# ST-HBase: A Scalable Data Management System for Massive Geo-tagged Objects

Youzhong Ma, Yu Zhang, and Xiaofeng Meng

School of Information, Renmin University of China, Beijing, China
{mayouzhong,zhangyu1990,xfmeng}@ruc.edu.cn

**Abstract.** In this paper, we propose ST-HBase (spatio-textual HBase) that can deal with large scale geo-tagged objects. ST-HBase can support high insert throughput while providing efficient spatial keyword queries. To the best of our knowledge, the existing approaches that deal with spatial keyword queries mainly focus on the static and medium-sized objects collections and cannot provide high insert throughput. In ST-HBase, we leverage an index module layered over a key-value store. The underlying key-value store enables the system to sustain high insert throughput and deal with large scale data, the index layer can provide efficient spatial keyword query processing. We propose two kinds of index approaches in ST-HBase: Spatial and Textual Based Hybrid Index(ST*b*HI) and Term Cluster Based Inverted Spatial Index(TC*b*ISI) which are suitable for different scenarios. We implement a prototype based on HBase that is a standard open-source key-value store. Finally we perform comprehensive experiments and the results show that ST-HBase has good scalability and outperforms the state-of-the-art approaches in terms of update and query performance.

**Keywords:** Spatial keyword query, Geo-tagged objects, HBase.

## 1 Introduction

With the development of positioning technology and the widely usage of smartphones, many objects on the web are being geo-tagged, such as the images in the Flickr, the tweets from Twitter, many spatial objects are also being associated with text descriptions. The combination of location and text results in a new kind of query: spatial keyword query. A spatial keyword query is like the follows: given a location or location area and a set of keywords, return a set of objects that satisfy the spatial requirements and are relevant to the query keywords. Such as finding the restaurants selling Peking Duck that are within 2 km from my current location or photos near by a given place whose text descriptions are similar to the query keywords.

Spatial keyword query processing has been studied in many literatures, the state-of-the-art approaches mainly employ a hybrid index combing R-tree index with textual inverted index. To the best of our knowledge, the existing R-tree based hybrid index mainly focuses on the static and medium-sized object collection. Although the hybrid index can improve the query performance by utilizing

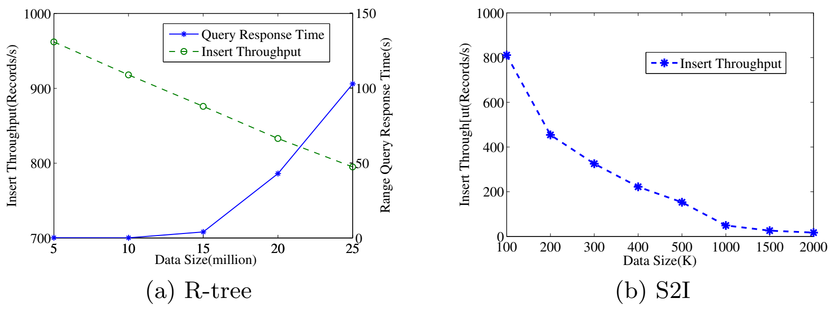

**Fig. 1.** Insert Throughput and Query Performance vs. Data Size

the spatial and textual pruning strategy, when the number of the targeted geo-tagged objects becomes vary large, the overlap between R-tree nodes increases which make the query performance decrease dramatically.

Figure 1 shows the insert throughput and query performance of the R-tree like index at different data size. Figure 1-(a) shows the performance of R-tree index, it just uses R-tree to index the spatial information but neglects the textual information. Figure 1-(b) shows that the performance of the Hybrid index S2I[1]. S2I[1] took both spatial and textual information into consideration and proposed a hybrid index. From Figure 1, we can observe that the insert throughput is very slow and decreases with the data size increasing, because the R-tree like index needs additional cost to adjust the R-tree node and the inverted index file; the response time increases with data size increasing, because the overlap between the R-tree nodes increases as the data size becomes bigger and bigger.

