Parallel Optimization of KNN Query Strategy based on Road Network

Boqi Hu\textsuperscript{a}, Hailong Sun\textsuperscript{a,b,*}, Fangsong Li\textsuperscript{b}, Chao Jiang\textsuperscript{b}, and Weitao Zou\textsuperscript{a}

\textsuperscript{a}College of Information and Computer Engineer, Northeast Forestry University, Harbin, 150040, China
\textsuperscript{b}Computing Center of Heilongjiang Province, Harbin, 150026, China

Abstract

K-nearest neighbor (KNN) query is one of the most important query types in spatial databases and have been widely used in intelligent transportation, roadside assistance, and other fields. In order to improve the query efficiency, in this paper we adopted the MapReduce parallel computing framework of the Hadoop large data processing platform and completed the query of K neighbor moving objects by designing Map, Reduce, Combiner, and other functions. Before the start of the query, the road network was divided into pieces, and each fragment was calculated. The final K-nearest neighbor moving objects were obtained by aggregating the calculated results of each slice to realize the parallel optimization of the KNN algorithm based on road network. The experimental results showed that the performance of the parallel KNN algorithm based on MapReduce was better than that of the serial KNN query algorithm in a large-scale road network environment and a larger K value of query requests.

Keywords: KNN query; parallel computing; MapReduce; moving object; road network

(Submitted on July 12, 2018; Revised on August 15, 2018; Accepted on September 10, 2018)

© 2018 Totem Publisher, Inc. All rights reserved.

1. Introduction

With the rapid development of technologies such as wireless communication, computing technology, and GPS space positioning, mobile phones and vehicle-mounted devices with positioning function are rapidly applied to daily life and are popularized on a large scale. Location Based Service has become a promising new industry that has received more and more attention, especially in the urban road network environment. In this environment, many new applications have been proposed, such as roadside assistance, traffic navigation, and location aware advertising services, which have aroused people’s widespread interest because they suit users’ actual demands.

Query services, such as K-nearest neighbor query in the road network environment, are increasingly being used by people and are becoming one of the most important query services. Location services, such as “Search the nearest 20 taxis”, meet the practical needs of people, are rapidly entering people’s lives, and in recent years have been widely concerned about and applauded application technology. At present, research on the serial KNN query algorithm based on road network has many production researches. Mouratidis [1] pioneered incremental network extension and group monitoring methods. Wang [2] proposed the MOVNet framework and MKNN algorithm and used the R-tree structure of road network based on disk and memory-based mobile object location grid index structure for continuous K-nearest neighbor query processing. Demiryurek [3] proposed the ER-CKNN algorithm. Liao [4] proposed a new road network directed graph model for continuous K-nearest neighbor query based on road network, which uses the memory-based hash tables and linear linked list structures to store and manage the current location of the moving object and the directed graph model of the road network. Through the introduction of one-way network distance measurements and two-way network distance measurements, the UNE algorithm and BNE algorithm were proposed to support continuous K-nearest neighbor query processing with different semantics and uses the influence tree and network expansion strategy to reduce the search cost of continuous K-nearest neighbor query updates.

The results of the parallel KNN query algorithm based on road network are limited. Sabeur [5] studied the shortest path problem of a large-scale real road network based on the Hadoop cloud computing model, but there are no further studies on

* Corresponding author.
E-mail address: sun_hai_long2000@163.com

In this paper, we optimize the KNN query based on the urban road network environment. In order to solve the problem of inefficient processing of large-scale data, this paper proposes a distributed parallel computing solution strategy that integrates the KNN algorithm with the MapReduce distributed parallel computing framework of the Hadoop large data processing platform to solve the problem that the computing efficiency is greatly reduced in the dense road network.

2. Related Work and Problem Description

2.1. MapReduce

MapReduce is a distributed parallel computing model that was proposed in Google and is designed to solve the parallel computing problem of large-scale data sets. The basic principle is that the large-scale data set is divided into small data sets firstly, and then the data slices are distributed to the nodes that perform the calculation. The data is processed and summarized by the user-written Map function and Reduce function to obtain the final result. Throughout the calculations, the data is passed and saved as <key, value> key-value pairs.

Figure 1 shows the MapReduce detailed workflow.

