International Journal of Database Theory and Application
Vol.10, No.4 (2017), pp.1-12
http://dx.doi.org/10.14257/ijdta.2017.10.4.01

Grid-based k-Nearest Neighbor Queries over Moving Object Trajectories with MapReduce

Ying Xia[1], Ruidi Wang[2], Xu Zhang[3*] and Hae Young Bae[4]

[1, 2, 3]Research Center of Spatial Information System, Chongqing University of Posts & Telecommunications, Chongqing, China
[4]Inha University, Incheon, South Korea
*[1]xiaying@cqupt.edu.cn, [2]wangruidi.jn@foxmail.com, [*3]zhangx@cqupt.edu.cn, [4]hybae@inha.ac.kr*

Abstract

k-Nearest Neighbor Trajectory (k-NNT) Query is a basic and important spatial query operation widely used in many fields, such as intelligent transportation and urban planning. However, with the rapid increase of trajectory data volume, traditional k-NNT query algorithms for centralized environment are not effective and scalable enough, because the computational complexity increases dramatically when the spatial continuity of trajectories is considered. To address this problem, we propose a distributed grid index for trajectory data which partitions the trajectory into grids under MapReduce framework. Furthermore, a parallel query approach MR-GB-KNNT is proposed based on the proposed grid index to improve the efficiency and scalability of the k-NNT query. The experiment demonstrates that MR-GB-KNNT could perform well in cloud computing environment and improve the querying performance of the k-NNT.

Keywords: *Nearest Neighbor Query, Moving Object Trajectory, Grid Index, MapReduce*

1. Introduction

In recent years, with the rapid development of GPS, mobile Internet and location-based services, a large amount of spatial-temporal trajectory data gathered in many fields such as traffic, environment, social networking and so on, put forward a higher requirement on the diversity and efficiency of spatial information services. Spatial query serves as a basis of spatial information services, of which processing efficiency is the key factor of system performance. Therefore, how to provide an efficient spatial query approach is one of the research hotspots for spatial information processing.

In this paper, we focus on the k nearest neighbor trajectory query. Different from traditional kNN query, k-Nearest Neighbor Trajectory (k-NNT) Query is to retrieve the top-K trajectories with the shortest aggregate distance from the query trajectory T in the trajectory dataset S. k-NNT has been widely used as classification and recommendation algorithm in trajectory databases [1]. For example, social network users want to retrieve the traces of some scenic spots visited by their friends as a reference for travel plans. This scenario can be simplified to a k-NNT query problem.

However, some of the traditional query methods are to find k nearest neighbor point objects [2], ignoring the spatial continuity of trajectories. Some other consider sequential dimension of trajectories, but only support some query of trajectory data instead of k-NNT query based on spatial-temporal distance [3]. Additionally, traditional k-NNT query algorithms are based on spatial index in a stand-alone, so it is difficult to deal with the massive trajectory data because the computation and memory are limited. As a result, we

[*] Corresponding Author

call for the cloud computing platform to provide a promising method to handle the trajectory of large volume and complex format. Therefore, in order to support the efficient k-NNT query algorithm in distributed environment, we provide a grid index for trajectory data and an auxiliary structure of the trajectory rebuilding table with considering of the spatial and temporal characteristics of location and trajectory data. Finally, we implement the distributed k-NNT query based on MapReduce.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 provides the preliminaries. In Section 4, we present the processing framework and the detailed solution of the k-NNT query in the MapReduce model. Section 5 evaluates the experimental results. We conclude in Section 6.

2. Related Work

There are many researches have been conducted on the k-NNT query problem for centralized environment. Chen [4] proposed IKNN algorithm applied depth-first strategy to find the trajectories which pass as close as possible to a few points. Tang [5] added a Global Heap structure to storage candidate trajectories on the basis of R-Tree, and gave the k-NNT query algorithm. These methods follow candidate generation and validation strategy, invoking the kNNT query algorithm with each query point as the center, and at last sort by the distance from the trajectory to the point set. Qi [6] designed a hybrid NN method by the means of a spatial range-based search, which addressed the problem of increasing I/O and CPU costs caused by running multiple NN searches independently. However, in the face of a large amount of spatial data, it is very time-consuming and even impossible to store and process these data on a stand-alone computer due to resource limit.