In this paper, we propose spatial textual HBase(ST-HBase) which has good scalability and is capable of dealing with large scale dataset, ST-HBase can also support high insert rates and efficient spatial keyword queries. In ST-HBase, we use HBase to store the geo-tagged objects and combine spatial textual index with HBase, we propose two different approaches: Spatial and Textual Based Hybrid Index(ST*b*HI) and Term Cluster Based Inverted Spatial Index(TC*b*ISI). In summary, the main contributions of the paper are:

– We propose spatial textual HBase(ST-HBase) which is a scalable data management system. It has good scalability and can deal with huge amounts of geo-tagged objects;
– We propose two spatial textual index approaches: Spatial and Textual Based Hybrid Index(ST*b*HI) and term cluster based inverted spatial index(TC*b*ISI). We implement these two index approaches based on HBase;
– We implement a prototype using HBase and perform comprehensive experiments to exploit the scalability and efficiency of ST-HBase.

The rest of the paper is organized as follows. In section 2, we review the related works about spatial keyword queries; in section 3, we give the problem statement;

section 4 and 5 describe Spatial and Textual Based Hybrid Index(ST*b*HI) and term cluster based inverted spatial index(TC*b*ISI) respectively; in section 6, we perform detailed experiments and section 7 concludes this paper.

## 2 Related Work

Many research works have been done to exploit spatial keyword queries problem. Zhou et al. [2] were the pioneers to try to improve the performance of spatial keyword queries in search engines. They proposed three approaches by combing the inverted index and R-trees:(1) Building both R-trees and inverted indexes for the web documents from spatial and textual perspectives respectively; (2)Building the inverted indexes firstly and then creating a R-tree for every distinct term; (3)Creating a R-tree index for all the web documents and then creating one inverted index for the documents that are contained in each R-tree leaf node. Their experiments showed that the second approach achieved better performance.

Cong et al. [3] proposed a new hybrid indexing framework for top-k spatial keyword queries, the index framework combines the inverted indexes for text retrieval and the R-tree for spatial query. Several index approaches were explored within the framework. In the baseline approach, they index the web documents with R-tree and simply attach the inverted index to each R-tree node to obtain an Inverted file R-tree called IR-tree. For each leaf node, an inverted index is created for the documents that are contained in the leaf node, while the inverted index for each internal node represents all the documents in the sub-trees of the internal node. In addition to the baseline method, Cong et al. also proposed other two advanced approaches: DIR-tree and CDIR-tree. In DIR-tree, they incorporated document similarity when computing Minimum Bounding Rectangles(MBR) and made sure that the textual descriptions of the objects that are in the same R-tree node are also similar. In CDIR-tree, Cong et al. clustered the documents attached to spatial objects and employed a pseudo-document to represent each cluster. So more tighter and precise textual relevance bound can be estimated and the query performance can be further improved.

Rocha-Junior et al. [1] proposed another novel method that can process top-k spatial keyword query more efficiently. Differently from [3], Rocha-Junior et al. proposed a new index structure called Spatial Inverted Index (S2I). They adopted different organization methods for the terms based on their frequency. It is well known that the distribution of terms is very skewed, the document frequency of terms in a corpus follows the Zipf's law, which means that only a small number of terms occur frequently, while most of the terms occur infrequently[1][4]. So S2I mapped each more frequent term to a distinct aggregated R-tree (aR-tree) that stores the objects with the given term, if the number of the objects corresponding to one given term does not exceed a given threshold, the objects are just stored in a file. Based on the S2I index, they also proposed two efficient algorithms(Single Keyword Algorithm: SKA and Multiple Keyword Algorithm: MKA) to process the top-k spatial keyword queries efficiently.

There are many other approaches that focus on spatial keyword queries. Ian De Felipe et al. [5] combined R-tree and signature file, and proposed a new efficient hybrid index called IR$^2$-tree(information retrieval R-tree). Each node of IR$^2$-tree contained two parts of information: space information and keywords information. Zhang et al. [6] proposed bR*-tree to process $m$-closet keyword queries; Zhang et al. [7] proposed a Light-Weighted Index to process the spatial keyword queries for the objects that are described by tags.

