Locating Traffic Hot Routes from Massive Taxi Tracks in Clusters
Zhiming Gui, Haipeng Yu, Yunlong Tang
School of Computer Science, Beijing University of Technology, Beijing, 100124, China

Abstract: The increasing availability of location-acquisition technologies has resulted in huge volumes of trajectories. The sheer volume of these data sets prevents their processing by traditional centralized technologies. In this paper, we propose a MapReduce-based extraction-and-group framework to locate traffic hot routes from taxis' track. In the proposed framework, massive trajectory data are partitioned into data chunks so that they can be processed in parallel on multiple machines. Then the low speed parts from each trajectory are extracted by a speed based clustering. Finally, a MapReduce inner-function based grouping method is used to locate traffic hot routes from all low speed parts. Based on this extraction-and-group framework, we develop a traffic hot route locating algorithm. The algorithm was evaluated through experiments on real life data sets, and was shown to have considerable potential to promptly and accurately locate traffic hot routes from massive trajectory data through various analyses on the experimental results.

Keywords: Trajectory clustering, Traffic hot route analysis, Map-Reduce, Cloud computing

1. INTRODUCTION

GPS-equipped taxis can be viewed as ubiquitous mobile sensors constantly probing a city’s rhythm and pulse, such as traffic flows on road surfaces and city-wide travel patterns. The large-scale digital tracks produced by them allow us to have a unique view of the underlying dynamics of a city’s road network [1]. Taxis' track data have been used for many applications of traffic management domain such as detecting hot spots [2, 3], urban computing [4, 5, 6, 7] and characterizing passenger finding strategies [8, 9, 10] amongst others.

Traditional hot spot detection related studies have two disadvantages. First, most studies adopt density-based clustering based on the Euclidean distance among points in trajectories. Since the GPS devices in taxis have different sampling frequencies as well as the signal loss problem, trajectory data sets usually have varying densities, which complicates the use of most Euclidean distance based density clustering methods. Furthermore, the dense areas identified by these kinds of algorithm have little practical meaning. For example, the green circled dots in Fig. 1 represent the dense parts got by DBCSAN, we could not give a precise definition for these areas. The precise meaning of these places is obscure. Second, traditional trajectory based traffic analysis methods are based on centralized technology and run on stand-alone machines. The huge trajectory data set often exceeds the main memory of most stand-alone computers. It prevents the efficient use of traditional centralized technologies. For example, statistics revealed that there are over 70 thousands taxis operating in Beijing. These taxis cover every corner of the city 24 hours a day and produce huge volume of trajectories. Facing trajectory data in huge volumes, cloud computing provides a promising paradigm to conquer the explosion of data and has shown good performance in many data intensive applications. Recently, the framework of MapReduce has gained a lot of attention around the world. It provides a framework that allows large clusters to easily process massive data sets. In this paper, we propose a MR-based extraction-and-group framework to locate traffic hot routes from taxi tracks. As indicated by its name, our traffic hot routes locating algorithm based on this framework consists of the following two phases:

Extraction phase: The low speed parts of each trajectory are extracted by speed-based clustering with consideration of traveled distance and time. Then we use the Douglas-Peucker algorithm to simplify the low speed parts. Thus each trajectory is partitioned into a set of line segments on which taxi are driving slowly. We call these line segments as low speed segment. The phase contributes to eliminate noise data and to prune middle points between low speed parts. These middle points are considered as useless for identifying traffic hot routes because taxis are moving quickly at these places.

Group phase: We identify the traffic hot routes via grouping on the low speed segments of all taxis. If the quantity of similar low speed segments is enough, we group them into a traffic hot route. Here, a frequent item mining algorithm for line segments is exploited. The algorithm is implemented easily via the auto sort/merge functions that are naturally contained in MR. In summary, the main contributions of this paper are as follows:

• We propose an extraction-and-group framework to locate traffic hot routes in parallel from massive taxi trajectory data.
• We apply MapReduce into our work and demonstrate that it represents good performance in massive trajectory data processing. We also argue that sort/merge functions in MapReduce can simplify the development of some frequent item mining algorithm.
• We propose a speed-based clustering algorithm to extract the useful parts of trajectories. We argue that this method can significantly diminish the negative effect of varying densities existed in Euclidean distance based clustering.

2. PRELIMINARY
2.1 Data Model and Traffic Hot Route

An original GPS trajectory \( Tr \) is recorded by a sequence of time-stamped points, \( Tr = p_0, p_1, \ldots, p_k \) where \( p_i = (x_i, y_i, t_i) \); \((x,y)\) are latitude and longitude respectively, and \( t \) is a timestamp. Due to the original trajectory sample points contain little semantic, it makes their analysis very complex from the application point of view. In our work, we adopt the popular Move-
Stop model [11] to preprocess trajectories. In this model, trajectories are observed as a set of stops and moves. A stop is a semantically important part of a trajectory that is relevant for an application, and where an object has stayed for a minimal amount of time. For instance, in a tourism application, a stop could be a touristic place, a hotel, an airport, etc. According to the application, the minimal stop duration can vary significantly. Moves are sub-trajectories between two consecutive stops.

