Finding Top-k Local Users in Geo-Tagged Social Media Data

Jinling Jiang† Hua Lu† Bin Yang† Bin Cui§

†Department of Computer Science, Aalborg University, Denmark
§ Key Lab of High Confidence Software Technologies (MOE), School of EECS, Peking University, China
{jinling, luhua, byang}@cs.aau.dk, bin.cui@pku.edu.cn

***Abstract*—Social network platforms and location-based services are increasingly popular in people's daily lives. The combination of them results in location-based social media where people are connected not only through the friendship in the social network but also by their geographical locations in reality. This duality makes it possible to query and make use of social media data in novel ways. In this work, we formulate a novel and useful problem called top-k local user search (TkLUS for short) from tweets with geo-tags. Given a location q, a distance r, and a set of keywords W, the TkLUS query finds the top-k users who have posted tweets relevant to the desired keywords in W at a place within the distance r from q. TkLUS queries are useful in many application scenarios such as friend recommendation, spatial decision, etc. We design a set of techniques to answer such queries efficiently. First, we propose two local user ranking methods that integrate text relevance and location proximity in a TkLUS query. Second, we construct a hybrid index under a scalable framework, which is aware of keywords as well as locations, to organize high volume geo-tagged tweets. Furthermore, we devise two algorithms for processing TkLUS queries. Finally, we conduct an experimental study using real tweet data sets to evaluate the proposed techniques. The experimental results demonstrate the efficiency, effectiveness and scalability of our proposals.**

I. Introduction

Social media like Twitter and Facebook generates huge amounts of data. For example, Twitter users sent 4,064 tweets in a single second when 2011's Super Bowl came to its final moments[1], making the highest tweets per second ever recorded for a sport event. In general, people use social network platforms for keeping in touch with friends and seeking various information that have social aspects.

Recently, the widespread of GPS-enabled mobile terminals and the success of mobile location-based services (LBS) make social media data acquire geo-location information. Geo-tagged microblogs (e.g., tweets with geo-locations in metadata) play an important role in sharing observations and opinions, getting news, and understanding the situations about real-world events. For example, local groups were quickly formed in Facebook in response to the earthquake in Haiti in 2010. As a result, so-called location-based social networks are becoming abundant sources of geo-related information.

Traditional information retrieval (IR) techniques behind search engines emphasize finding relevant information from long textual documents rich in keywords. They are not suitable for searching short-sized social media data that are characterized by few keywords. Twitter itself also offers a real-time search service[2], which returns highly-ranked tweets in response to user-input keywords. However, the spatial aspect is not handled in the search service.

With geo-tagged social media data, it is possible to offer novel services other than the traditional search services. For example, a couple with kids moving to Seoul may ask "Are there any good babysitters in Seoul?" This query contains keyword "babysitter" and spatial location "Seoul". In this paper, we tackle such location-dependent and contextualized social search that is among the top popular categories according to the social search engine Aardvark [10].

A straightforward approach for such social search is to directly retrieve tweets based on query keywords. However, this approach can return too many original tweets to a query issuer who may get lost and confused about what action to take. Instead, the information needed by such a location-dependent query lies in finding local social media users who are familiar with the relevant issues in a certain spatial region. Therefore, recommending local Twitter users for specific queries can be very useful. People in need can directly communicate with those recommended local users on Twitter platform. Also, Twitter maintains the social relationships among users, which can be exploited to score the users for the purpose of recommending local users.

Our approach in this paper is to recommend Twitter users instead of tweets, by taking into account users' tweet contents, locations, and tweets' popularity that indicates the users' social influence. In particular, we study the search of top-k local users in tweets that are associated with latitude/longitude information, namely geo-tagged tweets. To that end, we devise a scalable architecture for hybrid index construction in order to handle high-volume geo-tagged tweets, and we propose functions to score tweets as well as methods to rank users that integrate social relationships, keyword relevance and spatial proximity for processing such social search.

Note that existing works on microblog indexing and search focus on a real-time architecture as Twitter produces a large volume of data each day. In contrast, geo-tagged tweets occupy less than 1 percent of the whole tweets data set[3]. Therefore, we find it reasonable to separately process the geo-tagged tweets in a batch mode. This allows us to collect the geo-tagged tweets data in a periodic manner and to process them offline. In such a setting, we distinguish our research from the existing real-time indexing and search services in Twitter.

[1] http://blog.twitter.com/2011/02/superbowl.html

[2] https://twitter.com/search-home

[3] http://thenextweb.com/2010/01/15/twitter-geofail-023-tweets-geotagged/

We make the following contributions in this paper:

- We formulate *top-k local user search* (TkLUS) query to find local social media users who can provide relevant information to search needs (Section II).
- We propose *tweet thread* to capture the query-dependent popularity of tweets, and devise two local user scoring functions that integrate social relationships, keyword relevance and spatial proximity in ranking local users for answering a TkLUS query (Section III).
- We design a hybrid index, which is aware of keywords as well as locations, for organizing high volumes of geo-tagged tweets (Section IV).
- We devise an effective upper bound score and two algorithms for processing TkLUS queries according to the proposed user scoring functions (Section V).
- We conduct extensive experiments on real Twitter data sets to evaluate our proposals (Section VI).

In addition, Section VII reviews the related work; Section VIII concludes the paper with future work directions.

II. Problem Formulation

In this section, we formulate the search problem of TkLUS.

A. Modeling Social Media Data

In the context of social media, we identify three different but interrelated concepts. In particular, social media consists of many *posts* that are posted by *users* who form a *social network* through their interactions in the posts.

Definition 1: (**Social Media Post**) A social media post is a 4-tuple $p = (uid, t, l, W)$, where uid identifies the user who publishes the post, t is the timestamp when the post is published, l is the location where the user publishes the post, and W is a set of words $(w_1, w_2, \ldots, w_n)$ that capture the textual content of the post.

The location field may be unavailable for many social media posts, depending on whether localization is supported (enabled) by a social media platform (a user). For example, Twitter users can choose to record their locations or not in tweets if their devices are GPS-enabled. In this paper, we focus on social media posts that have non-empty location fields, and make use of them to find relevant results for user queries.

The size of $p.W$ is always small since social media platforms impose length constraints on posts. For example, a tweet contains at most 140 characters that can only convey a small number of words. On the other hand, not all textual words in an original social media post are included in the abstraction p. We assume the use of a vocabulary $\mathcal{W}$ that excludes popular stop words (e.g., this and that).

Definition 2: (**Social Network**) A social network referred in social media platform is a directed graph $G = (U, E_{reply}, l_{reply}, E_{forward}, l_{forward})$ where:

1) U is a set of vertices, each corresponding to a user.
2) E_{reply} is a set of directed edges that capture the "reply" relationships between users in U. Particularly, an edge $e = \langle u_1, u_2 \rangle \in E_{reply}$ means that user u_1 *replies to* user u_2 in one or more posts.
3) l_{reply} maps a reply edge to the set of involved posts. Given two users u_1 and u_2, $l_{reply}(u_1, u_2)$ returns all the posts in which u_1 replies to u_2.
4) $E_{forward}$ is a set of directed edges that capture the "forward" relationships between users in U. Particularly, an edge $e = \langle u_1, u_2 \rangle \in E_{forward}$ means that user u_1 *forwards* user u_2's post(s).
5) $l_{forward}$ maps a forward edge to the set of involved posts. Given two users u_1 and u_2, $l_{forward}(u_1, u_2)$ returns all u_2's posts forwarded by u_1.

The social network in a social media platform is complex as it involves both content-independent and content-dependent aspects. The definition above captures both aspects in different types of graph edges and auxiliary mappings. In particular, each reply edge in E_{reply} must involve at least one post where one user replies to another, whereas each edge in $E_{forward}$ must involve at least one post which one user forwards.

We use $P = \{(uid, t, l, W)\}$ to denote the set of all social media posts of interest, and $\mathcal{D} = (P, U, G)$ to denote the geo-tagged social media data. For convenience, we also use P_u to denote all the posts by a user $u \in U$.