## 3 Problem Statement

In this section, we briefly describe the problem addressed in the paper. Let $D$ be a geo-tagged database. Each geo-tagged object $O$ in $D$ can be represented as a triple ($O.id$, $O.loc$, $O.doc$), in which $O.id$ is the identifier of the object, $O.loc$ represents the location of the object and contains latitude and logitude, $O.doc$ is the text that describes the object. Table 1 is an example of $D$.

**Table 1.** Geo-tagged Objects Database

| $O.id$ | $O.loc$ | $O.doc$ |
|---|---|---|
| $o_1$ | (111.12, 108.23) | {apple banana } |
| $o_2$ | (112.21, 108.12) | {noodle pizza } |
| $o_3$ | (111,12, 110.21) | {Peking duck } |
| $o_4$ | (113.21, 110.54) | {great wall } |
| $o_5$ | (112.32, 109.65) | {coca cola } |

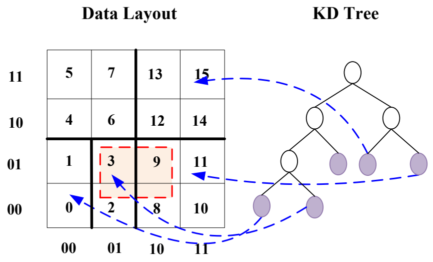

**Fig. 2.** Spatial Index

Given a spatial keyword query $Q(Q.loc, Q.keywords, \sigma)$, in which $Q.loc$ is a given location, $Q.keywords$ is the query keywords and $\sigma$ is the distance threshold, we need to find out all the geo-tagged objects which are far away from the location $Q.loc$ within $\sigma$ and contain the query keywords. After we obtain all the objects that satisfy the textual and spatial constraints of the query, we compute the relevance scores for all the objects and rank the objects according to their scores. The top-k spatial keyword query can be finished by extending the space range of the query incrementally. In this paper we mainly focus on the range query processing.

## 4 Spatial and Textual Based Hybrid Index

### 4.1 Overview of ST*b*HI

In this paper we want to manage the large scale geo-tagged objects using HBase. It is well known that HBase organizes the data based on the rowkeys and can provide very efficient queries on rowkeys. We try to design a suitable rowkey generation scheme that can combine the textual and spatial information of the geo-tagged objects together, so that we can deal with the spatial keyword queries more efficiently.

**Inverted Index Table Schema.** In order to implement textual filtering, we build the inverted index. While different to the traditional inverted index file, we use a HBase table called "Inverted Index Table" to finish the same task. Table 2 displays the scheme of the inverted index table. The rowkey is the composition of the term and the z-ordering value of the object. $t_i$ represents the $i_{th}$ term, $id_m$ is the object id, $z_m(e.g., 01011010)$ is the z-ordering value of $id_m$, $(x_m, y_m)$ represents the latitude and longitude of the object respectively. The z-ordering value of each geo-tagged object can be computed based on its location information(latitude and longitude), because z-ordering technique preserves the original locality of the objects and the z-ordering value of the objects that are nearby in the original space are likely close to each other. The advantage of such kind of design is that we can implement the textual filtering and spatial filtering simultaneously and translate a spatial keyword query into several range queries on the rowkey

**KD-Tree Index.** As long as we encode the objects using z-ordering technique, we can translate the spatial query into range query on the z-ordering values. For instance, given a query rectangle $\boldsymbol{R}[(x_{min}, y_{min}), (x_{max}, y_{max})]$, we can get the minimum z-ordering value $z_{min}$ and the maximum z-ordering value $z_{max}$ and then execute a range query by using [ $z_{min}$, $z_{max}$]. But in some cases, it is not feasible if we directly execute the range query from the query rectangle, as it will produce many false positives. Just as shown in Figure 2, the red bordered rectangle is the range query, and the range of the z-ordering value is [2, 9]. If we directly execute the range query [2,9], we have to scan every object from 2 to 9. But actually what we want to get just include 2, 3, 8 and 9, the objects from 4 to 7 are false positives. In order to eliminate the false positives as many as possible, we partition the objects by using KD-tree index, just like Figure 2 depicts, for the same query rectangle, we obtain two subspaces and get the required objects 2, 3, 8 and 9 finally.

