SPATIO-TEMPORAL KEYWORDS QUERIES IN HBASE

XIAOYING CHEN, CHONG ZHANG, ZONGLIN SHI AND WEIDONG XIAO

Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology
Changsha 410073, China

ABSTRACT. With the amount of data accumulated to tens of billions of scale, HBase, a distributed key-value database, plays a significant role in providing effective and high-throughput data service and management. However, for the applications involving spatio-temporal data, there is no good solution, due to inefficient query processing in HBase. In this paper, we propose spatio-temporal keyword searching problem for HBase, which is a meaningful issue in real life and a new challenge in this platform. To solve this problem, a novel access model for HBase is designed, containing row keys for indexing spatio-temporal dimensions and Bloom filters for fast detecting the existence of query keywords. And then, two algorithms for spatio-temporal keyword queries are developed, one is suitable for the queries with ordinary selectivity, the other is a parallel algorithm based on MapReduce aiming for the large range queries. We evaluate our algorithms on a real dataset, and the empirical results show that they are capable to handle spatio-temporal keyword queries efficiently.

1. **Introduction.** With the development of wireless communication and positioning technology, more and more data about locations are collected and used for many applications, such as location-based services. In particular, the format of the data could be generalized to (*location*, *time*, *texts*), where the *location* represents the geo-coordinates where event happens (or some object lies), and *time* means the valid time when the event happens at *location* (or the object is at *location*), and *texts* are used to describe or record the state of the event (or object). For instance, someone publishes a micro-blog associated with geo-location tag at 6:30 pm, in which some comments about a restaurant are attached. Another example about real-estate advertising is that every advertisement contains geo-coordinates of the property, the publishing time and texts describing the property. Users are often interested in the spatial, temporal and textual attributes of the data, and retrieve useful information through the combination of these three dimensions. Such queries are called spatio-temporal keyword (STK) queries. For instance, find the users publishing micro-blogs containing words *pasta* and *pizza* in the range of this shopping mall yesterday.

However, tens of billions of spatio-temporal data would be a serious problem for applications. HBase [1] is a distributed non-sql, key-value database, and is capable to store such a huge amount of data. However, due to the non-sql characteristics of HBase, it is inefficient to process spatio-temporal keyword queries using the

built-in functions of HBase. Consequently, it is highly necessary to design efficient algorithms for processing spatio-temporal keyword queries in HBase. A previous work [9] about processing spatial or multi-attributes queries in HBase is different from ours, because they merely consider spatial dimension in query processing. To the best of our knowledge, this is the first work for processing spatio-temporal keyword queries in HBase.

Our motivation is to adopt HBase to process STK queries efficiently. First, a space filling curve, Hilbert curve [8], is used to encode the spatial dimension into one-dimensional codes, i.e., for a location referred by geo-coordinates, it is represented by a number denoting the relative position in the original space. Based on this, we design a suitable access model for HBase, containing row keys for indexing spatio-temporal dimensions and Bloom filters [3] for fast detecting the existence of query keywords. After that, we develop two algorithms, one is suitable for the query with ordinary selectivity, the other is a parallel algorithm for the large range query. We evaluate our algorithms on a real dataset, and the results show that our algorithms are capable to handle spatio-temporal keyword queries efficiently. In summary, we make the following contributions:

- We propose spatio-temporal keyword queries processing problem in HBase, which is a new challenge for HBase platform.
- We design a novel access model for HBase to store spatial, temproal and textual information.
- We propose efficient algorithms for processing spatio-temporal keyword queries in HBase.

The rest of this paper is organized as follows. Section 2 reviews related works on spatio-temporal queries and HBase queries. Section 3 formally defines spatio-temporal keyword problem. Section 4 presents an access model for HBase. Algorithms for spatio-temporal keyword queries are presented in section 5. And we experimentally evaluate our algorithms in section 6. Finally, section 7 concludes the paper with directions for future works.

2. **Related work.** To our knowledge, the state of the art for spatio-temporal keyword queries in HBase is less well studied. However, some researches focusing on distributed index could be referenced. As an attractive choice for large-scale data processing, Cloud storage system currently adopts a hash-like approach to retrieve data that only support simple keyword-based queries, but lacks various forms of information search. To overcome this disadvantage, Zhou et al. [10] propose a novel SkipNet and B^+-tree based index structure, called SNB-index. SNB-index uses the B^+-tree to construct efficient local index in the lower layer, while it selects local index nodes to form a SkipNet-based global overlay in the upper layer. SNB-index also can be employed in systems such as GFS and Hadoop [2], which ensures it scalable and flexible. Experimental results show that SNB-index is valid and can be an alternative approach for constructing an auxiliary index in Cloud computing systems.

