Key Formulation Schemes for Spatial Index in Cloud Data Managements

Ya-Ting Hsu†, Yi-Chin Pan†, Ling-Yin Wei†, Wen-Chih Peng†, Wang-Chien Lee‡
†Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
‡Department of Computer Science and Engineering, The Penn State University, PA, USA
{ythsu.cs99g,beats2.cs99g,lywei.cs95g}@nctu.edu.tw,wcpeng@cs.nctu.edu.tw,wlee@cse.psu.edu

***Abstract*—Due to the flexibility and scalability in cloud computing, cloud computing nowadays plays an important role to handle a large-scale data analysis. For data processing operations, several cloud data managements (CDMs), such as HBase and Cassandra, are developed. Such CDMs usually provide key-value storages, where each key is used to access its corresponding value. Both HBase and Cassandra provide some basic operations (e.g., Get, Scan) to retrieve the values via keys specified by users. The exiting CDMs fully inherit the characteristics of cloud computing (i.e., high scalability and availability). With the aforementioned characteristics of cloud computing, CDMs are widely employed for Web data, especially for search engines. However, with the proliferation of smart phones and location-based services, data with spatial information, referring as spatial data, are dramatically increasing. Consequently, how to formulate keys for spatial data in the existing CDMs is a challenge issue. In this paper, we develop several key formulation schemes. In particular, we propose a novel Key formulation scheme based on R$^+$-tree (abbreviated as *KR$^+$-index*). With our design for keys of spatial data, the existing CDMs are able to efficiently retrieve spatial data. In light of KR$^+$-tree, two spatial queries, k-NN query and range query, are designed. Moreover, we implement the proposed key formulation schemes on HBase and Cassandra, and import real spatial data for spatial queries. The experimental results demonstrate that KR$^+$-tree outperforms other existing key formulations and MD-HBase.**

I. Introduction

In recent years, mobile devices, such as smart phones and tablet computers, become popular in our daily life. Simultaneously, with the increasing prevalence of Global Positioning System (GPS), a large number of location-based applications, such as Foursquare and Flickr, have been developed. People are able to share their real-time events with friends anytime and anywhere if the Internet is available. For example, people can check in to a specific location and can note their activities, and they can see their friends' shared real-time information by the Foursquare application. Those location-based applications induce that a huge amount of multi-attribute data, which at least consist of locations and time-stamps, are dramatically increasing. In order to retrieve and manage the huge amount of multi-attribute data well, different database management systems (DBMSs) have been developed. For traditional relational database management systems (RDBMSs), there are several index structures, such as k-dimensional (k-d) trees [2], quad trees [6], and R-trees [7]. However, RDBMSs is unable to deal with thousands of millions of queries efficiently. On the other hand, distributed relational database management systems (DRDBMSs) are developed and are able to deal with multi-attribute accesses. However, DRDBMSs are unable to maintain and retrieve data among servers efficiently, because DRDBMSs take much time to make data should be consistent by appropriately locking and updating data.

To deal with a huge amount of data efficiently and flexibly, cloud computing nowadays plays an important role and new cloud data managements (CDMs), which are NoSQL databases [14], have been developed. The most prevalent NoSQL CDMs, such as HBase [9], Cassandra [10] and Amazon Simple Storage [15], are developed based on a BigTable [5] management system. Compared with DRDBMSs, these management systems have the characteristics of high scalability, high availability and fault-tolerance because they can effectively and efficiently handle a large number of data updates even if failure events occur. In addition, a BigTable management system stores data as $\langle$key, value$\rangle$ pairs, and thus these BigTable-like management systems can retrieve data efficiently by the following characteristics: 1) each $\langle$key, value$\rangle$ pair is stored on multiple servers; 2) each key owns multiple versions of a value. In other words, the first characteristic benefits the efficiency of retrieving data, and the second characteristic eliminates the waiting time of making data be consistent. Due to the inherent restriction of a BigTable data structure, these management systems only support some basic operations, such as **Get**, **Set** and **Scan**. A **Get** operation retrieves values mapped by a key; a **Set** operation inserts/modifies values according to a corresponding key; a **Scan** operation returns all values mapped by a range of keys. However, these basic operations do not directly support multi-attribute accesses.