**Table 2.** Inverted Index Table Schema

| rowkey | col. fam.: cf | | |
|---|---|---|---|
| | col: id | col: x | col: y |
| $t_1$01011010 | $id_1$ | $x_1$ | $y_1$ |
| $t_1$01011011 | $id_2$ | $x_2$ | $y_2$ |
| $t_2$11011010 | $id_3$ | $x_3$ | $y_3$ |
| $t_3$01011010 | $id_1$ | $x_1$ | $y_1$ |
| $t_4$11011010 | $id_3$ | $x_3$ | $y_3$ |

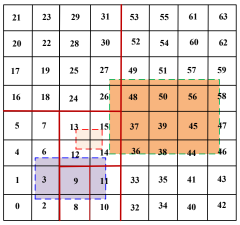

**Fig. 3.** Query Processing

The procedure of the data insertion is as follows. Given a geo-tagged object $O(O.id, \langle O.lat, O.lon\rangle, \{t_1, t_2, t_3, t_2\})$, firstly we compute the z-ordering value of $O.id$, represented as $Z_{O.id}$. Then insert the object $O.id$ into KD-tree, and decide whether the corresponding subspace needs to split or not, if the data size of a subspace exceeds the give threshold, the subspace needs to split; (This step is optional, if the KD-tree is built beforehand, this step can be omitted.) Finally we construct the inverted index data rows and insert them into Inverted Index Table: $(t_1 Z_{O.id}, 1, O.id)$, $(t_2 Z_{O.id}, 2, O.id)$, $(t_3 Z_{O.id}, 1, O.id)$.

## 4.2 Query Processing

Given a spatial keyword query *Q(Q.loc, Q.keywords, $\sigma$)*, in which *Q.loc* is a given location, *Q.keywords* is the query keywords, $\sigma$ is the distance threshold, *Q.loc* and $\sigma$ can be represented as a rectangle area $\boldsymbol{R}[(Q.loc_x$ - $\sigma$, $Q.loc_y$ - $\sigma)$, $(Q.loc_x$ + $\sigma$,$Q.loc_y$ + $\sigma)]$. When dealing with the spatial keyword query, we need to choose the suitable query strategy according to the area of the query rectangle, it can be divided into the following three categories:

– As shown in Fugure 3, if the area of the query rectangle is relative small, just like the red bordered rectangle, it just covers four distinct z-ordering values(12, 13, 14, 15), we can directly compute the z-ordering values of all the objects contained in the rectangle, then access the Inverted Index Table according to query keywords and the z-ordering values.
– As the area of the query rectangle becomes larger, the number of the distinct z-ordering value also increases, in such case, if we still compute each z-ordering value and query them from the Inverted Index Table one by one, it will cost too much time. So we can figure out the minimum and maximum z-ordering value of the query rectangle, then obtain the z-ordering value range, if the range is smaller than a given threshold, we can use the keywords and the z-ordering value range to request from the Inverted Index Table. In such case, there will be some false positives, just like the blue bordered rectangle, the minimum and maximum of the z-ordering value the rectangle covers are 2 and 14 respectively, what we actually want to get are 2, 3, 6, 8, 9, 10, 11, 12, 14 and 4, 5, 7, 13 are the false positives. But the number of the false positive is small, we can still gain more benefits.
– While if the z-ordering range of the query rectangle exceeds a given threshold, there will be much more false positives, in such cases, we need to resort to the KD-tree index. We can obtain the related subspaces by querying the KD-tree index using the query rectangle, then compute the desired ranges according to the area of the regions and the query rectangle. Just like the green border rectangle, by querying KD-tree index we can get three subspaces whose z-order value ranges are 12-15, 16-31 and 32-63 respectively. Supposing that the range of the query rectangle is 14-59, we get the intersects of the query range and each subspace, and the desired z-ordering value ranges are 14-15, 16-31 and 32-59, many false positives have been eliminated.