(1) Splitting a data set stored on HDFS (split). Usually, the split size is 64MB, and other values can be set.

(2) Job Tracker assigns Map tasks and Reduce tasks to the cluster. The number of Map tasks is determined by the number of splits in the process (1), and the number of Reduce tasks can be set by the programmer.

(3) Read the entire row of data and the execution of Map function, and the data will be parsed into a key-value pair and cached in memory.

(4) Classify Map results by key, sorting, and summarizing Generating class results like <key, list <value >>. Key-value pairs of the same key value are distributed to the same Reduce task.

(5) The Reduce task summarizes the results of the Map task and outputs the final calculation results. MapReduce work is completed.

In summary, the efficient Map function and Reduce function is the core of MapReduce work [9]. The data is processed into key-value pairs that can be handled by the MapReduce distributed parallel computing framework. The intermediate result in the form <key, list <value >> are obtained by the shuffle, sort, and other processes. The intermediate results of the same key value are passed to the reduce function, and the final solution is calculated. Here, to improve the computational efficiency of the Reduce function, you can set the Combiner procedure after the Map function is executed and before the
Reduce function is executed. The essence of the Combiner process can be understood as a Reduce function that works in the Map process, aggregating the intermediate results before entering the Reduce function. The principle of the Combiner function is to ensure that the total time efficiency of the MapReduce process is not affected by the time efficiency and that the correctness of the final output result is not affected. The MapReduce distributed parallel computing framework requires only programmers concerned about the Map function and Reduce function, which reduces the difficulty of programming and greatly improves the computational efficiency in large-scale data sets [10].

2.2. Road Network

The road network model plays an important role in the nearest neighbor query. Building a road network model is the use of nodes and edges that represent the real path and are stored in the computer. Building a road network requires consideration of road level, different parts of road density, and other macro factors as well as micro factors such as crossroads and one-way lines. The relevant definitions in the road network model are as follows:

Definition 1 [11] (Road Network): The road network G is defined as a two-tuple G = (R, J), where R is the set of routes in the road network, each route contains several links, and J is the intersection of the multiple routes in the road network.

Definition 2 (Route): A route is a complete path that can be independently named in a road network, defined as:

\[ r = (\text{rid}, \text{len}, (j_{id}, \text{pos})_{j \in J}) \]

Where \( \text{rid} \) is the route identifier and \( \text{len} \) indicates the route length, \( \text{len} \in [0,1] \). \((j_{id}, \text{pos})_{j \in J}\) is the set of locations on the route’s intersection, and its relative starting point on the route, \( \text{pos}_{j} \in [0,1] \).

Definition 3 (Segment): A road segment is a route between adjacent intersections, defined as:

\[ \text{seg} = (\text{sid}, \text{rid}, p_{s}, p_{e}, \text{dir}) \]

The \( \text{sid} \) and \( j_{id} \) respectively represent the road section and the identity of the route, \( p_{s} \) and \( p_{e} \) respectively indicate the start and end of the link. For \( \text{dir} \in \{-1,0,1\} \), a value of 1 indicates that the moving object is allowed to move from the starting point to the ending point on the link, a value of -1 means moving from the end point to the starting point of the road segment, and a value of 0 means that the road section permits bidirectional traffic.

Definition 4 (Intersection): Intersection refers to the intersection of multiple routes, defined as:

\[ j = (j_{id}, (\text{rid}, \text{pos})_{j \in J}^{m}, \text{adjList}) \]

Where \( j_{id} \) is the identifier of the intersection and \((\text{rid}, \text{pos})_{j} \) is the intersection’s position on the \( j^{th} \) route. \( \text{adjList} \) is the adjacency list of the intersection and stores the connection relation of each link at the intersection.

Definition 5 (Moving Objects) [12]: In a road network, a moving object is modeled as:

\[ o = (\text{oid}, x, y, \text{rid}, \text{pos}, \text{dir}) \]

Where \( \text{oid} \) and \( \text{rid} \) respectively represent the moving object and its alignment. \( x \) and \( y \) respectively represent the latitude and longitude coordinates of the moving object. \( \text{pos} \) represents the distance from the starting point of the route where the moving object is located, \( \text{pos} \in [0,1] \). \( \text{dir} \) indicates the moving direction of the moving object: a value of 1 indicates that the moving object is allowed to move from the starting point to the ending point on the link, while a value of -1 means moving from the end point to the starting point of the road segment.