B. Problem Definition

In the context of geo-tagged social media data, people may pose questions like *What are the events going on in New York City?* and *How can I find a babysitter in LA?* Among all social media users in U, there can be some local users, who have relevant knowledge indicated in their posts, to answer such questions. Without loss of generality, we formalize the questions as the following problem.

Problem Definition: **(Top-k Local User Search)** Given geo-tagged social media data $\mathcal{D} = (P, U, G)$, a location l, a distance r, and a keyword set W, a top-k local user search $q(l, r, W)$ finds a k-user set $\mathcal{E}_k \subseteq \mathcal{D}.U$ such that:

1) $\forall u \in \mathcal{E}_k$, $\exists p \in P_u$ such that $\|q.l, p.l\| \leq q.r$ [4] and $p.W \cap q.W \neq \emptyset$.
2) $\forall u \in \mathcal{E}_k$ and $\forall u' \in \mathcal{D}.U \setminus \mathcal{E}_k$, either u' does not satisfy condition 1 or $score(u', q) \leq score(u, q)$.

Here, $score(.)$ is a score function to measure the relevance of a user to a search request. Our score functions, to be detailed in Section III, consider both keyword and distance relevances.

We give a simplified example of top-1 local user search. Figure 1 shows a map where a TkLUS query is issued at the crossed location (43.6839128037, -79.37356590), with a single keyword "hotel" and a distance of 10 km. The tweets containing "hotel", as well as their users, are listed in Table I. The tweets' locations are also indicated in the map. We may use different scoring functions to rank users. If we favor users with more tweets, user u_1 with two tweets A and G will be returned. If we favor users having more replies/forwards (not shown in the table), user u_5 will be returned.

To find the top-k users for a TkLUS query is a non-trivial problem. The user set U and the post set P are very large on many social media platforms like Twitter. It is

[4] $\|l_1, l_2\|$ denotes the Euclidean distance between locations l_1 and l_2. The techniques proposed in this paper can be adapted to other distance metrics.

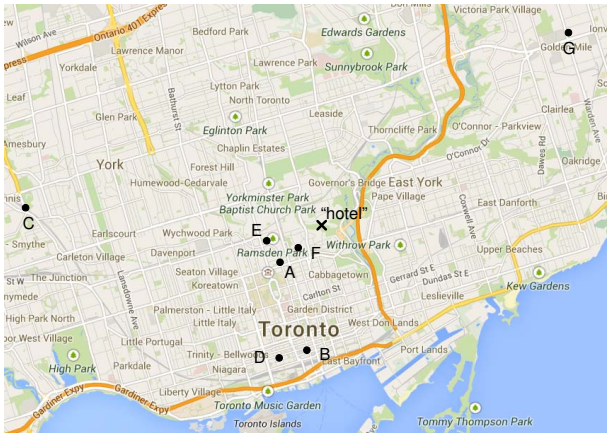


Fig. 1. TkLUS Example: Locations of Query and Tweets

definitely inefficient to check the sets iteratively. Also, it is not straightforward to measure the relevance of a user to a given query. Furthermore, a TkLUS query can have many relevant tweets and local users, and therefore we need to rank them and return the most relevant users as the query result. In this paper, we systematically address such technical challenges.

TABLE I. DETAILED INFORMATION OF EXAMPLE TWEETS

pid	uid	text
A	u_1	I'm at Toronto Marriott Bloor Yorkville Hotel
B	u_2	Finally Toronto (at Clarion Hotel).
C	u_3	I'm at Four Seasons Hotel Toronto.
D	u_4	Veal, lemon ricotta gnocchi @ Four Seasons Hotel Toronto.
E	u_5	And that was the best massage I've ever had.(@ The Spa at Four Seasons Hotel Toronto)
F	u_6	Saturday night steez #fashion #style #ootd #toronto #saturday #party #outfit @ Four Seasons Hotel Toronto.
G	u_1	Marriott Bloor Yorkville Hotel is a perfect place to stay.

III. SCORING TWEETS AND USERS

In this section, we propose a way to measure tweet popularity, followed by the scoring functions for tweets and users.

A. Tweet Thread and Tweet Popularity

A local user that offers useful information is likely to have tweets popular in the social media. Therefore, we first consider a tweet p's popularity with respect to a query $q(l, r, W)$. Intuitively, p is not interesting if it has no keywords from $q.W$. We only consider tweets with query keyword(s) and give priority to those with more of them. On the other hand, a tweet tends to be popular if it gives rise to visible responses in the social media. For example, if a user has useful information, her/his tweets tend to be popular with many replies and/or forwards. In this sense, we also quantify tweet responses and give priority to those having more responses. To support these considerations, we introduce the concept *tweet thread*.

Definition 3: (**Tweet Thread**) Given a query $q(l, r, W)$, a tweet thread T is a tree of tweets where

1) each node corresponds to a unique tweet;
2) the root $T.root$ is a tweet p that has keywords in $q.W$;
3) if tweet p_i is tweet p_j's parent node, p_j replies to or forwards p_i.

Figure 2 shows an example. Tweet p_1 has keywords requested in query $q(l, r, W)$. Tweets p_2, p_3, and p_4 reply to or forward p_1, each having further reply or forward tweets.

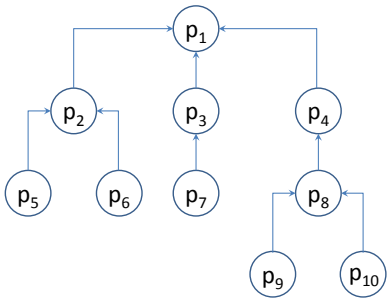


Fig. 2. Tweet Thread Example

Intuitively, the more tweets in a tweet thread, the more popular the root tweet is. If one tweet shares no connection with other tweets, it will form a tweet thread by itself only. Apparently, such a tweet is not popular and should be given a low score in our search. Furthermore, a tweet thread with a larger number of tweets is more important than those with fewer as this means its root tweet leads a conversation involving a lot of concerns. As a result, the user who posts such a root tweet is regarded as a relevant local user for answering the given query. The algorithm of constructing tweet thread and computing the thread score will be described in Section IV-A.

Based on the tweet thread concept, we define a tweet p's popularity as the score of tweet thread T whose root is p.

Definition 4: **Popularity of Tweet**

$$\phi(p) = \begin{cases} \epsilon, & \text{if } T.h = 1; \\ \sum_{i=2}^{n} |T_i| \times \frac{1}{i}, & \text{otherwise.} \end{cases}$$

In Definition 4, $T.h$ means the height of the tweet thread and $|T_i|$ represents the number of tweets in the i-th level. In particular, i starts from 2 at the second to top level in the tweet thread until n the deepest level of the tweet thread. In addition, ϵ is a smoothing parameter set for tweet thread consisting only one tweet. For example, in Figure 2, the score of tweet p_1 is $3 \times \frac{1}{2} + 4 \times \frac{1}{3} + 2 \times \frac{1}{4} = \frac{10}{3}$.

Our definition of tweet popularity takes into account the cascading effects of replies and forwards that are seen in social media. Such effects are more visible for users with many (layers of) followers.

B. Individual Tweet's Score

To define a single tweet p's score with respect to a given query $q(l, r, W)$, we need to take into account the distance between p and the query, as well as the p's popularity with respect to q. For the former, we define the distance score $\delta(p, q)$ of a tweet as follows.

Definition 5: **Distance Score of Tweet**

$$\delta(p,q) = \begin{cases} \dfrac{r - \|q.l, p.l\|}{r}, & \text{if } \|q.l, p.l\| \leq r; \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\delta(p,q)$ returns 0 if a tweet p is outside of the distance r from q. Within the distance r, a tweet gets a higher score if it is closer to q. Overall, the range of $\delta(p,q)$ is $[0,1]$.

Next, we define the keyword relevance between a tweet p with a given query $q(l,r,W)$ as follows.