## 4.3 Pre-Splitting for Inverted Index Table

It is well known that HBase organizes the data as regions and the data is stored in regions according to their rowkey. At the beginning, all the data is inserted into just one region, if the number of the records in the first region exceeds the given threshold, the region will split into two roughly equal sized child regions. As more and more data is inserted, the regions will split recursively. Such scheme has two pitfalls, one is the concurrency, the other one is the high splitting cost. Because all the data are inserted into one region or several regions at the beginning, the concurrency will be constrained by the limited regions, in order to enhance the concurrency, we can pre-split the regions for the Inverted Index Table.

In Inverted Index Table, the rowkey is composed of term and the z-ordering value of the geo-tagged objects, so the records will be stored into different regions according to the terms and the location information of the objects. We all know that, the distribution of the term frequency is always skewed and obeys to the $Zipf$-like distribution, what we want to do is that we can divide all the distinct terms into several segments, and make sure that the sum of the term frequency in each segment is roughly equal. The detail of the pre-splitting is described in Algorithm 1.

**Algorithm 1.** Pre-splitting for Inverted Index Table

```
Input: A geo-tagged object dataset $D$={$o_1, o_2, ..., o_N$};
       the number of the pre-splitting regions $K$;
       the length of the z-ordering value $L$;
Output: $K$ splitting keys:$S$={$\hat{t_1}0_1 0_2 0_3...0_L$, $\hat{t_2}0_1 0_2 0_3...0_L$,...$\hat{t_K}0_1 0_2 0_3...0_L$ }

1: Building a subset $\hat{D}$ = { $\hat{o_1}$,$\hat{o_2}$,...,$\hat{o_M}$} by sampling from the original dataset $D$
2: Calculate the frequency of each unique term and sort the terms in descending order:
   $T$={($t_1$, $f_1$), ($t_2$,$f_2$), ... ,($t_V$,$f_V$) }
3: Calculate the sum of the term frequency for all the terms: $S$
4: $S \leftarrow \varnothing$
5: $temp \leftarrow 0$ //temporal variable used to compute the sum of the frequency
6: while ($T$ != $NULL$) do
7:    get the first term $t_i$ and its frequency $f_i$ from $T$
8:    $temp \leftarrow temp + f_i$
9:    remove ($t_i$, $f_i$) from $T$
10:   if (($\lfloor \frac{S}{K} \rfloor$ - temp) $\leqslant \sigma$) then
11:      add $\hat{t_i}0_1 0_2 0_3...0_L$ into $S$
12:      $temp \leftarrow 0$
13:   end if
14: end while
15: return the pre-splitting key set $S$
```

## 5 Term Cluster Based Inverted Spatial Index

In ST$b$HI index approach, for a given geo-tagged object $o_i$, if $o_i$ has $n$ terms, then we have to insert one index entry for each term into the Inverted Index Table. That is to say, we have to do $n$ times insertion for each geo-tagged object $o_i$, so the index speed will decrease and the space overhead will be high. At the same time, when we deal with a spatial keyword query $Q$ that contains $m$ query keywords, we have to execute $m$ sub-queries and then combine the results of all the sub-queries, so the query performance will also be affected. Because of the above reasons, ST*b*HI Index approach is not the best suitable solution for managing the geo-tagged objects that contain many terms and dealing with the multi-keywords queries. In order to tackle the above problem, we propose another index solution: Term Cluster Based Inverted Spatial Index(TC*b*ISI). This idea is mainly inspired by the following observations: some terms are usually used together to describe the same objects and some terms often occur in the same query. According to the above observation, we can divide the terms into different clusters based on the co-occurrence relationship among the terms, the objects that correspond to one term cluster can be treated together, each cluster is mapped to a HBase table. The objects of a term cluster are stored in one HBase table and the organization of the objects is like that of ST*b*HI. By doing so, when we deal with a query that contains several keywords, the number of the HBase tables that we need to access is less than the number of the keywords, so the response time can be decreased.

