Outlier Detection over Distributed Trajectory Streams

Jiali Mao * Pengda Sun † Cheqing Jin ‡ Aoying Zhou §

Abstract

The wide deployments of GPS-embedded devices have produced multiple rapid voluminous trajectory streams, which needs to be analyzed to extract abnormal behaviors of moving objects in real-time. To date, outlier detection over distributed trajectory streams has not received enough focuses due to the constraint factors like skewness distribution and evolving nature of trajectory data, and on-the-fly execution requirement with minimal communication cost. In this paper, we present the first scalable decentralized outlier detection framework over distributed trajectory streams, called *ODDTS*. It consists of remote site processing and coordinator processing, with the aim of continuously providing *feature-grouping based* outliers detection over distributed trajectory streams. Extensive experiments over real data demonstrate high detecting validity, less communication cost and linear scalability of *ODDTS* method for online identifying outliers upon distributed trajectory streams.

Keywords distributed trajectory stream, feature-grouping, outlier detection, scalability

1 Introduction

The proliferating deployments of positioning devices and surveillance equipment have expedited exponential growth of position stream data arrived from disparate sources. For instance, the surveillance inspection spots deployed in city traffic crossroads record their nearby passing vehicles' location information and moving behavior characteristic (e.g., speed) in real time, meanwhile, transmit them through the leased lines to the servers of the traffic control center in their respective regions. The sequences of positions received continuously by the servers in various regions have formed into distributed trajectory streams. This necessitates efficient streaming analysis to extract timely insights to meet the needs of real-time applications. Centralizing the massively distributed trajectory streams to the central server and analyzing them afterward raises the issue with computing and storage capacities. Thus the whole analyzing process shall be distributed throughout the entire network of nodes (servers). In this paper we are primarily concerned with designing highly scalable decentralized outlier detection approach over distributed trajectory streams. With the advent of open-source distributed frameworks like Spark and Storm, exploiting the distributed solution for outlier detection becomes possible. It can facilitate various time-critical applications like route planning [1], ride-sharing planning [2], as well as road infrastructure optimization [3], etc.

Figure 1: Example of speed based outlier

Trajectory outlier is usually regarded as a trajectory that is obviously dissimilar with the majority of the others, such as the vessel behaves significantly different from the other ones in the same area of ocean [4], the hurricane that suddenly changes wind direction [5], and the taxi with detour behavior [6], etc. Despite huge efforts have been conducted to trajectory outlier detection, most of them distinguish outlier from inliers based on spatial proximity. Actually, trajectory outlier may manifest as the significant difference of motion behavior from between a trajectory and its spatial neighbors. Consider the following example, in road network scenario, at each time instant, traces of vehicles can be grouped into clusters in terms of the driving direction and spatial proximity. The mean speed of each cluster represents that of the road segment where per cluster locates on. From Fig.1 we can observe that the speeds of two road segments (represented by two clusters TC_1 and TC_2) are obviously lower than that of their respective neighbors. Here, TC_1 and TC_2 are

*School of Data Science and Engineering, East China Normal University, and School of Computing, China West Normal University. Email: jlmao1231@stu.ecnu.edu.cn.

†School of Computer Science and Software Engineering, East China Normal University. Email:51174500124@stu.ecnu.edu.cn

‡Corresponding author. School of Computer Science and Software Engineering, East China Normal University. Email: cqjin@sei.ecnu.edu.cn

§School of Data Science and Engineering, East China Normal University. Email: ayzhou@dase.ecnu.edu.cn

most likely the speed-based trajectory cluster outliers relative to their respective neighborhoods, indicating some contingency event like the traffic jam.

In a bid to identify the aforementioned outliers upon streaming trajectories, we presented a *feature-grouping based* outlier detection framework [7]. Nevertheless, it cannot be directly applied to distributed streaming cases owing to the following challenges. First, trajectory data distribution is highly skewed and changes over time. Accordingly, outliers may behave differently across various regions and evolve gradually. To address this issue, the instantaneous outlierness of per trajectory shall be measured on the local node, and the evolving outlierness of each object shall be estimated through continuously gathering the outlierness of its related trajectory from the local nodes. Another noticeable problem is that outlier detection shall be executed timely to ensure preventive actions to be taken as early as possible. This involves two key issues, one is to exploit efficient parallel detection process of the nodes, and the other is to reduce the amount of data to be transferred among the nodes during detection process.