For supporting similarity search in cloud system, Cheng et al. [4] propose VF-CAN, a novel indexing scheme, which integrates content addressable network (CAN) based routing protocol and the improved vector approximation file (VA-file) index. There are two index levels in this scheme: global index and local index. In the local index, VA-file approximation vectors are clustered by k-means according to their degree of proximity. In the global index, storage nodes are organized into an

overlay network CAN, and only clustering information of local index is issued to the entire overlay network through the CAN interface. The experimental results show that VF-CAN reduces the index storage space and improves query performance effectively.

For multi-dimensional data applications, Nishimura et al. [9] proposes MD-HBase, a scalable multi-dimensional data store supporting efficient multi-dimensional range and nearest neighbor queries. MD-HBase layers a multi-dimensional index structure over a range partitioned key-value store. Using a design based on linearization, its implementation layers standard index structures like K-D trees and Quad trees.

As we have mentioned above, many applications not only require finding objects closest to a specified location, but also that containing a set of keywords. Felipe et al. [6] present an efficient method to answer top-k spatial keyword queries. It adopts a structure called IR^2-Tree (Information Retrieval R-Tree) which combines an R-Tree with superimposed text signatures. Algorithms are proposed to construct and maintain an IR^2-Tree, and answer top-k spatial keyword queries. With experimentally comparison with current methods, superior performance and excellent scalability of the algorithms are shown.

Cong et al. [5] propose a novel indexing framework for location top-k text retrieval. It integrates the inverted file for text retrieval and the R-tree for spatial proximity query. Several hybrid indexing approaches are explored within the framework. Algorithms utilize the proposed indexes to compute the top-k query by taking into account both text relevancy and location proximity to prune the search space.

In practical applications, it is need to consider the nature of the movement of objects. Traditional indexes have good query performance but can not handle this data processing problem in that they are not efficient for update which is crucial for an index for moving objects, as they change their position frequently. Jensen et al. [7] represent moving-object locations as vectors that are time stamped based on their update time. Then B^+-tree based data structure is adopted to index moving objects according to their time stamp and otherwise preserves spatial proximity. This method supports range and nearest neighbor queries, as well as continuous queries. B^x-tree which is based on the B^+-tree, outperforms the TPR-tree in range queries and kNN queries concerning the current or near-future positions of the objects.

3. **Problem definition.** In this section, we first present the definitions for spatio-temporal keyword data and queries, and then describe the HBase logical table. For simplicity, only two-dimensional space is considered in this paper, however, our method can be directly extended into higher dimensional space.

Spatio-temporal data. A record r of spatio-temporal data can be denoted as $\langle x, y, t, W, A \rangle$, where (x, y) means the geo-location of the record, t means the valid time when the data is produced, W is a set of keywords, $W=\{w_1, w_2, \ldots, w_n\}$, A represents other attributes, such as user-id, object's shape, etc.

Spatio-temporal keyword query. Given a set U_{st} of records of spatio-temporal data, a spatio-temporal keyword (STK) query $Q=\langle R_q, t_s, t_e, W_q \rangle$, where $R_q=(x_l, y_l, x_u, y_u)$ means a query range with lower-left coordinate (x_l, y_l) and upper-right coordinate (x_u, y_u), $W_q=\{w_{q_1}, w_{q_2}, \ldots, w_{q_m}\}$ is a set of query keywords, aims to find a subset $S_{st}=\{r_s \mid r_s \in U_{st}\}$ satisfying that:

$$r_s.(x, y) \in R_q$$

$$r_s.t \in [t_s, t_e]$$

$$r_s.W \cap W_q \neq \phi$$

Logical view of HBase table. Without loss of generality, we give the descriptions for HBase table. HBase is a distributed key-value database which consists of a number of computing nodes cooperatively processing large-scale data. A physical table in HBase is partitioned into several regions each of which is maintained by a node. From the logical view, a table is similar to a grid, where a cell can be located by the given row identifier and column identifier. Row identifiers are implemented by row keys (rk) which are index by B$^+$-tree, and the column identifier is represented by column family (cf) + column key (ck), where a column family consists of several column keys. The value in a cell can be referred to as the format (rk, $cf : ck$). Table 1 shows a logical view of a table in HBase. For instance, value v_1 can be referred to as (rk_1, cf_1:ck_1).

