well to massive objects and subscriptions, we propose a distributed solution, namely DkM-SKS. Given m workers, DkM-SKS divides a set of subscriptions into m disjoint subsets based on a cost model so that each worker has almost the same kNN-update cost, to maintain load balancing. DkM-SKS allows an arbitrary approach to updating kNN of each subscription, so with a suitable in-memory index, DkM-SKS can accelerate update efficiency by pruning irrelevant subscriptions for a given new object. We conduct experiments on real datasets, and the results demonstrate the efficiency and scalability of DkM-SKS.

1 INTRODUCTION

Due to the recent prevalence of GPS-enabled devices, many applications have been generating objects that contain location information and keywords [15, 27]. They often provide services that retrieve
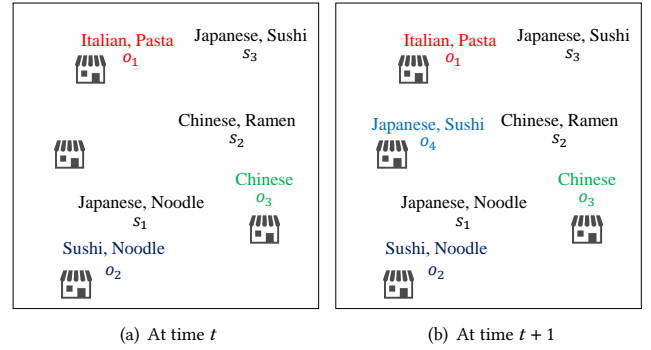
(a) At time t (b) At time $t + 1$

Figure 1: An example of kNN monitoring in a location-aware Pub/Sub system, where o_i and s_j respectively denote an spatial-keyword object and a subscription

objects useful to users from the generated ones, based on a location-aware publish/subscribe (Pub/Sub) model [22, 23, 29, 32, 33]. In this model, users register queries that specify query locations and keywords as their subscriptions on a Pub/Sub system, and this system delivers appropriate objects generated by publishers (e.g., Point of Interests) to subscriptions based on their query locations and keywords. It is well known that range and k nearest neighbor (kNN) queries support location-aware Pub/Sub systems. A range query retrieves all objects existing within a user-specified range from a query point, so it cannot control the result size. This means that users may not obtain any objects or may obtain a huge amount of objects, which is not desirable. On the other hand, a kNN query alleviates this drawback, since users can obtain a reasonable-sized result. In this paper, hence, we consider kNN queries.

1.1 Motivation

In Pub/Sub environments, objects are generated in a streaming fashion, so we have to *continuously update* the kNN objects for each subscription. For example:

EXAMPLE 1. *Figure 1 illustrates an example of kNN monitoring in a location-aware Pub/Sub system. Three subscriptions (s_1, s_2, and s_3) are registered, and the Pub/Sub system monitors kNN objects for each subscription. Assume $k = 1$ and focus on s_1, which specifies Japanese and Noodle as keywords. At time t, i.e., in Figure 1(a), the NN object for s_1 is o_2, because it contains the keyword Noodle and is the nearest to s_1 among $\{o_1, o_2, o_3\}$. Also, the NN object for s_2 (s_3) is o_3 (o_2). Assume further that a new object o_4 is generated at time $t + 1$,*

as shown in Figure 1(b). Since o_4 also contains the keyword Japanese, the NN object of s_3 is updated to o_4 (and the NN objects for the other subscriptions do not change).

Users require up-to-date results, so Pub/Sub systems have to efficiently update kNN objects of their subscriptions when new objects are given. However, this is a difficult task, because many applications employing Pub/Sub systems have to deal with a lot of (often million-scale) subscriptions [34]. Besides, due to the usefulness of location-aware Pub/Sub systems, the number of subscriptions is further increasing [31]. It is therefore hard for a single server to update the result for each subscription in real time [14]. This suggests that we need to make location-aware Pub/Sub systems efficient and scalable, motivating us to consider a distributed solution: given multiple workers, each registered subscription is assigned to a specific worker so that parallel kNN update is enabled.

Challenge. Although a distributed solution is promising, it has some challenges to scale well to massive objects and subscriptions (i.e., continuous spatial-keyword kNN queries).

(1) A distributed solution has to maintain load balancing. This is not trivial for continuous spatial-keyword kNN queries, because each subscription specifies arbitrary locations and keywords., i.e., the loads of subscriptions are different and not explicitly provided.

(2) It is necessary to deal with subscription insertions and deletions. Although some variants of the spatial-keyword kNN monitoring problem [22, 32] accept subscription insertions and deletions, these solutions consider centralized environments and extending them for decentralized environments is not trivial. In addition, [14, 31] assume subscription insertions and deletions in distributed processing environments. However, [14] considers not the costs of subscriptions but the number of them, which is not effective for load balancing, and [31] does not consider load balancing.

1.2 Contribution

We overcome these challenges and propose two baselines and DkM-SKS (Distributed kNN Monitoring on Spatial-Keyword data Stream). Our solutions employ

- **Cost models for subscriptions**: We design cost models for subscriptions, so that we can estimate the load of a given subscription when a new object is generated. Specifically, we propose keyword- and space-oriented cost models. Our models use a practical assumption and can deal with new subscriptions. Based on these models, we further propose a hybrid of these two models.
- **Cost-based subscription partitioning**: Based on our cost models, a set of subscriptions is divided into disjoint subsets, each of which is assigned to a specific worker. In particular, DkM-SKS considers both spatial and keyword information, so that kNN update costs can be minimized. We use a greedy algorithm for subscription partitioning, because optimal cost-based partitioning is NP-hard.

Furthermore, DkM-SKS allows an arbitrary exact algorithm for kNN update. This is a good property because it can implement a state-of-the-art to accelerate performance. To demonstrate the efficiency of DkM-SKS, we conduct experiments on two real datasets. From
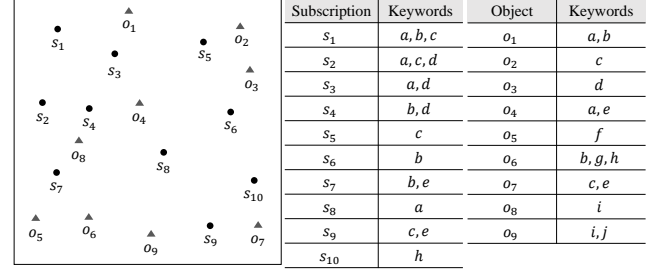
the experimental results, we confirm that DkM-SKS outperforms the baselines and a state-of-the-art technique.

Organization. The rest of this paper is organized as follows. We formally define our problem in Section 2. Then, we design baseline solutions in Section 3. We propose DkM-SKS in Section 4, and introduce our experimental results in Section 5. Related works are reviewed in Section 6. Finally, this paper is concluded in Section 7.

Subscription	Keywords	Object	Keywords
s_1	a, b, c	o_1	a, b
s_2	a, c, d	o_2	c
s_3	a, d	o_3	d
s_4	b, d	o_4	a, e
s_5	c	o_5	f
s_6	b	o_6	b, g, h
s_7	b, e	o_7	c, e
s_8	a	o_8	i
s_9	c, e	o_9	i, j
s_{10}	h		

Figure 2: A toy example of objects and subscriptions that have been respectively generated and registered at time t

2 PRELIMINARY

Problem definition. Let us first define spatial-keyword objects.

