Dynamically Ranked Top-K Spatial Keyword Search

Suprio Ray
University of New Brunswick
Faculty of Computer Science
Fredericton, N.B. Canada
sray@unb.ca

Bradford G. Nickerson
University of New Brunswick
Faculty of Computer Science
Fredericton, N.B. Canada
bgn@unb.ca

ABSTRACT

With the growing data volume and popularity of Web services and Location-Based Services (LBS) new spatio-textual application are emerging. These applications are contributing to a deluge of geo-tagged documents. As a result, top-k spatial keyword searches have attracted a lot of attention and a number of spatio-textual indexes have been proposed. However, these indexes do not consider the "recency" of the indexed documents. Part of the challenge is due to the fact that the textual relevance score measures that these indexes use, require all documents to be inspected.

To address these issues, we propose the idea of "dynamic ranking" of spatio-textual objects. We also introduce a novel index, called STARI, which uses this ranking method to retrieve the most recent top-k relevant objects. Experimental evaluation demonstrates that that our system can support high document update rates and low query latency.

CCS Concepts

•**Information systems** → **Query representation; Query reformulation; Top-k retrieval in databases; Retrieval efficiency; Search engine indexing; Search index compression; Web and social media search;** *Data encoding and canonicalization; Information retrieval diversity;* •**Applied computing** → **Document searching;** •**Social and professional topics** → *Geographic characteristics;* •**Computing methodologies** → *Concurrent algorithms;*

Keywords

spatio-textual; index; geo-tagging; spatial keyword search

1. INTRODUCTION

We are in the era of Big Data, in which vast quantities of data from a variety of sources are being generated. Web documents constitute a significant portion of this data. Increasingly, these documents contain location information. For instance, many Web documents contain information regarding popular places of attraction and restaurants. In recent times the proliferation of GPS-enabled mobile devices has made Location-Based Services (LBS) a pervasive technology. The confluence of the Web and LBS is contributing to the generation of unprecedented amounts of geo-tagged documents. This is driven by applications such as Twitter, Flickr, Foursquare, Facebook, Google Maps and Yellow Pages.

Top-k spatial keyword searches can provide significant insight into the hundreds of millions of daily tweets, Facebook comments, and blog updates as they relate to spatial location. A top-k spatial keyword query retrieves k objects based on both the distance of the generated documents (objects) from the query location and textual relevance. Efficient retrieval and update of geographically relevant textual information can be highly valuable for navigation, tracking news events that have an impact on business, and for entertainment. However, the value of such information often depends on the recency of the generated document. For instance, we pay more attention to the "breaking news" or recently reported tweets than those generated a week or a month ago. Similarly, "hot new" special deals or discounts are of interest to a customer in personalized mobile advertising. As time passes, these documents progressively lose their importance.

Due to the popularity of geo-textual search, a number of spatio-textual indexes have been proposed, such as [5, 13, 16]. However, they do not consider the "recency" or relevance based on aging of the indexed documents. These indexes use a ranking function that combines the spatial distance of the documents from the query location and textual relevance of the documents with the query keywords. Two popular measures to calculate textual relevance score are the classic tf-idf [8] and the language model [11]. For instance, the recently proposed I^3 index [16] uses tf-idf measure. However, these measures calculate the relevance score based on the entire history of all indexed documents and they do not consider "recency" of the documents. In order to calculate the score all documents must be inspected. For instance, the term idf in the tf-idf measure is calculated by dividing the total number of documents by the number of documents in which a term appears. Therefore existing indexes must calculate the term weights of each document statically during the pre-processing stage.

To address the limitations of existing indexes, we consider the efficient indexing and retrieval of most recent objects with the highest ranking scores that match query keywords and locations. We introduce the concept of "dynamic ranking" for geo-textual search. The notion of "most recent" uses exponential decay based on time to decide the weight of matching objects in the ranking process. More recently appearing matching objects rank higher than older matching objects with the same keyword and location score.

To support efficient most recent top-k spatial keyword search, we introduce an index called STARI (Spatio-Textual Adaptive Ranking-based Index). STARI is an indexing approach that supports con-

GeoRich'16, June 26-July 01 2016, San Francisco, CA, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4309-1/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2948649.2948655

current document updates and query execution. The updates are handled by a component that combines an in-memory grid with inverted list. The grid approach allows the handling of updates in a parallel fashion, and two different threads can concurrently handle updates for two different grid cells. Queries are directed to a fast Sort-Tile-Recursive (STR) R-tree that can perform efficient circular nearest neighbor search from the query location. The STR R-tree indexes the cells of the grid, but is not involved in any updates.