TABLE 1. **A HBase Logical Table**

	cf_1			cf_2	
	ck_1	ck_2	ck_3	ck_a	ck_b
rk_1	v_1	v_2	v_3	v_4	v_5
rk_2	v_6	v_7	v_8	v_9	v_{10}

4. **Spatio-temporal keyword access model for HBase.** In this section, we describe a spatio-temporal keyword access (STKA) model for the logical view of HBase table. The model is built based on Hilbert curve and Bloom filter. We first introduce Hilbert curve, which maps high-dimensional data into one-dimensional space and then describe the access model.

4.1. **Hilbert curve.** Hilbert curve is a kind of space filling curve which maps multi-dimensional space into one-dimensional space. In particular, the whole space is partitioned into equal-size cells and then a curve is passed through each cell for only once in term of some sequence, so that every cell is assigned a sequence number. Different space filling curves are distinguished by different sequencing methods. Due to information loss in the transformation, different space filling curves are evaluated by the criteria, locality preservation, meaning that how much the change of proximities is from original space to one-dimensional space. Hilbert curve is proved to be the best locality preserved space filling curve. With Hilbert curve, any object in the original space is transformed into $[0, 2^{2\lambda} - 1]$ space, where λ is called the order of Hilbert curve. Figure 1 shows four Hilbert curves in two-dimensional space with λ=1, 2, 3 and 4. In Figure 1 (b), there are two points p_1 and p_2, and the sequence number 2 and 13 is obtained through the curve for them, respectively. Note that p_1 and p_2 are neighbors in the original space, but apart from each other by 11 unit distances in one-dimensional space.

We describe two functions for Hilbert curve, one is mapping a point in the original space to a value in one-dimensional space, the other is mapping a range window to a series of intervals. Specifically, for a Hilbert curve with order=λ,

- *coorToCode*(p). Given a point p=(x_1, x_2, ..., x_n) in n-dimensional space $\mathbb{S}$, *coorToCode*(p) returns a cell number (between 0 and $2^{2\lambda} - 1$) referring the cell where p lies within $\mathbb{S}$.
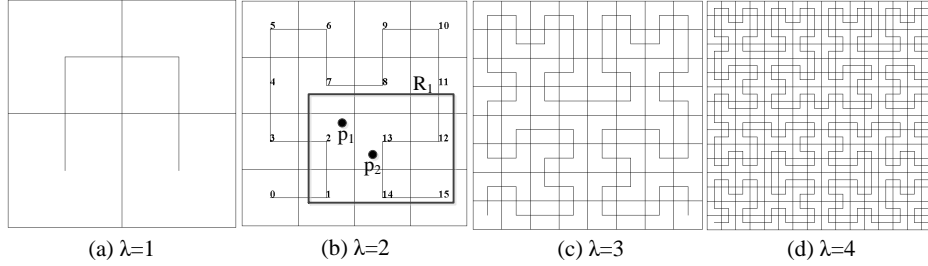
(a) λ=1 (b) λ=2 (c) λ=3 (d) λ=4

FIGURE 1. Hilbert Curves

- $rectToIntervals(R)$. Given a range window $R=(x_1^l, x_2^l, \ldots, x_n^l, x_1^u, x_2^u, \ldots, x_n^u)$ in n-dimensional space $\mathbb{S}$, where x_i^l and x_i^u $(1 \le i \le n)$ are the lower and upper bound of the ith-dimension, respectively, $rectToIntervals(R)$ returns a series of intervals representing the cells intersecting with R in $\mathbb{S}$.

For instance, in Figure 1 (b), $coorToCode(p_1) = 2$, $coorToCode(p_2) = 13$, and $rectToIntervals(R_1) = \{[1,2], [7,8], [11,15]\}$.