3. Research Content and Methods

The goal of this paper is to improve the query efficiency of K-nearest neighbor by combining the K-nearest neighbor query with the parallel computation method. The basic research idea is to split the road network, calculate the K-nearest neighbor query results for each subnet in the parallel framework, and summarize the final query results.
3.1. Road Network Division

To solve the parallel optimization problem of the large-scale network path, it is necessary to divide the network effectively and build the subnet [13]. Network partitioning is the process of dividing the huge road network into several subnets and handing these subnets to each task node for calculation. Each node computes the K-nearest neighbor in the current subnet.

The following matters need to be considered during network segmentation: (1) If the vertex is used as the segmentation, the number of vertexes on the subnet boundary needs to be as small as possible. If an edge is used as a partition, the number of edges of the connection needs to be as small as possible. This is related to the cost and efficiency of parallel computing. (2) In the partitioning process, the number of vertices in each subnet should be as even as possible to improve computing efficiency. Based on the above principles, we chose the K-Metis method, which is used in reference [14] to segment the road network. The division is shown in Figure 2.

3.2. KNN Parallel Computing

In this paper, the road network is partitioned based on the MapReduce parallel computing framework. A K-nearest neighbor query is designed for each partition, and then the result of each partition query is aggregated to form the final results. As mentioned earlier, the sorting phase of MapReduce is independent of the previous query phase. The KNN algorithm takes advantage of the feature, the sort process, and the calculation process on different nodes of the cluster, which significantly improves the computational efficiency. This feature is more pronounced when the data size and K value are larger.

The basic idea of parallel KNN query based on MapReduce is the following: firstly, the road network is divided according to the number of compute nodes of the Hadoop experimental cluster. The number of partitions is equal to the number of compute nodes. Each partitioned data is copied to the corresponding compute node. Secondly, the R-Tree index structure is queried to determine the location of the query point q, and the K candidate sets in each partition are calculated as the output of the Map task by using the adjacency list stored by each compute node. Finally, each Map is exported to the Reduce task to obtain the query result. The KNN parallel query pseudo-code as shown in Algorithm 1.

<table>
<thead>
<tr>
<th>Algorithm 1 ParallelKNNs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> query node q, query number K, road network data</td>
</tr>
<tr>
<td><strong>Output:</strong> KNNs</td>
</tr>
<tr>
<td>1. input q, K, data;</td>
</tr>
<tr>
<td>2. splitRoadNetwork (data);</td>
</tr>
<tr>
<td>3. search(R-tree);</td>
</tr>
<tr>
<td>4. For each Map task do</td>
</tr>
<tr>
<td>5. computeCandidateKNNs ();</td>
</tr>
<tr>
<td>6. End For</td>
</tr>
<tr>
<td>7. Reduce:</td>
</tr>
<tr>
<td>8. getCandidateKNNsAndSort ();</td>
</tr>
<tr>
<td>9. KNNs=getFinalResult ();</td>
</tr>
<tr>
<td>10. return KNNs;</td>
</tr>
<tr>
<td>11. <strong>end</strong></td>
</tr>
</tbody>
</table>

3.2.1. Map Function to Achieve Partition Calculation

The Map process reads the split road network data and calculates the K-nearest neighbor query results in the subnet. The result is stored as a key-value pair and passed to the reduce function for aggregation. The corresponding pseudo-code is shown in Algorithm 2.

<table>
<thead>
<tr>
<th>Algorithm 2 ComputeCandidateKNNs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> &lt;key, value&gt; Split road network data</td>
</tr>
<tr>
<td><strong>Output:</strong> &lt;key, vector&gt; Context context</td>
</tr>
<tr>
<td>1. For i=0 to n (training data set) do</td>
</tr>
<tr>
<td>2. buildRoadNetwork (i);</td>
</tr>
<tr>
<td>3. End For</td>
</tr>
<tr>
<td>4. For i=0 to m (moving Objects) do</td>
</tr>
<tr>
<td>5. dis=Dijkstra (i);</td>
</tr>
<tr>
<td>6. context.Write (key, vector (i, dis));</td>
</tr>
<tr>
<td>7. End For</td>
</tr>
<tr>
<td>8. <strong>end</strong></td>
</tr>
</tbody>
</table>

Here, the distance of moving objects obtained from the Map task must filter out the K results as CandidateKNNs. At this point, there are two possibilities. If the number of results calculated by Map task ≤ K, pass the result directly to the
Reduce program. If the number of results ≥ K, filter out the K of KNN query results as the CandidateKNNs and then pass to the Reduce function. Figure 3 shows the Combiner process.