In this paper, to support efficient multi-attribute accesses of skewed data on CDMs, we proposed a novel multi-dimensional index, called KR$^+$-index, on CDMs by designing Key names for leaves of R$^+$-tree. A challenge issue is to filter out data after querying result from large difference of volume of data between grids. In order to describe conveniently, we called the size of a gird as the volume of data in the grid. However, dividing a map more meticulously could reduce the differences of the grid sizes but it also reduces the efficiency of accessing data. For example, for a range query, we need to retrieve more grids for the same spatial range. According to the aforementioned observations, we expect the differences of the grid sizes could be smaller and the time of grid accesses could be less at the same time. Consequently, how to divide a map into grids to

978-0-7695-4713-8/12 $26.00 © 2012 IEEE
DOI 10.1109/MDM.2012.67

reach a balance between the two points plays an important role for CDMs. In this paper, we first use R$^+$-tree [13] to divide data, and the rectangles in the leave nodes of the tree index are treated as dynamic grids. The reasons of using R$^+$-tree are describes as follows. First, we could get a balance between the grids sizes and the times of grid accesses by adjusting two parameters, M and m, of the R$^+$-tree. Second, compared with other variants of the R-tree, the leaf nodes of R$^+$-tree do not overlap with each other, and thus it is benefit for no redundant retrieving the same data from different keys and easy to define different keys for each rectangle of a leaf node. Moreover, the second challenge is how to design key names of these grids to support efficient queries on BigTable management systems. We observed the characteristics of CDMs as follows: a CDM has a fast key-value search and to **Scan** keys which are ordered by a dictionary order is fast. Based on these characteristics, we propose an approach to define the key name of a grid to support efficient queries. In the experiment, we implement the proposed index on two well-known CDM systems, HBase and Cassandra, and we compare the performance of the proposed index with the existing index methods. The experimental results demonstrate that our proposed index outperforms the existing index methods via skewed data.

We summarize the contributions of this paper as follows:

- We propose an efficient multi-dimensional index structure, KR$^+$-index, on CDMs to support efficient multi-attribute accesses of skewed data.
- Based on KR$^+$-index, we define new efficient spatial query algorithms, range query and k-NN query.
- The KR$^+$-index uses the characteristics of CDMs effectively.
- The experimental results show that the proposed KR$^+$-index outperforms than other competitors.

The remainder of the paper is organized as follows. First, we illustrate the background of multi-attribute access, multi-dimensional index and Hilbert curve Section 2. We next propose the KR$^+$-index in Section 3. In Section 4, we evaluate the performance of the proposed KR$^+$-index for multi-attribute accesses on CDMs. Finally, we conclude the paper and give a discussion of the future work in Section 6.

II. Background

A. Multi-attribute Access

For multi-dimensional data search, multi-attribute access is used to restrict multiple attributes at the same time. For instance, *Range Query* and *k-NN Query* are common queries of multi-attribute access and are widely used in location-based services.

1) Range Query: Given a set of data points P and a spatial range R, a range query can be formulated as "searching the data points in P that locate in the spatial range R". Note that, in this paper, each data point has location information, e.g., a longitude and a latitude. Without loss of generality, in this paper, a spatial range is represented by a rectangular range.

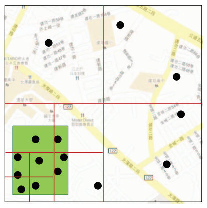


Fig. 1. Quad-tree with $M = 3$.

2) k-NN Query: Given a set of data points P, a query location $p = (p_x, p_y)$ and a constant k, a k-NN query can be formulated as "searching the data points in P that are the k nearest data points of p".

3) Sub-query: Given a query, the query needs one or more queries, called sub-queries, to be completed. In other words, the sub-queries composed of a query.