DEFINITION 1 (SPATIAL-KEYWORD OBJECT). *A spatial-keyword object o is defined as $o = \langle p, \psi, t \rangle$, where p is a 2-dimensional location of o, ψ is a set of keywords held by o, and t is the time-stamp when o is generated.*

Without loss of generality, hereinafter, we call o *object* simply. Note that we assume discrete time in this paper. Next, we define continuous spatial-keyword k nearest neighbor (kNN) queries.

DEFINITION 2 (CONTINUOUS SPATIAL-KEYWORD kNN QUERY). *A continuous spatial-keyword kNN query s is defined as $s = \langle p, \psi, k, t \rangle$, where p is a 2-dimensional location of interest for s, ψ is a set of keywords in which s is interested, k is the number of results required by s, and t is the time-stamp when s is registered. Let O be a set of objects generated so far, and let $O(s)$ be the set of objects $o \in O$ where $o.\psi \cap s.\psi \neq \varnothing$ and $s.t \leq o.t$. Given $O(s)$, this query monitors a set of objects A that satisfy (i) $|A| = k$ and (ii) $\forall o \in A, \forall o' \in O(s) - A, dist(o.p, s.p) \leq dist(o'.p, s.p)$, where $dist(p, p')$ evaluates the Euclidean distance between points p and p' (ties are broken arbitrarily).*

That is, we consider continuous spatial-keyword kNN queries with a *boolean* (i.e., OR) semantic for keywords [5, 8, 11] and a time constraint [6, 30] for obtaining fresh objects as much as possible. A subscription is corresponding to a continuous spatial-keyword kNN query in this paper, as shown in Example 1. We hence use them interchangeably. Then, our problem is defined as follows:

PROBLEM STATEMENT. *Given O and a set of registered subscriptions S, our problem is to exactly monitor A for each subscription $\in S$.*

EXAMPLE 2. *Figure 2 illustrates a toy example which is used throughout this paper. Assume that $s_1, ..., s_{10}$ ($o_1, ..., o_9$) have been registered (generated) at time t. Consider s_1, then $O(s_1) = \{o_1, o_4, o_6, o_7\}$, because they contain a, b, or c. Assuming $s_1.k = 2$, A of s_1 is $\{o_1, o_4\}$.*

This paper proposes a distributed solution to achieve real-time monitoring and scale well to large $|O|$ and $|S|$.

System overview. We assume that a location-aware Pub/Sub system employs a general distributed setting consisting of a main server and m workers [9, 24]. The main server (each worker) directly communicates with workers (the main server). (A worker can be a CPU core or a machine that can use a thread.) The main server takes the following roles: it

- assigns each subscription to a specific worker,
- receives a stream of objects and broadcasts them to all workers, and
- accepts subscription insertions and deletions.

The main operations of each worker are as follows: it

- accepts subscriptions assigned by the main server,
- removes requested subscriptions, and
- updates the kNN objects for each assigned subscription.

We see that kNN objects for each subscription are updated in parallel, thereby this approach is promising for massive subscriptions. An important problem to achieve this is load balancing. That is, distributed solutions have to consider how to make the computation time of each worker almost equal when new objects are generated. We below analyze this problem theoretically.

Let $C(s)$ be the kNN update cost of a subscription s (how to obtain $C(s)$ is introduced later). Furthermore, let $C(w_i)$ be the cost (load) of a worker w_i, which is defined as

$$C(w_i) = \sum_{s \in S(w_i)} C(s),$$

where $S(w_i)$ is a set of subscriptions assigned to w_i. We want to optimize the load difference between workers with a good subscription assignment. This can be formalized as follows:

DEFINITION 3 (OPTIMAL SUBSCRIPTION ASSIGNMENT PROBLEM). *Given a set of objects O, a set of subscriptions S, and m workers, this problem is to find a subscription assignment that minimizes*

$$\max_{i \in [1,m]} C(w_i) - \min_{j \in [1,m]} C(w_j).$$

We have the following theorem w.r.t. the above problem [7].

THEOREM 1. *The optimal subscription assignment problem is NP-hard.*

It can be seen, from this theorem, that it is not practical to obtain the optimal assignment, which suggests that we need a heuristic approach. We hence consider $C(s)$ to capture the load of s and then design an approach that partitions S into m disjoint subsets whose loads are well balanced. Note that $C(s)$ is dependent on a given cost model. In addition, we consider how to manage new subscriptions (we can easily deal with subscription deletions: the main server simply requests them to the corresponding workers).

3 BASELINES

Because this is the first work that proposes a distributed solution for processing continuous spatial-keyword kNN queries defined in Definition 2, we first design baseline solutions. We propose two baselines that respectively employ keyword- and space-oriented subscription partitioning. We assume that some subscriptions are registered at the initial time, and we partition S when O becomes sufficiently large. (This is common to DkM-SKS.) We use O_{init} to denote the set of objects when S is partitioned.

Algorithm 1: SUBSCRIPTION-ASSIGNMENT

Input: S (a set of subscriptions) and m workers

1 Sort S in descending order of cost
2 Set $C(w) = 0$ for each worker
3 **for** *each $s \in S$* **do**
4 $\quad$ $w \leftarrow \underset{m}{\arg\min} \, C(w)$
5 $\quad$ $S(w) \leftarrow S(w) \cup \{s\}, C(w) \leftarrow C(w) + C(s)$

3.1 Keyword-oriented Partition

To start with, we design keyword-oriented partition. One possible approach partitions S so that a set of distinct keywords of the subscriptions held by each worker can be disjoint between workers. This approach is not efficient, because it does not consider keyword frequencies. In other words, if a worker has subscriptions with keywords that are contained by many objects, its load becomes heavy, rendering load imbalance. Hence our keyword-oriented partition takes keyword frequencies into account.

Cost estimation. Similar to [33], we estimate the load of a subscription based on the distributions of keywords in O_{init}, because the distributions of large datasets rarely change in practice [36]. Assume that the appearance probability of each keyword is independent. Given an object o, the probability that a keyword λ is contained in $o.\psi$, $P(\lambda)$, is

$$P(\lambda) = \frac{|O_\lambda|}{|O_{init}|},$$

where $O_\lambda \in O_{init}$ is a set of objects o_i such that $\lambda \in o_i.\psi$. Recall that $O(s)$ is a set of objects o_j where $o_j.\psi \cap s.\psi \neq \varnothing$. Therefore, the kNN update cost of a subscription s, $C(s)$, can be estimated as:

$$C(s) = \sum_{\lambda \in s.\psi} P(\lambda). \tag{1}$$

Subscription partition. Keyword-oriented partition employs the cost model defined in Equation (1) and a 3/2-approximation greedy algorithm [20] for subscription partitioning. This approach first computes $C(s)$ for every $s \in S$, and sorts S in descending order of $C(s)$. Then, this approach sequentially accesses subscriptions while assigning an accessed subscription to w_i with the minimum $C(w_i)$. Algorithm 1 details this approach.

kNN update. Each worker w maintains an inverted file $w.I$ to index its assigned subscriptions. The inverted file is a set of postings lists $w.I[\lambda]$ that maintain subscriptions containing a keyword λ. Given a new object o (broadcast by the main server), each worker w computes subscriptions that contain keywords in $o.\psi$, from $w.I$, while pruning irrelevant subscriptions. After that, w updates the kNN of the corresponding subscriptions.