Definition 6: **Keyword Relevance Score of Tweet**

$$\rho(p,q) = \frac{|q.W \cap p.W|}{N} \cdot \phi(p)$$

This definition counts the normalized occurrences of a query keyword in tweet p, multiplies it by p's popularity $\phi(p)$, and uses the product as p's keyword relevance. Here, N is a normalized parameter and we do not necessarily further normalize $\phi(p)$ since $\rho(p,q)$ is allowed to exceed 1. In our experimentation, N is empirically set around 40 such that keyword relevance score is comparable to the distance score. Also, the occurrences are actually counted according to a bag model of keywords. Precisely, $q.W$ is a set whereas $p.W$ is a bag/multiset. For example, if a query is "spicy restaurant" and tweet p contains one "spicy" and two "restaurant", the occurrences of a query keyword in tweet p is 3.

C. Ranking Users for Query

With the tweet score defined above, we are able to rank users who post tweets. We first consider the keyword aspect of a user with respect to a given query. We propose two ranking functions to compute the keyword relevance scores with different emphases.

Given a user u, we can look at all the tweets that u has posted and sum up their keyword relevance scores as user u's keyword relevance score. The intuition behind this is that all tweets of a user may matter if they contain keywords in a query. We call this scoring function *Sum Score*.

Definition 7: **(Sum Score)** Given a tweet user u and a query $q(l,r,W)$, u's keyword relevance score is

$$\rho_s(u,q) = \Sigma_{p \in P_u} \rho(p,q)$$

Alternatively, a user u's keyword relevance score can be calculated as the maximum keyword relevance score of all his/her tweets. The intuition behind this function is that if one tweet thread gets a very high score, the user of the root tweet in the thread will probably be the local user whom we are interested in. We call this scoring function *Maximum Score*.

Definition 8: **(Maximum Score)** Given a tweet user u and a query $q(l,r,W)$, u's maximum keyword relevance score is

$$\rho_m(u,q) = \max_{p \in P_u} \rho(p,q)$$

On the other hand, the distance score of user u is defined as the average distance of all u's posts and the query location.

Definition 9: **(Distance Score of User)** Given a tweet user u and a query $q(l,r,W)$, u's distance score is

$$\delta(u,q) = \frac{\Sigma_{p \in P_u} \delta(p,q)}{|P_u|}$$

By combining the two aspects, we define the user score.

Definition 10: **(User Score)** Given a tweet user u and a query $q(l,r,W)$, u's score with respect to q is

$$score(u,q) = \alpha \cdot \rho(u,q) + (1-\alpha) \cdot \delta(u,q)$$

Here, the keyword part $\rho(u,q)$ and the distance part $\delta(u,q)$ are combined through a smoothing parameter $\alpha \in [0,1]$.

Refer to the running example in Figure 1 and Table I. Tweet threads are created for all the seven relevant tweets. If we use the sum score based ranking, user u_1 is ranked as the top local user because u_1 has two relevant tweets A and G (and thus two threads) and A is very close to the query location. The sum based ranking favors users with more relevant tweets. In contrast, the maximum based ranking returns u_5 as the top. In our data set, u_5's tweet E has considerably more replies and forwards than other tweets. As a result, E's tweet thread has a higher popularity than others, and u_5 yields the highest maximum score. Although E is not the closest to the query location, u_5 overall is still favored in the maximum score based user ranking due to the outstanding maximum score.

IV. Data Organization and Indexing

In this section, we detail the organization and indexing of huge volumes of geo-tagged tweet data in our system.

A. Architecture and Data Organization

The system architecture is shown in Figure 3. Twitter Rest API is commonly used to crawl sample data in JSON format from Twitter. After extraction, transform and load (ETL), the metadata of all the tweets is stored in a centralized database. Since geo-tagged tweets occupy less than 1 percent of the whole tweet data set, we find it reasonable to separately process the geo-tagged tweets in a batch mode other than the existing real-time architectures which process general tweets. To be specific, we can periodically (e.g., one day) collect the spatial tweets and then build the index for these tweets. In such a setting, a scalable spatial-keyword index is built under Hadoop MapReduce system and stored in Hadoop distributed file system (HDFS). The tweet contents/texts are stored in HDFS as well. The hybrid index is to be detailed in Section IV-B. For each query, the postings lists are retrieved from the distributed spatial-keyword index. Two query processing algorithms (sum score and maximum score based ranking), to be detailed in Section V, are achieved by computation on tweet metadata database and the retrieved postings lists. In the meantime, the system collects the tweet contents according to the postings lists for later user study that evaluates the effectiveness of the proposed user ranking.

All tweets in our system form a relation with the schema of $(sid, uid, lat, lon, ruid, rsid)$ which is stored in a centralized metadata database as shown in Figure 3. The meaning of each attribute is explained as follows. Attribute "sid" represents the

tweet ID which is essentially the tweet timestamp. Attribute "uid" indicates the user ID of this tweet. Attributes "lat" and "lon" contain the spatial information of this tweet. Attribute "ruid" expresses the user ID whose tweet is replied to or forwarded by this tweet and "rsid" corresponds to the tweet ID which is replied to or forwarded by this tweet. Furthermore, attribute "sid" is the primary key for which we build a B^+-tree. Another B^+-tree is built on attribute "rsid". These indexes are used to accelerate the query processing phase.

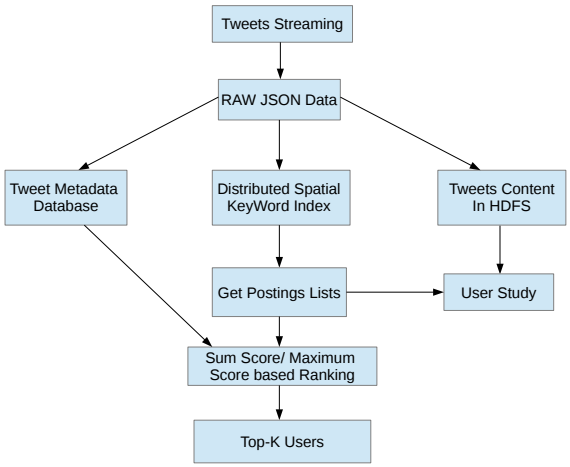


Fig. 3. System Architecture

Based on this database schema and Definition 4, Algorithm 1 reveals the process of constructing tweet thread (detailed in Section III-A) and computing the thread score. Line 7 is an SQL statement where I/Os happen. Note that in a practical implementation, a thread depth d is always set to constrain the construction process since constructing a complete tweet thread can incur quite a number of I/Os.

Algorithm 1: Construct a tweet thread and compute its popularity

Input: a tweetId tid, thread depth d
Output: score of tweet thread initiated by tid

```
1  AssociativeArray ⟨Integer, List⟨Integer⟩⟩ tweets;
2  Float popularity = ϵ;
3  Int i = 1;
4  tweets[i].add (tid);
5  while i ≤ d do
6      for j = 1; j ≤ tweets[i].size; j++ do
7          Id = tweets[i].get(j);
8          List temp = select all where rsid equals to Id;
9          if temp.size != 0 then
10             tweets[i + 1].add (temp);
11     popularity += tweets[++i].size × 1/i;
12 return popularity
```

B. Hybrid Index

1) Index Design: Our hybrid index design adapts the Geohash encode generally derived from quadtree index. Quadtree [9] is an easily-maintained spatial index structure which divides the spatial space in a uniform way. In a quadtree, each node represents a bounding square and contains four children if it is not a leaf node. In each node split, each quadrant is obtained by dividing the parent node in half along both the horizontal and vertical axes.

If we disregard the tree structure itself and encode each children by adding two bits (00 for upper-left, 10 for upper-right, 11 for bottom-right, 01 for bottom-left) to the parent encode such that each leaf node in a *full-height* quadtree is a complete geohash. The height of the tree determines the encode precision, for example, if we would like to encode a latitude/longitude pair (-23.994140625, -46.23046875) and make the encode precision at 20 bits, the encode will be "11001111111011010100". Then we change this into Base32 encode scheme which uses ten digits 0-9 and twenty-two letters (a-z excluding a,i,l,o). Every five bits will be encoded into a character so that the final geohash is "6gxp".