B. Multi-dimensional Index Techniques

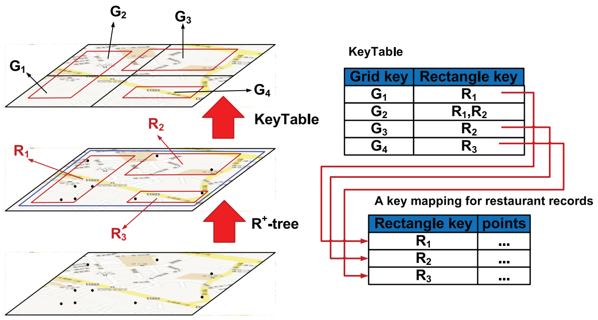
1) Tree Structures: R-tree, developed for indexing multi-dimensional data, are widely used in multi-attribute accesses. Because R-tree is a balance search tree by dynamically splitting and merging nodes and R-tree can restrict the number of elements in each node by controlling the M and m, R-tree benefits for searching skewed data. Moreover, to efficiently index different multi-dimensional data, different variations of R-trees have been developed, such as R$^+$-tree [13], R*-tree [1] and the Hilbert R-tree [8]. The R$^+$-tree developed a new rule of splitting and merging nodes to speed up the multi-attribute accesses.

Quad-trees [6] are another common tree structures for indexing multi-dimensional data. In quad-trees, each internal node has exactly four children. However, quad-trees are not balance trees because a region is split into four sub-regions until the number of data points in the region is less than or equal to a given parameter M.

2) Linearization: Linearization is a well-known technique for indexing multi-dimensional data by transforming multi-dimensional data into one-dimensional data. One of the most popular method of linearization is using space-filling curves [3], such as Hilbert curve [4] and Z-ordering [11]. Given a two-dimensional data, this method first divides the map into $2^n \cdot 2^n$ non-overlap grids, where n is a parameter, and assign a number for each grid according to the order of traversing all grids. Note that the number of each grid is regarded as a key. However, using space-filling curves to index data may be not efficient. If the value of n is set to be lower, it will result in querying more unqualified data, said false-positive, should be pruned. On the other hand, if n is set to be larger, it would increase the times to retrieve more grids. Thus, for this indexing technique, it is a trade-off to set a proper value of n for efficiency.

III. Multi-dimensional Index Structure

The CDMs provide key-value search, which retrieving a value by given a key, based on the data model of CDMs.

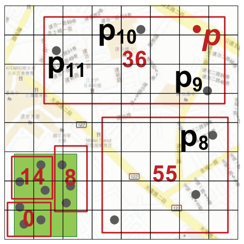


KeyTable

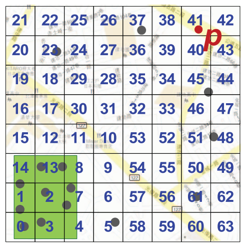
Grid key	Rectangle key
G_1	R_1
G_2	R_1, R_2
G_3	R_2
G_4	R_3

Rectangle key	points
R_1	...
R_2	...
R_3	...

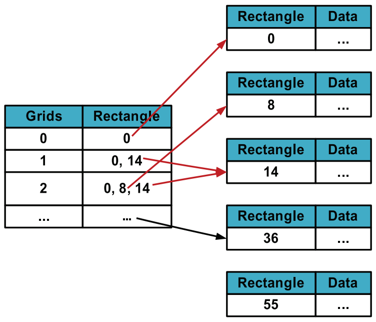
Fig. 2. The overview of KR$^+$-index.



(a) A key definition for rectangles.



(b) A key definition for grids.



Grids	Rectangle
0	0
1	0, 14
2	0, 8, 14
...	...

Rectangle	Data
0	...

Rectangle	Data
8	...

Rectangle	Data
14	...

Rectangle	Data
36	...

Rectangle	Data
55	...

(c) A KeyTable for rectangles.

Fig. 3. An example of KR$^+$-index.

The CDMs support basic operations to access data, but these operations do not directly support multi-attribute access. To deal with the problem of multi-attribute access, we develop a multi-dimensional index structure for CDMs. Furthermore, in this paper, we apply our developed index structure for range query and k-NN query on CDMs.