EXAMPLE 3. *We partition S in Figure 2 into two disjoint subsets for workers w_1 and w_2, based on keyword-oriented partition. Figure 3 illustrates an overview. The left part shows subscriptions and their costs obtained from Equation (1), and the right part shows the partition result, i.e., w_1 has s_1, s_6, s_7, s_9, and s_{10} while w_2 has s_2, s_3, s_4, s_5, and s_8. They are maintained by inverted files (the most right tables).*
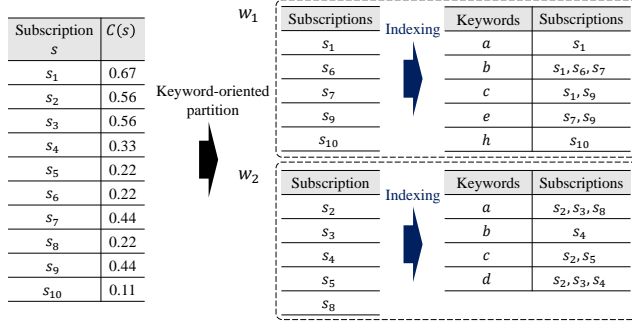
Figure 3: An example of keyword-oriented partition for two workers w_1 and w_2 (based on objects and subscriptions in Figure 2)

Subscription insertion. A new subscription s' also can obtain its estimated cost from Equation (1), because its cost model assumes that the keyword distribution rarely changes [34]. The main server maintains $C(w)$ for each worker w. (This is common to all of our solutions.) Given a new subscription s', the main server computes $C(s)$ from Equation (1). Then the main server assigns s' to the worker with the minimum $C(w)$.

Subscription deletion. For subscription deletion, the main server simply requests the worker, which has the corresponding subscription, to remove it, then updates $C(w)$. This is also common to our solutions.

3.2 Space-oriented Partition

We next design space-oriented partition. The most straightforward approach is to partition the data space into m equal-sized subspaces. Clearly, this is not efficient, because some of them have more objects than the others, which also provides load imbalance. We hence consider a space-based cost model below.

Cost estimation. Consider a set S_r of subscriptions that exist in a subspace r, and let $C(S_r)$ be its cost. Note that $C(S_r)$ can be a probability that kNN objects of subscriptions in S_r may be updated, given a new object o. Let o_k be the current k-th nearest neighbor object of a subscription s. Furthermore, let $B(s)$ be a ball whose center and radius are respectively $s.p$ and $dist(o_k.p, s.p)$. We see that new objects that are generated within $B(s)$ may become new kNN of s. Now consider a rectangle R that encloses all balls of S_r. It is also true that new objects that are generated within R may become new kNNs of $s \in S_r$.

The space-based cost also utilizes the distribution of O_{init}. Given a set O_R of objects existing within R, the probability that a new object is generated within R, $P(R)$, is

$$P(R) = \frac{|O_R|}{|O_{init}|}. \tag{2}$$

Then we define $C(S_r)$ as follows:

$$C(S_r) = P(R) \cdot |S_r| \tag{3}$$

It can be seen that $C(S_r)$ takes the number of subscriptions into account. Assume that R is small but contains many subscriptions. We see that the kNN update cost of R is not small when a new

object is generated within R. However, without $|S_r|$, $C(S_r)$ is small, which contradicts the above intuition. We therefore make Equation (3) an expected value, different from Equation (1).

Subscription partition. Here, we introduce how to obtain R (or r). Let $\mathbb{R}^2$ be the space where objects and subscriptions exist. We partition $\mathbb{R}^2$ in a similar way to quadtree [19], motivated by a recent empirical evaluation on a spatial-keyword stream that confirms the superiority of quadtree-based space partition [5]. Specifically, we partition $\mathbb{R}^2$ into four equal-sized subspaces and compute $C(S_r)$ for each subspace r. Then we pick the subspace that has the largest $C(S_r)$ and partition it in the same way. This is repeated until we have $n \geq \theta \cdot m$, where n and θ are the number of subspaces and a threshold (system parameter), respectively.

Now we have n disjoint subsets of S and determine their assignment in a similar way to Algorithm 1. Note that space-oriented partition considers the assignment of subsets S_r, different from keyword-oriented partition. That is, the input of the greedy algorithm is a collection of subsets S_r.

kNN update. Space-oriented partition takes a different approach from keyword-oriented partition. Assume that a worker w has a collection $S(w)$ of S_r. For each $S_r \in S(w)$, we build an inverted file $I(S_r)$ for S_r. This aims at pruning irrelevant subscriptions, i.e., we can prune S_r when a new object is generated within R but does not contain any keywords in S_r.

Given a new object o that is generated within R, w computes subscriptions that contain the keywords in $o.\psi$ by using $I(S_r)$. If there are such subscriptions, w updates their kNNs.

EXAMPLE 4. *We partition S in Figure 2 into two disjoint subsets for workers w_1 and w_2, based on space-oriented partition. Figure 4 illustrates an example. For simplicity, S is partitioned into four subsets S_1, S_2, S_3, and S_4, see Figure 4(a). Assume that their costs are 0.36, 0.11, 0.2, and 0.28, respectively. Then S_1 and S_2 (S_3 and S_4) are assigned to w_1 (w_2), as shown in Figure 4(b) (4(c)).*

Given a new object o_{10} that is shown in Figures 4(b) and 4(c), w_1 needs to deal with s_1, s_2, s_3, and s_4, because o_{11} exists within the rectangle of S_1. Similarly, w_2 needs to consider s_7.

New subscription. Our space-oriented partition provides a cost with a *set* of subscriptions. On the other hand, for new subscriptions, we should provide their respective costs, because the number of new subscriptions at a given time is much smaller than that of the initial set of subscriptions. We therefore estimate the cost of a new subscription based on Equation (2).

Given a new subscription s, the main server computes its kNN among O_{init} to obtain $B(s)$. (Note that its *exact* kNN is monitored after s is assigned to a worker, as O_{init} is not qualified for $O(s)$.) Then the main server has a rectangle R (i.e., a space r) that encloses $B(s)$. Now we can obtain its cost from Equation (2), because $S_r = \{s\}$, i.e., Equation (3) becomes Equation (2). How to assign s to a worker is the same as keyword-oriented partition.

Although the above approach can deal with new subscriptions, it loses the property of "space-oriented", because a new subscription s may be assigned to a worker w that does not have subscriptions close to s. This case may degrade the pruning performance, because the data space, where w has to care, becomes larger. For example, assume that a new subscription s_{11} is registered and its location is a
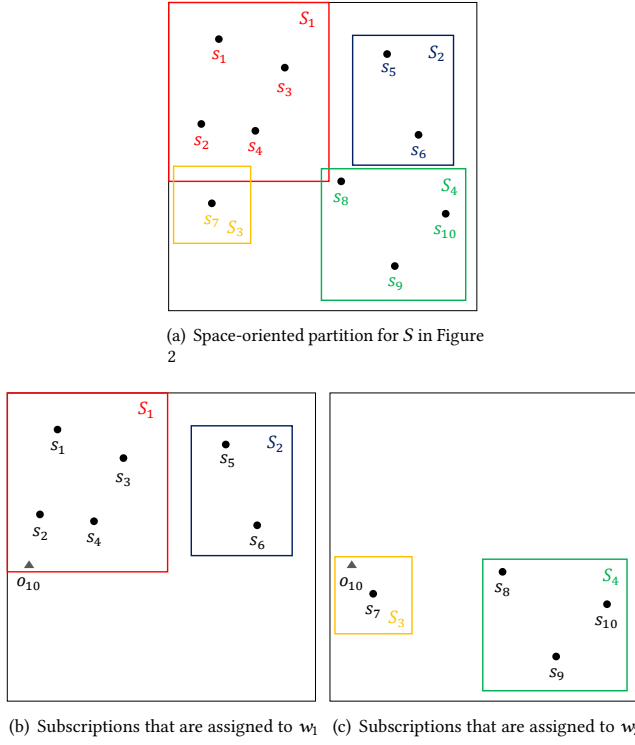
(a) Space-oriented partition for S in Figure 2

(b) Subscriptions that are assigned to w_1 (c) Subscriptions that are assigned to w_2

Figure 4: An example of space-oriented partition for two workers w_1 and w_2 (based on objects and subscriptions in Figure 2)

point in S_2 of Figure 4(a). Assume furthermore that s_{11} is assigned to w_2, then the space, where w_2 has to take care, becomes larger.