4.2. **STKA model.** For each record $r=\langle x, y, t, W, A\rangle$, the row key is generated as $t \oplus coorToCode(x,y)$. Note that for different records, they may share the same row key, which means a row key might correspond to several records. And we use Bloom filter to denote the union of keyword sets of records. A Bloom filter is a space-efficient probabilistic data structure which is used to test whether an element is a member of a set. If the returned value is false for testing whether element e belongs to set C, then it is definite that e is not in C. However, if the result is true, e may be in C, which means false-positives are possible and further inspection should be taken. In particular, for a set of records $S=\{r_1, r_2, \ldots, r_m\}$, the Bloom filter $B_S=addSet(\{w_{i,j} \mid w_{i,j} \in W_i, 1 \le j \le |W_i|, 1 \le i \le m\})$, where function $addset()$ means inserting keyword list into Bloom filter structure, and W_i is the keyword set of r_i $(1 \le i \le m)$.

The STKA model is $\langle rowkey, B, listof(x,y,W,A)\rangle$, where $rowkey$ identifies a set of records S, each of which has $t \oplus coorToCode(x,y)=rowkey$, and B is the Bloom filter of union of the keywords in S, and $listof(x,y,W,A)$ is a list, each element in which contains geo-location, keyword set, and other attributes of each records in S.

The STKA model is suitable for the logical view of HBase table. Table 2 shows an example of the model. For storing $listof(x,y,W,A)$, multiple column groups (loc, $wordlist$, $attr$) are used.

TABLE 2. **STKA model for HBase**

		stdata						
	bf	loc_1	$wordlist_1$	$attr_1$	loc_2	$wordlist_2$	$attr_2$	...
$rowkey_1$	B_1	(x_1,y_1)	W_1	A_1	(x_2,y_2)	W_2	A_2	...
...	...	...	...	...	...	...	...	...
$rowkey_k$	B_k	(x_m,y_m)	W_m	A_m	(x_n,y_n)	W_n	A_n	...

5. **Algorithms for STK queries.** In this section, we first describe the processing for STK queries on original model. Considering massive data processing, we use MapReduce [2] framework to design parallel algorithm for STK queries.

5.1. **STK queries.** For a STK query $Q=\langle R_q, t_s, t_e, W_q \rangle$, where $R_q=(x_l,\ y_l,\ x_u,\ y_u)$ means a query range, $W_q=\{w_{q_1},\ w_{q_2},\ \ldots,\ w_{q_m}\}$ is a set of query keywords, the basic idea for spatio-temporal range queries processing is to transform the spatial range and temproal range into a series of row keys, and using these row keys to retrieve the records and examin them whether they satisfy the query condition. To be more specific, firstly, the query spatial range is converted into a series of Hilbert value intervals. Then, according to row key design patterns, row keys are generated by time concatenating the Hilbert value. After that, we use *scan*, an interface of HBase, to retrieve the records within the row key range, and then we use Bloom filter to test whether these records intersect with the query keywords. Finally, to avoid false-positive, those satisfied records in the previous step must be further examined. Algorithm 1 describes the processing for STK queries in detail.

For the sake of legibility, some functions and variables of the algorithm are explained below:

1. $rectToIntervals(R)$ returns a series of intervals, each of which is denoted as $[Henter, Hexit]$;
2. $connectHTable()$ connects to HBase and returns the table;
3. $rowkeyRange$ is a range of row keys, represented by a starting row key and an ending row key;
4. $ResultScanner$ is a collection of rows retrieved by scanning table, whose row keys are in $rowkeyRange$;
5. $bf.intersect(W_q)$ is the function of Bloom filter bf to test whether bf intersects with W_q;
6. $getNextSTRecord()$ returns next spatio-temporal record in a row;
7. $Qlist$ is the result list for the query.

In line 1, R_q is transformed into a series of intervals. From line 3 to 20, the involved rows are retrieved and examined. In line 9, a collection of rows $ResultScanner$ is retrieved, corresponding to a time stamp in (t_s, t_e) and an area in R_q. The Bloom filter of the union of keywords in each row is examined whether it intersects with W_q in line 11. If the intersection is not null, every spatio-temporal record in the row is further examined on location and word list (from line 12 to 16). For the satisfied record, the result is added into $Qlist$ (line 14).

5.2. **Parallel STK queries.** As we have mentioned above, with the development of wireless communication and positioning technology, more and more data about locations are collected, so it is necessary to consider the problems of massive data processing. For a STK query with large range predicates, a great number of rows are retrieved, which is a bottleneck of performance. Consequently, it is necessary to design parallel algorithm which involved computing nodes to process the query simultaneously, so that throughput and efficiency increase. Based on MapReduce framework, we design a parallel algorithm for STK queries.