A. *KR$^+$-index*

Our design of multi-dimensional index is based on the observation of CDMs. We observe three characteristics of CDMs: 1) the time of retrieving a key that has n data is far less than the time of retrieving n keys that each has one data; 2) the time of retrieving a key has n data increases more than twice as n is large. 3) the operation **Scan** is more efficient than multiple **Get** that both retrieving the same keys. Considering the aforementioned characteristics, for a query should make the number of false-positive to be smaller from the characteristic 2 and let the number of sub-queries to be smaller from the characteristic 1. R$^+$-tree is balance tree that has M and m to control the size of each dynamic rectangle, we could use the M and m to meet the trade-off between false-positive and sub-queries. Considering the characteristic 3, we use the Hilbert curve to let the queried key to be as continuous as possible and then the rate of **Scan** is increased.

Figure 2 is the framework of KR$^+$-index. First, the data is constructed by the R$^+$-tree with given M, m and the restaurant records for each rectangle, $\{R_1, R_2, R_3\}$, are maintained. In order to retrieve the restaurant records efficiently, we proposed a mapping method for retrieving the queried rectangle keys. Second, the map is divided into $2^n \times 2^n$ non-overlap grids, $\{G_1, G_2, G_3, G_4\}$, uniformly. Then, for each grid maintains a list of rectangles that overlap with this grid. For instance, the grid G_2 overlaps with rectangles $\{R_1, R_2\}$ that the KeyTable store a record $\langle G_2, \{R_1, R_2\}\rangle$. Thus, a query could convenient transform into which grids need to be queried and then through the KeyTable could easily get the required rectangles.

For these key-value storages, it is crucial to define the key, because we use the key to access corresponding data. We construct R$^+$-tree to discover non-overlap minimum bounding rectangles. Considering the characteristic 3, we use Hilbert-curve to define the keys, since Hilbert-curve manifests superior data clustering compared with other multi-dimensional linearization technique. For each leaf rectangle, we use the Hilbert-value of the geographic coordinate of the centroid of rectangle be the key. Then, we split the space to non-overlap $2^n \times 2^n$ grids uniformly and each grid has a Hilbert-value which is transformed by Hilbert-curve. Take the Figure 3(a), for example, each rectangle is given a Hilbert value. Each grid also given a Hilbert value, the grid 1 in Figure 3(b) overlaps with the rectangles $\{0, 14\}$ that $\langle 1, \{0, 14\}\rangle$ is stored in KeyTable as showed in Figure 3(c). We could get the rectangle information though the KeyTable and the multi-attribute access can retrieve the data efficiently.

B. *Range Query*

The multi-dimensional range query is commonly used in location based applications. Algorithm 1 is the pseudo code for range query in HBase and Cassandra. (p_l, p_h) is the range for the query, p_l is the lower bound and p_h is the upper bound. Hilbert curve splits the space into grids, and each grid has one grid key. The algorithm first compute the coordinate of grids overlap with the range query. The GridKeys is the set of grid keys contained in the query range. For each coordinate of grid c, the function ComputeContainGridKeys() computes the corresponding grid keys via Hilbert curve and add to the list, GridKeys. Then, according to the key table we could find the rectangle keys in the query range. Line 5-8 find the queried key and line 9-10 fetch the points in the corresponding key. The function GetContainPoint() returns the queried data by first retrieving points from Cassandra and HBase with key k and then filtering out some points that is not in the query range.

As shown in Figure 3, take the green block as range query

Algorithm 1 Range Query

```
Input: $p_l$, $p_h$: the range for the query;
Output: points contained in the range;
1: Coordinate ← $ComputeCoordinateOfGrid(p_l, p_h)$;
2: Keys ← $\phi$;
3: RectKeys ← $\phi$;
4: Result ← $\phi$;
5: for each Coordinate $c$ ∈Coordinate do
6:     GridKeys ← GridKeys ∪$ComputeContainGridKeys(c)$);
7: end for
8: RectKeys← $GetRectKeys$(GridKeys);
9: for each Key $k$ ∈RectKeys do
10:     Result ← Result ∪$GetContainPoints(k)$);
11: end for
12: return Result;
```