4 DkM-SKS

Motivation. Our baselines partition S based only on either keyword or spatial information. However, given a subspace, a better partition approach is dependent on the space and keyword distributions of the subspace. For example:

EXAMPLE 5. *Figure 5 depicts two data distributions. Black points, dashed circles, and balloons represent the locations of subscriptions s, $B(s)$, and keywords of s, respectively.*

Focus on Figure 5(a) and let the solid rectangle show r. We see that $B(s)$ of each subscription s is small, thereby the entire cost is small if we use space-oriented partition for r, because the pruning probability becomes large. Next, consider Figure 5(b). Each subscription has a large $B(s)$ and it overlaps with the others. For this distribution, space-oriented partition is clearly not a good choice, because the size of the rectangle that encloses each ball does not change much even if r is partitioned.

Motivated by the above observation, DkM-SKS considers a better partitioning approach when it partitions a (sub)set of subscriptions, to minimize the entire load. Then DkM-SKS assigns each subscription to a specific worker based on the greedy algorithm [20] and an additional heuristic.

(a) A distribution in which space-oriented partition is effective (b) A distribution in which space-oriented partition is not effective
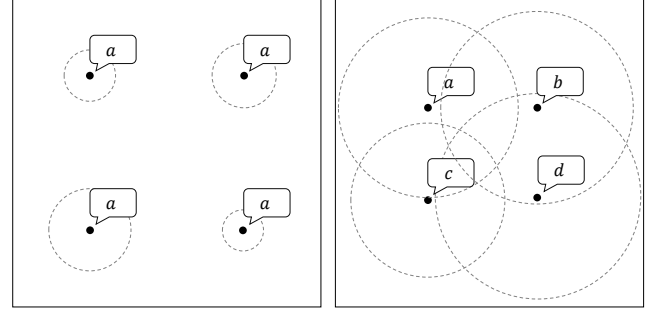
Figure 5: An example that depicts data distributions for considering a better partitioning approach. Black points, dashed circles, and balloons represent the locations of subscriptions s, $B(s)$, and keywords of s, respectively.

4.1 Cost Estimation

DkM-SKS also utilizes O_{init} to estimate the cost of a subscription. Different from keyword- and space-oriented partition, DkM-SKS considers both space and keyword information. Consider S_r a subset of S, and we have a rectangle R that encloses the balls of subscriptions in S_r. Based on a similar idea to Equation (2), the probability that an object o is generated within R and there exist the other objects in R which contain a keyword $\lambda \in o.\psi$ is

$$P(R, \lambda) = \frac{|O_{R,\lambda}|}{|O_{init}|}, \qquad (4)$$

where $O_{R,\lambda}$ is a set of objects $\in O_{init}$ that exist within R and contain λ in their keywords. Now take a subscription $s \in S_r$. Equation (4) focuses on a single keyword, thereby an estimated cost of s is

$$C(s) = \sum_{\lambda \in s.\psi} P(R, \lambda). \qquad (5)$$

Then the cost of S_r is defined as

$$C(S_r) = \sum_{s \in S_r} C(s). \qquad (6)$$

Note that DkM-SKS provides a cost both with a single subscription and a set of subscriptions.

4.2 Subscription Partition and Assignment

Subscription partition. Given S_r, DkM-SKS selects a better approach to S_r by considering the following two partitioning approaches.

SPACE-ONLY-PARTITION. Consider a space r where S_r exists. This approach partitions r into equal-sized disjoint subspaces r_i ($1 \leq i \leq 4$), as with space-oriented partition. Then DkM-SKS obtains a set of subscriptions S_{r_i} that exist in r_i.

HYBRID-PARTITION. This approach also partitions S_r into four disjoint subsets S_{h_i} ($1 \leq i \leq 4$), but in a different way from SPACE-ONLY-PARTITION. (The reason why this approach obtains four subsets is to be comparable to SPACE-ONLY-PARTITION.) This approach utilizes $C(s)$, which is defined in Equation (5), and considers both space

and keyword information. More specifically, DkM-SKS sorts S_r in descending order of $C(s)$, and runs the greedy algorithm to assign each $s \in S_r$ to S_{h_i} with the minimum $\sum_{s \in S_{h_i}} C(s)$.

We do not consider keyword-only partition, because it does not consider spatial information and cannot reduce the size of R.

We define a better partition as the one with less entire cost than the other after S_r is partitioned. The entire costs C_s and C_h, which are respectively provided by SPACE-ONLY-PARTITION and HYBRID-PARTITION, are defined as

$$C_s = \sum_{1 \leq i \leq 4} C(S_{r_i})$$

and

$$C_h = \sum_{1 \leq i \leq 4} C(S_{h_i}).$$

If $C_s < C_h$, DkM-SKS selects SPACE-ONLY-PARTITION. Otherwise, DkM-SKS selects HYBRID-PARTITION.

Algorithm description. Now we are ready to introduce how to partition S through DkM-SKS. Algorithm 2 describes the detail. The objective of this algorithm is to obtain at least $\gamma_1 \cdot m$ subsets of S, where γ_1 is a system parameter.

For ease of presentation, assume that we are given a subset S' of S. (At initialization, $S' = S$.) DkM-SKS considers partitioning S'. Because HYBRID-PARTITION needs to compute Equation (5) for each subscription in S', it incurs a large computational cost if $|S'|$ is large. Therefore, if $|S'| > \gamma_2$, where γ_2 is also a system parameter, DkM-SKS always utilizes SPACE-ONLY-PARTITION to partition S' into four subsets. On the other hand, if $|S'| \leq \gamma_2$, DkM-SKS tests both SPACE-ONLY-PARTITION and HYBRID-PARTITION. DkM-SKS then selects the result of SPACE-ONLY-PARTITION if $C_s < C_h$. Otherwise, DkM-SKS selects that of HYBRID-PARTITION. The four subsets obtained by a better partition are inserted into a collection $\mathcal{S}$ of subsets. After that, $\mathcal{S}$ is sorted in descending order of the estimated cost of subset. DkM-SKS checks $|\mathcal{S}|$, and if $|\mathcal{S}| < \gamma_1 \cdot m$, D$k$M-SKS picks the subset with the largest cost and repeats the above operations.

Subscription assignment. From the subscription partition, DkM-SKS has a collection $\mathcal{S}$ of subsets S'. It is important to note that S' tends to contain subscriptions with close locations and similar keyword sets. That is, given a new object o, the kNN of all subscriptions in S' may change by o. In this case, assigning each subscription $s \in S'$ to a different worker is better than assigning S' to a worker, to exploit the parallel kNN update. We use this heuristic for subscription assignment.

Algorithm description. DkM-SKS employs a similar approach to keyword-oriented partition for subscription assignment. In other words, DkM-SKS uses the greedy algorithm [20], but how to access subscriptions is different. Given $\mathcal{S}$, we first sort $\mathcal{S}$ in descending order of cost obtained from Equation (6). Then, for each $S' \in \mathcal{S}$, we sort S' as with $\mathcal{S}$ and run the greedy algorithm. Algorithm 3 elaborates this operation.