From the above description, points in proximity mostly will have the same prefix so that a trie, or prefix tree could be used for indexing the geohash. To answer a circle query, a set of prefixes need to be constructed which completely covers the circle region while minimizing the area outside the query region. The Z-order [22] curve is popularly utilized to construct such set of prefixes. The reason why we adapt geohash encode system is that the data indexed by geohash will have all points for a given rectangular area in contiguous slices. In a distributed environment, data indexed by geohash will have all points for a given rectangular area in one computer. Such advantage could save I/O and communication cost in query evaluation.

The hybrid index is illustrated in Figure 4. It contains two components: forward index and inverted index. Each entry in the forward index is in the format of $\langle ge_i, kw_i \rangle$ where ge_i is a geohash code and kw_i refers to a keyword. In our design, the forward index size is less than 12 MB if 4-length geohash encoding is used and all non-stop words in the twitter data set are considered. Therefore, it is kept in the main memory. The forward index associates each of its entry to a postings list ($\mathrm{P_i}$) in the inverted index that is stored in Hadoop HDFS. Based on the inverted index, $\langle ge_i, kw_i \rangle$ pairs are associated with tweet IDs in the tweet database. In other words, the inverted index refers each pair $\langle ge_i, kw_i \rangle$ to those tweets in the database that have kw_i in its content.

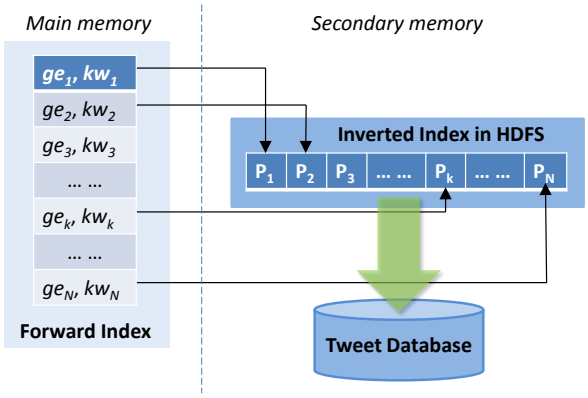


Fig. 4. Index Structure

In our implementation, the key of the inverted index is a pair of geohash code and a keyword. In this paper, we use "term" and "keyword" exchangeably as they essentially have the same meaning in the literature. Each entry in a postings list is a pair $\langle TID, TF \rangle$. Specifically, TID is the tweet ID that is essentially the tweet timestamp and TF represents the term frequency which is useful to count the occurrences of query keyword in a tweet when processing a query. The forward index for fetching the postings in query processing is implemented according to a previous proposal [16].

As we handle large volumes of geo-tagged tweet data, we choose to exploit the Hadoop MapReduce [6] [23] programming paradigm that offers scalability and fault-tolerance. Geohash encoding scheme fits well when we build the hybrid index for both spatial and text aspects in MapReduce.

2) Index Construction: Algorithms 2 and 3 illustrate the Map and Reduce steps of index construction algorithm respectively under the Hadoop MapReduce framework. The input to the mappers is a social media post p defined in Definition 1. In the map function, the content of each post is tokenized and each term is stemmed. Stop words are filtered out during the tokenization process. An associative array H is used to keep track of term frequency for each term. Then the map function traverses all terms: for each term, compute the geohash on the post's spatial information and create a posting. The posting is a pair consisting of timestamp $p.timestamp$ and term frequency $H\langle w \rangle$. Note that timestamp $p.timestamp$ corresponds to the tweet ID in our context since each timestamp is unique. Finally, the mapper emits a key-value pair with a pair $\langle geohash, term \rangle$ as the key and the posting as the value. In the reduce phrase, a new list P is initialized. Then the reduce function appends all postings associated with each pair $\langle geohash, term \rangle$ to the list. Each posting is a pair $\langle n, f \rangle$ where n represents tweet ID (timestamp) and f means the frequency. Finally, the postings are sorted by the timestamp $p.timestamp$ before they are emitted. The subsequent intersection operations on the sorted postings can be very efficient in the query processing (refer to lines 10–11 in Algorithms 4 and 5 in Section V).

Algorithm 2: Map Function of Constructing Inverted Index

Input: social media post p
1 AssociativeArray $\langle String, Integer \rangle$ H;
2 **for** *all term* $w \in p.W$ **do**
3 $\quad H\langle w \rangle = H\langle w \rangle + 1$;
4 **for** *all term* $w \in H$ **do**
5 $\quad$ Emit ($\langle geohash(p.l), w \rangle$, $\langle p.timestamp, H\langle w \rangle \rangle$);

Algorithm 3: Reduce Function of Constructing Inverted Index

Input: $\langle geohash\ g, term\ w \rangle$, postings $[\langle n_1, f_1 \rangle \ldots]$
1 List P;
2 **for** *all posting* $\langle n, f \rangle \in postings$ $[\langle n_1, f_1 \rangle \ldots]$ **do**
3 $\quad P$.Append ($\langle n, f \rangle$);
4 P.Sort( );
5 Emit($\langle geohash\ g, term\ w \rangle$, postings P)

To construct the aforementioned forward index (see Figure 4), another MapReduce job is run over the inverted index files in HDFS and a posting forward index is created to keep track of the position of each postings list in HDFS. Note that the Hadoop MapReduce framework can guarantee that the key of the inverted index is sorted. That means, the composite key $\langle geohash, term \rangle$ is sorted. As mentioned before, close points will probably have the same prefix of geohash so that close points associated with the same keyword are probably stored in contiguous disk pages. Such organization will substantially reduce the time accessing the postings lists which belong to close areas with the same keyword. In addition, by using MapReduce we can easily scale our index construction to terabyte-scale and even petabyte-scale data set.

V. Query Processing Algorithms

We detail query processing algorithms in this section.

A. Algorithm for Sum Score Based User Ranking

Algorithm 4 shows the sketch of the sum score based user ranking algorithm for processing TkLUS queries.

Algorithm 4: Query Processing for Sum Score Based Ranking

Input: a set of Keywords $W\{w_1, w_2, \ldots, w_n\}$, a location q, radius r, number of returning results k
Output: k UserIds $(u_1, u_2, \ldots, u_k)$
1 Array $Geohashes$ = GeoHashCircleQuery(q,r);
2 Array $PostingsLists$;
3 AssociativeArray $\langle Long, Float \rangle$ $users$;
4 **for** $i = 1; i \leq Geohashes.size; i++$ **do**
5 $\quad$ **for** $j = 1; j \leq W.size; j++$ **do**
6 $\quad\quad$ get postings list PL for key (GE_i, W_j);
7 $\quad\quad$ add PL into $PostingsLists$;
8 List P;
9 **if** *AND semantic* **then**
10 $\quad$ intersect the postlings lists from $PostingsLists$;
11 $\quad$ assign the intersect result to P;
12 **else if** *OR semantic* **then**
13 $\quad$ union the postlings lists from $PostingsLists$;
14 $\quad$ assign the union result to P;
15 **for** $j = 1; j \leq P.size; j++$ **do**
16 $\quad$ **if** *distance between* P_j *and* $q > r$ **then**
17 $\quad\quad$ **continue**;
18 $\quad$ Compute popularity according to Algorithm 1;
19 $\quad$ Compute keyword relevance score RS according to Definition 7;
20 $\quad$ **select** $userId$ **where** sid = $P_j.sid$;
21 $\quad$ **if** $userId \notin users$ **then**
22 $\quad\quad$ $users$.add($userId, RS$);
23 $\quad$ **else**
24 $\quad\quad$ $userId.score += RS$;
25 **for** *each user* $userId$ *in* $users$ **do**
26 $\quad$ Compute user score US according to Definition 10;
27 $\quad$ $userId.score = US$;
28 $users$.Sort( );
29 **return** the top k users in $users$ with the highest scores;

A list of geohash cells are returned by a circle query for given a location and radius (Line 1). Lines 2 to 14 get corresponding postings lists. Lines 9 to 14 show the difference between "AND" and "OR" semantics. The "AND" semantic requires the search results containing all the query keywords while the "OR" semantic relaxes the constraint by requiring a portion of the query keywords. Generally, "OR" semantic retains much more candidates than "AND" semantic. Lines 15 to 24 show the process of traversing the postings lists in P and calculating keyword relevance score RS for each user encountered according to Definition 7. All candidate users are put in an associate array $users$. Lines 25 to 27 computes user score US for each user according to Definition 10. Finally, $users$ is sorted and the top k users with the highest user scores are returned (Line 28 to 29).