Algorithm 2 $GetRectKeys$(GridKeys)

```
Input: GridKeys: the grid keys overlap with query range;
Output: the rectangle keys overlap with query range;
1: RectKeys ← $\phi$;
2: for each grid key $gk$ ∈GridKeys do
3:     RectKeys ← RectKeys ∪ KeyTable($gk$));
4: end for
5: return RectKeys;
```

that we will show an example how range query works. Using the query range to get the geographic coordinates of the overlapped grids, $\{(0,0),\ (0,1),\ (0,2),\ (1,0),\ (1,1),\ (1,2),\ (2,0),\ (2,1),\ (2,2)\}$, then get the Hilbert values of each geographic coordinate, {0, 3, 4, 1, 2, 7, 14, 13, 8}. Second, getting keys of rectangles through the KeyTable that grid 0 maps to rectangle {0}, grid 1 maps to rectangles {0, 14}, grid 1 maps to rectangles {0, 8, 14}, etc. Thus, we can get the queried rectangle keys, {0, 8, 14}, by union the rectangle sets got from the former steps. Finally, using the rectangle keys to retrieve data in the CDMs and then pruning the unqualified data to get the query result.

C. *k*-NN Query

The k-NN query is also commonly used in location based applications. Algorithm 3 shows the k-NN query algorithm in HBase and Cassnadra, K stores the result k nearest neighbors, QueryRect stores the rectangles could be scanned, dist is the range for rectangle search, Rect$_{scanned}$ stores the rectangles had been scanned, and the data structure of QueryRect is a queue. The k-NN query has two mainly parts: 1) set a range dist to search for rectangles overlap with a square range with centroid p and edge length 2·dist; 2) pick the nearest rectangle of p that is not scanned and add the nearest points in this rectangle into K. The algorithm keep repeat step 1, 2 until the distance of k-th nearest point and p is less than or equal to dist. The part 1) in Algorithm 3 is in line 6-11, where RectInRegion() is used to find the rectangles in square range and line 9 push the rectangles have not be scanned into QueryRect; 2) is in line 12-18, where line 12 pop the nearest rectangle, line 14 will add the points of R into K. The function RectInRegion(c, dist) in Algorithm 4 finds the rectangles overlap with the input square. It is designed by our methods for defining key in rectangles. Line 6-8 find the grids keys which overlap with the square and line 10 returns the rectangles overlap with grids through checking the KeyTable.

Algorithm 3 k-NN Query

```
Input: $k$: $k$ nearest neighbors; $p = (x, y)$: query point;
Output: $k$ nearest neighbors of $(x, y)$;
1: K← $\phi$;
2: QueryRect ← $\phi$;
3: dist ← 0;
4: Rect$_{scanned}$ ← $\phi$;
5: loop
6:     if QueryRect== $\phi$ then
7:         Rect$_{next}$←$RectInRegion(p$,dist)−Rect$_{scanned}$;
8:         for each Rectangle R∈Rect$_{next}$ do
9:             Push(R, MinDist($p$, R), QueryRect);
10:        end for
11:    end if
12:    R ←Pop(QueryRect);
13:    for each Point $t$ ∈R do
14:        K ← K ∪ ¡t, Dist($p$, $t$)¿ and sort K by dist;
15:    end for
16:    if dist($k$-th point in K, $p$) ≤ dist then
17:        break;
18:    end if
19:    Rect$_{scanned}$ ←Rect$_{scanned}$ $\bigcup$R;
20:    dist←Max(dist, MaxDist($p$, R));
21: end loop
22: return K;
```

Algorithm 4 $RectInRegion(p$,dist)

```
Input: $p = (x, y)$: query point; dist: means a square with edge length 2·dist and with $p$ as its centroid
Output: the keys of rectangles overlap with the input rectangle
1: RectKeys ← $\phi$;
2: xl ←x-dist;
3: xh ←x+dist;
4: yl ←y-dist;
5: yh ←y+dist;
6: for i=xl → xh do
7:     for j=yl → yh do
8:         GridKeys ← GridKeys ∪ Hilbert(i, j);
9:     end for
10: end for
11: return RectKeys←KeyTable(GridKeys);
```