4.3 kNN Update Algorithm

Actually, DkM-SKS can employ an arbitrary index for updating kNN of each subscription. This is a good property because it can always make use of a state-of-the-art. By default, in DkM-SKS, each

---

Algorithm 2: SUBSCRIPTION-PARTITIONING

Input: S (a set of subscriptions), m workers, γ_1, and γ_2 (system parameters)

1   $\mathcal{S} \leftarrow \langle S, 0 \rangle$
2   **while** $|\mathcal{S}| < \gamma_1 \cdot m$ **do**
3     $\langle S', C(S') \rangle \leftarrow$ the front of $\mathcal{S}$
4     $\mathcal{S} \leftarrow \mathcal{S} - \langle S', C(S') \rangle$
5   **if** $|S'| > \gamma_2$ **then**
6       $\mathbb{S} \leftarrow$ SPACE-ONLY-PARTITION(S')
7   **for** each $S_r \in \mathbb{S}$ **do**
8         $\mathcal{S} \leftarrow \mathcal{S} \cup \langle S_r, C(S_r) \rangle$
9   **else**
10       $\mathbb{S} \leftarrow$ SPACE-ONLY-PARTITION(S')
11       $\mathbb{S}' \leftarrow$ HYBRID-PARTITION(S')
12   **if** $C_s < C_h$ **then**
13   **for** each $S_r \in \mathbb{S}$ **do**
14           $\mathcal{S} \leftarrow \mathcal{S} \cup \langle S_r, C(S_r) \rangle$
15   **else**
16   **for** each $S_h \in \mathbb{S}'$ **do**
17           $\mathcal{S} \leftarrow \mathcal{S} \cup \langle S_h, C(S_h) \rangle$
18   Sort $\mathcal{S}$ in descending order of $C(S')$
19   **return** $\mathcal{S}$

Algorithm 3: SUBSCRIPTION-ASSIGNMENT for DkM-SKS

Input: $\mathcal{S}$ (a collection of subsets of S) and m workers
1   Set $C(w) = 0$ for each worker
2   Sort $\mathcal{S}$ in descending order of cost
3   **for** each $S' \in \mathcal{S}$ **do**
4   Sort S' in descending order of cost
5   **for** each $s \in S'$ **do**
6       $w \leftarrow \underset{m}{\arg\min}\, C(w)$
7       $S(w) \leftarrow S(w) \cup \{s\}$, $C(w) \leftarrow C(w) + C(s)$

---

worker w utilizes a hybrid structure of a grid and an inverted file, because this structure is update-friendly. The grid is a set of cells, and for each cell, we implement an inverted file. More specifically, consider a subscription $s \in S(w)$ and a cell g that overlaps with a ball of s, $B(s)$. This cell g maintains s by its inverted file $g.I$ (i.e., $g.I[\lambda]$ maintains s if $\lambda \in s.\psi$).

Given a new object o broadcast by the main server, each worker w obtains the cell g to which o is mapped. Then w considers whether or not it needs to update the kNN of subscriptions in $S(w)$ from $g.I$ (i.e., subscriptions that do not contain any keywords in $o.\psi$ are pruned). If necessary, w updates the kNN of corresponding subscriptions, then updates $g.I$ accordingly.

EXAMPLE 6. *DkM-SKS partitions S in Figure 2 into two disjoint subsets for two workers w_1 and w_2. Assume that the table in Figure 6(a) depicts the estimated cost of each subscription in DkM-SKS. Assume furthermore that the result of partitioning is $\{S_1, S_2, S_3, S_4, S_5\}$,*
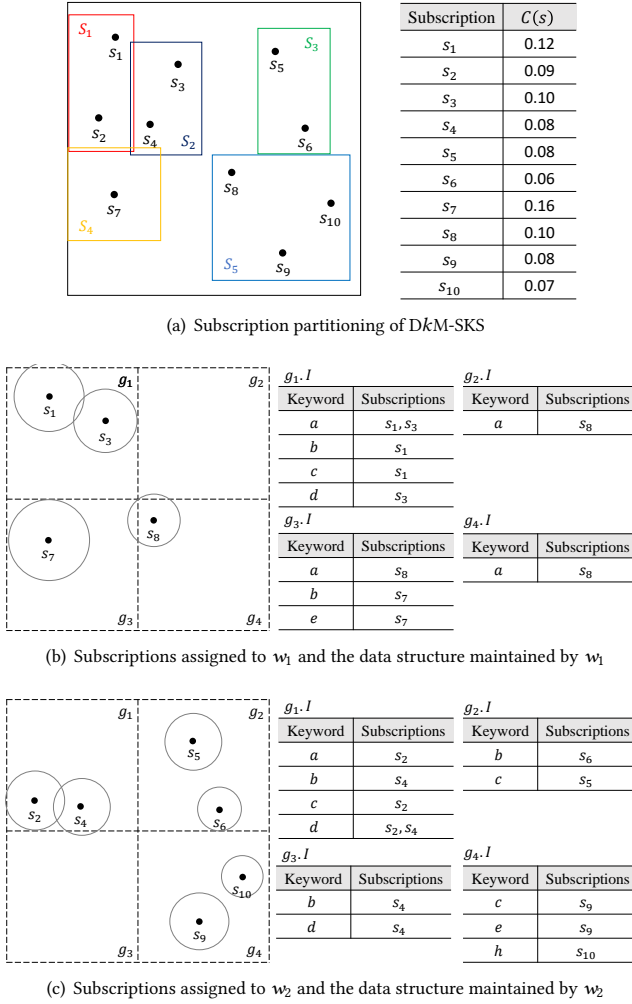
Subscription	$C(s)$
s_1	0.12
s_2	0.09
s_3	0.10
s_4	0.08
s_5	0.08
s_6	0.06
s_7	0.16
s_8	0.10
s_9	0.08
s_{10}	0.07

(a) Subscription partitioning of DkM-SKS

$g_1.I$

Keyword	Subscriptions
a	s_1, s_3
b	s_1
c	s_1
d	s_3

$g_2.I$

Keyword	Subscriptions
a	s_8

$g_3.I$

Keyword	Subscriptions
a	s_8
b	s_7
e	s_7

$g_4.I$

Keyword	Subscriptions
a	s_8

(b) Subscriptions assigned to w_1 and the data structure maintained by w_1

$g_1.I$

Keyword	Subscriptions
a	s_2
b	s_4
c	s_2
d	s_2, s_4

$g_2.I$

Keyword	Subscriptions
b	s_6
c	s_5

$g_3.I$

Keyword	Subscriptions
b	s_4
d	s_4

$g_4.I$

Keyword	Subscriptions
c	s_9
e	s_9
h	s_{10}

(c) Subscriptions assigned to w_2 and the data structure maintained by w_2

Figure 6: An example of subscription partitioning and assignment of DkM-SKS

which is also shown in Figure 6(a). Following Algorithm 3, the result of subscription assignment of DkM-SKS is $\{s_1, s_3, s_7, s_8\}$ for w_1 and $\{s_2, s_4, s_5, s_6, s_9, s_{10}\}$ for w_2, which are respectively illustrated in Figures 6(b) and 6(c).

Consider a case where a new object o_{10} (see Figure 4), which contains keyword b, is generated. Since it is mapped to g_3, w_1 needs to consider s_7, which can be seen from $g_3.I[b]$. Similarly, w_2 needs to consider s_4. Compared with Example 4, DkM-SKS shows a better load balancing.