There are also many other spatial query based on the cloud platform. Li [2] and Lu [7] used the Voronoi-based approach to partition the spatial data in MapReduce, and processed kNN [2] and kNN Join [7] in the iterative tasks. For spatial-temporal trajectory data, Ma [3] made full use of the computational ability of cluster and proposed a query processing framework for trajectory data based on MapReduce which solved the range query of trajectory data. Similar work, Ji [8] proposed the kNN query method based on the grid index, using the PCT algorithm to filter the grid and then validate the candidate set one by one during the verification phase. Besides, the join query of trajectory data based on the grid index was proposed [9]. Eldawy [10] presented a new spatial data processing platform SpatialHadoop, and implemented three basic spatial operations: kNN query, range query and spatial join query based on it. Unfortunately, few of the studies concern about the kNN query over moving object trajectories on the cloud platform.

3. Preliminaries

For simplicity, we use Euclidean space to describe the trajectory of moving objects. Related definitions are as follows:

Definition 1 A trajectory of a moving object is a sequence of spatial-temporal points. The trajectory *Tr* can be represented as $Tr = \{p_1, p_2, \ldots, p_n\}$, where p_j is the j-th member point of *Tr*. Each point is described as a triple (x, y, t), where *t* is the timestamp and (x, y) is the coordinates of the moving object at *t*.

Definition 2 Given trajectory $Tr = \{p_1, p_2, \ldots, p_n\}$ and point *q*. The minimum distance between the point *q* and *Tr* is denoted as $dist(p_j, q)$ .if $\forall p_k \neq p_j$, $dist(p_j, q) \leq dist(p_k, q)$ is the minimum distance and $\langle p_j, q \rangle$ is the shortest matching pair of *q* and *Tr*.

To compute the distance between the two trajectories, the closest-pair distance is usually adopted as the measure [11]. However, this approach is unreasonable because it ignores the contribution of other points in the query trajectory *t*. On the other hand, there are also other studies about trajectory distance measure [12], focusing on the shape of trajectory, while our method consider spatial distance. Definition 3 is our approach of the distance measure.

Definition 3 Given trajectory $Tr = \{p_1, p_2, \ldots, p_n\}$ and input query trajectory $t = \{q_1, q_2, \ldots, q_m\}$. The distance of shortest matching pair $\langle p_j, q \rangle$ is the distance between *Tr* and a query point *q*. The aggregated distance between *Tr* and *t* is defined as follows. It is the sum of distance of the shortest matching pairs from all points in *t*.

$$dist(Tr,t) = \sum_{q \in t} dist(Tr,q) = \sum_{q \in t} dist(p_j,q) \qquad (1)$$

In our method, we consider assigning the contribution to each matching pairs in the query trajectory *t*, and sum them as the aggregated distance. The method avoids an inaccurate distance caused by some noise points from the trajectory. As we can see in Figure 1, The aggregated distance between *Tr* and *t* is $dist(Tr,t) = dist(Tr,q_1) + dist(Tr,q_2) + dist(Tr,q_3) \ = 65m$.

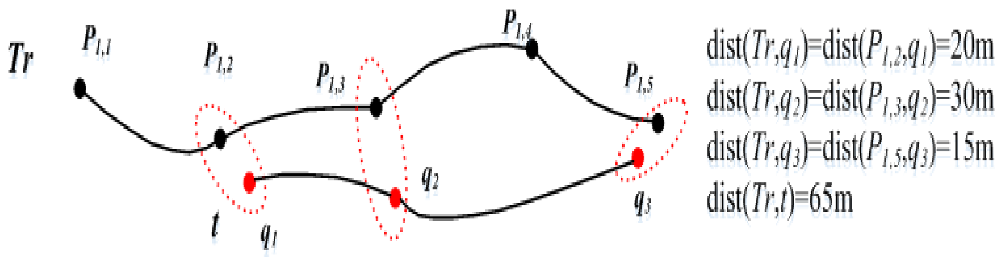
Tr
$P_{1,1}$
$P_{1,2}$
$P_{1,3}$
$P_{1,4}$
$P_{1,5}$
t
q_1
q_2
q_3
$dist(Tr,q_1)=dist(P_{1,2},q_1)=20m$
$dist(Tr,q_2)=dist(P_{1,3},q_2)=30m$
$dist(Tr,q_3)=dist(P_{1,5},q_3)=15m$
$dist(Tr,t)=65m$

Figure 1. Example of Aggregated Distance

With the above definition of distance measure, now we formally give a description of the k-NNT query problem.

