Using Collaborative Edge-Cloud Cache for Search in Internet of Things

Jine Tang, Zhangbing Zhou, Xiao Xue, and Gongwen Wang

***Abstract*—With the Internet of Things (IoT) becoming the infrastructure to support domain applications, IoT search engines have attracted increasing attention from users, industry, and research community, since they are capable of crawling heterogeneous data sources in a highly dynamic environment. IoT search engines have to be able to process tens of thousands of spatial–time–keyword queries per second, making query throughput a critical issue. To achieve this heavy workload, caching mechanisms in collaborative edge-cloud computing architecture, which can implement the caching paradigm in cloud for frequent *n*-hop neighbor activity regions, is first proposed in this article. With our design, the frequent query result can be gained quickly from the spatial–time–keyword filtering index of *n*-hop neighbor regions by modeling keywords relevance and uncertain traveling time. In addition, we use STK-tree proposed previously to directly answer nonfrequent queries. Extensive experiments on real-life and synthetic data sets demonstrate that our proposed method outperforms the state-of-the-art approaches with respect to query time and message number.**

***Index Terms*—Caching paradigm, collaborative edge-cloud computing, frequent query, spatial–time–keyword filtering index, spatial–time–keyword queries.**

I. Introduction

IN RECENT years, the Internet of Things (IoT) [1], [2] has become one of the hottest research topics, and it has already affected the way people study, work, and live [3], [4]. Typically, an IoT system is expected to connect and manage huge numbers of geographically distributed and heterogeneous smart object devices (SODs), such as sensors and/or monitoring devices. In this setting, users aim to get the state information of real-world objects, such as vehicles, buildings, forests, airports, traffic, and strive to make analysis, and decision in a near-real-time fashion. For instance, scientists want to realize the timely and intelligent situation analysis of disaster-affected regions from sensory data of SODs transmitted to the server side. However, it is still limited in timely data analysis and achievement. To reduce the processing latency, the data generated by SOD requires to be performed as close as possible to moving SODs or the base station. The use of nearby resources for computing is growing year by year, and it is called fog computing [5]–[7]. Further, the frequent data generated for SODs requires to be cached and performed in the cloud data center, which has high performance distributed computing and storage capacity, and can perform complex, long-term decision making. The intuition is that the fog and cloud cache make it possible to reduce network traffic and service latency to a large extent, thus promoting the spatial–temporal–keyword-based SOD search to be a fast and promising service in collaborative edge-cloud-cache-based IoT system [8]–[10].

Due to increasing application demands and rapid technological advances in IoT systems, the IoT search engine has been receiving a lot of attention from both industry and academia [11]–[14]. Most of the current research on IoT search engine deals with limited and simple query constraints leveraging layered architecture [15]–[17], such as keyword-based search [15]–[17], static-location-based search [16]–[18], or current-state-based search [19]. Snoogle [16] and Microsearch [11] cope with keywords constraints in a two-tier distributed architecture. Dyser [19] presents a two-tier centralized architecture, in which it supports real-world entities search with a user-specified current state through keyword matching. Recently, the research on the IoT search engine complicates the query request, adding spatial–temporal constraints along with keywords. IoT-SVKSearch [20] aims to resolve this kind of query constraints. It involves an *index master server* and multiple *index node servers*. Each index node server is composed of a set of hierarchical trees that are used to index either full-text keywords or the time-stamped, dynamically changing moving locations, of object devices. Therefore, the search engine cannot simultaneously process the spatial–temporal–keyword search of moving object devices. To resolve this problem, SMSTK search engine [21] proposes a coding-enabled index technology to seamlessly integrate spatial–temporal–keyword proximity. While providing efficient search techniques, the

Manuscript received June 5, 2019; revised September 10, 2019; accepted October 5, 2019. Date of publication October 10, 2019; date of current version February 11, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61702232, Grant 61772479, Grant 61972276, Grant 61572350, Grant 61832014, and Grant 41701133, in part by the National Key Research and Development Program of China under Grant 2017YFB1401200, in part by the Science and Technology Planning Project of Guangdong Province under Grant 2017A050506057, and in part by the Natural Science Foundation of Henan Province under Grant 162300410121. *(Corresponding author: Xiao Xue.)*

J. Tang is with the School of Artificial Intelligence, Hebei University of Technology, Tianjin 300401, China (e-mail: tangjine2008@163.com).

Z. Zhou is with the School of Information Engineering, China University of Geosciences Beijing, Beijing 100083, China, and also with Computer Science Department, TELECOM SudParis, 91000 Évry, France (e-mail: zhangbing.zhou@gmail.com).

X. Xue is with the School of Computer Software, College of Intelligence and Computing, Tianjin University, Tianjin 300072, China, and also with the School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454003, China (e-mail: jzxuexiao@tju.edu.cn).

G. Wang is with the School of Earth Sciences and Resources, China University of Geosciences Beijing, Beijing 100083, China (e-mail: gwwang@cugb.edu.cn).

Digital Object Identifier 10.1109/JIOT.2019.2946389

disperse search strategy or complicated query condition results in longer search latency and high communication cost when a series of consecutive query requests arrive.

Targeting at the above problems, we develop a collaborative edge-cloud-cache-based IoT search engine over moving SODs. To be specific, in this article, this search engine is constructed in the following procedures. We first analyze the system structure and give an overview of the collaborative edge-cloud-cache-based search framework. Then, we investigate how to construct the spatial–time–keyword filtering index of n-hop frequent neighbor regions. Finally, we explore the predictive range and KNN search algorithms of moving SODs. A comprehensive performance analysis is presented as well. We summarize our contributions as follows.

1) We establish a collaborative edge-cloud cache leveraging a three-tier search architecture in IoT system, where frequent n-hop neighbor regions are organized into a form of the spatial–time–keyword filtering index to facilitate frequent query and communication.
2) As for the nonfrequent query, we process it based on STK-tree [21] proposed previously.
3) We conduct extensive experiments on both real-life and synthetic data sets, and the result demonstrates the efficiency of our approach with respect to query time and message number.

The remainder of this article is organized as follows. In Section II, we present the collaborative edge-cloud-cache-based searching system architecture. In the following, we present the spatial–time–keyword filtering index of n-hop neighbor regions and collaborative edge-cloud-cache-based searching process in Sections III and IV, respectively. Section V contains the evaluation results. Section VI provides a review of related works and Section VII concludes this article.

II. System Design

In this section, we present our hierarchical searching architecture, then give the preliminary of STK-tree and an overview of the collaborative edge-cloud-cache-based searching framework. Table I lists the notations used in this article.

A. System Architecture

An overview of the three-tier collaborative edge-cloud-cache-based searching architecture is given in Fig. 1. We are now ready to describe the architecture and its tiers, as well as the cache embedded in cloud. Intuitively, the tiers should be closely associated with each other to finish the search efficiently and accurately.

1) *Device Tier:* It is the lowest level of the system architecture. This tier involves various SODs, such as sensors and/or monitoring devices, which continuously sample the states of real-world objects. The SODs always fall into some activity region rg_i at current timestamp, and intend to visit the next n-hop activity region rg_j in a future timestamp.
2) *Edge Tier:* An edge device is responsible for collecting and maintaining the data from root nodes of STK-trees,

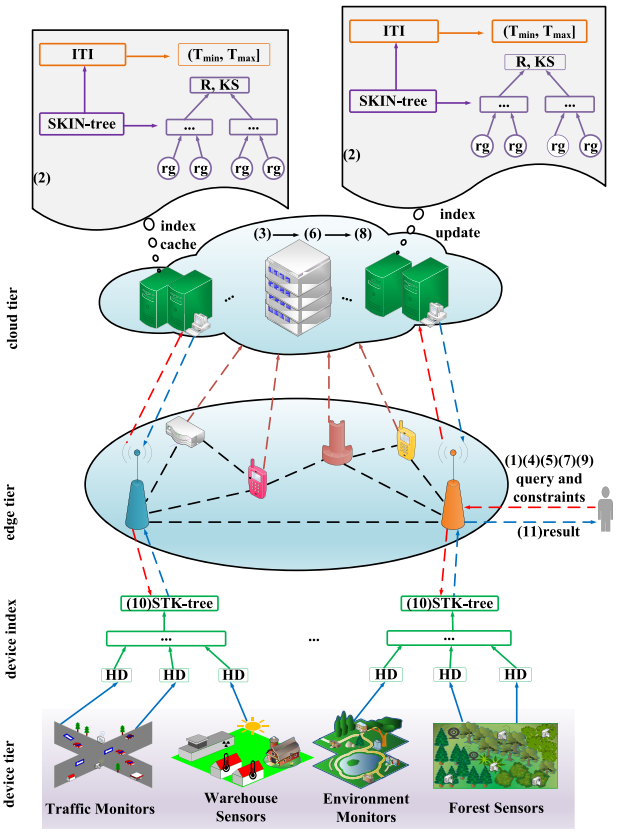

Fig. 1. Overview of CECSE search system architecture.

which are constructed for the SODs in its vicinity. It usually has more computational, communication, and energy resources, as well as a large amount of memory. A collection of edge devices forms a marginal edge network. When a query arrives, the edge device first communicates with its cache in cloud to see whether relevant query results can be directly returned from n-hop neighbor regions. Otherwise, the edge device turns its attention to STK-trees [21] for obtaining the query results. The search framework is shown in Fig. 2, which presents the detailed description of object devices search steps (1)–(11) in Fig. 1.

3) *Cloud Tier:* It is the highest level of system architecture. Each cloud data center manages the frequent activity regions of moving object devices in its corresponding edge device and is responsible for handing their indegree time interval (ITI) and spatial–keyword information in a single-index structure for local search of frequent interest regions. This facilitates the rapid search of moving objects that intend to visit: a) a rectangular range Q_r, which is composed of some activity regions frequently visited and b) at a future time instant t, which is the arrival timestamp closely relevant to the ITI of some activity regions. An edge device forwards the aggregated object device information to the cloud data center so that the cloud can hold the global information from the edge tier and directly return the query result relevant to a certain frequent user request.