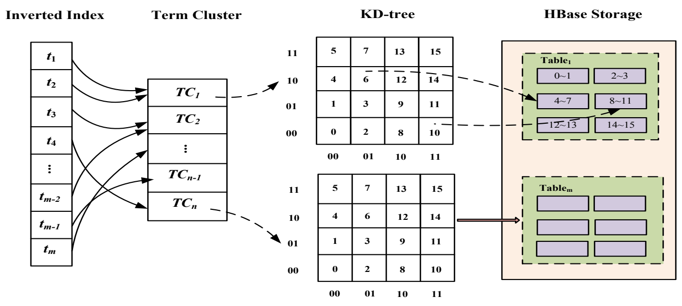

**Fig. 4.** Framework of Term Cluster based Inverted Spatial Index

## 5.1 Framework of Term Cluster Based Inverted Spatial Index

Figure 4 shows the framework of Term Cluster based Inverted Spatial Index. The TC*b*ISI index consists of four components: Inverted Index, Term Cluster Index, KD-tree and HBase Table.

- **Inverted Index**. The Inverted Index is used to implement textual pruning in which each term is mapped to a term cluster. Although the geo-tagged objects are unlimited, the total number of the distinct terms is always limited, we can suppose that the inverted index can be stored in the main memory.
- **Term Cluster Index**. Term Cluster Index is mainly used to map the term cluster to the KD-tree or HBase table, the number of the term clusters is just several thousand, so the term cluster index can also be stored in main memory.
- **KD-tree**. The KD-tree is used to divide the geo-tagged objects that corresponds to each term cluster into several partitions based on the z-ordering value of the objects, when dealing with spatial keyword queries, we can quickly located the related HBase regions using KD-tree.

– **HBase Table**. HBase Table is the actual storage of the objects, it uses the z-ordering value of the object as its rowkey. So the objects that are close in the original space are likely to be stored in the same HBase region.

### 5.2 Query Processing

According to the framework of TC*b*ISI we can finish a spatial keyword query in several steps: given a spatial keyword query *Q(Q.loc, Q.keywords, σ)*, firstly, getting the term clusters which contain *Q.keywords* through Inverted Index; Secondly obtaining the corresponding KD-trees or HBase table names through Term Cluster Index; Thirdly adopting suitable query methods according to the query range; Finally we merge the results returned from each term cluster and obtain the final results. The remaining query procedure is the same as that of ST*b*ISI.

### 5.3 Term Cluster Generation

Term cluster is the basis of TC*b*ISI, in order to obtain the high quality clustering result, we propose a Term Co-occurrence Graph based clustering method(TCG-Cluster), in which each node represents a term and the edge between two nodes represents the co-occurrence frequency of two terms. Because of the space limit, the detail of TCG-Cluster will not be listed here and it can be referred from the extended version of the paper.

## 6 Experimental Evaluations

In this section, we compare our approaches with the *Spatial Inverted Index*(S2I) proposed by Joao et al.[1], to the best of our knowledge, S2I is the state-of-the-art approach that addresses spatial keyword queries. The setup for S2I is the same as described in [1], the nodes of the aR-trees have a block size of 4KB and are able to store between 42 and 85 entries.

### 6.1 Experimental Settings

**Setup.** The experiments for S2I were executed on a server with a 3GHz Dual Core AMD processor and 4GB memory. Our approaches were implemented on HBase-0.20.6 and Hadoop-0.90.4, the cluster size is 16 nodes that are connected with 1Gbit Ethernet switch. The main parameters and values used in the experiments are described in Table 3. The default values are presented in bold.
**Datasets.** Table 3(b) shows the characteristics of the datasets used in the experiments. In this paper we mainly focus on querying from large scale geo-tagged objects, we have two real datasets presently, one is twitter dataset containing about 10,000,000 tweets, another one is a document corpus from Reuters containing 10,000 documents. Based on the two real datasets we generated several other datasets, the detailed statistics of them are listed in Table 3(b).