The basic work flow is that, firstly, an intermediate file containing all the involved row key ranges for the query is generated, in which each record corresponds to a row key range in HBase. Then Map procedure reads the records in the file and outputs them to Reduce procedure. And then in the Reduce, corresponding nodes

---

Algorithm 1 Spatio-temporal Keyword Query

Input:
    (t_s, t_e) //temporal range of query
    $R_q(x_l, y_l, x_u, y_u)$ //spatial range of query
    W_q //query keywords
Output:
    $Qlist$ //result list
1: $intervals = rectToIntervals(R_q)$;
2: $table = connectHTable()$; //connect to table in HBase for query
3: **while** $t_s < t_e$ **do**
4:     **for** each $interval$ in $intervals$ **do**
5:       $startrowkey = t_s + interval.Henter$; //starting row key
6:       $endrowkey = t_s + interval.Hexit$; //ending row key
7:       $rowkeyRange = (startrowkey, endrowkey)$; //row key range for scanning
8:       $Scan = scan(rowkeyRange)$; //scan table
9:       $ResultScanner = table.getScanner(Scan)$; //results of scanning
10:      **for** each $Result$ in $ResultScanner$ **do**
11:        **if** $Result.bf.intersect(W_q) \neq null$ **then**
12:          **while** $(str = Result.getNextSTRecord()) \neq null$ **do**
13:            **if** $(str.loc \in R_q)\&\&(str.wordlist \cap W_q \neq \phi)$ **then**
14:              $Qlist.add(str)$; //add result to query result list
15:            **end if**
16:          **end while**
17:        **end if**
18:      **end for**
19:     **end for**
20:     $t_s = t_s + 1$;
21: **end while**

---

retrieve and examine rows simultaneously to get results. Algorithm 2 describes the parallel algorithm for STK queries based on MapReduce framework.

Pseudo-codes between line 2 and 11 describe the generation for the intermediate file *Filerecord*. In line 8, the row key range is written into the file in terms of file record in the loops in line 3 and 4. Map procedure is showed between line 13 and 15, and *context.write(record)* means outputting *record* to Reduce step. Pseudo-codes between line 2 and line 11 describe Reduce procedure. In line 17, the record passed by Map is parsed into the row key range. The remaining codes are similar to Algorithm 1.

6. **Experimental evaluation.** We evaluate our algorithms on a real dataset, which contains trajectories of taxis in Beijing. In particular, each record in the dataset contains vehicle ID, geo-location, recording time stamp, etc. For each record, we randomly assign it a list of keywords with size varied from 7 to 15 words. For comparison, we extract 5 datasets in different sizes from the original one in terms of temporal range. Table 3 shows the datasets in detail.

Our algorithms are implemented in Java, and run on a three-node cluster with Hadoop 2.5.1 and HBase 0.98.6, in which each node is equipped with Intel(R) Core(TM) i3 CPU @ 3.40GHz, 4GB main memory, and 500GB storage, and operating system is CentOS release 6.5 64bit, and network bandwidth is 10Mbps.

6.1. **STK queries.** First, we evaluate the algorithm for STK queries. And we introduce two parameters to test the algorithm under various conditions. One is

---
Algorithm 2 Parallel Spatio-temporal Keyword Query

Input:
    (t_s, t_e) //temporal range of query
    $R_q(x_l, y_l, x_u, y_u)$ //spatial range of query
    W_q //query keywords
Output:
    $Qlist$ //result list
1: /*generating intermediate file $Filerecord$ for Map and Reduce*/
2: $intervals = rectToIntervals(R_q)$;
3: **while** $t_s < t_e$ **do**
4:     **for** each $interval$ in $intervals$ **do**
5:         $startrowkey = t_s + interval.Henter$; //starting row key
6:         $endrowkey = t_s + interval.Hexit$; //ending row key
7:         $rowkeyRange = (startrowkey, endrowkey)$; //row key range for scanning
8:         $Filerecord.write(rowkeyRange)$; //write $rowkeyRange$ range to $Filerecord$
9:     **end for**
10:     $t_s = t_s + 1$;
11: **end while**
12: /*Map input: file $Filerecord$ */
13: **while** $(record = Filerecord.getNextRecord()) \neq null$ **do**
14:     $context.write(record)$; //output the record
15: **end while**
16: /*Reduce input: output of Map */
17: $rowkeyRange = parse(record)$; //parse record into $rowkey$ range
18: $table = connectHTable()$; //connect to table in HBase for query
19: $Scan = scan(rowkeyRange)$; //start scanning
20: $ResultScanner = table.getScanner(Scan)$; //results of scanning
21: **for** each $Result$ in $ResultScanner$ **do**
22:     **if** $Result.bf.intersect(W_q) \neq null$ **then**
23:         **while** $(str = Result.getNextSTRecord()) \neq null$ **do**
24:             **if** $(str.loc \in R_q) \&\&(str.wordlist \cap W_q \neq \phi)$ **then**
25:                 $Qlist.add(str)$; //add result to query result list
26:             **end if**
27:         **end while**
28:     **end if**
29: **end for**