TABLE I
NOTATIONS AND THEIR DESCRIPTIONS

Notations	Description
CECSE	collaborative edge-cloud cache based search engine
SOD	smart object device
SKIN-tree	spatial-keyword index of n-hop neighbor regions
STK-tree	online mobile object devices index proposed in [21]
rg	frequent activity region
R	index node region
pr	parent node
HD	head node
ITI	indegree time interval of a certain activity region
T_{min}	the minimum traveling time for ITI
T_{max}	the maximum traveling time for ITI
KS	keywords summarization
KNN	k nearest neighbors
IoT-SVKSearch	hybrid real-time search engine for the Internet of Things based on spatial-temporal, value-based, and keyword-based conditions
SMSTK	search mechanism over STK-tree

B. Preliminary of STK-Tree

STK-tree [21] is based on B^x-tree [22], where each leaf node is augmented with an aggregated spatial–temporal–keyword proximity, denoted as $\text{STK}_{\text{key}} = [TD]_2 \oplus [HV]_2 \oplus [KC]_2$, for searching SODs. *TD* is the time partition in STK-tree. *HV* indicates the Hilbert-curve [23] value of SOD's location in *TD*. *KC* is the encoding value of SODs' keywords. As for nonleaf nodes, they are loaded an STK_{ckey} that involves all the index key values of child nodes for branch node pruning and judgment.

C. Overview of Collaborative Edge-Cloud-Cache-Based Search Framework

Fig. 2 outlines the collaborative edge-cloud-cache-based search framework. This framework includes three parts: 1) predictive query; 2) cache algorithm in an updated period with different traveling time t in cloud; and 3) query result. In this framework, there are interrelated cache actions and query actions. The main flows of the framework are the data flow. When an edge device receives a query requirement, it will go through its corresponding cache algorithms in cloud. The first is to search the spatial–time–keyword filtering index of n-hop neighbor regions by modeling keywords relevance (detailed in Section III), and then to compute the probability of traveling time between interest regions and their n-hop neighbor regions (detailed in Section III-D). Based on the interest regions from search and computation result, the query result can be directly returned from their n-hop neighbor regions. Otherwise, the query constraint will be sent along the original STK-tree for achieving the query result.

III. SPATIAL–TIME–KEYWORD FILTERING INDEX OF n-HOP NEIGHBOR REGIONS

In this section, we introduce the spatial–time–keyword filtering index of n-hop neighbor regions and the relevant filtering method by modeling keywords relevance and uncertain traveling time.

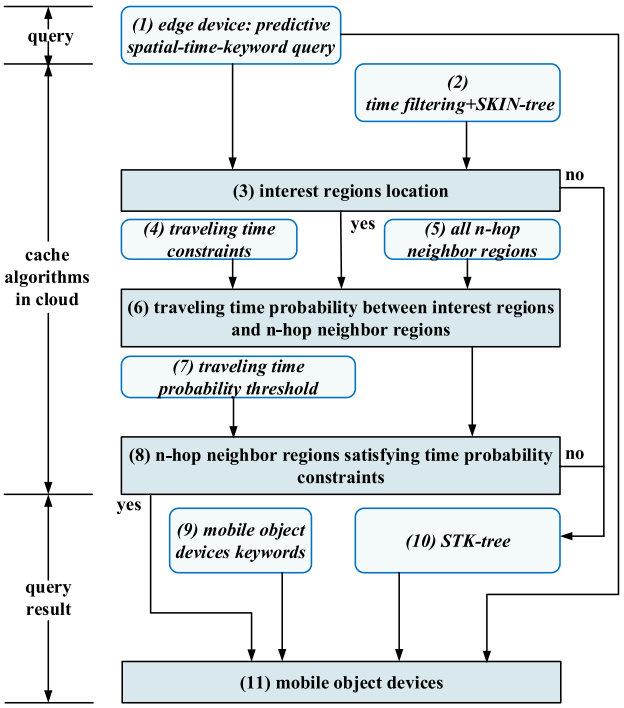

Fig. 2. Framework of collaborative edge-cloud-cache-based mobile object device search.

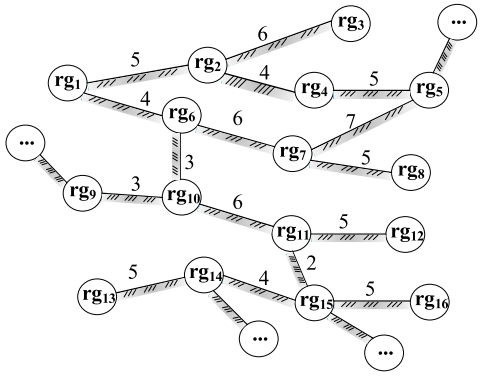

Fig. 3. Frequent updated activity subpaths. Each pair of 1-hop neighbor regions has a traveling time marked.

A. Overview of SKIN-Tree

Definition 1 [Frequent Activity Subpath Region Mining (FSRM)]: Given a set of spatiotemporal activity paths S consisting of n activity regions $Rg = \{rg_1, rg_2, \ldots, rg_n\}$, a subpath $s = (rg_i, \ldots, rg_j)$ in S is a frequent activity path such that the visiting frequency VF_s of s is bigger than the visiting threshold within a time interval t.

Definition 2 (ITI): Given an activity region $rg_i \in Rg$ ($i \in [1, n]$), the n-hop neighbor traveling time interval of rg_i, denoted as *ITI*, can be determined by different moving SODs with uncertain speed, transportation mode, and other impact factors.

We follow Definition 1 to perform FSRM. Let us take Fig. 3 as an example, that presents the frequent updated activity subpaths. Then, the 1-hop neighbor ITI of a certain region, e.g., $rg_i (i \geq 1)$ as shown in Fig. 4, can be recorded and stored.

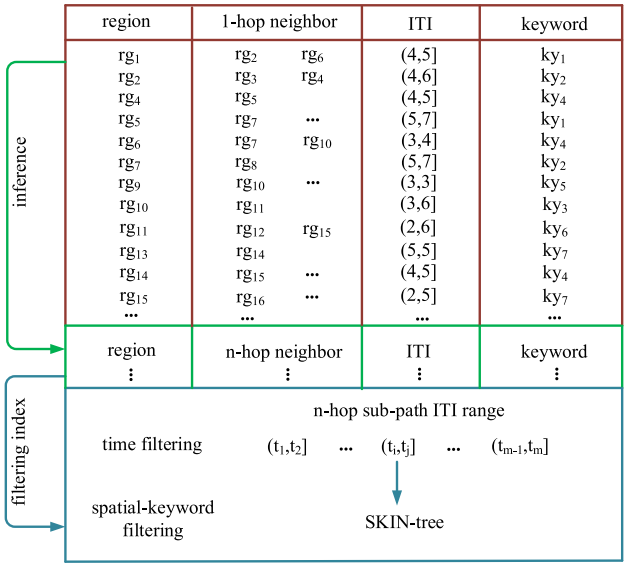

Fig. 4. Spatial–time–keyword filtering index of n-hop neighbor regions.

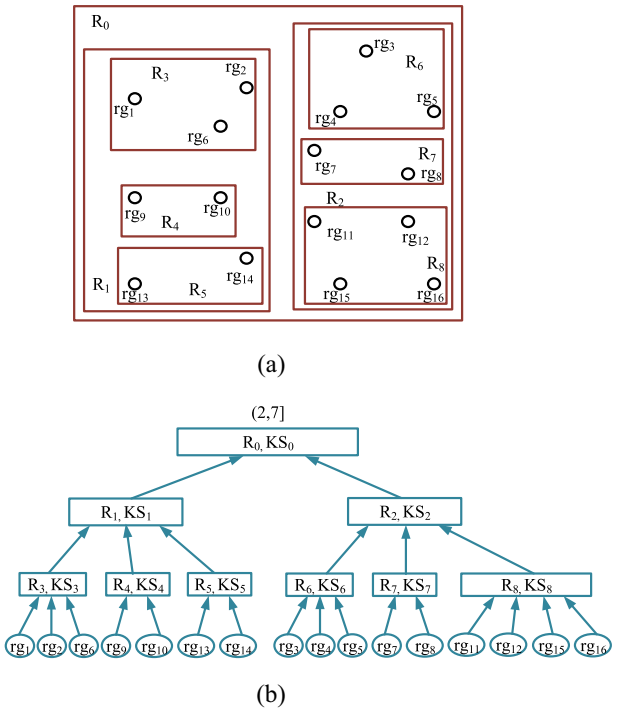

Fig. 5. Division and tree construction for 1-hop neighbor regions in ITI (2,7]. (a) Region division. (b) SKIN-tree construction.

Based on the 1-hop neighbor ITI, we can infer the n-hop neighbor ITI. Leveraging this, we propose a spatial–time–keyword filtering index of n-hop neighbor regions to locate *interest regions*. This index includes: 1) n-hop subpath time interval range, which is used for time filtering and 2) a spatial–keyword index of n-hop neighbor regions, denoted as SKIN-tree (see Section III-B), which can be loaded into each n-hop subpath ITI range for spatial and attribute filtering.

In this article, we adopt the clustering technique to build SKIN-tree. The purpose of clustering n-hop neighbor regions is to make activity regions in the same cluster to concentrate together, and activity regions in different clusters to separate with each other as far as possible. Therefore, the activity regions in the dense zone should be put into the same node region so as to avoid cascading overlap and large coverage, whereas the activity regions in the sparse zone can have simpler subtree structure and better operation performance. The tree built organizes the activity regions according to the clustering results. Each node of the tree corresponds to a cluster. Additionally, a keyword summarization (KS) is loaded into each node, which records the specified keywords and attribute information that the frequent regions in it possess of. Generally, SKIN-tree has the following properties.

1) The root node layer contains the whole minimum boundary region.
2) The interior nodes are the cluster regions, whose number of n-hop neighbor regions is larger than M.
3) The leaf nodes are the clusters, whose number of n-hop neighbor regions is within m and M.
4) Each node is augmented with two parts: a) the parent node, which contains all MBRs of children nodes and b) the KS, which stores the relevant keywords of child nodes.
5) An n-hop neighbor region is contained in only one upper parent node, and its query path is single.
6) SKIN-tree is often an unbalanced tree. Therefore, the leaf nodes are generally not at the same level.

Example 1: We give an example to demonstrate how to construct SKIN-tree based on n-hop subpath ITI range. It is worth emphasizing that multiple ITIs adapt to short query time condition while it requires fewer ITIs for long query time condition. In this example, we select 1-hop subpath ITI range for SKIN-tree construction. The construction method is also suitable to n-hop subpath ITI range.

First, we select the whole ITI (2,7] to construct spatial–keyword filtering. Fig. 5(b) gives an example of SKIN-tree constructed from the 1-hop frequent regions in ITI (2,7], as shown in Fig. 5(a), by setting M as 4. Each nonleaf node corresponds to a subregion of R_i ($i \in [0, 8]$). For example, the node R_4 corresponds to the left-middle subregion, as shown in Fig. 5(a), which includes frequent regions rg_9 and rg_{10}.