In our approach, the textual relevance score of the documents are **not** calculated during the index time. STARI uses our dynamic ranking measure to calculate textual relevance score of a small subset of documents based on the keywords in a given query in real-time. This on the fly computation of relevance score ensures that the search returns the most recent relevant objects. We implemented a prototype of our system and evaluated it with a real world dataset. Our experimental evaluation suggests that STARI can support high rates of document updates, while supporting low latency query execution. When compared with an existing spatio-textual index, such as IR-tree, we show that STARI can support real-time updates, whereas IR-tree does not.

The rest of the paper is organized as follows. In Section 2 we describe the related works. We introduce the problem and present our dynamic ranking scheme in Section 3. In Section 4 the system organization and in Section 5 our index STARI are presented. We evaluate STARI in Section 6 and finally, Section 7 concludes the paper.

2. RELATED WORK

Due to the rise of streaming data sources and the popularity of Web search, the topic of keyword search on data streams has been addressed by a number of research projects. For instance, Markowetz et al. [10] proposed a technique for keyword search on relational data streams without requiring any knowledge of the schema. However, their approach did not incorporate a ranking mechanism or return top-k results. The work of Cheng et al. [2] focussed on extracting representative and diversified posts from data streams, particularly in the context of microblogging data. None of these approaches considered spatio-textual keyword search.

In recent years spatial keyword search has become increasingly important, partly, due to the emergence of Location-based Services (LBS). Because of this, a number of spatio-textual indexes have been proposed. Depending on the structure of these indexes, they can be classified into three groups: R-tree based indexes, grid-based indexes and space filling curve based indices. Chen et al. [1] conducted an evaluation study of 12 of these indexes and reported that not all of them support the top-k spatial keyword search query.

The IR-tree [5] is one of the R-tree based approaches that support top-k spatial keyword search. It integrates an R-tree with an inverted file. It augments the nodes of the R-tree with summary information of the textual content of the node's corresponding subtree. In this approach an upper bound of the textual relevance score is computed from the summary information, and the spatial relevance is calculated from the MBR. This can be used for pruning search paths. The authors also present a few variants of the IR-tree, namely, the DIR-tree, the CIR-tree and the CDIR-tree. The S2I [13] index uses two different approaches for frequent terms and infrequent terms. For infrequent keywords, all the elements in the inverted file are stored sequentially for efficient I/O. Otherwise an aggregated R-tree (aR-tree) is used for pruning. The I^3 index [16] uses a similar textual partitioning approach as S2I. However, they use a Quadtree instead of an R-tree. Recently, Choudhury et al. [3] present an approach for batch processing of queries to take advantage of the spatial proximity of multiple queries when searching the IR-tree forming the basis of their spatial index.

SFC-QUAD [4] is one among only a few non-R-tree based approaches that support top-k spatial keyword search. It is essentially an inverted file in which the document id and object frequencies are compressed using the OPT-PFD algorithm [15]. In each inverted list the document ids are ordered based on their position in a Z-curve. It also maintains a Quadtree for efficient traversal.

None of the above-mentioned indexes considered the concept of retrieving most recent documents in real-time data streams. In [9] the authors present a system for real-time support of top-k spatio-temporal queries, but don't consider keyword search. Guo et al. [7] present a system to support continuous top-k spatial keyword queries on objects moving on road networks. Our approach differs from the existing approaches in that it considers the age of the indexed documents and retrieves the most recent document based on dynamic ranking.

3. PROBLEM STATEMENT

Our system supports top-k queries $q = (q.l, q.d, t)$ that are dynamically ranked, where $q.l$ is the query location, $q.d$ is the set of keywords being searched for and t is the time of the query. Each document $o = (o.id, o.l, o.d, t_o)$ in the set O of indexed documents to be searched has an id $o.id$, a location $o.l$, a set of words or terms $o.d$ describing the object or document, and a time t_o that the document was inserted into the database.

3.1 Spatial Match

Spatial match $S(o.l, q.l) \in [0, 1]$ is defined as the distance from the query location $q.l$ to the document location $o.l$, normalized by r_{max}, where r_{max} is the max distance between any two points in the spatial domain.

$$S(o.l, q.l) = \begin{cases} 0 & \text{if } o.l = q.l \\ 2(d/r_{max})^2 & \text{if } 0 < d \leq r_{max}/2 \\ 1 - 2((d - r_{max})/r_{max})^2 & \text{if } r_{max}/2 \leq d < r_{max} \\ 1 & \text{if } d \geq r_{max} \end{cases} \tag{1}$$

where d is the distance from $q.l$ to $o.l$. Equation 1 corresponds to a piecewise parabolic curve, and is an approximation of the sigmoid function.