4.4 Dealing with New Subscriptions

Recall that the estimated cost of a subscription s is obtained from $S_r \subset S$ ($s \in S_r$), as described in Equations (4) and (5). When a new subscription s_n is registered, it does not belong to any subset of S. A straightforward approach to providing an estimated cost with s_n is to re-conduct Algorithm 2. It is obvious that this approach is very computationally expensive. However, it is desirable that s_n is

handled as if s_n has been registered at the initial time. We therefore take an approximate approach to estimating the cost of s_n.

Consider a collection $\mathcal{S}$ of subsets of S obtained by Algorithm 2. The main server maintains R for each subset $S_r \in \mathcal{S}$. Given a new subscription s_n, we first do the same operation as space-oriented partition: the main server computes its kNN among O_{init}, and then computes the rectangle R_n that encloses $B(s_n)$. Let $R \cap R_n$ be the overlapping area between R and R_n. Now $|R \cap R_n|$ can be the overlapped area size. The main server computes

$$S^* = \arg\max_{S_r \in \mathcal{S}} |R \cap R_n|. \tag{7}$$

(In practice, $|\mathcal{S}|$ is small, so the cost of this computation is trivial.) Let R^* be the rectangle of S^*, and R^* is the rectangle that overlaps with R_n the most. By using Equation (5) with R^*, s_n obtains its estimated cost. Then s_n is assigned to the worker w with the minimum cost $C(w)$.

5 EXPERIMENT

5.1 Setting

We conducted experiments on a cluster of six machines. One of them is a main server with 3.0GHz Intel Xeon Gold with 512GB RAM. The others are equipped with 6-core 2.4GHz Intel Core i7-8700T and 32GB RAM. We used one core as a worker. The main server and workers communicate via a 1Gbps Ethernet network.

As with [32], we set $|O_{init}| = 1,000,000$. That is, when 1,000,000 objects were generated, we partitioned S into m workers. After that, we generated 1,000 objects and requested 100 subscription insertions and deletions *for each time-stamp*.

Dataset. We used two real spatial-keyword stream datasets, Place [1] and Twitter [2]. Table 1 shows the statistics of these datasets. We generated subscriptions for each dataset, so that they follow the distributions of the corresponding dataset [31]. When a subscription s was generated, we randomly picked one object to determine its location $s.p$ and then picked at most five keywords at uniformly random from its keyword set to obtain $s.\psi$. The value of $s.k$ was a random integer $\in [1, k_{max}]$.

Algorithm. We evaluated the following algorithms:

- PS^{2}Stream [14]: a state-of-the-art algorithm for continuous spatial-keyword *range* queries. We extend the original algorithm so that it can deal with our problem.
- KOP: our first baseline, keyword-oriented partition, introduced in Section 3.1.
- SOP: our second baseline, space-oriented partition, introduced in Section 3.2.
- DkM-SKS: our proposed solution in this paper.

All algorithms were implemented in C++.

Parameter. The default values of m, k_{max}, and the initial $|S|$ are 20, 10, and 10,000,000, respectively. When we investigated the impact of a given parameter, the other parameters were fixed. In addition, we set $\theta = 20$, $\gamma_1 = 100,000$, and $\gamma_2 = 20$ from preliminary experiments.

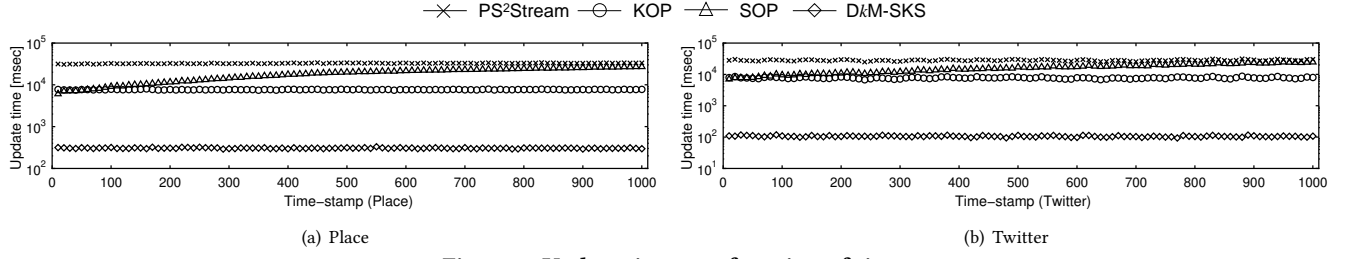**Criteria.** To evaluate the performance of each algorithm, we measured the following criteria:

Shohei Tsuruoka, Daichi Amagata, Shunya Nishio, and Takahiro Hara

(a) Place

(b) Twitter

Figure 7: Update time as a function of time

Table 1: Dataset statistics

Dataset	Place	Twitter
Cardinality	9,356,750	20,000,000
Distinct # of keywords	53,931	2,225,654
Avg. # of keywords in one object	2.94	5.51

Table 2: Load balance [msec] (default parameters)

Algorithm	PS^{2}Stream	KOP	SOP	DkM-SKS
Place	27490.11	549.23	14037.33	32.08
Twitter	21962.50	486.97	12265.40	51.03

Table 3: Decomposed time [msec] on Place

Algorithm	PS^{2}Stream	KOP	SOP	DkM-SKS
kNN update	32892.90	7684.48	18549.04	471.00
Subscription ins.	1.51	1.59	1110.65	35.36
Subscription del.	1.54	1.42	0.79	1.10

- Update time: this is the average computation time for updating kNN objects of all registered subscriptions and time for dealing with subscription insertions and deletions, per time-stamp.
- Load balance: this is the average difference between the maximum and the minimum time to finish kNN update between workers, per time-stamp.

5.2 Results

Justification of using O_{init} and analysis. We first empirically demonstrate that cost estimation based on O_{init} functions well. Figure 7 depicts the time-series of the update time of each algorithm. The result of DkM-SKS shows that its update time does not vary even as new objects are given, new subscriptions are inserted, and some subscriptions are removed, on both Place and Twitter. This suggests that DkM-SKS keeps load balancing, and its cost estimation yields this result. Table 2 depicts that the load of each worker in DkM-SKS is actually balanced.

We see that PS^{2}Stream also has this tendency. However, this result does not mean that PS^{2}Stream has load balancing. We observed that the initial partition of PS^{2}Stream incurs very imbalance partition, i.e., one worker w has a very heavy load at initialization. Because of this, new subscriptions are assigned to the other workers, but this does not overcome the load imbalance. Therefore, the update time of PS^{2}Stream is simply affected by the load of w. This is also confirmed from Table 2, which shows that the load in PS^{2}Stream is significantly imbalance.

Next, we focus on KOP. This algorithm also has a similar result. From Figure 7 and Table 2, we see that the load balance of KOP is not so bad. Because both subscription partition and kNN update in KOP consider only keyword information, they go well together. However, KOP is outperformed by DkM-SKS, which considers both spatial and keyword information. This result confirms the effectiveness of the partitioning approach in DkM-SKS.

Let us consider SOP here. Figure 7 shows that, different from the other algorithms, the update time of SOP increases, as time goes by. This is derived from low accuracy of its cost estimation for new subscriptions. Specifically, given a new subscription, its cost estimated

by SOP is usually small, although it is large in practice. Because of this, the load of a worker, which has new subscriptions, becomes heavy and bottleneck of the system. Table 2 also demonstrates this fact.