Second, we select ITI (2,4] and (4,7] together to construct spatial–keyword filtering. Fig. 6(b) gives an example of SKIN-tree constructed from the frequent regions in ITI (2,4], as shown in Fig. 6(a), by setting M as 4. Fig. 7(b) gives an example of SKIN-tree constructed from the frequent regions in ITI (4,7], as shown in Fig. 7(a), by setting M as 4.

B. SKIN-Tree Construction

In this article, we adopt the classic K-means algorithm for clustering activity regions. Roughly, this method selects k_0 activity regions as initial cluster centers. It then computes the distance between the activity regions and cluster centers, and assigns each activity region to a cluster when the distance between this activity region and the center of this cluster is the shortest. The algorithm continues to update the center of the updated cluster. When no update occurs to cluster centers, the clustering procedure terminates.

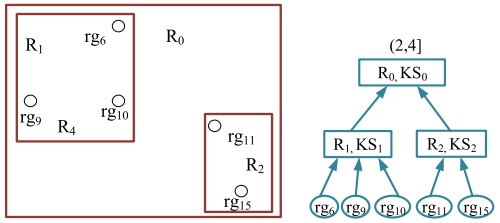

Fig. 6. Division and tree construction for 1-hop neighbor regions in ITI (2,4]. (a) Region division. (b) SKIN-tree construction.

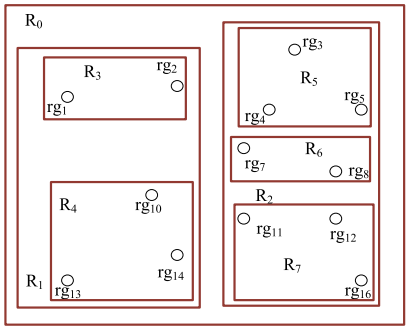

Fig. 7. Division and tree construction for 1-hop neighbor regions in ITI (4,7]. (a) Region division. (b) SKIN-tree construction.

Algorithm 1 presents the construction procedure of our SKIN-tree. We first select k_0 n-hop neighbor regions as the initial cluster centers, and apply the K-means method to group n-hop neighbor regions into k_0 clusters (line 3). If the number of n-hop neighbor regions in a subcluster is within m and M, this subcluster is set as a leaf node of SKIN-tree and assigned a KS according to keyword and attribute information (lines 6–8). When a subcluster covers n-hop neighbor regions in dense regions, the number of n-hop neighbor regions in this cluster is inevitably larger than the maximum value (i.e., M). We call such a cluster an overflow cluster, which needs to be reclustered to form smaller clusters and assigned a KS according to keyword and attribute information (lines 16–18). For a subcluster covering n-hop neighbor regions in sparse regions, the number of n-hop neighbor regions in this cluster may be less than the minimum number (i.e., m). In this case, this cluster should be merged with an overflow cluster nearby, and recluster the merged cluster (lines 9–14).

Given p as the reducing scale of the number of activity regions in each cluster process, the number of activity regions

Algorithm 1 : SKINTreeBuilt

Require: S_{nnr}: the set of n-hop neighbor regions
Ensure: nd_{rt}: the root node of constructed SKIN-tree

1: $k_1 \leftarrow$ the number of n-hop neighbor regions in S_{nnr}
2: **while** $k_1 > M$ **do**
3: $CLS \leftarrow$ select k_0 initial cluster centers ($m \leq k_0 \leq M$), and cluster n-hop neighbor regions using K-means method
4: **foreach** $cls \in CLS$ **do**
5: $k_1 \leftarrow$ the number of n-hop neighbor regions in cls
6: **if** $m \leq k_1 \leq M$ **then**
7: set cls as a leaf node of nd_{rt}
8: assign a keywords summarization to cls
9: **else if** $k_1 < m$ **then**
10: $cls_1 \leftarrow$ merge cls with the nearest overflow cluster denoted as cls_{of}
11: $CLS \leftarrow CLS - \{cls_{of}\}$
12: $nd_{cld} \leftarrow$ SKINTreeBuilt(n-hop neighbor regions in cls_1)
13: set nd_{cld} as a child node of nd_{rt}
14: assign a keywords summarization to nd_{cld}
15: **else**
16: $nd_{cld} \leftarrow$ SKINTreeBuilt(n-hop neighbor regions in cls)
17: set nd_{cld} as a child node of nd_{rt}
18: assign a keywords summarization to nd_{cld}
19: **end if**
20: **end for**
21: **end while**

in the following subcluster regions is appropriately $((n_{rg})/p^i)$ $(i = 0, 1, \ldots)$ from high level to low level, where n_{rg} is the total number of activity regions. The time complexity of clustering activity regions is $O((n_{rg}/p^i) \times k_c \times t_{iv})$, where k_c is the number of clusters and t_{iv} is the iterative time. Usually, $k_c \ll n_{rg}$ and $t_{iv} \ll n_{rg}$. In our clustering algorithm, k_c is a constant and t_{iv} is typically quite small. Therefore, the time complexity of Algorithm 1 is $O(n_{rg})$.

It is worth mentioning that activity subpath regions are frequently updated in a period with different traveling time intervals. In view of query results, which are conditioned on different query ranges and query locations (see Section IV), reflecting the frequently visited regions by moving object devices, it provides a good approximation for caching contents update. As SKIN-tree aims to store frequent updated regions, it suffers update from the insertion and deletion of frequent regions. In this article, we adopt the same insertion and deletion algorithms of R-tree [24] for SKIN-tree update since tree construction is similar for both.

C. Modeling Keywords Relevance

Notice that the frequent activity region path of a moving SOD is closely relevant to the interest regions' attributes. In turn, the specific attribute features of a moving SOD determines what activities it will engage in. Therefore, the attribute features of moving SODs have a certain relevance to the interest regions' attributes, which indicates that the SOD intends to visit the interest regions through n-hop neighbor regions. Motivated by the implicit attribute correlation, we design keywords-preference-based estimation for branch node pruning so as to improve search accuracy.

Definition 3 [Semantic Relevance (SR)]: Let $Ky_i = \{f_{i1} \cdot Ky_{i1}, f_{i2} \cdot Ky_{i2}, \ldots, f_{in} \cdot Ky_{in}\}$ be a KS of frequent region i,

$Ky_j = \{f_{j1} \cdot Ky_{j1}, f_{j1} \cdot Ky_{j2}, \ldots, f_{jn} \cdot Ky_{jn}\}$ be a KS of frequent region j, where f_{il} and f_{jl} ($l \in [1, n]$) are the frequency of Ky_{il} and Ky_{jl}, respectively. We utilize the Jaccard metric to measure the sematic similarity of Ky_i and Ky_j as follows:

$$\mathrm{Sim}\big(Ky_i, Ky_j\big) = \frac{\sum_{l=1}^{n} Ky_{il} \cdot Ky_{jl}}{\sum_{l=1}^{n} Ky_{il}^2 + \sum_{l=1}^{n} Ky_{jl}^2 - \sum_{l=1}^{n} Ky_{il} \cdot Ky_{jl}}. \tag{1}$$

Definition 4 [Keywords Preference Model (KPM)]: Let $S = \{Ky_1, Ky_2, \ldots, Ky_m\}$ be a set of m keyword summarizations relevant to a set of m frequent activity regions, and D be the set of SR between each pair of keyword summarizations (see Definition 3). For a query keyword summarization QKy_i, we define $S_{ij} = \mathrm{Sim}(QKy_i, Ky_j)$ as the SR between QKy_i and Ky_j, where $Ky_j \in S$. The kernel density of S_{ij} is defined as

$$F\big(S_{ij}\big) = \frac{1}{|D|b} \sum_{d' \in D} k\big(\big(S_{ij} - d'\big)/b\big) \tag{2}$$

where b is

$$b = \left(\frac{4\widehat{\sigma}^5}{3n}\right)^{\frac{1}{5}} \approx 1.06\widehat{\sigma} n^{-\frac{1}{5}}. \tag{3}$$

In the following, we introduce the keywords preference estimation (KPE) based on KPM.

Definition 5 (KPE): The preference of QKy_i on frequent region R_j in terms of Ky_j, denoted by KeyRating(QKy_i, Ky_j), can be calculated by taking the average of all its probability densities, i.e.,

$$\mathrm{KeyRating}\big(QKy_i, Ky_j\big) = \frac{1}{m} \sum_{j=1}^{m} F\big(d_{ij}\big). \tag{4}$$

A higher value of KeyRating(Qky_i, Ky_j) indicates that the smart object with Qky_i has a higher probability to visit frequent region in terms of Ky_j.

Lemma 1: Given a minimum similarity threshold α for a pair of KSs, the branch node R_j is pruned if $\mathrm{KeyRating}(QKy_i, Ky_j) < \alpha/n$.

Proof: Assume that $s = \{(rg_1, ky_1), \ldots, (rg_n, ky_n)\}$ is an n-hop frequent activity subpath that an SOD with certain specific feature $QKy_i = \{Qky_{i1}, \ldots, Qky_{in'}\}$ may visit. There is at least one pair of $\mathrm{Sim}(Qky_{ii'}, ky_s) > \alpha, i' \in [1, n']$, where ky_{sp} is the KS of s. If R_j contains the n-hop neighbor region (rg_n, ky_n) that the SOD intends to visit, the semantic similarity $\mathrm{Sim}(Qky_{ii'}, ky_{j'})$ is at least α/n since it requires n-hop neighbor regions to arrive at rg_n. Therefore, we have the result. ■

D. Modeling Uncertain Traveling Time

To model the uncertain traveling time [25], [26] for facilitating query result returned, we present two kinds of traveling time probability analysis as below.

Visiting Just Right at Predictive Time: In this case, we define $Dd_{i,j}(dt(i,j))$ as the probability distribution over traveling time duration $dt(i,j)$ between activity regions rg_i and rg_j. Assume we randomly select two continuous visiting actions of a moving SOD: (rg_i, t_i) and (rg_j, t_j), $Dd_{i,j}(dt(i,j))$ can be obtained by calculating the probability that $t_j - t_i = t_l$, where t_l is the given traveling time interval. In this article, we model $dt(i,j)$ as a thin Gaussian distribution and compare it with $Dd_{i,j}(dt(i,j))$ using symmetric KL divergence to obtain the possibility for visiting just right at predictive time instant.