3.2 Textual Match

Textual match $T(o.d, q.d)$ is computed as follows:

$$T(o.d, q.d) = \frac{\sum\limits_{i \in o.d \text{ and } q.d} \mathit{tfidf}(o.d.i, o.d)\mathit{tfidf}(q.d.i, q.d)}{\sqrt{\sum\limits_{i=1}^{|o.d|} \mathit{tfidf}(o.d.i, o.d)^2}\sqrt{\sum\limits_{i=1}^{|q.d|} \mathit{tfidf}(q.d.i, q.d)^2}} \tag{2}$$

where i refers to a specific term (e.g. a word) in either the document $o.d$ or the query $q.d$, and the notation $o.d.i$, $q.d.i$ refers to a term i in either the document text $o.d$ or the query text $q.d$, respectively. Equation 2 is called the vector space model, and was introduced by Salton et al. [14] for use in information retrieval. For a given term τ and document d, the term frequency inverse document frequency (tf-idf) weight $\mathit{tfidf}(\tau, d)$ is computed as

$$\mathit{tfidf}(\tau, d) = \frac{f(\tau, d)}{|d|} \log \frac{|O|}{|\{d' \in O | \tau \in d'\}|} \tag{3}$$

where O is the complete set of documents being indexed, $f(\tau, d)$ is the number of times term τ appears in document d, $|d|$ is the

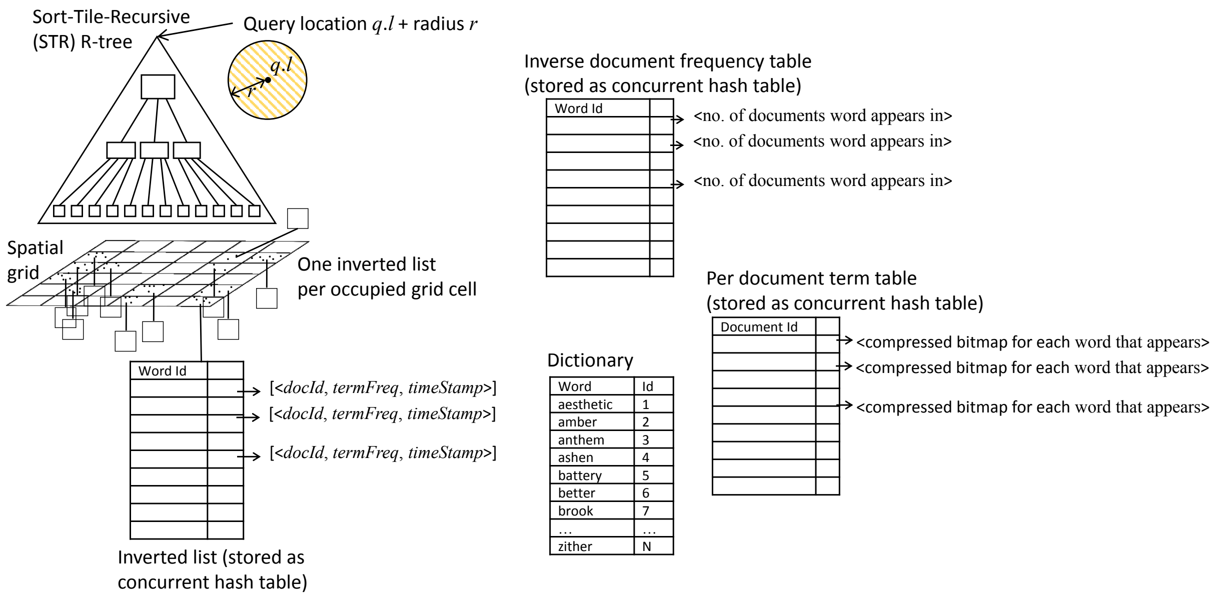
Sort-Tile-Recursive (STR) R-tree
Query location $q.l$ + radius r
$q.l$
r
Spatial grid
One inverted list per occupied grid cell
Word Id
[<docId, termFreq, timeStamp>]
[<docId, termFreq, timeStamp>]
[<docId, termFreq, timeStamp>]
Inverted list (stored as concurrent hash table)
Inverse document frequency table (stored as concurrent hash table)
Word Id
<no. of documents word appears in>
<no. of documents word appears in>
<no. of documents word appears in>
Dictionary
Word
Id
aesthetic 1
amber 2
anthem 3
ashen 4
battery 5
better 6
brook 7
… …
zither N
Per document term table (stored as concurrent hash table)
Document Id
<compressed bitmap for each word that appears>
<compressed bitmap for each word that appears>
<compressed bitmap for each word that appears>