From an application point of view, traffic hot route is a general path in the road network which contains heavy traffic flow. Speed plays an essential role in traffic hot route mining applications. In our work, we propose our trajectory model that uses the low speed parts series to represent the trajectory with the objective to find traffic hot routes. Informally, traffic hot routes can be described as road sections on which many vehicles are driving slowly during a certain period. So, we define the traffic hot route as the road sections which contain enough low speed segments.

To be formal, we assume there is a number $s$, called the support threshold. $R$ is a road section, $L_1, \ldots, L_n$ are low speed segments. The support for $R$ is the number of low speed segments that match $R$. $R$ is a traffic hot route if its support is $s$ or more. Figure 2 exemplifies two traffic hot routes. In figure 2, road segments are represented by thin-solid lines. The dotted lines represent the low speed segments extracted from trajectories. There are four low speed segments matching road segment A and five low speed segments matching road segment B. With a support threshold number of 4, the thick line A and B are traffic hot routes. It should be noted that due to road matching is time-consuming and not suitable for exploiting large scale parallelism, we use a similarity judgment function instead of road matching in our experiment.

Traffic hot routes are often around the potential sites of interests due to the higher likelihood of the events and opportunities (e.g., traffic jam, exhibitions, and commercial promotions). Taxi’s speed will be intuitively lower during driving in these places. So, the slow moving parts of a taxi track are most likely within a traffic hot route. Following this reason, we define the low speed parts of a trajectory as Stops. Since the low speed parts could not be predefined as the normal move-stop model requires, we use a speed-based clustering algorithm to find the stops in each trajectory.

Traffic hot routes mining study has many useful applications. An immediate application is that we can predict vehicle speeds based on the crowdedness distribution. Furthermore, matching traffic hot routes to known road segments or key locations helps in understanding of the city event such as sport games and exhibition.

Traffic hot route (THR) is a general path congested with vehicles. They are often around the potential sites of interest due to higher likelihood of events (e.g., traffic jam and commercial promotions). Taxi’s speed will be intuitively lower in these regions. So speed plays an essential role in THR mining application. In our work, we propose our trajectory model that uses LSS series to represent trajectories with the objective to find THR. Since the LSS could not be predefined as normal move-stop model requires, we use a speed-based clustering algorithm to find the “stop” in each trajectory. To be formal, we assume there is a number $s$, called support threshold. If $k$ is a LSS, the support for $k$ is the number of trajectories for which $k$ is a subset. We say $k$ is a THR if its support is $s$ or more. Figure 2 shows a simple road network and four trajectories. Road segments are represented by light shaded thin rectangles and trajectories by arrow lines. The solid lines represent the LSSs extracted from each trajectory. The dotted lines are normal parts. If the minimum support threshold is 0.5, the set of THR is \{A,B\}.

![Fig.2. An example of traffic hot routes in the road network.](image)

2.2 MapReduce

MapReduce provides a framework that easily enables to process massive amount of data over large clusters. Programs written according to MapReduce framework are automatically parallelized and executed on a large cluster of commodity machines. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

However, due to the default split strategy in MapReduce may corrupt the data integrity of ordered data, this framework does not directly support trajectory data, which consists of ordered data elements whose time dimension is monotonously increasing and whose latter location is dependent on the former one.

2.3 DBSCAN

Density-Based Spatial Clustering and Application with Noise (DBSCAN) is a clustering algorithm based on density. This algorithm is particularly suited to deal with large datasets, with noise, and is able to identify clusters with different sizes and shapes. DBSCAN’s definition of a cluster is based on the notion of density reachability. The key feature of DBSCAN is that for each object in a cluster, the neighborhood of a given radius has to contain at least a specified minimum number of objects.

Let $D$ be a data set of points and given parameters are $Eps$ and $MinPts$. $MinPts$ is a density measure that indicates the amount of points needed in a neighborhood of point in order to assign that point and its neighbors to a cluster. $Eps$ is a distance used to delimiting the neighborhood.
Then DBSCAN can be specified by the following definitions.
1. **Core object**: A point is a core point if there is enough number (MinPts) of neighbors in Eps semi-diameter.
2. **Eps-neighborhood** of a point p: denoted by \( N_{Eps}(p) = \{ q \in D | \text{dist}(p, q) = Eps \} \). The Eps neighborhood of \( p \) represents all points inside a circle centered in \( p \) and with radius \( Eps \). The \( \text{dist}(p, q) \) is a function that returns the distance between point \( p \) and \( q \), considering the Euclidean distance on a 2D surface.
3. **Directly Density Reachable**: A point \( p \) is directly density reachable to a point \( q \) with respect to \( \epsilon \) Minpts if:
   - \( p \in N_{Eps}(q) \)
   - \( |N_{Eps}(q)| \geq \text{MinPts} \)
4. **Density reachable**: A point \( p \) is density reachable to a point \( q \) with respect to \( Eps \), MinPts if there is a chain of points \( p_1, p_2, \ldots, p_n \), \( p_1 = q, p_n = p \), for each \( p_i, p_{i+1} \), is directly density reachable from \( p_i \).
5. **Density connected**: A point \( p \) is density connected to a point \( q \) with respect to \( Eps \), MinPts if there is a point \( o \) such that both \( p \) and \( q \) are density reachable from \( o \) with respect to \( Eps \), MinPts.
6. **Cluster**: A cluster \( C \), with respect to \( Eps \), MinPts is a non-empty subset of \( D \) satisfying the following conditions:
   - \( \forall p, q \in D \) and \( q \) is density reachable from \( p \) with respect to \( Eps \) and \( \text{MinPts} \) then \( q \in C \).
   - \( \forall p, q \in C \), \( p \) is density connected to \( q \) with respect to \( Eps \) and \( \text{MinPts} \).