![Figure 2. Road network division](image)

![Figure 3. Combiner process](image)

### 3.2.2. Combiner Function Design

As described in Section 3.2.1, the Combiner function is responsible for filtering out CandidateKNNs from the results of the Map task. The Combiner function itself can be understood as the Reduce task running in the Map side which reduces the communication overhead of the Map output to the Reduce input. The pseudocode is shown in Algorithm 3.

**Algorithm 3 Shuffle**

**Input:** <key, vector>

**Output:** <key, vector> Context context

```
/*
* Sort the value and get the nearest K
* Values if the result number > K
*/
1. If result number > K do
2. For all key and value do
3.   ArrayList.add (value);
4.   End For
5.   Sort (ArrayList);
6.   For i=0 to k do
7.     context. Write (ArrayList.get (k));
8.   End For
9.   End If
10. end
```
3.2.3. Reduce Function Design

The Reduce function is responsible for summarizing the output of the Map function to produce the final result. The Reduce function obtains the key-value pair (<key, vector>) of the same key generated by each Map task node, and the final result is stored as a key-value pair in the HDFS. Its code is similar to the Combiner function as shown in Algorithm 4.

Algorithm 4 GetCandidateKNNsAndSort

<table>
<thead>
<tr>
<th>Input: &lt;key, vector&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: &lt;key, value&gt; Context context</td>
</tr>
<tr>
<td>1. For all key and value do</td>
</tr>
<tr>
<td>2. ArrayList.add (i);</td>
</tr>
<tr>
<td>3. End For</td>
</tr>
<tr>
<td>4. Sort (ArrayList);</td>
</tr>
<tr>
<td>5. For i=0 to n do</td>
</tr>
<tr>
<td>6. context.Write (ArrayList.get (i));</td>
</tr>
<tr>
<td>7. End For</td>
</tr>
<tr>
<td>8. end</td>
</tr>
</tbody>
</table>

4. Comparative Analysis of Experimental Results

4.1. Experimental Environment

This experiment used a server-level virtual software xenserver6.2 virtual 4 Sugon I450-G10 tower server (Inter Xeon E5-2407 Quad core 2.2GHZ CPU, 8GB RAM) into 16 hosts and one HP Compaq dx 2308 (Intel Pentium E2160 1.8GHZ CPU, 1G RAM) as master. The Hadoop 2.2.1 cloud cluster was set up on the Cenos6.4 final (Kernel 2.6.32) system. The cluster consists of 16 Data Nodes and 16 Task Trackers, and the HDFS storage space is 3.34T.

4.2. Experimental Data Set

The road network data from Los Angeles and San Francisco, California, USA, was downloaded from TIGER / Line. The former consists of 79,800 nodes and 110,000 edges and the latter consists of 9,322 nodes and 14,809 edges. The downloaded .shp format data generated the node data and the edge data as the input of the moving object generator. The mobile object generator designed by Brinkhoff [15] was used to obtain a point-of-interest (POI) object set based on road network. In this experiment, two types of data sets with large differences in size were used to test the efficiency difference between the traditional KNN query strategy and the KNN query strategy with parallel computing under different data scales.

4.3. Experimental Comparative Analysis

Experiments were carried out to compare the efficiency of the ParallelKNNs algorithm in this paper and the IMA/GMA algorithm when executing a KNN query by changing the number of POI, nearest neighbor number K, road network size, and other parameters. The default values of the experimental parameters are shown in Table 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Defaults</th>
<th>Range of parameter values</th>
</tr>
</thead>
<tbody>
<tr>
<td>POI number</td>
<td>15K</td>
<td>5,10,15,20,25(K)</td>
</tr>
<tr>
<td>KNN number</td>
<td>30</td>
<td>10,20,30,40,50</td>
</tr>
<tr>
<td>Road Scale</td>
<td>50K</td>
<td>5,10,30,50,60,70,80(K)</td>
</tr>
</tbody>
</table>

4.3.1. The Effect of the Number of POIs on Query Time

Figure 4 shows the comparison of CPU response times for the ParallelKNNs algorithm and IMA / GMA algorithm in LA and the SF road network environment, when the number of POI objects is evenly distributed in the range of 5K to 25K.