**Table 3.** Experimental Settings

(a) Experimental Setup

| Parameter | Values |
|---|---|
| Num. of keywords | 1, 2, **3**, 4, 5 |
| Spatial range | 5, 50, **100**, 200, 300, 500 |
| Datasets(M) | 0.5,**1**,1.5,2,100 |
| Comuter Node | 4, **8**, 12, 16 |

(b) Characteristics of the datasets

| Datasets | Tot. no. of objects | Avg. no. of unique terms/obj | Tot. no. of unique terms |
|---|---|---|---|
| Twitter5 | 0.5M | 6.42 | 226,969 |
| Twitter10 | 1M | 6.44 | 380,727 |
| Twitter15 | 1.5M | 6.47 | 515,676 |
| Twitter20 | 2M | 6.46 | 648,518 |
| Data-Syn | 100M | 9.07 | 1,485,225 |

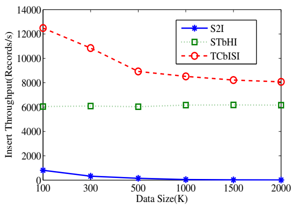

**Fig. 5.** Insert Throughput vs. Data Size

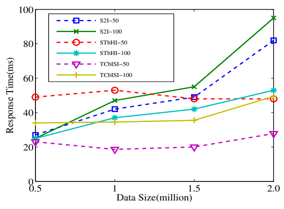

**Fig. 6.** Query Performance vs. Data Size

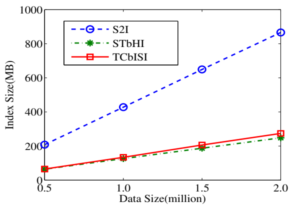

**Fig. 7.** Index Space Requirement vs. Data Size

## 6.2 Performance Varying the Data Size

**Insert throughput.** Figure 5 shows that the insert throughput of S2I, ST*b*HI and TC*b*ISI with different data size. We can see that the insert throughput of S2I is the worst one, and the insert throughput decreases dramatically with the data size increasing. That is because S2I adopts the R-tree like index and R-tree like index needs additional cost to keep the tree balanced which affects the insert performance. The insert performance of TC*b*ISI is better than that of ST*b*HI, the main reason is as the follows: a given geo-tagged object $o_i$ containing $m$ keywords needs $m$ times insertion for ST*b*HI approach, while the insertion times of TC*b*ISI is equal to the number of the clusters which the keywords of $o_i$ belong to, so the insertion times of TC*b*ISI is less than that of ST*b*HI. We can also observe that the insert throughput of ST*b*HI and TC*b*ISI has good scalability, they can keep high insert throughout with the data size increasing. The peak insert throughput of TC*b*ISI is over 10,000 objects per second.

**Query Performance.** Figure 6 mainly describes the query performance of the different index approaches. When the data size and the query range are both relative small(e.g., the data size is less than about 1,500,000 and the query range is smaller than 50), S2I can obtain the better query performance than ST*b*HI. While as the data size increases, the query performance of S2I-50 is becoming more inferior than that of ST*b*HI-50 and TC*b*ISI-50. For the large range query, the query performance of ST*b*HI and TC*b*ISI is always better than that of S2I, e.g., ST*b*HI-100 and TC*b*ISI-100.

**Index Space Requirement.** Figure 7 shows the storage space required for S2I, ST*b*HI and TC*b*ISI. The size required by S2I is larger than the size required by ST*b*HI and TC*b*ISI. The reason is that ST*b*HI and TC*b*ISI are both implemented on HBase, while the data in HBase is actually stored on HDFS and HDFS always adopts many compression algorithms to improve the space utilization. In order to keep fault tolerant, HDFS always keep 3 replicas for each data block, even so, the total size requires is still smaller than that of S2I.