Definition 4 Given the trajectory dataset *S*, and a query trajectory *t*, the k-NNT query is to retrieve *k* trajectories *K* from *S*, $K = \{Tr_1, Tr_2, \ldots, Tr_k\}$ that for $\forall Tr_i \in K$ and $\forall Tr_j \in D - K$, $dist(Tr_i, t) \le dist(Tr_j, t)$.

To solve the problem mentioned above, we usually adopt the divide-and-conquer strategy when the dataset becomes too much, and the MapReduce framework provides a viable way. Therefore, we introduce a baseline parallel solution of k-NNT query problem using MapReduce and denote the approach as MR-Base-KNNT. It scans the total trajectory dataset and computes all the aggregated distances between each trajectory and the query trajectory in the map() and sorts their distances in the reduce(). Finally, it outputs the k nearest neighbor trajectories to the HDFS. The main drawback of MR-Base-KNNT is expensive computational cost, since all the trajectories are retrieved.

4. Query Processing

4.1. Framework

The framework of the k-NNT query system is composed of storage module, grid index module and query module, which is shown in Figure 2. First, all trajectory data are imported into HDFS to store as replications for reliability. Second, because the cost of performing a k-NNT query is measured by the number of the input trajectory it handles, we need to build the index to speed up this process. The grid cells and Voronoi diagrams are both the spatial-aware partitioning strategy which organize the trajectory data into different regions, so that they could reduce the number of the trajectory candidate set, hence reducing computing cost. In this work, we adopt the grid-based index for spatial-temporal trajectories data with MapReduce. Grid-based index is flat and easy for parallelization, because its multiple local sub-grid indexes could be constructed by the divide-and-conquer strategy in MapReduce. Moreover, it is particularly more suitable for dealing with dynamic data set, while Voronoi-based index suffers from rebuilding the index or other complex changes of the index when processing dynamic data. As a result, we use the grid partition strategy to divide an n-dimensional space into multiple grid units. Finally, after building the grid index, with the help of the trajectory rebuilding strategy, the results will be returned quickly when a k-NNT query is submitted.