Figure 1: Structure of STARI index

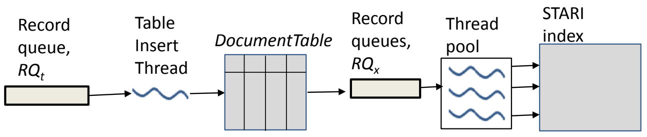

Figure 2: Handling document updates

number of terms in d, and $|O|$ is the size of set O. The log term on the right of equation 3 is the inverse document frequency; its denominator $|\{d' \in O|\tau \in d'\}|$ is the number of documents in O in which term τ appears.

3.3 Dynamic Ranking

We introduce the half-life decay parameter $H(t)$ defined as

$$H(t) = e^{-\lambda t}, \tag{4}$$

where $\lambda > 0$ is a decay constant, e is the base of the natural logarithms, t is time and $H(t) \in (0, 1]$. The decay constant λ is related to the half-life $t_{1/2}$ as follows:

$$\lambda = \frac{\ln(2)}{t_{1/2}} \tag{5}$$

Half-life $t_{1/2}$ defines the time period during which an object loses $1/2$ of its original quantity.

We adapt the ranking scheme of Choudhury et al [3] to include a factor depending on the time t_o the object was entered into the database. Objects $o \in O$ in a database matching a spatial-textual query q are ranked as follows:

$$R(t, o, q) = \alpha S(o.l, q.l) + \frac{1}{H(t - t_o)}(1 - \alpha)(1 - T(o.d, q.d)), \tag{6}$$

if $H(t - t_o) \leq 1$, where $S(o.l, q.l) \in [0, 1]$ is the spatial match between o and q, $o.l$ and $q.l$ are the spatial locations of object o and query q, respectively, $T(o.d, q.d)$ is the textual match between the query keywords $q.d$ and the object keywords $o.d$, and $\alpha \in [0, 1]$ is the weight given to the spatial query. Note that objects closely matching the query have a lower rank $R(t, o, q)$. Other textual match models such as the language model [11] could also be used for $T(o.d, q.d)$ in equation 6 as long as their maximum value is 1.

We generally consider t to be the current time, and $t \geq t_o$ always holds. If an object o_i was not yet inserted into the database at time t, then $t < t_o$, so o_i should be discarded as a possible query match.

The term $H(t - t_o)$ is a time-varying half-life decay parameter with t = time of the query. When $t = t_o$, then $H(t - t_o) = H(0) = 1$, and the textual match receives its full weight of $1 - \alpha$. We thus have a dynamic, time-based ranking of an object matching a query. The object's location at the time of insertion into the database does not change, so the time-varying part applies only to the textual component of the rank. The factor $\frac{1}{H(t-t_o)}$ could be applied to both the spatial and textual components if it is known that object locations are time dependent.

The value $R(t, o, q)$ of a database object matching a query based on equation 6 increases as time increases, thus giving a lower rank, and eventually the factor $1/H(t - t_o)$ exceeds 1 as $H(t - t_o)$ decreases. The time at which $1/H(t - t_o)$ exceeds 1 depends on the half-life $t_{1/2}$ chosen to define $H(t)$.

4. SYSTEM ORGANIZATION

In this section we describe the overall system organization, particularly, how document updates are handled. Figure 2 shows the update workflow. When a new geo-tagged document, such as a tweet, $o = (o.id, o.l, o.d, t_o)$ is received by our system, it is enqueued in RQ_t. Once, it is retrieved by the table insert thread from the queue, a document record is created and inserted into in-memory table *DocumentTable*. It has the following schema:

{*DocumentId,Datestamp,Latitude,Longitude,Content*}.

The columns in this table corresponds to the document fields $o.id$, t_o, $o.l.lat$, $o.l.lon$ and $o.d$. The content, $o.d$, of a small document such as a tweet or a Facebook comment, can be stored directly in the *Content* column. For a larger document, the *Content* column can store the pointer to its location in the disk file. The in-memory table can store a certain number of records based on the available RAM. When the age of a document record exceeds a configured threshold, it is moved from the in-memory store to a disk store.