Last, we investigate the detail of update time. Table 3 decomposes the update time of each algorithm on Twitter into kNN update time, subscription insertion time, and subscription deletion time, each of which includes index update time. The result on Twitter is omitted, because its tendency is similar to that on Place. It can be seen that the main part of the update time is kNN update time, and subscription deletion needs a trivial cost. DkM-SKS significantly outperforms (is more than 10 times faster than) the other algorithms and exploits available workers to reduce kNN update time. We see that the query insertion time of DkM-SKS is slower than those of KOP and PS^{2}Stream. This is because DkM-SKS needs to compute Equation (7), which incurs a more cost than Equation (2). Also, we can observe that the subscription insertion time of SOP is much longer than those of the others. As explained earlier, (most) new subscriptions are assigned to a single worker w. Hence w incurs a long index update time.

Varying m. We next study the impact of m, the number of workers. Figure 8 depicts the result. Since PS^{2}Stream is significantly outperformed by DkM-SKS, we omit its result.

We see that each algorithm reduces its update time as m increases. This is an intuitive result, since subscriptions are distributed to more workers. The load balances of KOP and DkM-SKS are not affected by m so much, since their cost estimations yield balanced load. On the other hand, as m increases, the load balance of SOP decreases. The reason is simple: the update time of the worker with the largest load becomes shorter as m increases.

(a) Update time (Place)　　(b) Update time (Twitter)　　(c) Load balance (Place)　　(d) Load balance (Twitter)

Figure 8: Impact of m

(a) Update time (Place)　　(b) Update time (Twitter)　　(c) Load balance (Place)　　(d) Load balance (Twitter)

Figure 9: Impact of $|S|$

(a) Update time (Place)　　(b) Update time (Twitter)　　(c) Load balance (Place)　　(d) Load balance (Twitter)
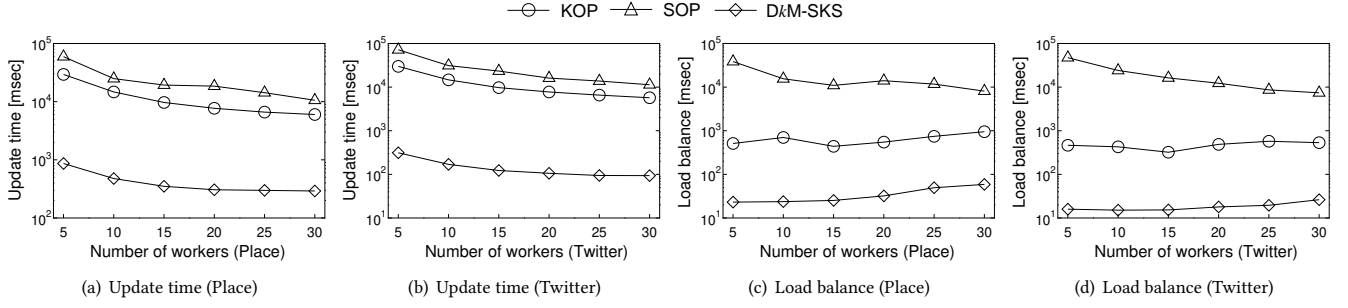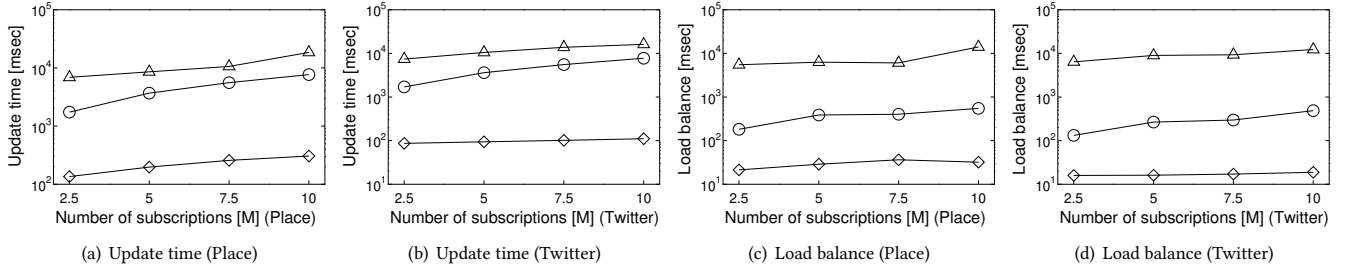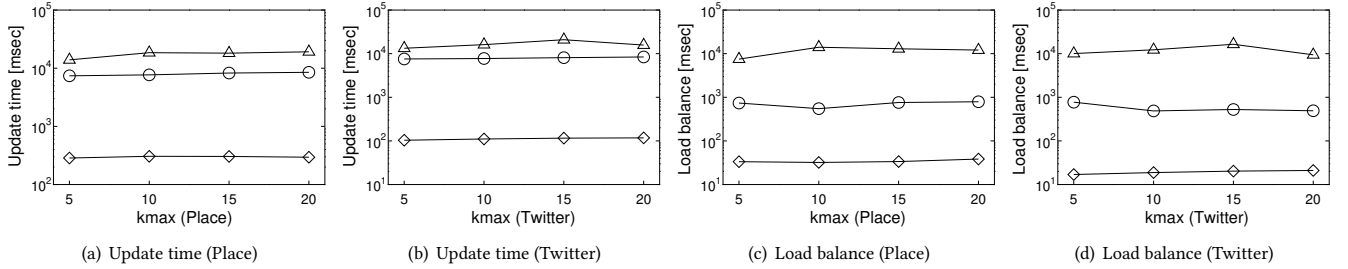
Figure 10: Impact of k_{max}

Varying $|S|$. To investigate the scalability of each algorithm, we studied the influence of $|S|$. Figure 9 shows the result. Due to the load imbalance, the update time of SOP is slow, even when $|S|$ is small. KOP and DkM-SKS scale linearly to $|S|$, so DkM-SKS always outperforms KOP. Note that the linear scalability shows that their subscription assignment functions well. From Figs. 10(c) and 10(d), we see that the load balance of each algorithm normally becomes larger, as $|S|$ increases. It is however trivial increase. For example, in DkM-SKS on Twitter, the difference is only 40[msec] between the cases of $|S| = 2.5 \cdot 10^6$ and $|S| = 10 \cdot 10^6$.

Varying k_{max}. Last, we study the impact of k_{max}, and the result is shown in Figure 10. The update time of KOP and DkM-SKS increases slightly as k_{max} increases. This is also a reasonable result, because, as k_{max} increases, the probability that new objects update the kNN of some subscriptions becomes higher. SOP shows a different pattern to KOP and DkM-SKS, because of its load imbalance. It can be seen that the update time of SOP is simply affected by its load balance.

6 RELATED WORK

Spatial-keyword search. Due to the prevalence of spatial-keyword objects, algorithms for efficient searching them have been extensively devised [11]. For indexing a given dataset, these algorithms employ hybrid structures of spatial indices, e.g., R-tree and quadtree, and textual indices, such as inverted files [17, 37]. A famous index is IR-tree [17]. This is an essentially R-tree, but each node contains an inverted file. The hybrid structure of grid and inverted file, which is employed by DkM-SKS, is derived from IR-tree. An R-tree is not update-efficient, because it partitions data space based on a given dataset. We therefore did not employ R-tree-like structures. It is also important to notice that these works consider snapshot queries and static datasets, whereas we consider continuous queries and streaming data.

Some related queries support moving users [35], find potential users [16], and analyze the result of spatial-keyword search [12]. These works are also totally different from our problem.