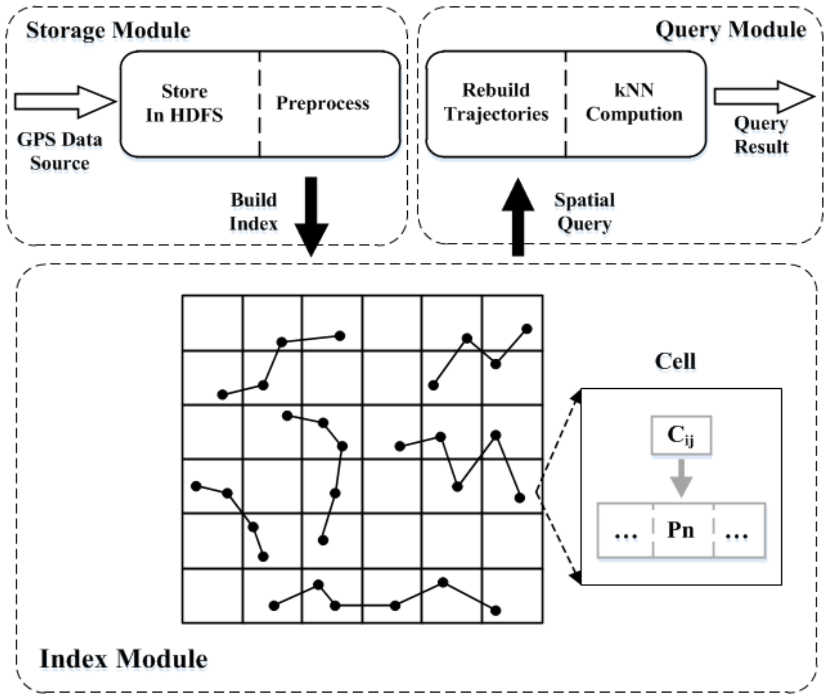

Figure 2. Framework of k-NNT Query Processing

4.2. Grid-based Index Structure

Without loss of generality, we could assume that the space is a rectangle. Given a two-dimensional space trajectory dataset *S*, a point p of any trajectory in S can be represented in coordinates $(p.x, p.y)$. For the point *p*, there is a Function $index(p)$ which returns a grid that contains the point *p*. Clearly, the point $p(p.x, p.y)$ falls into the cell $c[p.x/\delta, p.y/\delta]$, where δ is the grid size. After we use the single-layer grid with a size of $\delta * \delta$ to partition the space area into regular grid cells, all trajectory data should be allocated to at least one grid. It can be easy to explain that a trajectory line segment is assigned to the grid if it is fully covered by one grid. There is another situation, the trajectory should be split at the boundary if the trajectory spans a spatial grid boundary, and then should be inserted into two grids. The grid index is managed with key-value pairs, and the trajectory segments in the same grid are stored in the area mapping the same grid key.

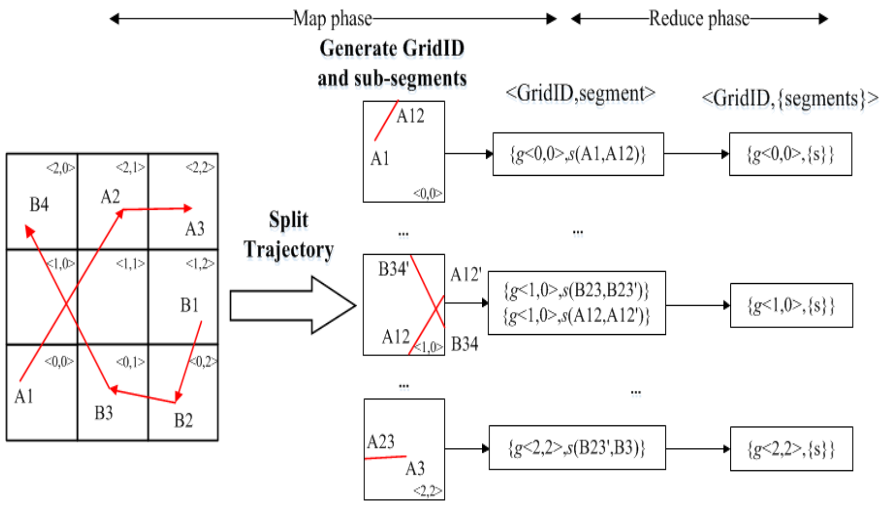

Figure 3. Process of Building Grid Index

Figure 3 shows the process of building grid index with MapReduce model. After the preprocessing phase, each input file contains one trajectory per line, and then we build the trajectory grid index on the input file. In map phase, each datanode reads and splits a trajectory *Tr* into m sub-trajectory segments on the basis of its spatial dimension, and transforms m trajectory segments to a list of $\langle GridId, Tr_i^{sub}\rangle, i \in [1,\ldots,m]$, such as $\{g\langle 0,0\rangle, s(A_1, A_{12})\}$, $\{g\langle 1,0\rangle, s(A_{12}, A_{12'})\}$. In key-value pair, the key is the *GridId* of the grid index while the value is sub-trajectory belonging to this grid. Before reduce phase, we reconstruct the custom partitioning strategy of MapReduce to make the grid id $GridIdPair(x,y)$ sorted and split in the order of x value, and then output to different reduce nodes to group. In reduce phase, reducer groups trajectory segments with the key of $GridIdPair(x,y)$ and outputs as $\langle GridIdPair, Set\{\text{segment}\}\rangle$, such as $\{g\langle 1,0\rangle, \{s(B_{23}, B_{23'}), s(A_{12}, A_{12'})\}\}$. Algorithm1 gives the details of the grid index building process.