B. Algorithm for Maximum Score Based User Ranking

In running the query processing algorithm proposed in Section V-A, we observe that the bottleneck of query processing lies in constructing the tweet thread for each relevant posting since every construction will cost several I/Os. To alleviate this problem, we define the upper bound score of a single tweet thread and make use of it to speed up query processing according to the maximum score (Definition 8).

In the following, we define the upper bound popularity of a tweet thread, where t_m means the maximum number of replied tweets a tweet can have in our database.

Definition 11: **Upper Bound Popularity of Tweet Thread**

$$\phi(p)_m = \sum_{i=2}^{n} |t_m| \ \times \ \frac{1}{i}$$

Algorithm 5 sketches the query processing according to the maximum score. Lines 1 to 17 are the same as the counterpart in Algorithm 4, except that we now use a priority queue $topKUser$ to maintain the current top-k candidate users (Line 3). The pruning process is shown at Lines 18 and 19. In Line 18, UpperBound is calculated as a linear combination (dependent on smoothing parameter ϵ) of the current tweet's upper bound popularity (Definition 11) and the maximum distance score (for the distance part in Definition 10, the maximum distance score can be 1), and topKUser.peek() gets the element with the least user score from the priority queue $topKUser$. If the current tweet thread's UpperBound score is less than the least score maintained in the priority queue, there is no need to construct the thread and to calculate the actual popularity for that tweet. Lines 22 to 33 update the user and score in the priority queue if necessary, such that a candidate user's maximum based user score is always kept in the queue.

In the running example (Figure 1 and Table I), user u_1 has two relevant tweets A and G. Algorithm 5 makes use of the upper bound popularity (Definition 11) and avoids constructing the threads for u_1 (Line 20) if u_1's overestimated user score based on the upper bound is less than u_5's user score that is already computed.

Definition 11 is the global upper bound for all keywords that can appear in all the tweets in a specific tweet thread, since t_m is the maximum number of replied tweets that a tweet can have in our database. As a matter of fact, the upper bound of any "specific keyword related" tweet threads should be much smaller than t_m. Therefore, it is desirable to maintain a specific bound for some hot keywords. Such a bound is expected to be lower than the global upper bound and thus tighter.

Algorithm 5: Query Processing for Maximum Score Based Ranking

Input: a set of keywords $W\{w_1, w_2, \ldots, w_n\}$, a location q, radius r, number of returning results k
Output: k UserIds $(u_1, u_2, \ldots, u_k)$

1 Array $Geohashes$ = GeoHashCircleQuery(q,r);
2 Array $PostingsLists$;
3 PriorityQueue $\langle PairOfLongFloat \rangle topKUser$;
4 **for** $i = 1; i \leq Geohashes.size; i++$ **do**
5 **for** $j = 1; j \leq W.size; j++$ **do**
6 get postings list PL for key (GE_i, W_j);
7 add PL into $PostingsLists$;
8 List P;
9 **if** *AND semantic* **then**
10 intersect the postlings lists from $PostingsLists$;
11 assign the intersect result to P;
12 **else if** *OR semantic* **then**
13 union the postlings lists from $PostingsLists$;
14 assign the union result to P;
15 **for** $j = 1; j \leq P.size; j++$ **do**
16 **if** *distance between* P_j *and* $q > r$ **then**
17 continue;
18 **if** $topKUser.size() == k \ \wedge \ UpperBound < topKUser.peek()$ **then**
19 continue;
20 Compute popularity according to Algorithm 1;
21 Compute user score US according to Definition 8 and Definition 10;
22 **select** $userId$ **where** sid = $P_j.sid$;
23 **if** $topKUser.size < k$ **then**
24 **if** $userId \notin topKUser$ **then**
25 $topKUser$.add($userId, US$);
26 **else**
27 $userId.score = US$;
28 **else if** $topKUser.size == k$ **then**
29 **if** $userId \notin topKUser \ \wedge \ topKUser.peek() < US$ **then**
30 $topKUser$.remove();
31 $topKUser$.add($userId, US$);
32 **else if** $userId \in topKUser \ \wedge \ userId.score < US$ **then**
33 $userId.score = US$;
34 **return** $topKUser$;

In our implementation, we identify from the data set top-10 frequent keywords as shown in Table II. For each top frequent keyword, a specific upper bound popularity is pre-computed by offline constructing tweet threads and selecting the largest

thread score as the "specific keyword related" upper bound. For example, for hot keyword "restaurant", we may maintain a upper bound popularity for all tweet threads initiated by the tweet containing keyword "restaurant". When queries with keyword "restaurant" come in, we use the "restaurant related" upper bound popularity instead of the global upper bound popularity. For queries without any hot keyword, global upper bound popularity is still used. We evaluate the effect of such hot keyword upper bounds in Section VI-B5.

It is possible that one user has many tweet threads but none of them gets a top-k highest score. In such a case, the maximum score based query processing algorithm will not include the user as a top-k local user in the query result. Note that the user may be included in the query results by the sum score based query processing algorithm, due to the user's large number of tweet threads. We compare the two query processing algorithms by measuring the Kendall Tau coefficient of their query ranking results in Section VI-B.

TABLE II. TOP-10 FREQUENT KEYWORDS

Frequency Rank	Keyword	Frequency Rank	Keyword
1	restaurant	2	game
3	cafe	4	shop
5	hotel	6	club
7	coffee	8	film
9	pizza	10	mall

VI. EXPERIMENTAL STUDY

We conduct an extensive experimental study to evaluate the proposed techniques for processing TkLUS queries. We sample a real Twitter data set with geographical coordinates collected by using Twitter REST API. The 20.3GB data set covers the period from September 2012 to February 2013, having nearly 514 million geo-tagged tweets in total. In this section, we report on the relevant experimental results.

A. Index Construction Time and Storage Cost

Our first experiments evaluate the index construction. We vary Geohash configuration and measure the index construction time and index size. We construct the proposed hybrid index (Section IV-B) in a Hadoop MapReduce cluster of three PCs (see Table III for details). The CPUs of three PCs are all Quad Core Intel(TM) i7 Intel(R) Core(TM) i7. One PC is configured as the master and the others are used as slaves.

TABLE III. CLUSTER SUMMARY

Node Type	Memory	Hard Disk	CPU
Master	8GB	220GB	Quad Core Intel(TM) i7
Slave	8GB	500GB	8 Core Intel(TM) i7
Slave	8GB	500GB	8Core Intel(TM) i7

Figure 5 shows that the index construction efficiency is insensitive to the Geohash configuration. The index construction time is steady around 850 minutes. The state-of-the-art spatial keyword index I^3 [25], built in a single machine (Quad-Core AMD Opteron (tm) Processor 8356 and 64GB memory), processes 15 million geo-tagged tweets in around 1,500 minutes. In contrast, our MapReduce index construction algorithm processes one order of magnitude more tweets but incurs one order of magnitude less time than I^3 index. Note that our index is built and stored in a distributed way while the others like I^3 and IR-tree variants are centralized and unable to process large scale data; neither can they solve TkLUS queries.

Figure 6 shows the results of the hybrid index sizes. The index size is very steady as the Geohash configuration varies, occupying about 3.5 gigabytes in HDFS. This index size is almost the same as I^3, although our indexes handle one order of magnitude more tweets.

Fig. 5. Index Construction Time

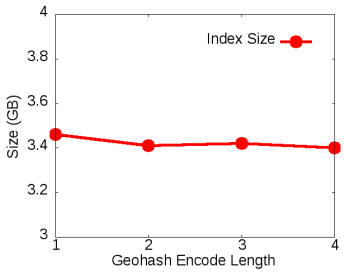


Fig. 6. Index Size

B. Evaluation of Query Processing

In this part of experiments, we still store the index and tweets on HDFS, and keep tweet metadata in a central database. The cluster parameters are given in Table III.