Distributed query processing system. Recently, distributed spatial query processing systems have been developed on Hadoop,

Spark, and Storm. (DkM-SKS is orthogonal to these systems.) For example, Hadoop-GIS [4] and SpatialHaddop [18] support efficient processing of spatial queries, e.g., range and kNN queries on MapReduce environments. However, they do not consider keyword information and cannot deal with our problem.

Tornado [26] is a system based on Storm [3] and supports spatio-textual queries. The main focus of this system is to achieve efficient spatial-keyword query processing and not to support massive subscriptions. Hence, it is not trivial for Tornado to provide subscription partitioning for continuous spatial-keyword kNN queries. SSTD [13] is also a system that supports spatio-textual queries on streaming data. However, SSTD imposes, for objects, the condition that they have to contain all keywords specified by queries to match the queries. This is too strict, resulting in no matching objects.

Location-aware Pub/Sub. There are many studies that addressed the problem of dealing with spatio-textual subscriptions. Literatures [23, 25, 27, 33] considered continuous boolean *range* queries as subscriptions. Although PS^{2}Stream [14] also deals with boolean range queries, this is the most related work to ours, because it also assumes the same distributed setting as ours. Our empirical study has demonstrated that the cost model proposed in [14] is not efficient for our problem and DkM-SKS significantly outperforms PS^{2}Stream.

Some studies [10, 28, 29, 32] also tackled the problem of spatio-textual kNN (or top-k) monitoring. [10] considers a decay model for streaming data, while [29, 32] do a sliding-window model. In addition, they consider an aggregation function for object scoring, i.e., spatial proximity and keyword (textual) similarity are aggregated to a score through a weighting parameter α. Based on this scoring function, they monitor top-k objects for each subscription. Their techniques are specific to this scoring function and their assumed streaming model (decay or sliding-window), thereby cannot deal with our problem. Besides, it is well-known that specifying an appropriate α is generally hard for ordinary users [21]. We therefore consider boolean-based kNN monitoring, which is more user-friendly.

7 CONCLUSION

In this paper, to scale well to massive objects and subscriptions in location-aware Pub/Sub environments, we proposed DkM-SKS, a distributed solution to the problem of spatial-keyword kNN monitoring of massive subscriptions. DkM-SKS employs a new cost model to effectively reflect the load of a given subscription. Besides, DkM-SKS partitions a set of subscriptions so that the entire load becomes as small as possible, then assigns each subscription to a specific worker while considering load balancing. We conducted experiments on two real datasets, and the results demonstrate that DkM-SKS outperforms baselines and a state-of-the-art and scales well to massive subscriptions.

REFERENCES

[1] https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces.
[2] http://www.ntu.edu.sg/home/gaocong/datacode.html.
[3] http://storm.apache.org/.
[4] 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. *PVLDB*, 6(11):1009–1020, 2013.
[5] A. Almaslukh and A. Magdy. Evaluating spatial-keyword queries on streaming data. In *SIGSPATIAL*, pages 209–218, 2018.
[6] D. Amagata and T. Hara. Diversified set monitoring over distributed data streams. In *DEBS*, pages 1–12, 2016.
[7] D. Amagata and T. Hara. Identifying the most interactive object in spatial databases. In *ICDE*, pages 1286–1297, 2019.
[8] D. Amagata, T. Hara, and S. Nishio. Distributed top-k query processing on multi-dimensional data with keywords. In *SSDBM*, pages 10:1–10:12, 2015.
[9] D. Amagata, T. Hara, and M. Onizuka. Space filling approach for distributed processing of top-k dominating queries. *IEEE Transactions on Knowledge and Data Engineering*, 30(6):1150–1163, 2018.
[10] L. Chen, G. Cong, X. Cao, and K.-L. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, pages 255–266, 2015.
[11] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. *PVLDB*, 6(3):217–228, 2013.
[12] L. Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu. Answering why-not questions on spatial keyword top-k queries. In *ICDE*, pages 279–290, 2015.
[13] Y. Chen, Z. Chen, G. Cong, A. R. Mahmood, and W. G. Aref. Sstd: A distributed system on streaming spatio-textual data. *PVLDB*, 13(11):2284–2296.
[14] Z. Chen, G. Cong, Z. Zhang, T. Z. Fuz, and L. Chen. Distributed publish/subscribe query processing on the spatio-textual data stream. In *ICDE*, pages 1095–1106, 2017.
[15] F. M. Choudhury, J. S. Culpepper, Z. Bao, and T. Sellis. Batch processing of top-k spatial-textual queries. *ACM Transactions on Spatial Algorithms and Systems*, 3(4):1–40, 2018.
[16] F. M. Choudhury, J. S. Culpepper, T. Sellis, and X. Cao. Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. *PVLDB*, 9(6):456–467, 2016.
[17] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
[18] A. Eldawy and M. F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE*, pages 1352–1363, 2015.
[19] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
[20] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.
[21] Z. He and E. Lo. Answering why-not questions on top-k queries. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1300–1315, 2012.
[22] H. Hu, Y. Liu, G. Li, J. Feng, and K.-L. Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, pages 711–722, 2015.
[23] G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish/subscribe. In *KDD*, pages 802–810, 2013.
[24] S. Luo, Y. Luo, S. Zhou, G. Cong, J. Guan, and Z. Yong. Distributed spatial keyword querying on road networks. In *EDBT*, pages 235–246, 2014.
[25] A. R. Mahmood, A. M. Aly, and W. G. Aref. Fast: frequency-aware indexing for spatio-textual data streams. In *ICDE*, pages 305–316, 2018.
[26] A. R. Mahmood, A. M. Aly, T. Qadah, E. K. Rezig, A. Daghistani, A. Madkour, A. S. Abdelhamid, M. S. Hassan, W. G. Aref, and S. Basalamah. Tornado: A distributed spatio-textual stream processing system. *PVLDB*, 8(12):2020–2023, 2015.
[27] A. R. Mahmood, A. Daghistani, A. M. Aly, M. Tang, S. Basalamah, S. Prabhakar, and W. G. Aref. Adaptive processing of spatial-keyword data over a distributed streaming cluster. In *SIGSPATIAL*, pages 219–228, 2018.
[28] S. Nishio, D. Amagata, and T. Hara. Geo-social keyword top-k data monitoring over sliding window. In *DEXA*, pages 409–424, 2017.
[29] S. Nishio, D. Amagata, and T. Hara. Lamps: Location-aware moving top-k pub/sub. *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[30] M. Qiao, J. Gan, and Y. Tao. Range thresholding on streams. In *SIGMOD*, pages 571–582, 2016.
[31] X. Wang, W. Zhang, Y. Zhang, X. Lin, and Z. Huang. Top-k spatial-keyword publish/subscribe over sliding window. *The VLDB Journal*, 26(3):301–326, 2017.
[32] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang. Skype: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB*, 9(7):588–599, 2016.
[33] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*, pages 1107–1118, 2015.
[34] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: efficiently support location-aware publish/subscribe. *The VLDB Journal*, 24(6):823–848, 2015.
[35] D. Wu, M. L. Yiu, and C. S. Jensen. Moving spatial keyword queries: Formulation, methods, and analysis. *ACM Transactions on Database Systems*, 38(1):1–47, 2013.
[36] S. Yoon, J.-G. Lee, and B. S. Lee. Nets: extremely fast outlier detection from a data stream via set-based processing. *PVLDB*, 12(11):1303–1315, 2019.
[37] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top k spatial keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1706–1721, 2016.