Algorithm 1 Grid Index Build Method
Input: Trajectory dataset
Output: Grid index files

```
1.   procedure MAP(k1,v1)
2.       trSegMap ← SpatialPartition(v1); //Split trajectory to grids
3.       for each trSeg ∈ trSegMap do
4.       k2 ← trSeg.gridIdPair, v2 ← trSeg
5.       OUTPUT(k2,v2);
6.       end for
7.   end procedure

8.   procedure REDUCE(k2,v2)
9.       sort(k2); //Sort the GridId with order of x
10.      OUTPUT(k2,v2); // Output to different grid files
11.  end procedure
```

4.3. Trajectory Rebuild Table

With the purpose of retrieving and rebuilding the entire trajectory when processing a k-NNT query, we have to keep track of the sub-trajectory segments on the grid index. As a result, we propose Trajectory Rebuild Table (TRT) which is a table-like structure and managed with key-value pairs. The key in the pair is a trajectory id while the value is a list of references to grid index which the trajectory crosses. We could load and store the TRT to memory in map-setup() phase in order that all the nodes could read it to retrieve and rebuild the entire trajectory before finding the candidate trajectory. The MapReduce program idea is as follows. In map phase, each datanode splits a trajectory and maps it to a list of pairs $\langle TrId, GridIdPair\rangle$, which is composed of trajectory identifier and the grids it overlaps with. In reduce phase, the reducer groups *GridIdPair* with the key of trajectory identifier and outputs as $\langle TrId, Set\{GridIdPair\}\rangle$.

4.4. k Nearest Neighbor Trajectory Query

In this section, we introduce the MapReduce process of the k-NNT query. The CircularTrip algorithm [13] is usually used as an efficient method to access around grids. However, the traditional CircularTrip algorithm doesn't support the distribution and trajectory query. Therefore, we propose grid-based k-nearest neighbor trajectory query algorithm under MapReduce framework and denote the approach as MR-GB-KNNT.

Before searching candidate trajectories in k-NNT, we locate the input query trajectory into the grids and then determine the center of these grids in map-setup(). Around this center, the CircleTrip() algorithm is executed once by default to find the candidate grid set *candidateGridList*. After that, we also load TRT from HDFS to memory in map-setup() for avoiding reading duplication in every map(). Besides, we rewrite the class RecordReader in Hadoop and reconstitute the map method to read one index file rather than one line at one time. In map phase, the datanode reads the whole grid index file and finds the candidate trajectory id set with no duplication at the first time. CircularTrip() is executed continuously to find candidate trajectories until $candidateNum \geq K\_NUM$ we want to retrieve. And then, it would search the rebuildTable to get the entire trajectory. In reduce phase, we provide *treeMap* to store the trajectory segment and restore to full trajectory so that we can easily compute the distances between the entire candidate trajectories and our input trajectory. Finally, we sort and output k nearest neighbors to HDFS. The following is the pseudo-code for MR-GB-KNNT.

Algorithm 2 MR-GB-KNNT
Input: Grid index file
Output: k nearest neighbor trajectories

```
1.  procedure MAP-SETUP(k1,v1)
2.      inputGridSet←Φ; candidateGridList←Φ;
3.      centerX=0; center=0;
4.      inputGridSet=locTraj2Grid(inputTraj); // Locate the grids of input
    trajectory;
5.      findCenterGrid(inputGridSet,centerX,centerY);
6.      candidateGridList.addAll(CircularTrip(1)); // Execute CircularTrip once
    by default;
7.      readRebuildTableFromHDFS(context);
8.  end procedure
9.  procedure MAP(k1,v1)
10.     candidateNum=0;wholeFileString=null;candidateIdSet←Φ;
11.     // Load grid index file;
12.     wholeFileString=loadWholeIndexFile();
13.     //find id of the candidate trajectory segment;
14.     candidateIdSet=traverseFileString(wholeFileString,candidateGridList);
15.     while candidateNum < K_NUM do
```

```
16.             candidateIdSet.addAll(traverseFileString(wholeFileString,
17.             CircularTrip(++cycle_num))); //Execute CircularTrip until find K
    result;
18.         end while
19.         //find other traj segments by trajId in TRT
20.         findRebuildEntireTraj(candidateTrajectoryIdSet,wholeFileString,context);
12.         k2 ← trId, v2 ← traj segment;
21.         OUTPUT(k2,v2);
22.     end procedure
23.     procedure REDUCE(k2,v2)
24.         tempMap←Φ;
25.         for each seg ∈ v2s do  // Use TreeMap to restore the entire traj
26.             tempMap.put(trId, seg);
27.         end for
28.         calculateTraj2TrajDist(tempMap,inputTrajectory);// Compute aggregated
    distances;
29.         select and ouput k nearest neighbors to HDFS;
30.     end procedure
```