As showed in Figure 3, take p as the query point, $k = 3$ and given a initial dist=0. First, we will get a rectangle 36 through KeyTable with a square range of length 2·dist and then insert the location points $\{p_9,\ p_{10},\ p_{11}\}$ of rectangle 36 into K, in that location points are ordered by the distance from p. Second, resizing the dist to the minimum distance of k-th/—K—-th location points in K from p, the dist=dist(3-th location point in K, p) in this example. The algorithm continues the first and second steps, it will add the rectangle 55 into Rect$_{next}$ and add the location points in rectangle 55 into K. The algorithm is stoped by dist(3-th location point in K, p) ≤ dist, and we get the fist three location points $\{p_{10},\ p_9,\ p_8\}$ in K as the query result.

TABLE I
CASSANDRA-RANGE QUERY(SECOND)

km^2	$1 \cdot 1$	$5 \cdot 5$	$10 \cdot 10$	$20 \cdot 20$	$40 \cdot 40$
KR$^+$: M1250 m625	0.118	0.696	1.221	2.003	4.899
KR$^+$: M2500 m1250	0.007	0.036	0.206	0.642	1.734
KR$^+$: M5000 m2500	0.234	0.776	1.454	1.665	3.748
Hilbert: order4	4	7.8	7.2	7	9.4
Hilbert: order5	5	8.9	9.1	9.1	11.3
Hilbert: order6	7.6	13.7	13.5	13.7	19.8
Scan DB	105	110	127	155	203

TABLE II
CASSANDRA-k-NN QUERY(SECOND)

k	10	50	100
KR$^+$: M1250 m625	0.675	1.387	1.668
KR$^+$: M2500 m1250	0.039	0.088	0.657
KR$^+$: M5000 m2500	0.433	0.944	1.329
Hilbert: order4	8	11.201	14.35
Hilbert: order5	10.348	15.455	19.349
Hilbert: order6	14.6	18.866	24.545
Scan DB	105	110	127

TABLE III
HBASE-RANGE QUERY(SECOND)

km^2	$1 \cdot 1$	$5 \cdot 5$	$10 \cdot 10$	$20 \cdot 20$	$40 \cdot 40$
KR$^+$: M1250 m625	1.446	5.172	6.290	10.713	13.114
KR$^+$: M2500 m1250	1.019	2.656	6.481	12.810	18.552
KR$^+$: M5000 m2500	3.508	5.129	10.895	14.998	21.617
Hilbert: order4	44.65	47.93	50.04	52.16	56.47
Hilbert: order5	48.82	50.99	54.77	59.83	67.12
Hilbert: order6	54.10	57.31	62.50	66.32	79.51
Scan DB	135.06	134.99	135.08	134.84	135.01

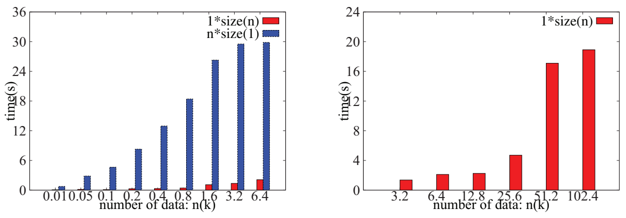
TABLE IV
HBASE-k-NN QUERY(SECOND)

k	10	50	100
KR$^+$: M1250 m625	2.13	3.558	7.729
KR$^+$: M2500 m1250	1.679	2.896	8.354
KR$^+$: M5000 m2500	6.732	8.024	14.2
Hilbert: order4	54.282	60.04	67.31
Hilbert: order5	57.354	64.13	73.52
Hilbert: order6	67.492	70.458	77.341
Scan DB	135.12	135.250	135.22

IV. EXPERIMENT

In this section, we show the experiments about the time of range query and k-NN query in HBase and Cassandra with different methods: scan databases, Hilbert curve and our method, KR$^+$, and compare the KR$^+$HBase with MD-HBase[12].