Definition 6 [Just Right Visiting Possibility (JRVP)]: Given a frequent activity region subpath $s : rg_1 \to rg_2 \to \cdots \to rg_{n-1} \to rg_n$ and its departing timestamp t_{ds}, we can calculate the possibility that how good the path s just right visited at the given predictive time instant t is by defining an excellent duration function

$$f_{\mathrm{dur}}(s) = \left(\prod_{i=1}^{n-1} D_{KL}\Big(g\Big(t, dt(i, i+1), \sigma^2\Big)\Big) \Big\| Dd_{i,i+1}(dt(i, i+1))\right)^{\frac{-1}{n-1}} \tag{5}$$

where $\sum_{i=1}^{n-1} dt(i, i+1) = t - t_{ds}$.

A path s with a higher value of $f_{\mathrm{dur}}(s)$ indicates that such a path can be visited by SODs at just right predictive time instant.

Visiting Before Predictive Time: It is noteworthy that the real traveling time remains uncertain due to many uncertain factors that could affect the traffic. To model this uncertainty, we shall treat the traveling time $dt(i,j)$ between a pair of activity regions rg_i and rg_j as a random variable complying with a certain thin Gaussian distribution with the probability density function $f_{i,j}(\cdot)$. To put it simply, we suppose that $dt(i,j)$ between each pair of (rg_i, rg_j) is independent. In the following, we give the probability estimation of visiting before predictive time instant.

Definition 7 [Before Possibility Estimation (BPE)]: Let x denotes the total traveling time of a subpath s. Let t_{ds} denote the departing timestamp of s and t be the predictive time instant. For any path $s : rg_1 \to rg_2 \to \cdots \to rg_{n-1} \to rg_n$, the probability of x less than the traveling time $t - t_{ds}$ is estimated by

$$p(x < t) = \int \cdots \int_D f_{1,2}\big(\xi_{1,2}\big) \cdots f_{n-1,n}\big(\xi_{n-1,n}\big) d\xi_{1,2} \cdots d\xi_{n-1,n} \tag{6}$$

where $D = \{(\xi_{1,2}, \ldots, \xi_{n-1,n}) \in R_{>0}^{n-1} : 0 < \xi_{1,2} + \cdots + \xi_{n-1,n} < t - t_{ds}\}$. This probability can be computed, given that all the probability density functions $f_{1,2} \cdots f_{n-1,n}$ are known.

Based on the above discussions, n-hop neighbor regions that satisfy the traveling time probability constraint are returned. Therefore, the query result can be returned directly from the n-hop neighbor regions. Otherwise, we adopt the original STK-tree to query the mobile object devices at predictive time instant when there are no frequent n-hop neighbor regions returned.

IV. Collaborative Edge-Cloud-Cache-Based Search Mechanism

In this section, we present SOD search based on the collaborative edge-cloud cache query mechanism in detail.

Algorithm 2 : RangeQuery

Require: Q_r: query range
Q_{ky}: query keywords
α: minimum similarity threshold for a pair of keywords summarizations
Q_{tl}: traveling time interval
s_{rg}: all the frequent n-hop neighbor regions
Ensure: S_{SOD}: SODs satisfying user request

1: $s_{ph} \leftarrow$ the frequent activity regions and traveling time mining from s_{rg}
2: $ITI \leftarrow$ the indegree time interval calculated from s_{ph}
3: SKIN-tree $\leftarrow$ SKINTreeBuilt(s_{ph})
4: $r_{rt} \leftarrow$ the root node of SKIN-tree
5: **if** $Q_{tl} \in ITI$ **then**
6: $s_{nr1} \leftarrow$ SKINQuery(Q_r, Q_{ky}, α, r_{rt})
7: **if** $s_{nr1} = Q_r$ **then**
8: $s_{nr2} \leftarrow$ n-hop neighbor regions that can be achieved from s_{nr1} by computing the traveling time probability
9: **if** $s_{nr2} \neq \emptyset$ **then**
10: $S_{SOD} \leftarrow$ returning the mobile object devices with Q_{ky} in s_{nr2}
11: **end if**
12: **else if** $s_{nr1} \bigcap Q_r \neq \emptyset$ **then**
13: $S_{SOD} \leftarrow$ returning the mobile object devices from original STK-tree for the unresolved query part
14: **else**
15: $S_{SOD} \leftarrow$ returning the mobile object devices from original STK-tree
16: **end if**
17: **else**
18: $S_{SOD} \leftarrow$ returning the mobile object devices from original STK-tree
19: **end if**

Algorithm 3 : SKINQuery

Require: Q_r: query range
Q_{ky}: query keywords
α: minimum similarity threshold for a pair of keywords summarizations
r_{rt}: SKIN-tree root node of n-hop neighbor regions
Ensure: S_{rg}: n-hop neighbor regions satisfying query request

1: **if** $r_{rt} \subseteq Q_r$ and $KeyRating(Q_{ky}, r_{rt}.ky) = 1$ **then**
2: $S_{rg} \leftarrow r_{rt}$
3: **else**
4: **foreach** child node region r_{cld} of r_{rt} **do**
5: **if** $r_{cld} \bigcap Q_r \neq \emptyset$ and $KeyRating(Q_{ky}, r_{cld}.ky) \geq \alpha/n$ **then**
6: **if** r_{cld} corresponds to a leaf node **then**
7: $p_{rg} \leftarrow$ all the partial n-hop neighbor regions with the query keywords in this overlap region
8: $S_{rg} \leftarrow S_{rg} \cup p_{rg}$
9: **else**
10: SKINQuery(Q_r, Q_{ky}, α, r_{cld})
11: **end if**
12: **end if**
13: **end for**
14: **end if**

A. Predictive Range Query

Given a rectangular range Q_r, a future time instant Q_t, as well as the current departing time instant Q_d, a *predictive range query* aims to retrieve all SODs whose moving locations before/at Q_t fall into Q_r. When such a query arrives, the following steps are conducted on the CECSE search engine.

1) The user first queries marginal edge network, that returns edge devices which may include SODs relevant to query request.
2) The returned edge devices first give an examination on frequent activity regions in the corresponding cloud cache, either directly getting query result or through SKIN-tree based on keywords relevance and traveling time probability.
3) When there are no query results returned from cloud cache, edge devices turn their attention to the original STK-tree for query processing.
4) If cached aggregates in cloud for an edge device alone cannot be used to fully compute a query, the cloud generates subqueries for the unresolved portions and send it to edge device for query processing based on the original STK-tree.

Algorithm 2 presents the range query operation based on cache structure in cloud, which aims to return SODs satisfying user query request efficiently. We first examine the traveling time constraint, if the traveling time is out of ITI, the query result will be returned from the original STK-tree (line 18). Otherwise, we carry on search along SKIN-tree based on Algorithm 3 (lines 5 and 6). If the query range is fully searched (line 7), the query result can be returned from n-hop neighbor regions (lines 8–10). If the query range is partially searched (line 12), the query result can be returned from the original STK-tree for unsolved parts (line 13). If the query range is not searched at all, the query result can be returned from the original STK-tree (line 15). Let the number of SODs be n_{sod}, the fanout of SKIN-tree be k_0 and the total number of n-hop neighbor regions be n_{rg}, the time complexity of Algorithm 2 can be derived as $O(\log_{k_0} n_{rg}) + O(n_{\text{sod}}) \approx O(n_{\text{sod}})$, since in most cases, k_0 is a constant and $n_{rg} \ll n_{\text{sod}}$.

Algorithm 3 gives the spatial–keyword filtering operation, which aims to return n-hop neighbor regions satisfying the filtering condition. If the root node region of SKIN-tree is contained in the query region and the whole region matches with query keywords completely, it is returned (lines 1 and 2). Otherwise, it is required to check all child nodes regions. If the child nodes are the leaf nodes which have: 1) overlap with query region and 2) match with query keywords to some extent, then all the partial n-hop neighbor regions matching with query keywords in this overlap region are included into S_{rg} (lines 4–14). If the child nodes are not leaf nodes, the same procedure is processed recursively (line 10). Let the fanout of SKIN-tree be k_0 and the total number of n-hop neighbor regions be n_{rg}, the time complexity of Algorithm 3 can be derived as $O(\log_{k_0} n_{rg})$.

Example 2: We present a 1-hop neighbor example to explain the range query process in Algorithm 2. Given a query range constraint Q, a query keyword constraint ky_6, and a future time instant 4, as shown in Fig. 8, we present the corresponding query process in Fig. 9. One part of Q, i.e., Q_1, falling into the frequent regions, we process it from SKIN-tree through modeling traveling time probability. The remaining part of Q, i.e., Q_2, not in the frequent regions, we process it from STK-tree.

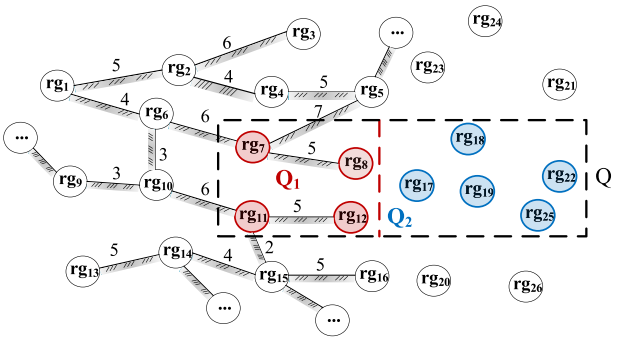

Fig. 8. Example of predictive range query.

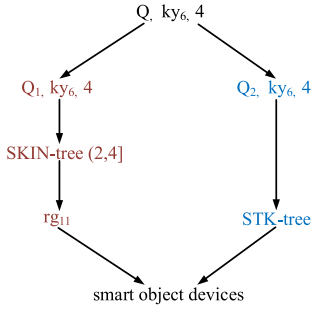

Fig. 9. Range query process of the example in Fig. 8.