1) Query Settings: Based on the data set statistics, we select 30 meaningful keywords including the top-10 frequent ones shown in Table II. In the experiments, a 1-keyword query randomly gets one out of the 30. Queries with 2 and 3 keywords are constructed from AOL query logs[5] that contain the single keyword from Table II. For example, queries "restaurant seafood" and "morroccan restaurants houston" occurring in AOL query logs are used in our experiment. Each query is randomly associated with a location that is sampled according to the spatial distribution in our data set. Finally, random combinations of keywords and locations form a 90-query set used in our experiments. Each query type in terms of keyword number (single keyword, two keywords and three keywords) has 30 corresponding queries in the query set.

According to Equation 10, we set α as 0.5 so that the two factors are considered as having the same impact. Both HDFS caches and database caches are set off in order to get fair evaluation results. Before query processing starts, the system first loads the postings forward index into memory since it is always small (less than 12MB in our experiments). Random access to inverted index in HDFS is disk-based.

When issuing the queries in the experiments, we vary the number of query keywords ($|q.W|$) from 1, 2 to 3, and the number k of local users to return from 5 to 10. On each configuration, we vary the query radius from 5 km to 20 km. The ϵ in Definition 4 is set 0.1 in our implementation.

2) Effect of Geohash Encoding Length: We first vary the Geohash configuration from 1-length encoding to 4-length encoding, and see the effect on the query processing. As an example, suppose the coordinate is (-23.994140625, -46.23046875), Table IV shows the geohash at different lengths.

For each query radius, we issue 10 queries randomly chosen from the query set described in Section VI-B1. We report the average query processing time in Figure 7.

[5]http://www.gregsadetsky.com/aol-data/

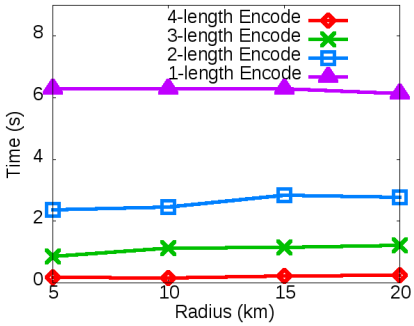


Fig. 7. Effect of Geohash Encoding Lengths

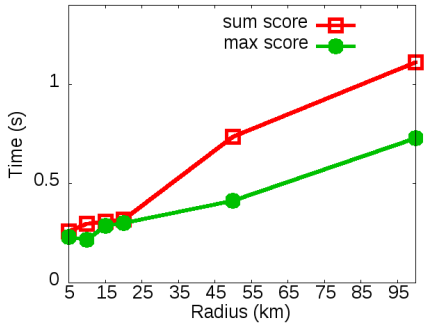


Fig. 8. Single Keyword Efficiency

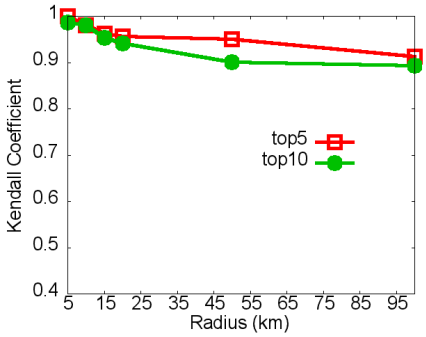


Fig. 9. Kendall Tau for Single Query Keyword

TABLE IV. GEOHASH ENCODING LENGTH EXAMPLE

Length	Geohash	Length	Geohash
1	6	3	6gx
2	6g	4	6gxp

A shorter length Geohash encoding means a coarser grid, i.e., fewer but larger grid cells. Although a longer encoding complicates the construction of a set of prefixes completely covering the query range and thus results in more candidate grid cells in search, it does not necessarily lead to more I/Os since the close points associated with the same keyword are probably stored in contiguous disk pages. With almost the same I/O cost, a shorter length encoding yields larger grid cells and needs to process more points in each cell. Thus, longer length encodings benefit TkLUS query processing.

Nevertheless, it does not mean that the longest Geohash encoding is favorable since the query efficiency also depends on the query range. If the query range is large and the encoding is long, I/O cost can increase substantially. In our experiments, the queries focus on searching for local users within the distance of 5 km to 20 km from the query location. These are practical settings for many location based services, for which the 4-length encoding works well. Moreover, the 4-length encoding is sufficient to handle the precision since the latitudes and longitudes have at most 8 bits of decimals in our data set. Based on these observations, we decide to use 4-length geohash encoding in the remaining experiments.

3) Results on Queries with Single Keyword: Next, we vary the query radius from 5 km to 100 km and use only the single keyword queries described in Section VI-B1.

We compare the query processing efficiency of the aforementioned two ranking methods (namely sum score based and maximum score based), and report the results in Figure 8. Both methods incur longer query processing time when the query range increases. For a query range not greater than 20 km, the two methods perform closely and the maximum score based ranking method is only slightly better. For larger query ranges, the maximum score based ranking method clearly outperforms the sum score based ranking method. The superiority of maximum score based ranking method for large query ranges is attributed to its pruning power that works more visibly when there are more candidates involved in large query ranges. Small query ranges contain less tweets and thus the pruning does not make a big difference.

We investigate Kendall tau rank correlation coefficient between the query results using two different ranking functions. Suppose that the two top-k query results are ρ_b and ρ_d respectively. Let cp be the number of user pairs (u_i, u_j) whose rankings in ρ_b and ρ_d are concordant, i.e., if u_i is ranked before (after or in tie with) u_j in both ρ_b and ρ_d. Let dp be the number of user pairs (u_i, u_j) whose rankings are discordant in the two results. The Kendall tau coefficient is thus defined as

$$\tau(\rho_b, \rho_d) = \frac{cp - dp}{0.5k(k-1)}.$$

Since the query results using two different ranking functions are not exactly the same, a variant of Kendall tau rank correlation coefficient is proposed here to compare the two rankings. For example, suppose $k = 3$, ρ_b is $\langle A, B, C\rangle$ and ρ_d is $\langle B, D, E\rangle$. In order to measure the Kendall tau coefficient between ρ_b and ρ_d, we modify ρ_b and ρ_d to $\langle A, B, C, D, E\rangle$ and $\langle B, D, E, A, C\rangle$ respectively. The elements we add into a ranking have the same ordering value, e.g., elements D and E both are ranked the 4th in ρ_b. The same applies to ρ_d.

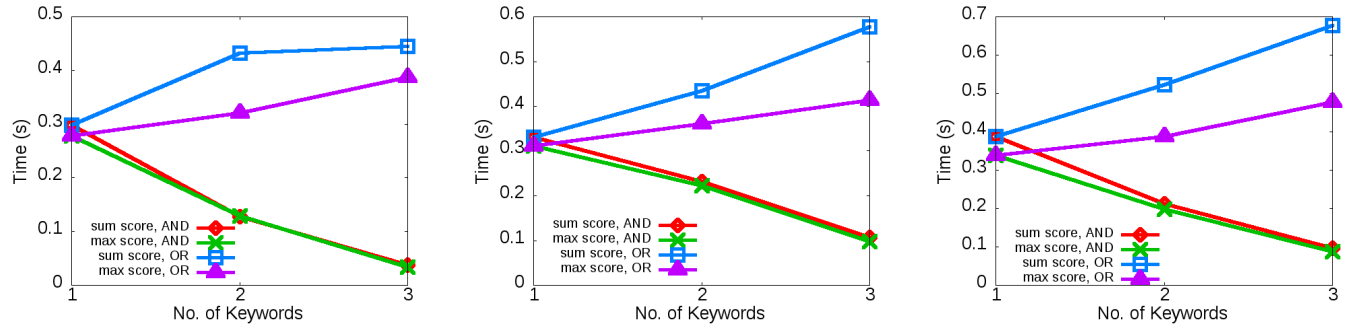
We thus measure the variant Kendall tau coefficient for single keyword top-5 and top-10 query results, and show the coefficient in Figure 9. In all tested settings, the Kendall tau coefficient is higher than 0.863. This suggests that the two different ranking methods are highly consistent.

4) Results on Queries with Multiple Keywords: In this set of experiments, we vary the number of keywords in each query from 1 to 3 and see the effect on the query processing.