However, DBSCAN cannot detect meaningful clusters in data of varying density, whereas trajectory data sets usually have varying densities. Furthermore, it only takes distance into account and takes no account of time. Thus the standard DBSCAN is unsuitable for our work.

### 3. PARALLEL EXTRACTION GROUPING FRAMEWORK

#### 3.1 Problem Statement
Given a set of small text files SF in which each file represents a taxi’s trajectory during a time period and each line in a file represents a GPS location with the format as (id, x, y, t), where \( id \) represents the identification of the taxi. The total size of these files is very large. Our work aims to find the traffic hot routes associated with different time period of a specified date from the data set in parallel.

#### 3.2 Overall Workflow
Figure 3 shows the overall procedure of trajectory clustering in our extraction-and-group framework. The step 1 is mainly response for identification of LSS. The step 2 is response for grouping all the similar low speed segments into traffic hot routes by counting the frequency of similar low speed segments. In our current work, the traffic hot route is represented by a line segment whose \( x, y \) value of each node is the mean \( x, y \) value of all similar low speed segments. We do not conduct the road match work at present.

#### 3.3 Extracting Low Speed Parts
To identify low speed part from trajectories, both the time and traveled distance are needed to be taken into account. In aforementioned definition of neighborhood in DBSCAN, it is defined as an area around points based on Euclidean distance. Here, we define the neighborhood of a point \( p_i \) as the set of points before and after \( p_i \) in the trajectory whose reporting time is within the \( \epsilon \) time interval.

\( \epsilon \) time neighborhood of a point: Let \( T_r \) be a trajectory as: \( T_r = \{ p_0, p_1, \ldots, p_k, p_{k+1}, \ldots, p_n \} \). The \( \epsilon \) time neighborhood of \( p_k \), denoted by \( \text{TimeEps}(p_k) = \{ p_m, p_{m+1}, \ldots, p_n \} \), is the maximal set of points \( p_k \) such that: \( p_k, p_m, \epsilon / 2 \) and \( p_k, p_{n+1}, \epsilon / 2 \).

\( \epsilon \) is a positive number that represents the maximum time interval between a point \( p \) and its neighbors on the trajectory. Because we are target to finding low speed parts in a single trajectory during the first step, the neighborhood should contain only points in the considered trajectory. Furthermore, instead of considering a minimal number of points for a region to be dense, we will use the notion of maximum traveled distance, which are named as MaxDist in the following part.

\( \delta \) Let \( st \) be a sub-trajectory of \( T_r \) as \( \{ p_m, \ldots, p_n \} \), all points in \( st \) is ordered by time. Traveled distance of \( st \) denoted as:
\[
t_{dist} = \sum_{i=1}^{n} (dist(p_i, p_{i+1}))
\]

Definition 8 defines the distance between two points as the real distance traveled by taxi rather than the direct distance between two points. An average speed \(v\) associated with \(st\) can be calculated according to:

\[
v = \frac{t}{\text{distance}(\text{TimeEps}(p))} \leq \text{MaxDist}
\]

(9) Core object: A point \(p=(x,y,y,p)\) of a trajectory is called Core point with respect to \(e\) and MaxDist if \(t_{dist}(\text{TimeEps}(p)) \leq \text{MaxDist}\).

Definition 9 corresponds to the maximum speed condition. The ratio \(\frac{\text{MaxDist}}{e}\) gives the speed limit of the respective neighborhood. By decreasing the MaxDist parameter the related speed limit increases. Besides, using distance instead of number of points will also avoid problems like the absence of points or data noise because of some equipment failure.

(10) Directly Speed Reachable: A point \(p\) is directly speed reachable to a point \(q\) with respect to \(e\), MaxDist if:

a. \(p \in \text{TimeEps}(q)\).

b. \(t_{dist}(\text{TimeEps}(q)) \leq \text{MaxDist}\).

(11) Speed reachable: A point \(p\) is Speed reachable to a point \(q\) with respect to \(e\), MaxDist if there is a chain of points \(p_1, \ldots, p_n\) where \(p_i=q, p_i=p\), for each \(p_i, p_{i+1}\) is speed reachable from \(p_i\).

(12) Speed connected: A point \(p\) is Speed connected to a point \(q\) with respect to \(e\), MaxDist if there is a point \(o\) such that both \(p\) and \(q\) are Speed reachable from \(o\) with respect to \(e\), MaxDist.

Having these definitions, the low speed part can be defined as follows.

(13) LSP: A low speed part of a trajectory \(Tr\) with respect to \(e\) and MaxDist, is a non-empty subset of \(Tr\) formed by a set of contiguous time-space points such that:

a) \(\forall p, q \in Tr\); if \(p \in \text{LSP}\) and \(q \text{ is speed-reachable from } p\) with respect to \(e\) and \(\text{MaxDist}\), then \(q \in \text{LSP}\).

b) \(\forall p, q \in \text{LSP}\); \(p\) is speed-connected to \(q\) with respect to \(e\) and \(\text{MaxDist}\).