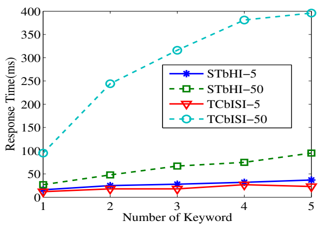

**Fig. 8.** Response Time vs. Num. of Keywords

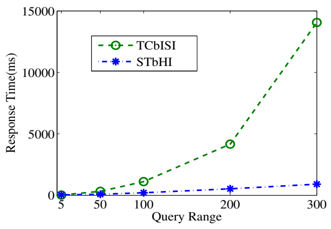

**Fig. 9.** Response Time vs. Query Range

## 6.3 Varying the Number of Keywords

Figure 8 depicts the response time while varying the number of keywords(from one keyword to five keywords) for large scale dataset(100M geo-tagged objects), the default query range is set to 0-50. From section 6.2, we know that the time consumed by S2I for inserting two million objects is about 32 hours, it will spend more than 66 days to insert 100,000,000 objects, so in this section we just test the response time of ST*b*HI and TC*b*ISI. The response time of ST*b*HI increases almost linearly with the number of the keywords. For the query with small range, the query performance of TC*b*ISI is better than that of ST*b*HI, while for the large range query, the performance of ST*b*HI is much better than that of TC*b*ISI. The main reason is as the follows: for the large range query, many records need to be scanned and verified, in ST*b*HI we can finish the filtering and verification based on the rowkey, so that is very fast. While in TC*b*ISI, we need additional time to filter and verify the records on the non-rowkey column.

## 6.4 Varying the Query Range

In this section we mainly test the query response time of ST*b*HI and TC*b*ISI for the queries with different range varying from 5, 50, 100 to 200, 300, 500, the query contains three keywords, the number of geo-tagged objects is the same as in section 6.3. In order to obtain the relative accurate response time, We execute ten times queries and use the average time as the final response time.

As the query range increases, more objects need to be scanned and transformed to the client side, so the response time increases as the query range becomes larger. For the small range queries($\leqslant 5$), the performance of ST*b*HI is better than that of TC*b*ISI, as the query range increases, the performance of ST*b*HI is becoming much better than that of TC*b*ISI, the reason is as the same described in the above subsection.

## 7 Conclusions and Future Work

In this paper, we propose a scalable data management system for massive geo-tagged objects called ST-HBase (Spatial Textual HBase) in which we use HBase as the storage so that it can provide high insert throughput. In order to support efficient spatial keyword query, we layer KD-tree index on HBase and propose two kinds of index approaches. In the future, we plan to extend our work to ad-hoc spatial temporal analysis about the large scale geo-tagged objects, because many geo-tagged objects also contain time dimension, such as photos in Flickr, tweets from Twitter, and the time dimension is also important for analyzing the geo-tagged objects, for instance we can compute the temporal and spatial distribution of the tweets about some specific topics.

**Acknowledgments.** This research was partially supported by the grants from the Natural Science Foundation of China (No. 61070055, 91024032, 91124001); the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University( No. 11XNL010); National 863 High-tech Program (2012AA010701, 2013AA013204).

## References

1. Rocha-Junior, J.B., Gkorgkas, O., Jonassen, S., Nørvåg, K.: Efficient processing of top-k spatial keyword queries. In: Pfoser, D., Tao, Y., Mouratidis, K., Nascimento, M.A., Mokbel, M., Shekhar, S., Huang, Y. (eds.) SSTD 2011. LNCS, vol. 6849, pp. 205–222. Springer, Heidelberg (2011)
2. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.Y.: Hybrid index structures for location-based web search. In: CIKM 2005, pp. 155–162 (2005)
3. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. In: PVLDB, pp. 337–348 (2009)
4. Joachims, T.: A statistical learning model of text classification for support vector machines. In: SIGIR 2001, pp. 128–136 (2001)
5. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE 2008, pp. 656–665 (2008)
6. Zhang, D., Chee, Y.M., Mondal, A., Tung, A.K.H., Kitsuregawa, M.: Keyword search in spatial databases: Towards searching by document. In: ICDE 2009, pp. 688–699 (2009)
7. Zhang, D., Ooi, B.C., Tung, A.K.H.: Locating mapped resources in web 2.0. In: ICDE 2010, pp. 521–532 (2010)