5. Experimental Evaluation

5.1. Experiment Setting

The experiment chooses MR-Base-KNNT discussed in Section 3 for comparison. The comparison focuses on the value of k number of nearest neighbors, the size of dataset, the width of cell in grid and the length of input query trajectory.

Experiments are performed on a Hadoop cluster with 9 nodes, one is Master node, and the others are as Slave nodes. The specific cloud computing experimental environment is shown in the Table 1.

Table 1. Experimental Environment

CPU	OS	Memory	Hadoop	Programming Language	Master	Slaves
2.2GHz	CentOS6.5	8G	2.5.2(64bit)	jdk1.7.0_07	1	8

We use Beijing Taxi Dataset for our experiment: one is from Microsoft GeoLife(DS1)[14] and the other is from DataTang (DS2) [15]. The size of DS1 and DS2 is 2G and 30G, respectively. The DS1 contains trajectories with a total distance of 1,251,654 kilometers and a total duration of 48,203 hours. The DS2 is collected by 12000 taxis in a period of October-December 2012 and is about 4.5 million trajectories. They are both the taxi trajectory data and have similar attributes, such as the trajectory id, latitude, longitude, timestamp, speed, *etc*.

5.2. Experimental Result Analysis

In this experiment, we evaluate the effect of the number of query trajectory we want to retrieve on query performance. By default, we choose dataset DS1, and the width of grid δ is 0.01. As can be seen in Figure 4, MR-GB-KNNT always outperforms MR-Base-KNNT as k grows because MR-GB-KNNT searches only part of the candidate set rather than the whole dataset. For MR-GB-KNNT, when k is small, the number of candidate trajectories found by CircularTrip() at first time is always more than k and there is no need to execute more CircularTrip(). So it takes almost same time. However, when k raises, the number of candidate trajectories becomes less than k, it costs more time on executing more CircularTrip(). But for MR-Base-KNNT, the query time is always growing slowly.

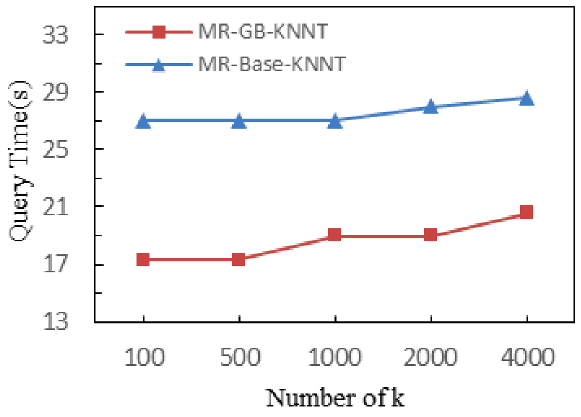

Figure 4. Effect of Query Number of k on the Query Time

In this experiment, we evaluate how the size of dataset affects the query time of the system. By default, we use dataset DS2, δ is 0.01 and k is 20. Figure 5 shows the overall results of the experiment. The response time of the two approaches would gradually increase when the amount of data continues to increase, but the increase in MR-GB-KNNT is relatively small. That is because that our grid index helps us locate and search only part of the candidate set, while MR-Base-KNNT always searches for the whole dataset. The scalability of our MR-GB-KNNT is better.