TABLE 3. **Dataset**

Dataset	Temporal Range	Size (records)
dataset1	0:00-0:48, Nov. 1st	1 million
dataset2	0:00-8:25, Nov. 1st	10 million
dataset3	0:00-23:59, Nov. 1st	30 million
dataset4	Nov. 1st & Nov. 2nd	60 million
dataset5	Nov. 1st, Nov. 2nd & Nov. 3rd	100 million

selectivity θ defined as:

$$\theta = \frac{L_{(t_s,t_e)}}{L_t} \cdot \frac{A_{R_q}}{A_{\mathbb{S}}}$$

where $L_{(t_s,t_e)}$ means the length of query temporal range (t_s, t_e), L_t means the length of temporal extent of the dataset, A_{R_q} means the area of query spatial range R_q, and $A_{\mathbb{S}}$ means the area of the whole space. Selectivity specifies the size of the query range, and the larger θ is, the more spatio-temporal records are involved. The

other parameter is the number of query keywords κ, and a larger κ will recall more records.

Firstly, we fix κ=5(in reality, the number of keywords users usually query is less than 5), and vary θ from 3% to 50%. For each value of θ, we issue 10 queries with different temporal ranges and spatial ranges, and collect the average response time as the measurement of performance. Figure 2 shows the results.
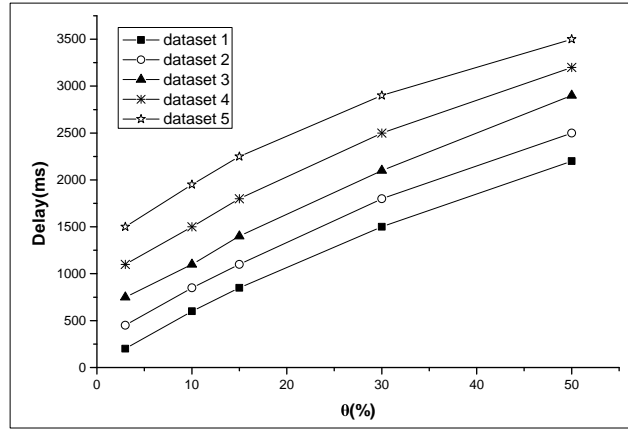

FIGURE 2. Response Time vs. Selectivity (κ=5)

We can see that response time increases with θ for all the datasets. This is because a larger selectivity would access more records to be retrieved and examined, which increases the processing time. Furthermore, the response time grows linearly with θ. This can be explained by the transformation from spatio-temporal dimensions into one-dimensional space, and by the fast detection of Bloom filter for keywords matching. Another observation is that with the enlargement of dataset size(the amount of data from dataset1 to dataset5 increases), the response time also increases. The explanation is that for a fixed selectivity θ, STK query would recall more records with the increase of data amount .

We fix θ=15%, and vary κ from 3 to 10. The average response time is computed similarly to the previous one. Figure 3 shows the results.

Similarly, the response time increases with κ. The explanation is similar to the previous case, i.e., a larger κ would involve more records to be accessed.

6.2. **Parallel STK queries.** After studying the performance of STK queries, we compare parallel STK query with the original one to observe the improvement. For both of them, we issue 5 different queries on dataset 1, and Table 4 shows the conditions in detail.