Algorithm 4 : KNNQuery

Require: Q_l: query location
Q_{ky}: query keywords
α: minimum similarity threshold for a pair of keywords summarizations
Q_{tl}: traveling time interval
k: nearest neighbors number
s_{rg}: all the frequent n-hop neighbor regions
Ensure: S_{SOD}: smart object devices satisfying user request

1: $D_k = \frac{2}{\sqrt{\pi}}[1 - \sqrt{1 - (\frac{k}{N})^{\frac{1}{2}}}$
2: $r_q = \frac{D_k}{k}$
3: *flag* $\leftarrow 1$
4: **while** *flag* **do**
5: $R \leftarrow GetRange(Q_l, r_q)$
6: $u \leftarrow$ RangeQuery(r_q, Q_{ky}, α, Q_{tl}, s_{rg})
7: $S_{SOD} \leftarrow S_{SOD} \cup u$
8: **if** k neighbors are found **then**
9: $R \leftarrow GetRange(Q_l, r_k)$
10: $u \leftarrow$ RangeQuery(r_q, Q_{ky}, α, Q_{tl}, s_{rg})
11: $S_{SOD} \leftarrow S_{SOD} \cup u$
12: *flag* $\leftarrow 0$
13: **else**
14: $r_q \leftarrow Next_{radius}()$
15: *flag* $\leftarrow 1$
16: **end if**
17: **end while**

B. Predictive KNN Query

Given a location Q_l, a future time instant Q_t, as well as the current departing time instant Q_d, a *predictive KNN query* aims to retrieve k nearest SODs predicted to be with the highest probability to show up around Q_l before/at Q_t. Similar to the predictive range query, when a predictive KNN query comes, the following steps are executed on the CECSE search engine.

1) The user first queries marginal edge network, which returns a ranked list of edge devices which may include k nearest SODs with respect to distance constraint.
2) Then, the user performs a distributed KNN query from the returned list of edge devices to retrieve k qualified answers (see Algorithm 4).

It is noteworthy that KNN query is a gradual extended range query. It starts search with an initial radius r by Algorithm 2. The search radius extended with an increment is required when k SODs are not found in the initial search region. The setting of r and increment is D_k/k, the same as that in [22], where $D_k = (2/\sqrt{\pi})[1-\sqrt{1-(k/N)^{(1/2)}}$, with N as the total number of locations in a unit space.

On the basis of Algorithm 2, Algorithm 4 presents the KNN query procedure in each local SEN region. Initially, the query radius r_q is estimated by the distance between Q_l and its k'th nearest SOD (lines 1 and 2). Then, we use *RangeQuery()* to retrieve SODs within the search range R that is with Q_l as range center and r_q as radius (lines 5 and 6). If k nearest SODs are returned, the query region will be refined by computing the distance between Q_l and its kth nearest SOD found so far (lines 8–11). In case less than k SODs are found, the query radius is extended with an increment for initiating a new round of search (line 14). The time complexity of Algorithm 4 is $O(n_{\text{sod}})$, since the time complexity of carrying on RangeQuery for achieving k nearest SODs is $O(n_{qd} \times n_{\text{sod}})$, where n_{qd} is the number of query rounds and n_{sod} is the number of SODs. Usually, n_{qd} is a constant and $n_{qd} \ll n_{\text{sod}}$.

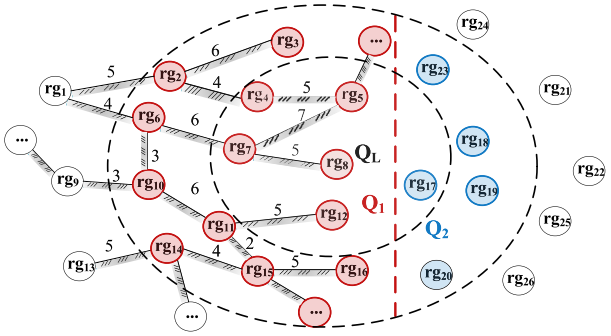

Fig. 10. Example of predictive KNN query.

Example 3: We present a 1-hop neighbor example to explain the KNN query process in Algorithm 4. Given a query location constraint Q_L, query keyword constraint ky_4, and a future time instant 10, as shown in Fig. 10, we present the corresponding KNN query process in Fig. 11. The whole query procedure needs to extend the radius twice for searching 100 nearest SODs to Q_L. In each expanded query range, one part of the query region, i.e., Q_1, falling into the frequent regions, we process it from SKIN-tree through modeling traveling time probability. The remaining part of the expanded region, i.e., Q_2, not in the frequent regions, we process it from STK-tree.

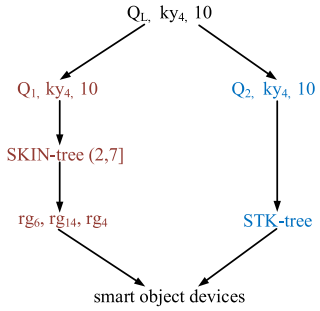

Fig. 11. KNN query process of the example in Fig. 10.

V. Implementation and Evaluation

In this section, we evaluate the performance of our searching scheme via the baseline methods and parameter effect. The goal of our experiment is mainly to evaluate query response time, and also to evaluate query message complexity. We mainly focus on SODs, edge devices, and cloud cache interaction because the SODs, edge node, the head nodes communicating between them, and the cloud cache are computationally challenged devices. This makes the performance of SODs, edge devices, head nodes, and cloud cache crucial for the validity of the searching system.

A. Setup

We use Beijing smart taxi cabs data set [20] to evaluate our proposed searching method. Beijing smart taxi cabs data set contains 2 325 708 check-in records at 920 023 activity regions from 207 232 smart taxi cabs. From this data set, we extract two groups of check-in records over 10 000 smart taxi cabs. The first group, denoted as gp_1, includes relatively frequent activity regions, thus the moving locations intensively scattered. The other group, denoted as gp_2, contains relatively less frequent activity regions, thus the moving locations widely dispersed. We describe an activity region road map according to the extracted check-in records and divide the corresponding spatial region into $N \times N$ squared grid cells. The grid width is set according to the average size of activity regions. We use the spatial filling curve to map the location of each grid cell into a Hilbert value and a list to store the smart taxi cabs frequently updated in it. Therefore, all recorded location-time information of a smart taxi cab formulates a grid series trajectory in terms of interest regions. Notice that the check-in records of each smart taxi cab contain its ID, grid coordination, interest region, as well as the check-in time. Then, we can obtain the ITI of each activity region by calculating the traveling time that is taken by smart taxi cabs moving among activity regions at a time interval t_l. Hence, we can efficiently make full the ITI before starting experiments offline. In addition, we randomly choose documents from a real data set such as China Daily [27] and assign them to each smart taxi cab as keywords. All the index structures and algorithms are implemented by Java. All experiments are conducted on a laptop with an Intel Core i5-5200U CPU 2.20GHZ, 8-GB memory, 64-bit operating system. The parameters used in the experiment are listed in Table II.

TABLE II
Parameters and Their Settings

Parameters	Setting
object devices number	1000, 3000, 5000, 7000, 9000
query number	10, 20, 30, 40, 50
k	100, 200, 300, 400, 500
keywords number	10
keywords length	20
update rounds	1, 3, 5, 10

For the sake of understanding the effectiveness of our collaborative edge-cloud-cache-based searching scheme, we give the following searching schemes to compare.

1) *Chained Structure (CST):* A set of object devices connected in the form of chained structure.
2) *Snoogle (SGL) [16]:* An information retrieval system constructed on sensor networks for the real world.
3) *IoT-SVK (IVK) [20]:* A real-time multimodal search engine mechanism implemented for the IoT.
4) *STK-Tree (STK) [21]:* This strategy directly searches the online spatial–time–keyword coded index of mobile object devices for query result achievement.
5) *SKIN-Tree+STK-Tree (SKIN+STK):* This strategy first searches the spatial–time–keyword filtering index of n-hop neighbor regions. If there is no matched neighbor regions, it searches STK-tree for query result achievement.

B. Experimental Evaluation

Comparison With Baselines: Table IV shows the experimental results on three query requests given in Table III. We see that our proposed SKIN-tree+STK-tree searching method outperforms the state-of-the-art searching methods in terms of query time and message number. The pruning and collaborative edge-cloud cache are dominant factors in the query process, and hence, have more impact on query performance. SKIN-tree+STK-tree has collaborative edge-cloud cache and update capability, as well as the most pruning judgment conditions and strongest pruning capability. The query result with overlap query constraints can be directly returned from n-hop neighbor regions. STK-tree also has the most pruning judgment conditions and strongest pruning capability, followed in descending order of *IoT-SVK*, *Snoogle*, and *Chained Structure*. However, all the four methods do not have collaborative edge-cloud cache and update capability. Without loss of generality, the less number of the pruning judgment conditions, a searching scheme sets up; the stronger the pruning hierarchy, a searching scheme constructs; and the better the query performance, the searching scheme can achieve. The collaborative edge-cloud cache mechanism also improves query performance for the overlap query constraints, which can be further illustrated by the following experiments on parameter effect. In general, SKIN-tree+STK-tree performs best while *Chained Structure* does worst. We also observe that the performance on the gp_1 data set is better than that on the gp_2 data set. This is because: 1) the number of frequent activity regions extracted from the gp_2 data set is less than that

TABLE III
QUERY SETTING

Query	Range Time Interval (Region Query)	Location Time Interval (KNN Query)	K	Update Round
Query 1	[5,15][5,25], [5,10][5,13], [8,19][8,17], [23,35][8,20], [20,34][27,45]	5 [10,16], 8 [20,25], 10 [25,29], 13 [15,20], 16 [19,30]	100	1
Query 2	[5,15][5,24], [5,8][6,13], [8,17][9,17], [21,35][8,20], [22,30][25,45], [25,40][20,33], [50,64][25,45], [55,67][55,67], [52,69][55,75], [75,85][65,95]	7 [5,10], 11 [15,18], 13 [11,16], 16 [15,17], 19 [14,18], 20 [23,27], 24 [27,30], 25 [28,29], 28 [30,34], 29 [31,35]	100	1
Query 3	[5,15][3,14], [5,8][5,23], [9,17][10,17], [23,35][8,20], [25,30][25,45], [25,40][20,33], [52,64][25,45], [45,67][52,63], [50,69][52,77], [70,85][55,85]	9 [5,11], 12 [23,28], 15 [25,30], 19 [23,28], 22 [12,29], 26 [14,23], 29 [17,24], 31 [25,32], 32 [32,38], 33 [36,40]	300	1