Intuitive, a low speed part with respect to \(Tr\) is a set of points that represent a part of \(Tr\) in which the speed is lower than in other none-low speed parts. The taxi has traveled in each low speed part for a minimal amount of time.

Note that a low speed part may have a long and complicated path as exemplified in Fig. 4 (the green parts are the low speed parts extracted by our algorithm), clustering on low speed part as a whole could not find the common sub-parts of the low speed parts. Our solution is to further partition each low speed part into a set of line segments and then group similar line segments. The partition algorithm we use is the Douglas-peucker algorithm, which is widely used for reducing the number of points in a curve. We connect the adjacent points of the simplified low speed part in sequence. Then each low speed part is represented as a sequence of low speed segments. We count occurrence frequency of the similar low speed segments and use a frequency-based algorithm to group them.

![Fig.4 LSP example from a real trajectory](image)

3.4 Mapreduce Inner-function based Grouping

In our approach, we seek to identify hot routes by counting the number of similar low speed segment. The similar of two low speed segments means that two taxis pass a same road segment at a low speed. Here, we use the inner function of MapReduce to group similar LSS. Besides map and reduce, MapReduce dataflow relies on three inner functions. Function “part” partitions the map output and thereby distributes it to the available reduce tasks. All keys are then sorted by a comparison function “comp”. Finally, each reduce task employs a grouping function “group” to determine the data chunks for each reduce function call. The use of extended keys and an appropriate choice of part, comp, and group supports self-defined grouping behavior and will be utilized in our approach. We define the low speed segment as key object and implement our own “comp” function. This allows us to compare the similarity of two low speed segments and sent all similar low speed segments to the same reducer. Then the count job can be easily done.

Our own key data type contains the coordinates of the two endpoints of line segment. The “comp” function that judges whether two keys are equal is determined in terms of the following conditions:

Suppose there are two line segments \(e_1 = AC\) and \(e_2 = BD\). Here, \(A, B, C\) and \(D\) represent time stamped 2-dimensional points. The time stamp of \(A\) is ahead of \(C\), the same as \(B\) and \(D\). Suppose the projection points of the points \(A\) and \(C\) onto \(e_1\) are \(P_1\) and \(P_2\), respectively. \(l_1\) is that between \(C\) and \(P_2\); \(l_2\) is that between \(A\) and \(P_1\); They are intuitively illustrated in Fig.5.a. We use the following equation to define the similarity of two LSS.

![Fig. 5 Similarity function related figures](image)

\[
\text{Similar}(e_1, e_2) = \begin{cases} 
\text{true} & \text{if } l_1 \leq 50 \text{ m } \land l_2 \leq 50 \text{ m} \\
\text{false} & \text{else}
\end{cases}
\]

Considering the normal mean width of the road (as exemplified in Fig.5 (b)) and the precision of GPS, we set the distance
threshold for similarity to 50 meters in the equation. We do not use the segment distance function defined in TRACLUS [12]. Although it is more accurate, it is high computational-complexity and more suitable for trajectories without network constraint. Since trajectories of taxi are intuitively constrained by road network, our method is much simple and can be implemented easily. It also can ensure the transitivity of the similarity function.

Prove: Suppose \( \text{Compare}(A, B) = \text{true} \) and \( \text{Compare}(B, C) = \text{true} \), then \( \text{Compare}(A, C) = \text{true} \). Because if the \( \text{Compare} \) function is true, it means that the two LSSs were lying on a same road segment. A and B were lying on a same road segment, B and C were lying on a same road segment, the road width in each direction is less than 50m, certainly the \( l_1 \) and \( l_2 \) between A and C were less than 50m. So, \( \text{Compare}(A, C) = \text{true} \).

Intuitively a traffic hot route is correlated with time dimension. The equation 1 only defines the spatial similarity of two segments. It needs to further define the temporal similarity of two segments. However, it is unnecessary in our work. Our parallel processing mechanism has ensured that low speed segments to be grouped are within the same target time slot.

3.5 Parallel Processing Mechanism

In our work, we design the parallel mechanism taking account of data volume, and time distribution. Fig.6 illustrates our parallel workflow. There are two phases in our framework. We arrange for each phase an independent MapReduce job. The MR job for the extracting phase mainly aims at the issue of data volume. The MR jobs for the grouping phase mainly aims at the issue of time distribution.

During the extracting phase, the auto parallel processing mechanism of MP is used. The MR library in the program first shards the input trajectory data set into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.

![Fig. 6 Parallel workflow of our program](image)

It must be noted that the clustering algorithm is performed over each trajectory iteratively during this phase. Due to the default split method used in MapReduce uniformly cuts the data set into several parts and puts them into different nodes, the continuity of some trajectories may be broken. To preserve the integrity of each trajectory, we need to create a custom input split to ensure the data with same taxi identification be split into same input shard. It is a technical problem. Here we do not make detailed instruction. The parallel strategy in this phase is focused on the size of data volume. The output of this phase is the low speed segments from all trajectories. Each low speed segment set within a same time slot is output into a file as the input for the next phase.