In the query processing, either OR or AND semantic can be used to handle the multiple keywords included in a query. The results on query processing efficiency are reported in Figure 10. Overall, more keywords in the query incur longer query processing time in OR semantic while the opposite in AND semantic. The reason is straight forward since AND semantic filters out more candidates.

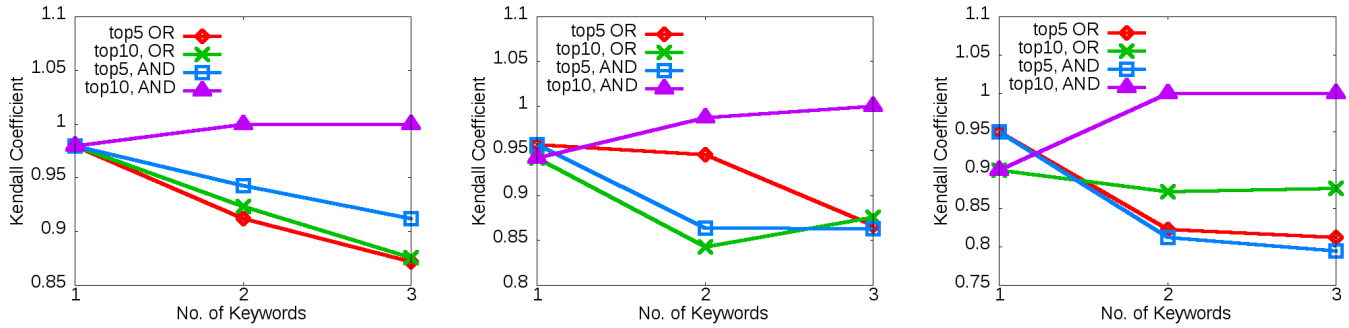
Generally, the maximum score based user ranking yields a better performance compared to the sum score based ranking, especially for 20 km and 50 km query ranges. The intersection operation used in processing the AND semantic prunes a lot of candidates, such that there is not much room for the pruning power for maximum score based ranking to take effect. In contrast, the OR semantic is processed by the union operation that results in more candidates and more room for pruning.

In the same set of experiments, we also measure the Kendall tau coefficient in the same way as described in Section VI-B3. The experimental results are shown in Figure 11.



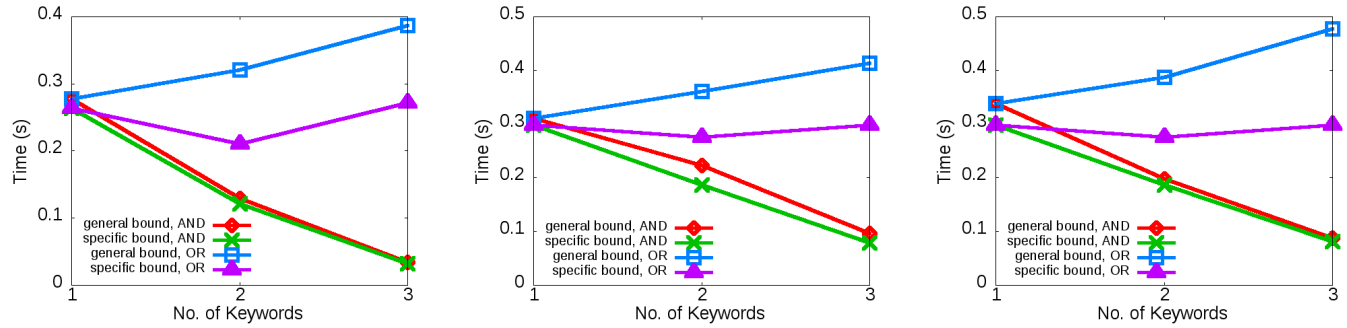
(a) 10 km query range (b) 20 km query range (c) 50 km query range

Fig. 10. Query Efficiency with Multiple Query Keywords



(a) 10 km query range (b) 20 km query range (c) 50 km query range

Fig. 11. Kendall Tau for Multiple Query Keywords



(a) 10 km query range (b) 20 km query range (c) 50 km query range

Fig. 12. Effect of Specific Tweet Popularity Bound

For the AND semantic, the Kendall tau coefficient is always higher than 0.95 at different query ranges. This indicates that two user ranking methods always return highly similar query results. For the OR semantic, the lowest Kendall tau coefficient is slightly below 0.8. The two ranking methods are still consistent.

5) Effect of Specific Popularity Bound for Maximum Score Based User Ranking: As described in Section V-B, the maximum score based query processing algorithm utilizes specific upper bound tweet popularity derived from hot keywords. Such specific bounds are pre-computed and expected to improve the performance for queries that contain those hot keywords. We maintain upper bound popularity for 10 hot keywords shown in Table II and Figure 12 reports the results about the query improvements in comparison to the use of the general upper bound (Definition 11 in Section V-B). For multiple keywords, "AND" semantic uses the smallest upper bound among the query keywords whereas "OR" semantic chooses the largest upper bound. For example, if "Mexican restaurant" is in a query and the upper bound popularity of "restaurant" is larger than "Mexican", "AND" semantic will use the upper bound popularity of "Mexican" whereas "OR" semantic chooses the upper bound popularity of "restaurant".

It is clear that using such specific popularity bound of hot keywords fastens the query processing for both semantics. As the query range increases, the performance gain becomes more visible. The specific popularity bound is effective because those hot keywords help rule out irrelevant tweets when the query processing algorithm computes tweet threads.

6) User Study: We conduct a user study to evaluate the effectiveness of our scoring mechanisms (Section III) for top-k local users. A total of 30 queries with one to three keywords are issued at random. For each query, we output the top-10 query results with different query ranges (5, 10, 15, and 20 km). Each line of a top-10 query result is formulated as a pair (userId, tweet content), where userId identifies a user and the

tweet content is the corresponding tweet keywords found by the query.

Six participants familiar with Twitter are invited to give relevance feedbacks for the query results. A participant gives 1 to a query result line if she/he thinks it is relevant to the query, or 0 otherwise. We assign 20 top-10 query results to each participant so that each query result are evaluated four times. If a particular Twitter user's tweets are considered relevant twice or even more, that user will be regarded as a relevant user for the corresponding query.

We adapt precision as the effectiveness metric in the user study. Precision in our context is defined as the fraction of the returned local users that are regarded as relevant by the user study. We measure the precision for top-5 and top-10 query results, and the results are shown in Figure 13.

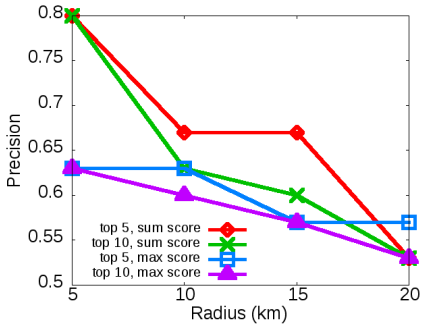


Fig. 13. User Study Results

Overall, both user ranking methods (sum score based and maximum score based) are very effective for small and moderate query ranges—the precision is always between 60% and 80% for query ranges not larger than 10 km. The precision roughly decreases as the query range increases. These findings justify our distance score that assigns higher importance to tweets created at locations closer to the query location.

On the other hand, top-5 query results always achieve higher precision (up to 80%) than the top-10 counterparts. This certainly indicates that our user ranking methods are effective in prioritizing more relevant local users in the query results.

VII. Related Work

A. SIR and Spatial Keyword Search

Spatial Information Retrieval (SIR) refers to providing access to geo-referenced sources by indexing, searching, retrieving and browsing [12]. "Web-a-Where" [1] attempts to build an IR system for tagging each web page with a geographic focus. "Steward" [15] provides a complete architecture of a spatio-textual search engine.

IR-trees combine R-tree with inverted indexes [5], [14]. In an IR-tree, each node represents an MBR that is associated with an inverted file. That inverted file refers to all the documents whose locations fall into the associated MBR. Although IR-tree family utilize both spatial and text attributes to prune the search space to facilitate query evaluation, it suffers from the scalability issue since node splitting in such a data structure incurs considerable I/O costs. Based on IR-tree, Rocha-Junior et al. [21] proposed the S2I index which improves the query evaluation performance. Recently, the I^3 [25] index is proposed to offer better scalability for large data sets and significantly outperforms in data update and query processing.