TABLE IV
PERFORMANCE COMPARISON FOR DIFFERENT METHODS

Group	Query	Performance	CST	SGL	IVK	STK	SKIN+STK
gp_1	Query 1	range query time (ms)	$4 * 10^{11}$	$1.5 * 10^{11}$	$1.1 * 10^{11}$	10^{10}	10^4
		range query message number	$4.1 * 10^5$	$3.6 * 10^5$	$1.2 * 10^5$	10^5	10^3
		KNN query time (ms)	$3.8 * 10^{10}$	$1.6 * 10^{10}$	$1.2 * 10^{10}$	10^{10}	10^7
		KNN query message number	$5.8 * 10^7$	$1.5 * 10^7$	$1.1 * 10^7$	10^7	10^3
	Query 2	range query time (ms)	$10 * 10^{11}$	$3 * 10^{11}$	$1.3 * 10^{11}$	10^{11}	10^5
		range query message number	$15 * 10^6$	$13 * 10^6$	$1.4 * 10^6$	10^6	10^4
		KNN query time (ms)	11^{12}	$1.9 * 10^{12}$	$1.6 * 10^{12}$	10^{12}	10^8
		KNN query message number	$10.2 * 10^8$	$2.1 * 10^8$	$1.3 * 10^8$	10^8	10^4
	Query 3	range query time (ms)	$12.52 * 10^{11}$	$2.28 * 10^{11}$	$1.32 * 10^{11}$	10^{11}	10^5
		range query message number	$15.11 * 10^6$	$13.1 * 10^6$	$1.39 * 10^6$	10^6	10^4
		KNN query time (ms)	11^{15}	$2.1 * 10^{15}$	$1.8 * 10^{15}$	10^{15}	10^{10}
		KNN query message number	$11 * 10^{10}$	$2.5 * 10^{10}$	$1.6 * 10^{10}$	10^{10}	10^6
gp_2	Query 1	range query time (ms)	$4.44 * 10^{10}$	$1.776 * 10^{10}$	$1.44 * 10^{10}$	$1.2 * 10^{10}$	$1.1 * 10^4$
		range query message number	$5.72 * 10^5$	$4.42 * 10^5$	$1.69 * 10^5$	$1.3 * 10^5$	$1.1 * 10^3$
		KNN query time (ms)	$72 * 10^{10}$	$30 * 10^{10}$	$28 * 10^{10}$	$20 * 10^{10}$	$1.0095 * 10^7$
		KNN query message number	$180.54 * 10^5$	$47.2 * 10^5$	$38.64 * 10^5$	$29.5 * 10^5$	$10.5 * 10^3$
	Query 2	range query time (ms)	$14 * 10^{11}$	$4.34 * 10^{11}$	$2.1 * 10^{11}$	$1.4 * 10^{11}$	$1.02 * 10^5$
		range query message number	$25.6 * 10^6$	$22.4 * 10^6$	$2.4 * 10^6$	$1.62 * 10^6$	$1.13 * 10^4$
		KNN query time (ms)	$2.04e + 14$	$1.38e + 14$	$1.32e + 14$	$60 * 10^{12}$	$1.0396 * 10^8$
		KNN query message number	$1047.15 * 10^8$	$232.7 * 10^8$	$152.15 * 10^8$	$89.5 * 10^8$	$3.95 * 10^4$
	Query 3	range query time (ms)	$16.624 * 10^{11}$	$3.76 * 10^{11}$	$2.608 * 10^{11}$	$1.6 * 10^{11}$	$1.02 * 10^5$
		range query message number	$24.528 * 10^6$	$22.576 * 10^6$	$2.592 * 10^6$	$1.61 * 10^6$	$1.12 * 10^4$
		KNN query time (ms)	$3.4947e + 15$	$2.2458e + 15$	$2.5416e + 15$	$1.059e + 15$	$1.1205 * 10^{10}$
		KNN query message number	$973.76 * 10^{10}$	$236.28 * 10^{10}$	$162.89 * 10^{10}$	$89.5 * 10^{10}$	$1.2296 * 10^6$

extracted from the gp_1 data set; 2) more moving locations are extracted from the gp_2 data set; and 3) the locations extracted from the gp_2 data set are more scattered than that extracted from the gp_1 data set.

Parameters Effect: In the following, we focus on evaluating the searching performance based on *collaborative edge-cloud cache* in terms of parameters effect in detail. Since *IoT-SVK*, *Snoogle*, and *Chained Structure* have weaker searching performance than STK-tree, and do not have collaborative edge-cloud cache and update capability, we intend to give comparison only with STK-tree.

Effect of Query Correlation: Considering parts of a query result can be directly achieved from the *n*-hop neighbor regions that satisfy the overlap constraint with the query q already answered (see Table V), we compare the query time based on SKIN-tree+STK-tree in descending of query correlation $cr_1 \succ cr_2 \succ cr_3 \succ cr_4$ through setting query number as 10, 20, and 30, respectively. From Fig. 12, we observe that the query time increases with the query number for each kind of query correlation, while the query time grows up in descending of query correlation. This indicates that query correlation has an important impact on the searching performance for SODs. Correlation improves the overlap possibility for query result, thus reducing the query processing time.

TABLE V
QUERY AND THEIR CONSTRAINTS

Query	Time	Range
q_1	$q_1 \bigcap q \neq 0$	$q_1 = q$
q_2	$q_2 = q$	$q_2 \bigcap q \neq 0$
q_3	$q_3 \bigcap q \neq 0$	$q_3 \bigcap q \neq 0$
q_4	$q_4 \bigcap q = 0$	$q_4 \bigcap q = 0$

Effect of Update Rounds: As update rounds can effectively improve query correlation, we compare the impact on query time based on SKIN-tree+STK-tree by setting update rounds as 1, 5, and 10, respectively, under a variant number of unrelated query number. In this set of experiments, the total query number is set as 20. The result in Fig. 13 shows that when the number of queries having a higher correlation is large,

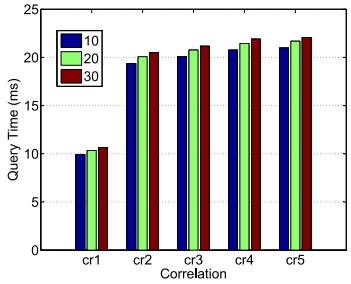

Fig. 12. Query time comparison with variant query correlation.

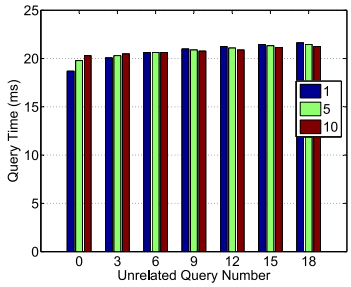

Fig. 13. Query time comparison with variant unrelated query number.

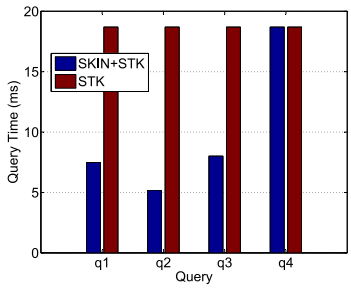

Fig. 14. Query time comparison with four query situations.

the query time in fewer update rounds is less. Following the increasing of unrelated query number, the query time in more update rounds is less since the unrelated query result may have overlap in a new round of update. The whole query time increases with unrelated query number since it only improves the overlap possibility in a certain extent other than completely.

Effect of Query Situations: In this set of experiments, given a query q and its query result, we study SKIN-tree to achieve the query result of another query for the four query situations listed in Table V. Fig. 14 shows the query time comparison for the four query situations. The query time of q_1, q_2, and q_3 based on SKIN-tree+STK-tree tends to be much less than that directly based on STK-tree, while the query time of q_4 based on SKIN-tree+STK-tree is almost the same to that directly based on STK-tree. This is because part of the query result can be directly achieved from the n-hop neighbor regions that satisfy the overlap query constraint with query q.

Effect of Query Number: In this set of experiments, we examine the effect of query number on two methods under variant update rounds. For readability, we plot the query response time in the log scale in Fig. 15. We observe that the query time and message number rise with the update

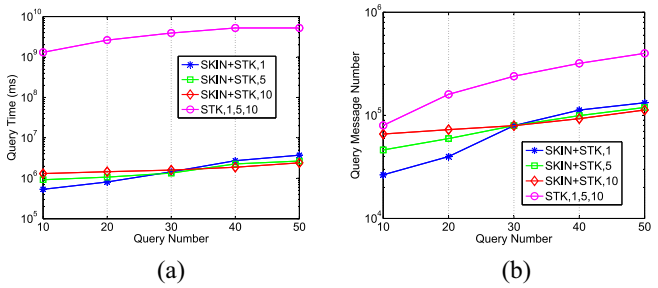

Fig. 15. Varying query number. (a) Query time. (b) Query message number.

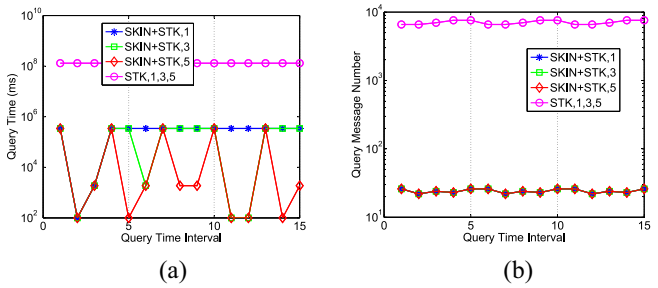

Fig. 16. Varying query time interval. We give a sequence of 15 query time intervals as [5, 15], [5, 8], [8, 17], [20, 35], [20, 30], [25, 40], [50, 60], [55, 65], [52, 68], [75, 95], [80, 85], [85, 95], [100, 115], [105, 110], and [100, 124]. (a) Query time. (b) Query message number.

rounds when the query number grows up within a small range for SKIN-tree+STK-tree-based approach. However, the query time and message number trend to reduce with the update rounds when the query number increases beyond a certain range. This is because more update rounds improve the query result overlap possibility, especially when more queries arrive. The SKIN-tree+STK-tree-based approach achieves a better query performance since the query result with overlap query constraints can be directly returned from n-hop neighbor regions. Compared to STK-tree, the query time and message number improve about 10^4 times and 10^1 times by the SKIN-tree+STK-tree-based approach.

Effect of Query Constraints: To further explore the effect of the collaborative edge-cloud-cache-based approach, we vary the query time interval, query range interval, and both the query time and query range interval to compare the query time and query message number. Figs. 16–18 show that the SKIN-tree+STK-tree-based approach still achieves better query performance compared to the STK-tree-based approach. For the first query in each round of update, it undergoes SKIN-tree to return query result. For the following queries, the actual query range shrinks because a part of query result can be directly answered from the n-hop neighbor regions with overlap query constraints as that of the query already answered. As expected, the query time of the SKIN-tree+STK-tree-based approach in more times of update is less than that in fewer times of update. However, the query time of the STK-tree-based approach keeps stable all the time, at about 10^8 ms. With regard to the query message number, the SKIN-tree+STK-tree-based approach explores the approximately same variation trend with different query constraints because the communication mainly results from edge device to cloud cache.