During the grouping phase, the parallel strategy mainly focuses on the time distribution of the low speed segments. In regard to the temporal dimension, we partition it into five time slots in accordance with the time period of interest. So, during this phase, five MapReduce jobs are called in parallel. Each job is responsible for processing a low speed segment set within the same time slot. In each job, the Map function reads the input and creates low speed segment as the key object. Similar low speed segments that share the same key are automatically sent to the same reducer. The reducer counts the quantity of similar low speed segment and outputs the results.

4. TRACP ALGORITHM

We develop a parallel trajectory clustering algorithm based on the extraction-and-group framework, which we call trajectory clustering in parallel-TRACP.

Given a set of trajectories \( \{T_{r_1}, \ldots, T_{r_n}\} \), our algorithm generates a set of traffic hot routes \( H = \{H_1, H_2, \ldots, H_n\} \) in a specified date simultaneously, where each \( H_i \) in \( H \) represent all discovered traffic hot routes during a time period of that day. The Speed-DBSCAN algorithm is designed according to definition 7-13 and listed in table 1. Table 2 shows the skeleton of our traffic hot routes discovering algorithm TRACP. It goes through the two phases. Each phase is executed as a Map-Reduce job. When the job 1 ends, it will call the rest job automatically. The extracting phase mainly executes the Speed-DBSCAN algorithm to extract low speed segments. The grouping phase is mainly responsible for counting the quantity of similar low speed segments.

<table>
<thead>
<tr>
<th>Table 1 Speed based DBSCAN algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed-DBSCAN(ε, ρ, MaxDist)</td>
</tr>
<tr>
<td>Input: ( \epsilon ), Maximum time interval;</td>
</tr>
<tr>
<td>// MaxDist, maximum traveled distance;</td>
</tr>
<tr>
<td>//D, a trajectory</td>
</tr>
<tr>
<td>1. SET SP=null</td>
</tr>
</tbody>
</table>

![Fig. 6 Parallel workflow of our program](image)
2. FOR each unprocessed point p in dataset D DO
3. \( \text{MARK} \) p as visited
4. neighbors\( = \text{TimeEps}(p, \varepsilon) \) \( \cup \) return all points within p’s crime interval
5. \( \text{IF} \) t-distance( neighbors \( \cup \) MaxDist mark p as NOISE
6. ELSE
7. \( \text{SP} \rightarrow \text{next Stop} \)
8. expandStop(p, neighbors, SP, ε, MaxDist)
9. ENDIF
10. FOR (each Stop) run Douglas–Peucker
11. expandStop(p, neighbors, SP, ε, MaxDist)
12. Add p to Stop SP
13. FOR each point p’ in neighbors
14. \( \text{IF} \) p’ is not visited
15. mark p’ as visited
16. neighbors’ = TimeEps(p’, ε);
17. \( \text{IF} \) t-distance(neighbors) \( \cup \) MaxDist neighbors=neighbors \( \cup \) neighbors’
18. \( \text{IF} \) p’ is not yet member of any Stop add p’ to SP
19. ENDIF

<table>
<thead>
<tr>
<th>Table 2 Parallel trajectory clustering algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm: Trajectory clustering in parallel</td>
</tr>
<tr>
<td>Input: T // set of trajectories,</td>
</tr>
<tr>
<td>// Maximum time interval,</td>
</tr>
<tr>
<td>MaxDist // Maximum traveled distance</td>
</tr>
<tr>
<td>DOI // Date of Interest</td>
</tr>
<tr>
<td>Support // Support Number</td>
</tr>
<tr>
<td>TimeSlot // target time slot</td>
</tr>
<tr>
<td>Output: set of hot traffic routes in different time slot</td>
</tr>
<tr>
<td>// Grouping phase, Job-2</td>
</tr>
<tr>
<td>Mapper: // partition the dataset using the user-defined split</td>
</tr>
<tr>
<td>1. FOR every points in T DO</td>
</tr>
<tr>
<td>2. t = reporting time of the point</td>
</tr>
<tr>
<td>3. i = identification of the taxi</td>
</tr>
<tr>
<td>4. ( \text{IF} ) (i belongs to DOI)</td>
</tr>
<tr>
<td>5. write(key(I), value(report position)) : // Set taxi ID as KEY</td>
</tr>
<tr>
<td>6. ENDIF</td>
</tr>
<tr>
<td>7. ENDIF</td>
</tr>
<tr>
<td>Reducer: // parallel compute the LSSs for each trajectory</td>
</tr>
<tr>
<td>8. Partition the trajectory into 5 subsection according to the time slot</td>
</tr>
<tr>
<td>9. for (each subsection)</td>
</tr>
<tr>
<td>Speed-DBSCAN(D, ε, MaxDist) //D is the subsection associated with a time slot</td>
</tr>
<tr>
<td>// Grouping phase, Job-2</td>
</tr>
<tr>
<td>Mapper: // input all LSSs during a certain timeslot generated by the previous job</td>
</tr>
<tr>
<td>10. For (each line segment) set the two endpoints as the key</td>
</tr>
<tr>
<td>Reducer: // parallel compute the THRS during this time slot</td>
</tr>
<tr>
<td>11. Count the number of the same key</td>
</tr>
<tr>
<td>12. If the number &gt; support threshold, output a traffic hot route</td>
</tr>
</tbody>
</table>