Next the document record is enqueued in one of the queues in RQ_x. An index update thread from the thread pool dequeues the record and updates the data structure in our index STARI. The index update and query execution mechanisms are described in the next section.

5. STARI SPATIO-TEXTUAL INDEX

In this section we describe our spatio-textual index. Figure 1 shows the structure of our index. First we describe the internal data structures of STARI. Then, we describe the update and query processing algorithms.

5.1 Index structure

STARI organizes the spatial domain as a grid (*GRID*) by dividing it into regular sized grid cells. Whenever a new document object is received, the corresponding grid cell (*GCELL*) is determined from its location (i.e. $o.l.lat$, $o.l.lon$). For each *GCELL* entry an inverted list (*ILIST*) is maintained for the documents that are indexed at that grid cell. The *ILIST* is a concurrent hash table, in which the key is a word id and value is a list of tuples. Each word id corresponds to the numeric identifier in the global word dictionary (*DICT*). Each tuple consists of 3 entries: the document id, the term frequency - the number of times a word appears in that document, and the timestamp when the document was generated.

The dictionary, *DICT*, is a global data structure. In addition, there are three global data structures: an inverse document frequency table (*IDFT*), a per document term table (*PDTT*) and a global document counter (*DC*). Any time a new document object is indexed, the counter *DC* is incremented. The *IDFT* keeps track of inverse document frequency for each word i.e. the number of unique documents in which a word appears. The *PDTT* data structure is used to maintain a concise representation of each document. It is a concurrent hash table in which the key is a document id and value is a compressed bitmap (*CBMAP*). A position in the bitmap, p, is set if a word appears in the document whose document id in *DICT* is p. *CBMAP* is compressed to be memory efficient.

To efficiently support circular range search given a query location $q.l$ and radius r, a Sort-tile-recursive R-tree, (*STR*) is maintained. This can used to tightly pack the spatial objects, which results in more efficient spatial search. The grid cells of *GRID* are indexed by *STR*. However, *STR* is solely used for query processing and it is not involved in the document update.

5.2 Update processing

The update processing algorithm enables ingesting and indexing continuously arriving geo-tagged documents. For an incoming document object $o \in O$, a new record id *RID* is obtained when the corresponding record is inserted into *DocumentTable*. Then the record is enqueued in order to be picked up by one of several index updater threads. An adaptive load balancing algorithm is used [12] to ensure that the index updater threads are kept busy and data skew is minimized.

A partial listing of the update processing algorithm is presented in Figure 3. To update the index STARI with a new document object, first its location field $o.l$ is used to determine which grid cell, *GCELL*, it belongs to. Then the corresponding inverted list object, *ILIST*, is updated by registering a new entry for each word in that document (lines 10 to 19). Furthermore, the global data structures *IDFT*, *PDTT* and *DC* are updated (lines 20 and 22).

```
Require: o is a given document object and ILIST is the inverted list of
    the grid cell in which o.l belongs to.
 1: initialize term frequency hash table termFreqHM
 2: for term in o do
 3:     if term exists in termFreqHM.keys() then
 4:         value ← termFreqHM.get(term)
 5:         termFreqHM.put(term, value+1)
 6:     else
 7:         termFreqHM.put(term, 1)
 8:     end if
 9: end for
10: for term in termFreqHM.keys() do
11:     termFreq ← termFreqHM.get(term)
12:     if term exists in ILIST.keys() then
13:         invList ← ILIST.get(term)
14:     else
15:         instantiate a new invList
16:     end if
17:     invListEntry ← (o.id, termFreq, $t_0$)
18:     add invListEntry to invList
19:     ILIST.put(term, invListEntry)
20:     IDFT.update(term)
21: end for
22: PDTT.update(o)
```