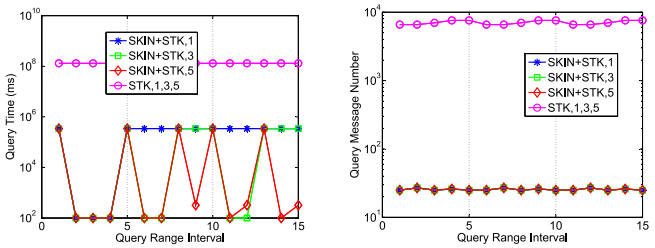

Fig. 17. Varying query range interval. We give a sequence of 15 query range intervals as [5, 20], [5, 13], [8, 17], [8, 20], [25, 45], [20, 30], [25, 40], [55, 60], [55, 65], [65, 75], [68, 71], [67, 78], [85, 95], [88, 90], and [90, 100]. (a) Query time. (b) Query message number.

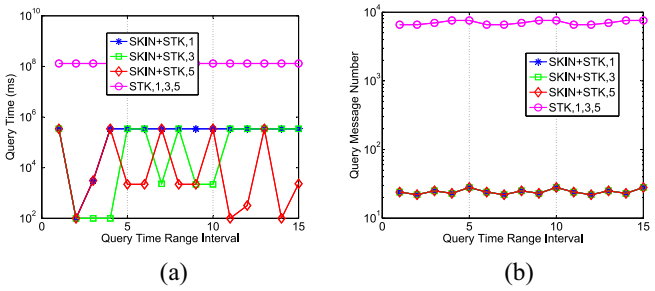

Fig. 18. Varying query time range interval. We give a sequence of 15 query time range intervals as [5, 15] [5, 20], [5, 8] [5, 13], [8, 17] [8, 17], [20, 35] [8, 20], [20, 30] [25, 45], [25, 40] [20, 30], [50, 60] [25, 40], [55, 65] [55, 60], [52, 68] [55, 65], [75, 95] [65, 75], [80, 85] [68, 71], [85, 95] [67, 78], [100, 115] [85, 95], [105, 110] [88, 90], and [100, 124] [90, 100]. (a) Query time. (b) Query message number.

The STK-tree-based approach also presents the same query performance trend because it requires to search STK-tree for each variant query. Note that the query message number is about 10^3 times less than that of the STK-tree-based approach. The whole improvements over the SKIN-tree+STK-tree-based approach are quite significant.

Effect of Object Device Number: Figs. 19 and 20 plot the spatial–time range query performance by varying the object device number. In this set of experiments, we set the query number as 50. No matter the query is fixed or nonfixed, we see that the query time for the SKIN-tree+STK-tree-based approach in each round of update is higher, at about 10^7 ms, when the number of object devices is 1000. Similar to the previous experimental result, we see that more update rounds result in less query time. As the number of object devices rises with time, the query time sustains at about 10^4 ms as the n-hop neighbor regions are stable with the same query constraint based on SKIN-tree+STK-tree. However, the query time of the STK-tree-based approach increases with the object device number, since it requires to sweep more object devices for achieving the final query result. In terms of the query message number, it increases to about 10^5 with the object device number for the STK-tree-based approach due to more branch nodes involved in the search process. Instead, the query message number keeps approximately stable at about 10^3 for the SKIN-tree+STK-tree-based approach since SKIN-tree remains unchanged with the increase of object device number.

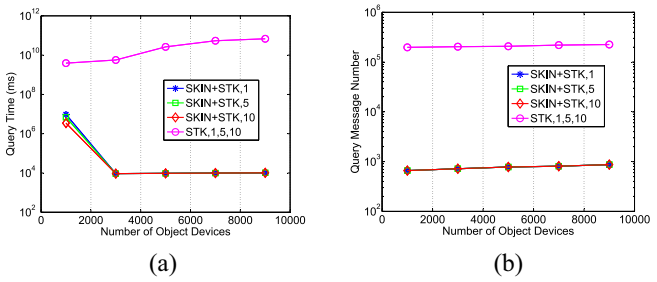

Fig. 19. Varying object device number with a fixed spatial–time range query. (a) Query time. (b) Query message number.

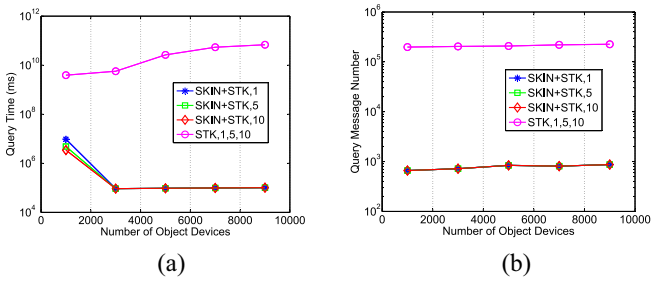

Fig. 20. Varying object device number with a nonfixed spatial–time range query. (a) Query time. (b) Query message number.

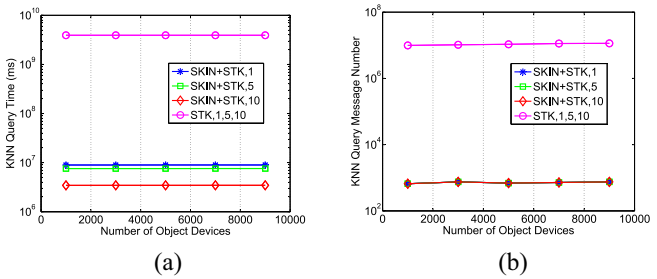

Fig. 21. Varying object device number with $k = 100$. (a) KNN query time. (b) KNN query message number.

Fig. 21 shows the query performance by varying the object device number with $k = 100$. The SKIN-tree+STK-tree-based approach achieves a significant 10^3 times and 10^4 times reduction in terms of query time and message number compared with the STK-tree-based approach. Such improvement results from the stable n-hop neighbor regions and the only communication from edge device to cloud cache. Similar to the previous experimental result, we also see that more update rounds result in less query time.

Fig. 22 illustrates the query performance changing over the object device number. When a frequent query arrives, the query result can be achieved directly from the cloud cache. Hence, the query message number mainly comes from the communication from the edge device to cloud cache. We see it drops from 10^7 for the STK-tree-based approach to 10^3 for the SKIN-tree+STK-tree-based approach. It is obvious that the cloud cache provides whole performance improvement. When an infrequent query arrives, both the SKIN-tree+STK-tree-based approach and the STK-tree-based approach take 329 450 to 366 650 message numbers to conduct the same operation with the increase of object device number. The results are

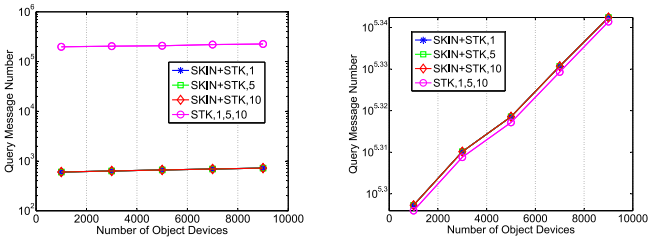

Fig. 22. Varying object device number with (a) frequent query and (b) infrequent query.

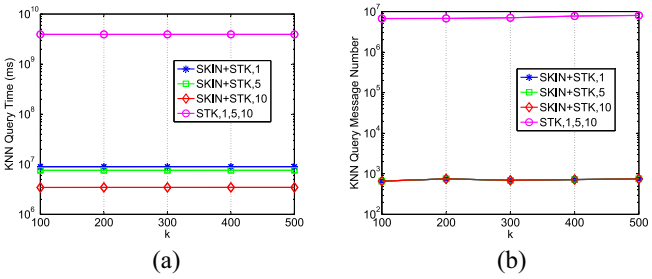

Fig. 23. Varying k. (a) KNN query time. (b) KNN query message number.

expected since the cloud-cache-based approach has to search STK-tree when there are no neighbor regions returned.

Effect of k: Next, we consider the effect of varying k. In this set of experiments, we also set the query number as 50 for each KNN query. When setting k from 100 to 500, the SKIN-tree+STK-tree-based approach takes about 10^7 ms to generate the KNN query result, while the STK-tree-based approach takes about $10^{9.5}$ ms to conduct the same operation (see Fig. 23). In addition, we notice that the SKIN-tree+STK-tree-based approach can generate 10^4 less message number since the computation can be shifted to the cloud cache system without loss of query result data. Similar to the previous experimental result, we also see that more update rounds result in less query time.

The whole experimental results confirm the benefits of the SKIN-tree+STK-tree-based approach, which achieves better query performance than directly searching STK-tree, especially when more updates and more queries arrive. This attributes to collaborative edge-cloud cache mechanism that possesses: 1) more computational, communication, and energy resources, as well as low latency access capability on edge device and 2) high performance distributed computing, storage, cache, and update capacity on cloud.

VI. Related Work and Comparison

In this section, we present an overview of previous efforts on the two most relevant works: 1) IOT search engine and 2) spatial–temporal–keyword query.

IOT Search Engine: Some of the notable works in this area are given as follows. Wang *et al.* [16] proposed Snoogle, a two-tier hierarchical search engine in pervasive environments. The entity is stored in the sensor node in the form of a set of keywords. When a query arrives, the users exploit the keywords to query the entities that are matched. However, in the face of large-scale dynamic network environment, the Snoogle is not suitable due to weak data transmission mode. Yap *et al.* [17] designed MAX system. The physical entities are sensed by tag, which also stores their textual information. MAX is fitting for the mobile queries with frequently changed content due to pull mode adopted. However, it is ineffective for large-scale network environments as the messages broadcast results in high communication overhead. Ostermaier *et al.* [19] designed Dyser, a real-time search engine that aims to search physical entities having a certain state at query time. The real entities and sensors are represented by Web pages. Dyser is suitable to be applied in resource-constrained networking environment because a predictive mechanism used can improve the searching efficiency, and reduce the searching overhead. Further, considering the spatial–temporal, value, and keyword-based constraints, Ding *et al.* [20] proposed IoT-SVK, which a hybrid real-time search engine constructed in the IOT. IoT-SVK uses the same grid region to describe the original path. However, the search engine cannot construct an integrated spatial–temporal–keyword index, which reduces the searching efficiency in a certain extent. To resolve this problem, SMSTK search engine [21] proposes a coding-enabled index technology to seamlessly integrate spatial–temporal–keyword proximity. The key value in each node provides search and pruning foundation. However, it does not consider the query frequency and information overlap for improving searching efficiency.