Figure 4 shows the results. We can see that at the beginning, i.e, the number of scanned records is not much, STK query is faster than the parallel one, because the cost of writing and parsing the intermediate file in the MapReduce procedure impacts the performance. However, with the query range enlarged, the parallel STK query is more efficient than the original one. This can be explained by the parallelism of processing the query, making accessing and comparing to operate
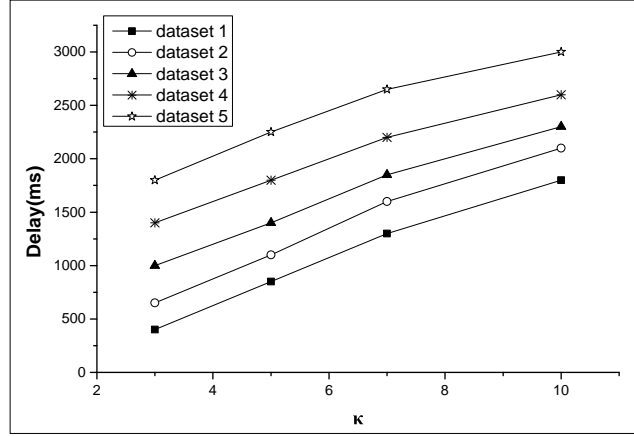
FIGURE 3. Response Time vs. Number of Query Keywords ($\theta=15\%$)

TABLE 4. **Query Conditions**

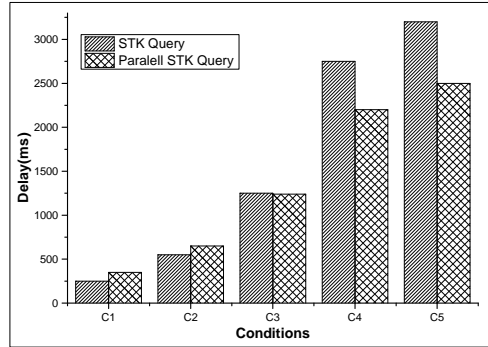| Conditions	Temporal Range(min)	Spatial Range(km)	Number of Keywords	Number of Scanned Records
c1	10	1	3	3987
c2	20	2	5	18786
c3	30	3	5	42457
c4	40	4	7	99789
c5	50	5	10	108040

FIGURE 4. STK Query vs. Parallel STK Query

simultaneously. Therefore, the parallel STK queries are applicable for the condition with large selectivity.

7. **Conclusion.** In this paper, we propose the spatio-temporal keyword queries in HBase, which is a new problem for HBase platform. We devise a proper access

model for HBase, utilized Hilbert curve and Bloom filter. Two algorithms are developed suitable for ordinary and large query selectivities, respectively. We conduct experiments on a real dataset, and the results show our methods are capable for large scale spatio-temporal keyword queries.

In the future, we plan to extend our work to the inner structure of HBase index table, in order to improve efficiency of spatio-temporal keyword queries.

REFERENCES

[1] HBase, 2015. Available from: `http://hbase.apache.org`.
[2] Hadoop, 2015. Available from: `http://hadoop.apache.org`.
[3] J. Blustein and A. El-Maazawi, Bloom filters. a tutorial, analysis, and survey, *Halifax, NS: Dalhousie University*, (2002), 1–31.
[4] C. Cheng, C. Sun, X. Xu and D. Zhang, A multi-dimensional index structure based on improved VA-file and CAN in the cloud, *International Journal of Automation and Computing*, **11** (2014), 109–117.
[5] G. Cong, C. S. Jensen and D. Wu, Efficient retrieval of the top k most relevant spatial web objects, *VLDB Endowment*, **2** (2009), 337–348.
[6] I. D. Felipe, V. Hristidis and N. Rishe, Keyword search on spatial databases, *In ICDE*, (2008), 656–665.
[7] C. S. Jensen, D. Lin and B. C. Ooi, Query and update efficient B$^+$-tree based indexing of moving objects, *VLDB Endowment*, **30** (2004), 768–779.
[8] B. Moon, H. V. Jagadish, C. Faloutsos and J. H. Saltz, Analysis of the clustering properties of the Hilbert space-filling curve, *IEEE Transactions on Knowledge and Data Engineering*, **13** (2001), 124–141.
[9] S. Nishimura, S. Das, D. Agrawal and A. E. Abbadi, MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services, *In MDM*, **1** (2011), 7–16.
[10] W. Zhou, J. Lu, Z. Luan, S. Wang, G. Xue and S. Yao, SNB-index: A SkipNet and B+ tree based auxiliary Cloud index, *Cluster Computing*, **17** (2014), 453–462.

Received May 2015; revised August 2015.

E-mail address: `1473550256@qq.com`
E-mail address: `leocheung8286@yahoo.com`
E-mail address: `997860224@qq.com`
E-mail address: `wilsonshaw@vip.sina.com`