We implement our experiments using HBase 0.20.6 with Hadoop 0.20.2 and Cassandra 0.8.2 as the underlying system. We have 8 machines, each consists of 2 virtual machines, 2GB memory and 500GB HDD and 64bit Ubuntu 8.04.4. The arguments in our experiments with $(M, m) = (1250, 625)$, $(2500, 1250)$ and $(5000, 2500)$ of KR$^+$ are about 352, 176 and 88 rectangles respectively. The number of grids is divided into $2^4 \times 2^4$, $2^5 \times 2^5$ and $2^6 \times 2^6$. The range of range query are $1km \times 1km$, $5km \times 5km$, $10km \times 10km$, $20km \times 20km$ and $40km \times 40km$. The k-NN query with $k = 10, 50, 100$. The datasets has 440,912 GPS location points collected by our Carweb, it is a data collection machine. The data collected by Carweb is extremely skewing. We also generate a uniform datasets has 440,912 GPS location points. In the following we show the different result with different data distribution.

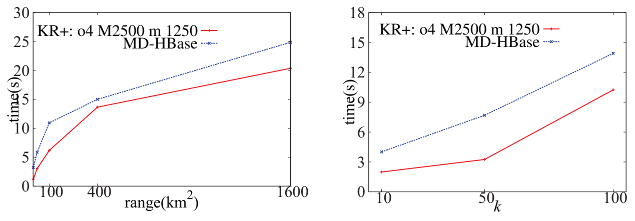


(a) One **Scan** v.s n **Get**.
(b) **Scan** with large n.

Fig. 4. The features of the CDMs.

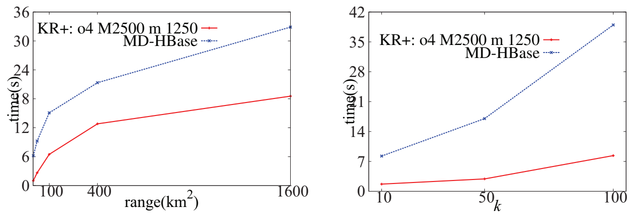
Before evaluating our method with others, we have done some experiments on HBase and Cassandra. It is necessary to find the features of these CDMs, and we design the index structure according to these features. We observe that it is more efficient to fetch a set of keys continuously than fetch single key repeatedly, and it has bad performance when one key stores too much data for these CDMs. Figure 4(a) shows the evaluation between scanning a set of data once and getting one key many times that scanning is quite outstanding. The Figure 4(b) shows that the response time is increasing rapidly while the number of data n from 25600 to 51200.

Table I and II show the range query and the k-NN query on Cassandra respectively. We compare our KR$^+$ with Hilbert curve and no index. With no index, we scan the databases to find the location points in the query. And the Hilbert curve with order 4, 5, 6 is uniform dividing the map along x-axis and y-axis into $2^4 \times 2^4$, $2^5 \times 2^5$ and $2^6 \times 2^6$. The method of scan databases is obviously very slow, about 105s to 203s in range query and 105s to 127s in k-NN query. The method of Hilbert curve in range query is much faster than scan databases, the fastest in range $1km \times 1km$ is 4s and $40km \times 40km$ is 9.4s with order 4. The time increases as the order of Hilbert curve increases since the number of sub-queries increases as the order increases. Our KR$^+$ with order 4 of grids is mush faster than Hilbert curve since it has the feature of balancing the number of false-positive and the number of sub-queries. We observe that the KR$^+$ with $M = 1250$ and $m = 625$ is slower than with $M = 2500$ and $m = 1250$ since the former has the more sub-queries, and the KR$^+$ with $M = 5000$ and $m = 2500$ is slower than with $M = 2500$ and $m = 1250$ owing to the former has the more false-positives. Thus, the KR$^+$ with arguments about $M = 2500$ and $m = 1250$ closest the target of the trade-off between the number of false-positives and the number of sub-queries. The k-NN query has the same effect with range query in Hilbert curve and KR$^+$, but it is slower than range query because it may needs to resize the distance of searching rectangles and query again as the number of location points is less than k.