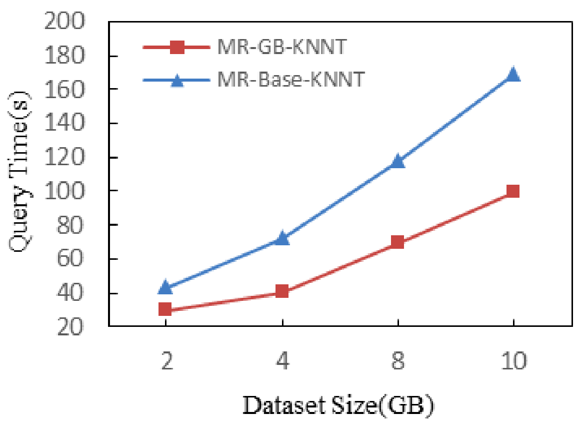

Figure.5 Effect of Data Size on the Query Time

Figure 6 shows the system query time on different width of grid and dataset. By default, we randomly select the sub-dataset with the same size(2G) from two datasets, and k is 20. The value of δ depends on the distribution of the data in the specific dataset. In the experiment, we take δ as 0.1, 0.01, 0.001, and 0.0005, which is the measure of latitude and longitude. We can see that when the width of cell is 0.1, the grid is too sparse to store so many trajectories and make CircularTrip() scan a large set of candidate trajectories. So it takes the most time. However, the smaller δ is, the more grid files it has. If we choose 0.0005, the cluster initializes too much splits which cause performance degradation. As a result, it is important to choose an appropriate grid width, such as 0.01.

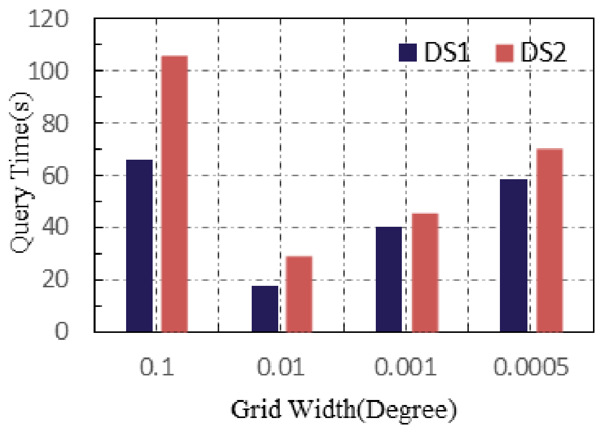

Figure 6. Effect of Width of Grid on the Query Time

Figure 7 demonstrates the performance of two approaches with different length of input query trajectory. By default, we use dataset DS1, δ is 0.01 and k is 20. For MR-GB-KNNT, the effect of trajectory length on the query time is small, but it crosses more grids when the length increases, causing more candidate grids to be scanned and more query time. However, for MR-Base-KNNT, the longer trajectory length is, the more query time it costs. That is because of the dramatic increase of computational cost in global scanning.

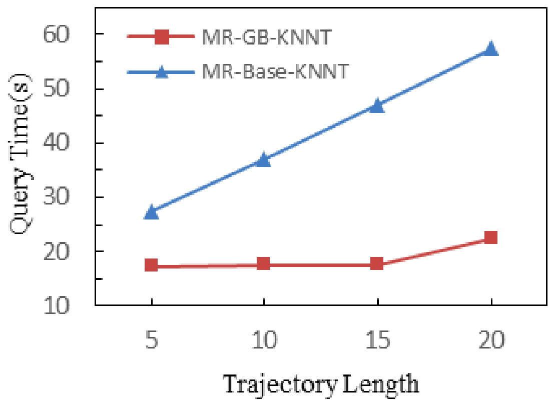

Figure 7. Effect of Trajectory Length on the Query Time

6. Conclusion

In order to support the k-NNT query for large-scale trajectory data efficiently in cloud computing environment, we propose a distributed trajectory grid index, which is a spatial data partitioning approach under MapReduce framework and divides the space into multiple grid units. We also represent the details of how to perform distributed k Nearest Neighbor Trajectory queries. As well, we apply the Trajectory Rebuild Table to retrieve and rebuild the entire trajectory. In addition, we propose MR-GB-KNNT algorithm based on the grid index using MapReduce, which locates the query trajectory and executes CircleTrip() to find the candidate trajectories in map() and gets the k nearest neighbor trajectories in reduce(). The experiment based on the two datasets demonstrates the MR-GB-KNNT could improve the performance of the query and has good scalability for spatial trajectory data.