In a whole, most of the existing proposals cannot directly enable effective searching of mobile SODs in the IoT due to disperse search strategy or complicated query condition. In this article, we propose the collaborative edge-cloud-cache-based searching scheme, in which two kinds of spatial indexes that are used by distinguishing activity region frequency can efficiently help to improve searching efficiency and reduce searching latency.

Spatial–Temporal–Keyword Query: Most of the works about spatial–temporal–keyword query use separate or combined data structures to treat one or several aspects of space, time, and text issues. Li *et al.* [28] proposed IR-tree, which supports top-k document retrieval via integrating spatial and textual filtering as well as relevance and document ranking computation. It is also able to search documents with different weights assigned to textual and spatial aspects. However, the time factor is not included. Khodaei *et al.* [29] further considered the time factor. They devise a hybrid index via combining both time and textual factors in a unified inverted list manner to process time–textual query. However, it does not study the time–textual query along with spatial factor. Nepomnyachiy *et al.* [30] further considered the time and spatial–keyword aspects. They first propose to shrink searching space in time dimension based on dynamical time instant data. As for spatial and textual dimensions, a shallow tree is adopted to further shrink searching space. The time and spatial–keyword aspects are processed via a large time domain with some local spatial–textual domains. The three factors are not considered simultaneously. Mehta *et al.* [31] considered resolving spatial–temporal and spatial–keyword

queries through two hybrid indexes. For the remaining keyword information and temporal dimension, two hybrid indexes further incorporate them to realize spatial–temporal–keyword query. Unfortunately, the three factors are still required to be treated into two structures. Hoang-Vu *et al.* [32] further proposed a single-index structure that integrates keywords, time, and space features fully for efficiently answering spatial–temporal–keyword queries. However, mapping of keywords numbers is strictly monotone for constructing spatial index structure.

In general, most of the existing index technologies adopt *combined structures* to search the documents satisfying user's request. Instead, we adopt two kinds of *single-index structure*, which explore the activity region frequency and ITI in SKIN-tree, and the concatenated spatial–temporal–textual key value in STK-tree. Therefore, our collaborative edge-cloud-cache-based searching scheme can well improve the retrieval performance compared with the state of the art.

VII. Conclusion

This article proposed a novel collaborative edge-cloud-cache-based search engine over mobile object devices. The focus is to construct a spatial–time–keyword filtering index of n-hop neighbor activity regions by incorporating the keywords relevance and uncertain traveling time into the search mechanism. Further, we use an online spatial–time–keyword coded index of mobile object devices to process the infrequent query. To evaluate the performance, we conduct a series of experiments and results demonstrate our proposed method has a better searching efficiency than the state-of-the-art's techniques.

References

[1] S. Li, L. D. Xu, and S. Zhao, "The Internet of Things: A survey," *Inf. Syst. Front.*, vol. 17, no. 2, pp. 243–259, 2015.

[2] J. Pan, R. Jain, S. Paul, T. Vu, A. Saifullah, and M. Sha, "An Internet of Things framework for smart energy in buildings: Designs, prototype, and experiments," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 527–537, Dec. 2015.

[3] E. Park, Y. Cho, J. Han, and S. J. Kwon, "Comprehensive approaches to user acceptance of Internet of Things in a smart home environment," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2342–2350, Dec. 2017.

[4] F. Zhang, M. Liu, Z. Zhou, and W. Shen, "An IoT-based online monitoring system for continuous steel casting," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1355–1363, Dec. 2016.

[5] D. C. M. Segura, R. D. S. Stabile, S. M. Bruschi, and P. S. L. D. Souza, "Providing computing services through mobile devices in a collaborative way—A fog computing case study," in *ACM Int. Conf. Model.*, Miami, FL, USA, 2017, pp. 117–121.

[6] B. Alturki, S. Reiff-Marganiec, and C. Perera, "A hybrid approach for data analytics for Internet of Things," in *Proc. Int. Conf. Internet Things*, Linz, Austria, 2017, pp. 1–8.

[7] S. Fong, "Big data mining algorithms for fog computing," in *Proc. Int. Conf. Internet Things*, London, U.K., 2017, pp. 57–61.

[8] P. Zhang, Y. Liu, F. Wu, S. Liu, and B. Tang, "Low-overhead and high-precision prediction model for content-based sensor search in the Internet of Things," *IEEE Commun. Lett.*, vol. 20, no. 4, pp. 720–723, Apr. 2016.

[9] A. Shemshadi, Q. Z. Sheng, and Y. Qin, "ThingSeek: A crawler and search engine for the Internet of Things," in *Proc. 39th Int. ACM SIGIR conf. Res. Develop. Inf. Retrieval*, Pisa, Italy, 2016, pp. 1149–1152.

[10] Y. Hua, Y. Hua, and K. Kyriakopoulos, "Semi-edge: From edge caching to hierarchical caching in network fog," in *Proc. 1st Int. Workshop Edge Syst., Anal. Netw.*, 2018, pp. 43–48.

[11] C. C. Tan, B. Sheng, H. Wang, and Q. Li, "Microsearch: A search engine for embedded devices used in pervasive computing," *ACM Trans. Embedded Comput. Syst.*, vol. 9, no. 4, pp. 1–43, 2010.

[12] M. Shah and A. Sardana, "Searching in Internet of Things using VCS," in *Proc. Int. Conf. Security Internet Things*, Kollam, India, 2012, pp. 63–67.

[13] H. Ma and W. Liu, "A Progressive search paradigm for the Internet of Things," *IEEE Multimedia*, vol. 25, no. 1, pp. 76–86, Jan.–Mar. 2018.

[14] Y. Zhou, S. De, W. Wang, and K. Moessner, "Search techniques for the Web of Things: A taxonomy and survey," *Sensors*, vol. 16, no. 5, pp. 1–29, 2016.

[15] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *Proc. Int. Conf. Mobile Data Manag.*, Mannheim, Germany, 2007, pp. 198–205.

[16] H. Wang, C. C. Tan, and Q. Li, "Snoogle: A search engine for pervasive environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 8, pp. 1188–1202, Aug. 2010.

[17] K.-K. Yap, V. Srinivasan, and M. Motani, "MAX: human-centric search of the physical world," in *Proc. 3rd Int. Conf. Embedded Netw. Sensor Syst.*, San Diego, CA, USA, 2005, pp. 166–179.

[18] W. I. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE Multimedia*, vol. 14, no. 4, pp. 8–13, Oct.–Dec. 2007.

[19] B. Ostermaier, K. Römer, F. Mattern, M. Fahrmair, and W. Kellerer, "A real-time search engine for the Web of Things," in *Proc. Internet Things (IOT)*, Tokyo, Japan, 2010, pp. 1–8.

[20] Z. Ding, Z. Chen, and Q. Yang, "IoT-SVKSearch: A real-time multimodal search engine mechanism for the Internet of Things," *Int. J. Commun. Syst.*, vol. 27, no. 6, pp. 871–897, 2014.

[21] J. Tang and Z. Zhou, "Searching the Internet of Things using coding enabled index technology," in *Proc. Int. Conf. Green Pervasive Cloud Comput.*, 2018, pp. 79–91.

[22] C. S. Jensen, D. Lin, and B. C. Ooi, "Query and update efficient B^{+}-tree based indexing of moving objects," in *Proc. 30th Int. Conf. Very Large Data Bases*, Toronto, ON, Canada, 2004, pp. 768–779.

[23] R. Uddin, C. V. Ravishankar, and V. J. Tsotras, "Indexing moving object trajectories with Hilbert curves," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Seattle, WA, USA, 2018, pp. 416–419.

[24] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Boston, MA, USA, 1984, pp. 47–57.

[25] H.-P. Hsieh, C.-T. Li, and S.-D. Lin, "Measuring and recommending time-sensitive routes from location-based data," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 3, pp. 1–27, 2014.

[26] C. Zhang, H. Liang, K. Wang, and J. Sun, "Personalized trip recommendation with POI availability and uncertain traveling time," in *Proc. CIKM*, Melbourne VIC, Australia, 2015, pp. 911–920.

[27] [Online]. Available: http://www.chinadaily.com.cn/

[28] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 585–599, Apr. 2011.

[29] A. Khodaei, C. Shahabi, and A. Khodaei, "Temporal-textual retrieval: Time and keyword search in Web documents," *Int. J. Next Gener. Comput.*, vol. 3, no. 3, pp. 288–312, 2012.

[30] S. Nepomnyachiy, B. Gelley, W. Jiang, and T. Minkus, "What, where, and when: Keyword search with spatio-temporal ranges," in *Proc. Workshop Geograph. Inf. Retrieval*, Dallas, TX, USA, 2014, pp.1–8.

[31] P. Mehta, D. Skoutas, and A. Voisard, "Spatio-temporal keyword queries for moving objects," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Seattle, WA, USA, 2015, pp. 1–4.

[32] T.-A. Hoang-Vu, H. T. Vo, and J. Freire, "A unified index for spatio-temporal keyword queries," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manag.*, Indianapolis, IN, USA, 2016, pp. 135–144.

Jine Tang received the Ph.D. degree from the China University of Geosciences Beijing, Beijing, China, in 2014.

She is an Associate Professor with the School of Artificial Intelligence, Hebei University of Technology, Tianjin, China. Her current research interests include process-aware information system, spatial–temporal database, sensor network middleware, and data security.

Zhangbing Zhou received the Ph.D. degree in computer science from Digital Enterprise Research Institute, Galway, Ireland, in 2010.

He was a Software Engineer with Huawei Technologies Company Ltd., Beijing, China, for one year. He served as a Member of the Technical Staff with Bell Laboratories, Lucent Technologies, Beijing, for five years. He is currently a Professor with the China University of Geosciences Beijing, Beijing, and as an Adjunct Professor with TELECOM SudParis, Évry, France. He has authored over 100 referred papers. His current research interests include process-aware information system and sensor network middleware.

Prof. Zhou has served as an Associate or a Guest Editor for over 10 journals.

Xiao Xue was born in 1979. He received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2007.

He is a Professor with the School of Computer Software, College of Intelligence and Computing, Tianjin University, Tianjin, China, and also an Adjunct Professor with the School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, China. His current research interests include service computing, computational experiment, and Internet of Things.

Gongwen Wang received the Ph.D. degree from the China University of Geosciences Beijing, Beijing, China, in 2006.

He is a Professor with the School of Earth Sciences and Resources, China University of Geosciences Beijing. His current research interests include geosciences big data mining, machine learning, deep learning, artificial intelligence, digital mine, intelligent mine, and virtual reality.