Figure 3: Algorithm ProcessUpdate

```
Require: q is a given query and RCells are the grid cells returned by STR
    that are within r from which q.l. Priority queue PQueue stores the
    matched results. We are interested to find top k objects
 1: for cell2process in RCells do
 2:     ILIST ← cell2process.invertedList()
 3:     initialize perDocTextualScore
 4:     for term in q.d do
 5:         if term exists in ILIST.keys() then
 6:             invList ← ILIST.get(term)
 7:             for (o.id, termFreq, $t_0$) in invList do
 8:                 cbitmap ← PDTT.getbitmap(o.id)
 9:                 if matches(cbitmap,term) then
10:                     textRelScore ← calculate textual relevance score per
                        Equation 2
11:                     perDocTextualScore.update(o.id,textRelScore)
12:                 end if
13:             end for
14:         end if
15:     end for
16:     for o in perDocTextualScore.keys() do
17:         textualRelScore ← perDocTextualScore.getScore(o)
18:         spatialRelScore ← calculate spatial relevance score per Equa-
            tion 1
19:         combinedScore ← calculate combined relevance score per
            Equation 6
20:         topKthScore ← PQueue.getKthObjCombinedScore(k)
21:         if topKthScore < combinedScore then
22:             PQueue.add(o.id,combinedScore)
23:         end if
24:     end for
25: end for
```

Figure 4: Algorithm ProcessQuery

5.3 Query processing

The goal of our system is to support top-k spatial keyword search over dynamically ranked streaming documents. A partial listing of the query processing algorithm is shown in Figure 4. For a spatial-textual query q, the *STR* is used to perform a circular range search in which the point of origin is $q.l$ and the radius r (here, r is configurable). The *STR* returns a list of grid cells that are within distance r from $q.l$. For each cell, the corresponding inverted list *ILIST* is consulted to compute the ranking. This involves first computing the textual relevance score for each document that has a match for one of the query keywords (lines 3 to 15). Then for each of those documents the spatial relevance score and the combined score is calculated (lines 16 to 19). A priority queue $PQueue$ is used to maintain the current top-k documents. If the calculated combined

Table 1: Trace file details

Dataset name	Num. of objects
20k	200,000
2mi	2,000,000
20mi	20,000,000

Table 2: Parameter settings

Parameter	Settings
Grid resolution	64, 256, **1024**, 4096
Updates (num. document objects)	200 thousand, **2 million**, 20 million
Number of queries	1000
k	5
Number of query keywords	5
Half-life, $t_{1/2}$	7 days
Query selectivity	5%, 10%, **20%**

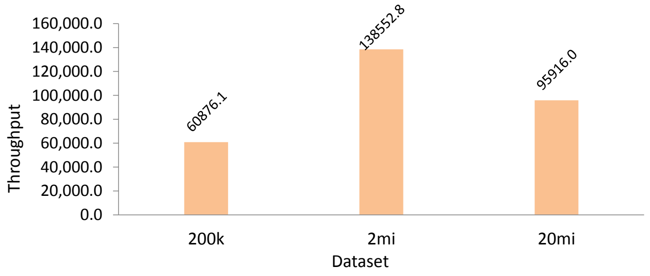

Figure 5: Average update throughput

score of a document is less than the score of the k-th document in $PQueue$, it is enqueued (lines 21 to 23).

6. EVALUATION

In this section we evaluate STARI in various settings. We first describe the datasets and the settings used in our experiments.

6.1 Experimental setup

We use a real-world spatio-textual dataset: TWITTER. The tweets were geo-tagged by the authors of [1] using a real road network dataset [6]. Two additional datasets were generated to conduct scalability experiments. The three datasets contain 200 thousand, 2 million and 20 million tweets respectively. A summary of the datasets is presented in Table 1. In order to simulate live streaming of the data, we implemented a driver program that pushed the tweets into an incoming queue from where our system picked up the data objects.

The query set consisted of 1000 queries and they were freshly generated before each new experiment run. The queries were generated based on the TWITTER dataset. The query keywords were selected from the dictionary generated by preprocessing the entire dataset. An important parameter used during the query generation was selectivity. For example, a 5% selectivity implies that there is a 5% chance that the current query contains all its keywords from a known data object in the dataset and its location of search (origin) is near to that document's location. Otherwise there is a 95% chance that the query location will be random and the query keywords are selected randomly from the dictionary.

The experiments were conducted on a machine having 16 GB memory, eight 2.6 GHz Intel Core I7 processors, and an 700 GB 7200-RPM SATA disk. We run Ubuntu 14.04 LTS 64-bit as the OS. Some of the key parameters are shown in Table 2.