Acknowledgment

This work is funded by Natural Science Foundation of China (41571401), Chongqing Natural Science Foundation (cstc2014kjrc-qnrc40002), Scientific and Technological Research Program of Chongqing Municipal Education Commission (KJ1500431), and WenFeng Creative Foundation of CQUPT (WF2014-05).

References

[1] Z. Feng and Y. Zhu, "A Survey on Trajectory Data Mining: Techniques and Applications", IEEE Access, vol. 4, **(2016)**, pp. 2056-2067.

[2] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang and W. Yi, "Processing moving k NN queries using influential neighbor sets", Proceedings of the Vldb Endowment, vol. 8, no. 2, **(2014)**, pp. 113-124.

[3] Q. Ma, B. Yang, W. Qian and A. Zhou, "Query processing of massive trajectory data based on mapreduce", International CIKM Workshop on Cloud Data Management, Clouddb 2009, Hong Kong, China, **(2009)**.

[4] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng and X. Xie, "Searching trajectories by locations: an efficiency study", Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, Indianapolis, USA, **(2010)** June 06-10.

[5] L. A. Tang, Y. Zheng, X. Xie, J. Yuan, X. Yu and J. Han, "Retrieving k-Nearest Neighboring Trajectories by a Set of Point Locations", Advances in Spatial and Temporal Databases - International Symposium, SSTD 2011, Mn, USA, **(2011)** August 24-26.

[6] S. Qi, P. Bouros, D. Sacharidis and N. Mamoulis, "Efficient Point-Based Trajectory Search", Proceedings of the 14th International Congress of Advances in Spatial and Temporal Databases, Hong Kong, China, **(2015)** August 26-28.

[7] W. Lu, Y. Shen, S. Chen and B. C. Ooi, "Efficient processing of k nearest neighbor joins using MapReduce", Proceedings of the Vldb Endowment, Vol. 5, No. 10, **(2012)**, pp. 1016-1027.

[8] C. Ji, Z. Li, W. Qu, Y. Xu and Y. Li, "Scalable nearest neighbor query processing based on Inverted Grid Index", Journal of Network & Computer Applications, Vol. 44, No. 9, **(2014)**, pp. 172-182.

[9] Y. Fang, R. Cheng, W. Tang and S. Maniu, "Scalable Algorithms for Nearest-Neighbor Joins on Big Trajectory Data", IEEE Transactions on Knowledge & Data Engineering, Vol. 28, No. 3, **(2015)**, pp. 785-800.

[10] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A MapReduce framework for spatial data", IEEE International Conference on Data Engineering, Seoul, South Korea, **(2015)** April 13-17.

[11] S. Arumugam and C. Jermaine, "Closest-Point-of-Approach Join for Moving Object Histories", International Conference on Data Engineering, Atlanta, Georgia, **(2006)** April 3-7.

[12] L. Chen, M. T. Zsu and V. Oria, "Robust and fast similarity search for moving object trajectories", ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, **(2005)** June 14-16.

[13] M. A. Cheema, Y. Yuan and X. Lin, "Circulartrip: an effective algorithm for continuous kNN queries", International Conference on Advances in Databases: Concepts, Systems and Applications, Bangkok, Thailand, **(2007)** April 9-12.

[14] https://www.microsoft.com/en-us/download/details.aspx?id=52367.

[15] http://www.datatang.com/data/45888/.

Authors

Ying Xia, a professor of Chongqing University of Posts and Telecommunications. Her research area mainly includes database and data mining, spatial information processing, etc.

E-mail: xiaying@cqupt.edu.cn

Ruidi Wang, a graduate student at the College of Computer Science and Technology, Chongqing University of Posts and Telecommunications. His research area mainly includes spatial information processing and cloud computing.
E-mail: wangruidi.jn@foxmail.com

Xu Zhang, an associate professor of Chongqing University of Posts and Telecommunications, received Ph.D degree from Inha University, South Korea. His research area mainly includes large scale data processing, database, etc.
E-mail: zhangx@cqupt.edu.cn

Hae Young Bae, is tenured full professor of Inha University of Korea and honorary professor of the Chongqing University of Posts and Telecommunications of China. His research area mainly includes database and spatial information processing.
E-mail: hybae@inha.ac.kr