All these works above are centralized solutions and cannot scale seamlessly in a distributed way for extremely large data sets that a single computer cannot handle efficiently. In contrast, this work proposes an indexing mechanism to handle high-volume social media data in a decentralized manner. Furthermore, this work focuses on finding local users in social media, differing from mobile local search [18], [19] that considers no social context when recommending businesses close to query locations.

B. Microblog and Social Media Data Management in General

TI [4] addresses real-time microblog indexing and search. The core part of TI is to classify tweets into distinguished and noisy tweets. The classification algorithm mainly depends on the statistics of past query log. The intuition behind the classification is that tweets getting high score on past queries are more probably to be retrieved in the future. After classification, the distinguished tweets are indexed immediately, namely in real time while the rest (noisy tweets) are processed in batch operation. In that way, TI substantially reduces the number of tweets it needs to process in real-time. However, TI suffers from scalability because of its centralized system design and the classification algorithm is not applicable when future queries are different from past query log.

Earlybird [2] is the proprietary distributed system developed by Twitter to overcome the practical system issues inside Twitter such as inverted index organization and concurrent control. The most recent academic work on real-time microblog indexing and search is Pollux [17] which is also a distributed system aiming to solve the concerns of fault tolerance and global storage effectiveness.

The provenance model [24] targets at the connections between microblog posts, clustering them into so-called bundles with respect to how the posts are originated and how they evolve in the microblog platform. Indexing and retrieval techniques are also proposed [24] to efficiently manage such microblog bundles. The provenance model of microblogs is query independent. In contrast, the tweet thread proposed in this paper is dynamically generated according to the current user query. Moreover, the provenance model does not support location-dependent search in the microblog data.

Since geo-tagged tweets occupy less than 1 percent of the whole tweets data set, we propose a batch processing setting in Section IV-A other than the existing real-time architectures.

Our TkLUS problem differs from the microblog based jury selection problem [3] that involves no locations or keywords but employs a boolean voting to find a set of jurors that achieve high votes. Our TkLUS problem also differs from a friend search approach [20] in which only the query issuer's friends are explored and returned according to multiple scores.

A recent work [8] mines influential event organizers from online social networks. Compared to this paper, it solves a different problem with a different model of social network, and it considers no spatial locations.

C. Exploiting Locations in Social Media

Spatial-aware search and analysis of social media becomes an important research topic [7]. Lee et al. [13] develop algorithms to mine tweets stream for real-time and geo-spatial event information. Kim et al. [11] construct and visualize spatiotemporal trends of topics based on geo-tagged tweets.

A location-based social network (LBSN) consists of social structures made up of individuals, who are connected by the interdependency derived from their locations in the physical world, as well as their location-tagged media content such as photos, video, and text [26].

Aardvark [10] is a social search engine. When users ask a question in Aardvark, the system would route the question to other users who are most likely to answer this question. The key point in such social search engine is to find the right *person* to satisfy one's information need instead of retrieving the right *document*. In this social engine, location is a key factor utilized in the process of routing the question since many of the questions are locality sensitive.

Unlike related works, TkLUS studied in this paper is a new problem in the context of geo-tagged tweets. This paper distinguishes itself from the literature by the tweet thread concept, the scoring functions for tweets and users, the MapReduce-enabled hybrid index for tweets, and TkLUS query algorithms.

VIII. Conclusion and Future Work

This paper tackles top-k local user search (TkLUS) queries in geo-tagged social media. Given a location q, a distance r, and a set W of keywords that describe user needs, a TkLUS query finds the top-k local users who have posted tweets relevant to the keywords in W at a place within the distance r from q. A set of relevant techniques are designed to process such TkLUS queries. The concept of tweet thread is proposed to capture tweet popularity, based on which ranking functions that take into account both keyword relevance and spatial proximity are defined for tweets and users. A hybrid index is design for high volume geo-tagged tweets in a distributed mode. Pruning bounds and efficient algorithms are developed for processing TkLUS queries. The proposed techniques are evaluated through experiments on a large set of real tweets with geo-tags. The experimental results demonstrate that our proposals are scalable, efficient and effective.

Several directions exist for future research. The definition of TkLUS query can be extended by introducing temporal considerations in prioritizing tweets and local users. For example, we can define a query for a particular period of time and only search the tweets that are posted in that period. Also, we can still search all tweets but give priority to more recent tweets (and their users) in ranking.

There are also tweets that lack longitude/latitude in the metadata but mention place name(s) in the short content. It is worth studying how to exploit the implicit spatial information in such tweets to serve user needs like TkLUS queries.

We focus on geo-tagged tweets in this paper. It is also interesting to make the search for local users across the platform boundary, such that more informative query results can be obtained by involving different social networks.

Acknowledgments

The authors would like to thank Anders Skovsgaard for his generous help on collecting the geo-tagged tweet data used in the experimental study. Bin Cui is supported by the National Natural Science Foundation of China under Grant No. 61272155.

References

[1] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *SIGIR*, pages 273–280, 2004.

[2] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-time search at twitter. In *ICDE*, pages 1360–1369, 2012.

[3] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.

[4] C. Chen, F. Li, B. C. Ooi, and S. Wu. Ti: an efficient indexing mechanism for real-time search on tweets. In *SIGMOD Conference*, pages 649–660, 2011.

[5] 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.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[7] L. Derczynski, B. Yang, and C. S. Jensen. Towards context-aware search and analysis on social media data. In *EDBT*, pages 137–142, 2013.

[8] K. Feng, G. Cong, S. S. Bhowmick, and S. Ma. In search of influential event organizers in online social networks. In *SIGMOD Conference*, pages 63–74, 2014.

[9] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[10] D. Horowitz and S. D. Kamvar. The anatomy of a large-scale social search engine. In *WWW*, pages 431–440, 2010.

[11] K.-S. Kim, R. Lee, and K. Zettsu. *m*trend: discovery of topic movements on geo-microblogging messages. In *GIS*, pages 529–532, 2011.

[12] R. R. Larson and P. Frontiera. Geographic information retrieval (gir): searching where and what. In *SIGIR*, page 600, 2004.

[13] C.-H. Lee, H.-C. Yang, T.-F. Chien, and W.-S. Wen. A novel approach for event detection by mining spatio-temporal information on microblogs. In *ASONAM*, pages 254–259, 2011.

[14] 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. *TKDE*, 23(4):585–599, Apr. 2011.

[15] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. Steward: architecture of a spatio-textual search engine. In *GIS*, page 25, 2007.

[16] J. J. Lin, T. Elsayed, L. Wang, and D. Metzler. Of ivory and smurfs: Loxodontan mapreduce experiments for web search. In *TREC*, 2009.

[17] L. Lin, X. Yu, and N. Koudas. Pollux: towards scalable distributed real-time search on microblogs. In *EDBT*, pages 335–346, 2013.

[18] Y. Lv, D. Lymberopoulos, and Q. Wu. An exploration of ranking heuristics in mobile local search. In *SIGIR*, pages 295–304, 2012.

[19] D. Lymberopoulos, P. Zhao, A. C. König, K. Berberich, and J. Liu. Location-aware click prediction in mobile local search. In *CIKM*, pages 413–422, 2011.

[20] A. Nandi, S. Paparizos, J. C. Shafer, and R. Agrawal. With a little help from my friends. In *ICDE*, pages 1288–1291, 2013.

[21] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222, 2011.

[22] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[23] T. White. *Hadoop - The Definitive Guide: MapReduce for the Cloud*. O'Reilly, 2009.

[24] J. Yao, B. Cui, Z. Xue, and Q. Liu. Provenance-based indexing support in micro-blog platforms. In *ICDE*, pages 558–569, 2012.

[25] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013.

[26] Y. Zheng. Location-based social networks: Users. In *Computing with Spatial Trajectories*, pages 243–276. 2011.