As can be seen from Figure 4, the CPU response time is inversely proportional to the number of POI objects. With the increase in the number of POI objects, the CPU response time decreases significantly; however, when the POI number reaches 20K or more, the variation tends to be milder. In addition, the road network size affects the CPU response time. The performance of the ParallelKNNs algorithm is better than that of the IMA / GMA algorithm in a large-scale road network environment like LA. In the small-scale road network environment of SF, the performance of the two algorithms is very close. When the number of POIs exceeds 15K, the KNNs performance of the IMA / GMA algorithm is even higher than that of the ParallelKNNs algorithm because in small-scale networks, when the number of POIs increases, the speed of the KNN
is increased by the network expansion method. The ParallelKNNs algorithm needs to partition the road network and schedule Map, Reduce, and other tasks, which increases the computing time.

4.3.2. The Effect of the Nearest Neighbor K on the Query Time

Figure 5 shows the effect of the number of nearest neighbors K on the query performance of the ParallelKNNs algorithm and IMA/GMA algorithm in LA and the SF road network environment.

![Figure 5](image)

It can be seen from Figure 5 that as the number of nearest neighbors K increases, the CPU response time also increases. The performance of the ParallelKNNs algorithm is much better than that of the IMA/GMA algorithm in the LA large-scale road network environment. In the SF small-scale road network environment, when the number of nearest neighbors K is greater than 30, the performance of the ParallelKNNs algorithm can be better than that of the IMA/GMA algorithm.

From the experimental results, it can be explained that the ParallelKNNs algorithm does not show good performance under any circumstances. It shows the advantages of parallel computing only in a large-scale network environment or when the number of query requests is very large.

5. Conclusion

In this paper, we focus on parallel optimization of KNN query strategy based on road network. It was proposed to optimize the efficiency of KNN query by partitioning the road network and cluster parallel computing in a large and complex road network environment. Over the course of the experiment, we segmented the complex road network data to create a road network subnet and stored it in HDFS. We also made full use of the MapReduce parallel computing framework independent of each computing process features and summarized the results of the road network subnetwork query in each task node to obtain the final KNN query result. The experimental results showed that the proposed parallel KNN algorithm was superior in a large-scale road network environment or when the number of query requests is large. Future research will focus on reducing the consumption of different tasks during the switching process, such as the IO consumption during data disk read. In addition, whether to use other ways to optimize the road network model and whether to use the current popular NoSQL database storage query needs and the resulting data should also be explored in depth.
Acknowledgement

This work was partly financially supported by the Fundamental Research Funds for the Central Universities (No. 2572017CY09) and the National Undergraduate Innovation Training Program (No. 201510225046). The authors thank the anonymous reviewers for their hard work.

References


Boqi Hu will receive his B.S. degree in the College of Information and Computer Engineering at Northeast Forestry University, China in July 2018. Currently, he is an intern at an IT company in Beijing. His research areas include big data analysis and mobile application development.

Hailong Sun received his M.S. degree in Computer Science from Qiqihar University in April 2006 and his Ph.D. in Forestry Engineering from Northeast Forestry University in July 2014. He is currently a teacher at the Dept. of Computer Science and Technology at Northeast Forestry University, China. His current research interests include moving objects databases and big data mining.

Fongsong Li is a head engineer of the Heilongjiang Province Computing Center and one of member of the expert group on manufacturing informatization in Heilongjiang Province. His research interests include enterprise management informatization and parallel and distributed computing.

Chao Jiang is an engineer of the Heilongjiang Province Computing Center. His research interests include computer software and scientific research management informatization.

Weitao Zou is a college student and has received the master’s quota of the College of Information and Computer Engineering at Northeast Forestry University. His research areas include big data analysis and mobile application development.