5. EXPERIMENTAL RESULTS

5.1 Experiment Setting

Our experiment is built on Hadoop version 0.20.2. Our Hadoop cluster consists of a master node and four slave nodes. Each node runs Ubuntu Linux version 11.04 on Intel Xeon (R) E5606 2.13GHZ CPU. The master node has a 4GB memory and each slave node has a 2GB memory. The program is implemented in JDK 1.7. The experimental data are T-Drive trajectory dataset [13, 14] from Microsoft Research China. The dataset contains one week (from 2008-2-2 to 2008-2-8) trajectories of 10,357 taxis in Beijing. The total data volume is 756MB and total number of points in the dataset is about 15 million and total distance of the trajectories reaches 9 million kilometers. Each taxi track is stored in a single text file. Each line of file stores one position information in format as: vehicle ID, time, latitude, longitude. (There is large number of small files in T-Drive data set because it uses a single file to store the entire trajectory of one taxi rather than store all the trajectories in one file. This is not an ideal environment for MapReduce process. In our work, we use the CombineFileInputFormat class to deal with the trajectory files in HDFS rather than the default implementation class of Map Reduce FileInputFormat. The class packs many files into each split so that each Mapper has more to process. The detail usage of the class can be found at the Hadoop website.) The map data we use is the road network data within sixth ring road in Beijing city.

In regard to the parameters of maximum time interval and maximum traveled distance, it is difficult to specify a good value for this parameter without knowing well the characteristics of each trajectory and the road grade. Considering this, we specify the two parameters according to the local standard DB11/T 785-2011 “Urban road traffic performance index” of Beijing city [15]. In this standard, the severe congestion means that the vehicles speed is no higher than 10 Kilometers per hour on road, 15 Kilometers per hour on trunk road, 20 Kilometers per hour on express way and main ring road. The statistics interval is 15 minutes. So the maximum time interval is set to 900 seconds. Though we have the road class information in our map data, we still set the maximum traveled distance with a same speed threshold value for all roads in our current work for convenience.

Then, we further segment time of day into five continuous slots in terms of the typical traffic conditions. The time slots are 8:00-11:00, 11:00-14:00, 14:00-17:00, 17:00-20:00, and 20:00-23:00. The support threshold parameter in group phase is set varies according to number of taxis in the test data. The InputSplit size is set using the default value 64MB. The number of Mapper has a great impact on the efficiency of the algorithm. Here we use the default value and do not discuss the optimum
number of it. It should be noted that the real life data we downloaded are not big enough. To simulate an actual big data processing, we use the default JVM heap size 64MB to ensure that the dataset could not be easily fitted to the main memory. And in our experiments, these data are scanned by File Steam rather than put them into memory.

5.2 Experiment Evaluation

5.2.1 Efficiency Analysis

To evaluate the efficiency of our method, we implement a traditional centralized version and a single node multi-processor (MP) parallel version of our extraction-group framework and make performance comparison between the three versions. The centralized version and MP parallel version only run on the main node. The MP version is programmed based on Fork/Join framework. According to the principle in [16] “For computer intensive tasks, an N CPU processor system usually achieves optimum utilization with a thread pool of N+1 threads”, the number of threads in MP version is set to five. Each thread reads the trajectories files allotted to it from disk in sequence and performs the extracting and grouping job. We made experiments with trajectories of 100, 256, 318, 726 and 4000 taxis (corresponding to 100, 256, 318, 726 and 4000 txt files, these files are relatively complete. Some files in the data set just contain little reporting position and less than 1KB) in the data set respectively. The total data sizes of these files are 42M, 108M, 172M, 307M and 570M respectively. The performance graph is shown in Fig.7. The fastest growth of time overhead with the increase of the amount of data came from centralized version. The time overhead of MP version also grows rapidly. The time overhead of cluster version in cluster environment is relatively stable. Because the communication time between machines account for a large proportion of entire time overhead in cluster environment, the centralized version and MP version have a less time overhead in terms of small amount of data. With the increasing of data volume, the growth of time overhead of MP version is rapider than that of cluster version. This phenomenon is more obvious in extraction phase. Let’s take input data sizes 42Mb and 570Mb as an example. During extraction phase, the time overhead of MP version and cluster version is 43 seconds and 152 seconds respectively in the case of 42Mb. In the case of 570Mb, both the time overhead of the two algorithms are close to 400 seconds. Because the output data volume of extraction phase was small, the performance of MP version still has a big advantage over cluster version in group phase. To further evaluate the efficiency of our method with a larger scale data set, we copy the 4000 txt files and replace the taxi ID in each file with a new taxi ID. We made an experiment with the new dataset whose size is 1.14Gb. As it shows in Fig.7, the time overhead of MP version has exceed that of cluster version at this size (We do not make this test on centralized version). On the whole, we argue that the cluster version would be more efficient than MP version in the case of a larger amount of data.