6.2 Update Performance

The update task involves inserting a new document into the data

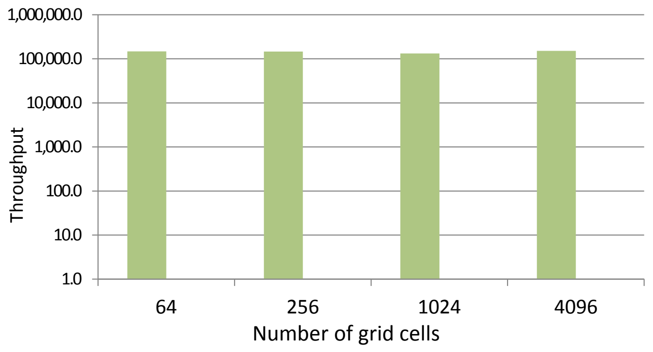

Figure 6: Average update throughput

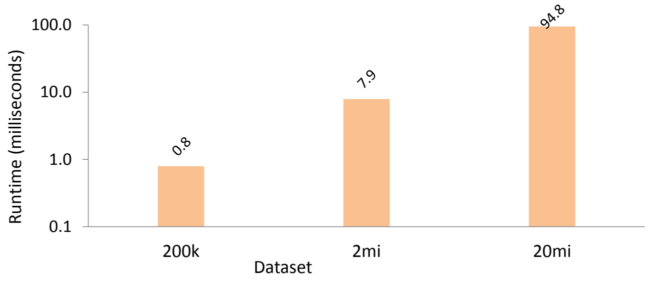

Figure 7: Average query latency

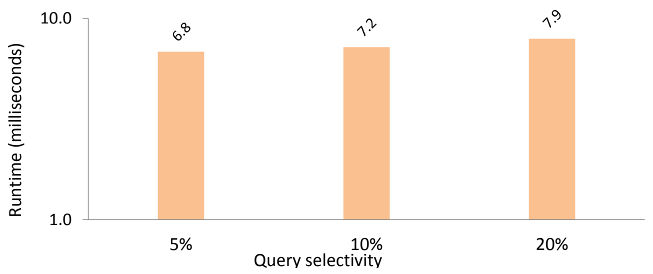

Figure 8: Average query latency

table *DocumentTable* and updating the index STARI. We evaluate the update performance of our system in this section. We used the above-mentioned three datasets (200k, 2mi and 20mi). In our experimental setup we utilized 1 dedicated thread to handle inserts into the *DocumentTable* and 4 threads to update STARI. For all experiments we used a driver program to stream data from the three datasets into STARI.

The update throughputs of STARI with the three datasets is shown in Figure 5. The best throughput is achieved with dataset 2mi, which is over 138 thousand document objects per second. Even with the largest dataset 20mi, this throughput is close to 100 thousand document objects per second. With the smallest dataset 200k, the throughput is low because it does not saturate the machine capacity.

6.3 Handling skew

Geo-tagged streaming text data, such as microblogs, are generated from various parts of the world, and more data is generated where population density is high. So, there is a significant skew in terms of source of data.

In Figure 6 we plot the update throughput of STARI for the dataset 2mi while varying the grid resolutions to 64, 256, 1024 and 4096 cells. As shown, the throughput remains relatively stable regardless of the grid resolution. This demonstrates that STARI's load-balancing algorithm does a fine job of handling data skew.

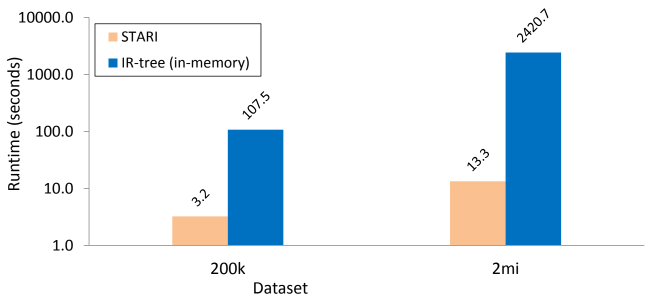

Figure 9: Total index building time (seconds): IR-tree (in-memory) vs. STARI

6.4 Query performance

To evaluate query performance we use the parameters specified in Table 2 for the three datasets. The key feature of our system is that queries can be executed concurrently with the updates. The average query latencies for these datasets are shown in Figure 7. For the dataset 200k, this latency is just 0.8 milliseconds. Even with the largest dataset 20mi, this is quite low (94.8 milliseconds).

Next we vary the selectivity of the queries. The higher the selectivity, the more processing cost is involved. We report the average query latencies while using 5%, 10% and 20%. For this experiment the dataset 2mi was used. As can be seen, a change in selectivity from 5% to 20% only causes the average latency to increase from 6.8 milliseconds to 7.9 milliseconds.