(a) Range query. (b) k-NN query.

Fig. 5. Uniform data distribution.



(a) Range query. (b) k-NN query.

Fig. 6. Skewing data distribution.

In Table III and IV show the range query and k-NN query on HBase. We compare KR$^+$ with Hilbert curve and no index as well. As no indexing, it needs to scan whole data then filters out points which not match the queried. The Hilbert curve is also uniform dividing with order 4, 5, 6. As we have seen, the no index has bad performance due to scanning the whole data. Hilbert curve is much better than no index both on range query and k-NN query, with order 4 the range query of $1km \times 1km$ is 44.6s, $40km \times 40km$ km is 56.4s and the k-NN query of is about 54.2s as $k = 10$. As the same evaluation of Cassandra, KR$^+$ is faster than Hilbert curve and when $M = 2500$, $m = 1250$ the KR$^+$ will get the best performance. The KR$^+$ with $M = 2500$ and $m = 1250$ of range query for $1km \times 1km$ is about 1.01s and k-NN query for $k = 10$ is about 1.67s.

At last, we compare KR$^+$ with MD-HBase for skewing data and uniform data. Figure 5(a) and Figure 5(b) are range query and k-NN query for uniform data. KR$^+$ improves a little efficiency comparing with MD-HBase for uniform data. KR$^+$ with M = 2500 and m = 1250 for $1km \times 1km$ range query is about 1.2s and k-NN is about 2.0s as $k = 10$. In addition, with skewing data, in Figure 6(a) and Figure 6(b) show the KR$^+$ is much faster than MD-HBase. KR$^+$ could balance the number of false-positive and the number of sub-queries so that the KR$^+$ improves the efficiency of range query and k-NN query a lot. KR$^+$ for range query $1km \times 1km$ is about 1.0s and k-NN query as $k = 10$ is about 1.7s. However, MD-HBase for range query $1km \times 1km$ is about 6.2s and k-NN query as $k = 10$ is about 8.2s. The result of evaluation shows our KR$^+$ has much better performance for range query and k-NN query. Besides, KR$^+$ overcomes the trade-off between the number of points for getting one key and the number of keys for scanning so that KR$^+$ get more efficient than MD-HBase, especially for skewing data.

V. Conclusion

We proposed a scalable multi-dimensional index, KR$^+$-index, based on now existing CDMs, such as HBase and Cassandra. It supports efficient multi-dimensional range queries and nearest neighbor queries. We used R$^+$ to construct index structure and designed the key for efficient accessing data. In addition, we redefined spatial query algorithm, including range query and k-NN query for our KR$^+$. KR$^+$ took the characteristics of these CDMs into account so that KR$^+$ shows much more efficient than other index methods in experimentation. The experiments verified our ideas and showed that KR$^+$ outperformed the other competitors especially using skewing data distribution.

[1]

References

[1] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.
[2] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
[3] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *Information Theory, IEEE Transactions on*, 15(6):658–664, 1969.
[4] A.R. Butz. Convergence with hilbert's space filling curve*. *Journal of Computer and System Sciences*, 3(2):128–146, 1969.
[5] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.
[6] R.A. Finkel and J.L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
[7] A. Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.
[8] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. 1993.
[9] A. Khetrapal and V. Ganesh. Hbase and hypertable for large scale distributed storage systems. *Dept. of Computer Science, Purdue University*.
[10] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
[11] G.M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. *IBM, Ottawa, Canada*, 1966.
[12] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services.
[13] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507–518. Citeseer, 1987.
[14] M. Stonebraker. Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11, 2010.
[15] J. Varia. Cloud architectures. *White Paper of Amazon, jineshvaria. s3. amazonaws. com/public/cloudarchitectures-varia. pdf*, 2008.

[1]Wen-Chih Peng was supported in part by the National Science Council, Project No. 100-2218-E-009-016-MY3, 100-2218-E-009-013-MY3 by Taiwan MoE ATU Program, and by Microsoft.