5.2.2 Effectiveness Analysis

To check the effectiveness of TRACP, we test it against a variety of settings. Figure 8 shows several traffic hot routes extracted from 4000 taxis with a speed threshold 15K/h, support number threshold 25. Each traffic hot route is drawn in thick dotted line. To the authors own life experience, there are three obvious traffic hot routes that have been labeled at both Fig.8(a) and Fig.8(b). The first one is on the way to the capital airport, at where taxis are assembled in queue for picking up/dropping off passengers. The second is on the way to Wangjing residential area, which is one of the largest communities in Beijing. The third one is in the surrounding areas of Beijing west railway station, at where traffic congestion is always serious. Table 3 lists statistics of the result. At first sight of these figures, we doubt that there should be something wrong in our algorithm.

It was generally believed that there should be more traffic hot routes during peak hours than in other time, but there is much less number of traffic hot routes during 8:00-11:00 than in other time periods in our statistics. After examining original data and the number of low speed segments, we argue that there are two reasons that lead to this result. First, the data is significantly less in the morning. Many trajectories contain no records during that time. Maybe many taxi drivers did not turn on GPS device at that time. Furthermore, there are news say that many taxi drivers in Beijing are unwilling to work at that time, because traffic jam is so serious that they could not make profit. Second, because traffic congestion is more serious during that time, there is more similar low speed segments associated with a traffic hot route. From table 3, we can

---

(a) THR during of 14:00-17:00 02-02-2008

(b) THR during of 17:00-20:00 02-02-2008

Fig. 8 Traffic hot routes distribution map
get that there are 123 low speed parts found during 8:00-11:00, but there are only two traffic hot routes extracted during that time. A traffic hot route can be found from an average of 62 low speed parts. The number was reduced to 28 during 11:00-14:00. In summary, we argue that the results are reasonable.

However, for efficiency reasons, we did not conduct road matching in our current work. If matching the low speed part with the road network, the grouping result would be more effective and accurate. We will enhance our method to contain road matching in the future work.

<table>
<thead>
<tr>
<th>Time slots</th>
<th>Low speed parts</th>
<th>Traffic hot routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00-11:00</td>
<td>123</td>
<td>2</td>
</tr>
<tr>
<td>11:00-14:00</td>
<td>393</td>
<td>14</td>
</tr>
<tr>
<td>14:00-17:00</td>
<td>1212</td>
<td>38</td>
</tr>
<tr>
<td>17:00-20:00</td>
<td>2330</td>
<td>47</td>
</tr>
<tr>
<td>20:00-23:00</td>
<td>3424</td>
<td>51</td>
</tr>
</tbody>
</table>

### 5.2.3 Varying the Parameters

To validate our reasoning, we make a further experiment by varying the value of maximum speed threshold during extraction phase and the value of support threshold during group phase. Figure 9(a) and 9(b) show the effect of varying support threshold on the number of traffic hot routes with a speed of 15K/h and 10K/h respectively. Here, horizontal axis represents the value of support threshold. The vertical axis represents the number of traffic hot routes discovered. We can see from the figure that when the value of support threshold increases, the number of traffic hot routes during non-peak hours decreases rapidly whereas it remains relatively stable during peak hours. Figure 9(c) shows the effect of varying speed on the number of traffic hot routes with a support threshold value of 20. Because we use a same speed threshold for all kinds of road, we can see that speed has a less impact on the number of traffic hot routes than support threshold in our experiments.

In summary, we argue that time distribution of traffic hot routes in our experiments is not reasonable. It is because the sample data cannot reflect the full day traffic status of the whole city. But spatial distribution of our result is in accordance with the actual case. It might also be noted that the results are highly dependent on user-specified threshold values of DBSCAN and the support value. The parameters value we used in this paper is the experience value after several tests. It is difficult to evaluate the impact on the results of these values. We do not discuss how to get a perfect parameter value in this paper.