6.5 Comparison with IR-tree

Since STARI uses adaptive ranking, it is not possible to do a direct comparison with an existing spatio-textual index. However, we wanted to demonstrate that STARI's index building time is fast enough to support real-time updates, whereas existing indexing approaches are not. STARI is an in-memory approach and so to do a fair comparison we used IR-tree [5] in an in-memory setup. IR-tree is a popular spatio-textual index for top-k spatial keyword search. We constructed IR-tree index with the three datasets (200k, 2mi and 20mi) and observed the total index build time. With STARI, we streamed records using a driver program from the same three datasets and reported the time to completely ingest all the records.

With dataset 20mi, the IR-tree index building process did not complete, as it needed more memory than the available memory (RAM) in the machine. So, we removed the results for dataset 20mi and present the results for datasets 200k and 2mi in Figure 9. As can be seen in this figure, our approach takes 2 orders of magnitude less time than that of an IR-tree in an in-memory setup. For instance, with dataset 2mi STARI took 13.2 seconds and IR-tree took 2420.7 seconds. Moreover, IR-tree index building process involves several steps. The first step requires the entire dataset and each step must be completed before the next step can begin. Therefore, existing spatio-textual indexing approaches, such as IR-tree, may not be suitable for real-time streaming applications.

7. CONCLUSION

Due to the rapid growth of spatial-textual, top-k spatial keyword queries are becoming increasingly important. The "recency" of the documents in the search results is very important, because we tend to value most recent objects, such as "hot new" deals. However, existing spatio-textual indexes that support top-k spatial keyword queries do not consider the age of the documents.

To address the mentioned challenges, we have presented a novel spatio-textual index, STARI, that retrieves most recent documents. It uses a dynamic ranking scheme to calculate textual relevance measure that enables document relevance to decrease over time. Our system uses a grid-based inverted index to support live document updates and a compact R-tree based index to perform spatial search queries. With a real-world dataset we demonstrate that our system scales well.

8. ACKNOWLEDGEMENTS

This research was supported in part by Natural Sciences and Engineering Research (NSERC) Discovery Grants (no. 36866-2011-RGPIN and RGPIN-2016-03787) and NBIF Start-Up Grant (RIF 2016-008).

9. REFERENCES

[1] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. In *PVLDB*, pages 217–228, 2013.

[2] S. Cheng, A. Arvanitis, M. Chrobak, and V. Hristidis. Multi-Query Diversification in Microblogging Posts. In *EDBT*, pages 133–144, 2014.

[3] F. M. Choudhury, J. S. Culpepper, and T. Sellis. Batch Processing of Top-k Spatial-textual Queries. In *GeoRich*, pages 7–12, 2015.

[4] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. Space: Efficient Geo-search Query Processing. In *CIKM*, pages 423–432, 2011.

[5] G. Cong, C. S. Jensen, and D. Wu. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *VLDB*, 2(1):337–348, 2009.

[6] DIMACS Implementation Challenge - Challenge benchmarks. http://www.dis.uniroma1.it/challenge9/download.shtml.

[7] L. Guo, J. Shao, H. H. Aung, and K.-L. Tan. Efficient continuous top-k spatial keyword queries on road networks. *GeoInformatica*, 19(1):29–60, 2014.

[8] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search in Relational Databases. In *SIGMOD*, pages 563–574, 2006.

[9] A. Magdy, M. F. Mokbel, S. Elnikety, S. Nath, and Y. He. Mercury: A memory-constrained spatio-temporal real-time search on microblogs. In *ICDE*, pages 172–183, 2014.

[10] A. Markowetz, Y. Yang, and D. Papadias. Keyword Search on Relational Data Streams. In *SIGMOD*, pages 605–616, 2007.

[11] J. M. Ponte and W. B. Croft. A Language Modeling Approach to Information Retrieval. In *SIGIR*, pages 275–281, 1998.

[12] S. Ray, R. Blanco, and A. K. Goel. Supporting Location-Based Services in a Main-Memory Database. In *MDM*, pages 3–12, 2014.

[13] J. a. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient Processing of Top-k Spatial Keyword Queries. In *SSTD*, pages 205–222, 2011.

[14] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620, 1975.

[15] H. Yan, S. Ding, and T. Suel. Inverted Index Compression and Query Processing with Optimized Document Ordering. In *WWW*, pages 401–410, 2009.

[16] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable Top-k Spatial Keyword Search. In *EDBT*, pages 359–370, 2013.