### 5.2.4 Comparison with Other Methods

To further evaluate the effectiveness of our method, we implement an approximate solution of TRACLUS algorithm and conduct a comparative analysis of our method and this method. TRACLUS is a density-based line-segment clustering algorithm proposed in [12]. It is used to find common sub-trajectories in trajectory database. The meaning of frequent common sub-trjectories is similar to traffic hot route in transportation applications. The main difference between our approximate algorithm and TRACLUS is that we use Douglas–Peucker algorithm to partition a trajectory into a set of line segments whereas a minimum description length principle-based algorithm is used in TRACLUS. We argue that the simplification principle of Douglas–Peucker algorithm is similar to minimum description length when they are used to simplify a piecewise linear curve. The clustering phases of both two algorithms are based on DBSCAN and use the same segment-based distance function. The perpendicular distance threshold for Douglas–Peucker algorithm and the minimum support number for DBSCAN are set to the same value as we used in TRACP. Our approximate algorithm is also evaluated with the aforementioned dataset. Figure 10 shows the distribution of 48 common sub-trajectories during 14:00-17:00 generated by the algorithm. We can see from figure 8(a) and 10 that much of the results of the two methods are overlapped and located in same area. However, there is more number of common sub-trajectories found in the approximate algorithm than the number of traffic hot routes found by TRACP. Furthermore, the common sub-trajectories are relatively rough. Some of the results occupy a long distance. Because our TRACP algorithm has filtered out fast moving parts of the trajectories, it got more accurate results. Furthermore, Douglas–Peucker algorithm based partition simply depends on the geometric information of the trajectory, thus the partition includes little semantic information of trajectories. The common sub-trajectories found in this way are more suitable to represent road sections passed by many taxis during certain period. It can be seemed as popular routes that taxi driver like to choose. These common sub-trajectories may have a smooth flow of traffic rather than traffic congestion. Due to taxi drivers are usually experienced in finding the fastest route to a destination based on their knowledge, we argue that the method is more suitable for mining smart driving directions from trajectories. Furthermore, due to the Douglas–Peucker or minimum description length principle based segment extraction may generate many useless segments at road intersections, these algorithm are not a good choice for trajectories with network constraint. Though we also use Douglas–Peucker in the grouping phase of TRACP, such influence has been weakened by filtering out fast moving parts.
Moving object clustering is a well-studied problem with a great deal of research efforts being devoted in. Related work focuses on moving object based traffic analysis are mainly interested in detecting areas of high traffic load. Density-based clustering algorithms are most popular used for identifying hot spots or popular places in current studies. Linsey Xiaolin Pang [17] transforms trajectories into grid series and uses density based clustering on the grids to find traffic hot area. G.Gidonfalvi [18] uses a similar approach to find dense spatio-temporal areas and frequent routes. N.S.Savage [19] uses grid series to model trajectory and adopts a frequent sequence-pattern mining algorithm to find hot routes. The problem with grid-density based clustering is that the results are very sensitive to grid size. It also introduces the problem of answer loss due to edge effects. The TRACLUS framework first partitions trajectory using the Minimum Description Length principle, then it uses a density-based line-segment clustering algorithm to discover common sub-trajectories. Xia Ying [20] uses minimum bounding box series to model trajectories and discovers traffic congestion area through density of minimum bounding box. Li et al [21] studied traffic flow patterns in road networks and proposed a density-based algorithm called FlowScan, which uses the density of traffic in sequences of road segments to discover hot routes. Especially in the previous study scenario, they all consider the density or quantity of objects is enough to cluster. Thus when the density or quantity of objects is not good enough for special application scenarios, they will fail. Siyuan Liu [3] proposed a non-density-based approach called mobility-based clustering. The key idea is that sample objects are employed as “sensors” to perceive the vehicle crowdedness in nearby areas using their instant mobility, rather than the “object representatives”. They argue that mobility based clustering have three advantages: less sensitive to the size of the sample object set, does not require accurate location information, mobility-based clustering naturally incorporates the mobility of vehicles. Atsuo Tachibana [22] proposes a method to locate congested segments over the Internet by clustering the delay performance of multiple paths. They use a hierarchal clustering to cluster the monitored paths based on packet delay variation. We think the main idea of this approach can be modified to adapt to be used for locating congestion in road network.

However, there are only a few works that consider parallel processing on trajectories. B.Yang [23, 24] presents a framework for query processing over trajectory data based on MapReduce. In this work, historical trajectory data set is partitioned into data chunks according to predefined spatial grid. The work mainly focuses on parallel executing spatio-temporal range queries on historical trajectories.

7. DISCUSSION AND CONCLUSION

In this paper, we proposed a MapReduce-based framework to locate traffic hot routes from massive trajectory data. In the framework, massive trajectory data are partitioned into data chunks and therefore they can be processed at different nodes in the cluster. A two phase trajectory clustering algorithm is performed on the data chunks in parallel. As the algorithm progress, each trajectory is partitioned into a set of low speed segments by a speed-based DBSCAN algorithm which can resolve the problem of various densities. A MapReduce inner function based frequent item set mining method is used to locate frequent low speed segments. The method is efficient and easy to implement. To show the effectiveness and efficiency of our algorithm, we have performed extensive experiments on real taxis data of Beijing city. The preliminary experiments have demonstrated that our method has advantages in efficiency and accuracy in terms of massive data.

The main weakness of our approach is that the trajectories are not registered with the underlying road network. It may mark small roads as congested segments, when in reality the speed limit of the road segments may be low. Furthermore, matching the low speed parts with the road network can improve the accuracy of the result greatly. The segment similarity function also can be more simple and accurate. However, synchronous parallel road matching within Mapreduce framework needs to be studied at first.

We argue that MapReduce provides a good alternative for parallel massive trajectory data processing. Traditional trajectory data modeling, indexing, similarity query processing and road matching technologies need to be further studied so that they can fully utilize the highly parallel processing power of large-scale clusters. There are also many other challenging issues in integrating MapReduce into trajectory data mining. We are also considering improving our algorithm with an extended MapReduce framework for multi-core as a further study.

Reference


[13] Jing Yuan, Yu Zheng, Xing Xie, “Driving with knowledge from the physical world,” Proc. of Int. Conf. on Knowledge discovery and data mining (SIGKDD’11) 316-324
[21] Xiaolei Li, Jiawei Han, Jae-Gil Lee, “Traffic Density-Based Discovery of Hot Routes in Road Networks,” Proc. of 10th int. conf. on Advances in spatial and temporal databases(SSTD’07)441-459.
[23] Qiang Ma, Bin Yang, Weiming Qian, and Aoying Zhou, “Query Processing of Massive Trajec- tory Data based on MapReduce,” Proc of the first int. workshop on Cloud data manage- ment (CloudDB’09)16.