To tackle the above issues, we propose the first distributed *feature-grouping based* outlier detection framework to identify outliers upon trajectory streams. It consists of parallel outlier detection on the remote sites, and evolving anomaly object detection on a single coordinator. For the remote sites, the trajectory fragments derived by trajectory simplification are grouped into clusters, and then implemented detection to output the trajectory fragment (or fragment cluster) outlier based on the behavior dissimilarity in relation to their respective neighborhoods. For the coordinator, upon receiving the trajectory fragment outliers detected by the remote sites, the outlierness duration of the related object is updated and checked whether exceeding the given anomaly timebin count threshold to detect the evolutionary anomaly object. In more detail, the contributions of this paper are summarized below.

- We first address the problem of outlier detection over distributed trajectory streams.
- We present trajectory outlier definitions to characterize the anomaly trajectory fragment, the anomaly fragment cluster and the evolutionary anomaly object in distributed streams.
- We propose a two-phase framework, termed as *ODDTS*, to support trajectory outliers detection upon distributed streams.
- We conduct a comprehensive series of experiments on two real datasets. Experimental results manifest the efficiency and effectiveness of our proposal.

Outline. The remainder of this paper is structured as follows. Section 2 reviews related work in literature. In Section 3, the preliminary concepts are introduced and the problem is defined formally. In Section 4, we outline the scheme of *ODDTS*. In Section 5, we show the results of our experiments. Finally, the last section presents conclusion of the work.

2 Related work

Trajectory outlier detection has received considerable attentions. The existing techniques involve classification-based approach [8], historical similarity based approach [4, 6, 9–12], distance-based approach [5, 13, 14], direction- and density-based approach [15], isolation-based approach [16], etc (see [17] for a survey). Due to huge volume, rapid updating and highly skewed nature of streaming trajectories, there exist relatively few researches on outlier detection upon trajectory streams [13,14,16]. They mainly distinguish outlier from inliers based on spatial proximity. In our previous work [7], we proposed a *feature-grouping based* outlier detection framework with two detection methods to identify outliers upon streaming trajectories.

However, the above centralized solutions are not tailored to distributed trajectory streams. Even scalable techniques of them still require excessive execution overheads for handling continuously increased distributed stream data when compared to the stringent response time requirements of actual applications. Additionally, distributed distance-based [18] and distributed local outlier [19] detection methods cannot be directly applied to identify trajectory outlier. Recently, in the aspect of distributed trajectory analysis, a surge of researches have been devoted to trajectory querying in distributed dataset [20, 21]. Zhang et al. proposed a communication cost-saving approach to process distributed top-k similarity query over trajectory streams by utilizing the multi-resolution property of *Haar wavelet* [22]. But the distributed trajectory query solutions are not well suited for our proposed outlier detection problem (depicted in Section 1). In order to capture the behavior outlierness of each trajectory in relation to its local neighborhood upon distributed streams, it is imperative to design a scalable decentralized *feature-grouping based* outlier detection mechanism.

3 Problem Definition

The notations used in the rest of this paper are summarized in Table 1. Some of the notations are adapted from [7] and listed here for completeness.

Consider a distributed-computing environment with a single coordinator and M remote sites. Let S denote a trajectory stream and it is composed of M subsets $S_1, \ldots, S_M$, with each local stream S_k located at the kth remote site, i.e., $S = \cup_{k=1}^{M} S_k$.

Table 1: List of notations

Notation	Definition
S_k	the local stream of the kth remote site
M	the number of remote sites
T_c	the current timebin
N	the window size
p_i	the location of an object at the timestamp t_i
tf	the trajectory fragment
d	the proximity threshold within cluster
d_c	the proximity threshold between clusters
$\rho(\rho_c)$	the local outlier threshold of fragment (cluster)
thr_a	the anomaly timebin count threshold

Definition 3.1. (Local trajectory stream) *The local trajectory stream $S_k = \{(p_1, t_1), (p_2, t_2), \ldots\}$ refers to the infinite sequence of position points of multiple moving objects that received by the kth remote site, and p_i is the location (latitude and longitude) of one object at timestamp t_i in 2-D space, i.e., $p_i = (x_i, y_i)$.*

To guarantee various moving objects of different sampling rates report their locations at least once in a time interval, the term "timebin" is used to describe a basic time interval (represented by m timestamps, here $m \geq 1$). As new position points arrive continuously, the local trajectory stream is typically processed in a *sliding window*. Here, *time-based sliding window* model is leveraged. Let N represent the window size and T_c denote the current timebin, only the most recent stream elements p_{T_i} $(T_c - N + 1 \leq T_i \leq T_c)$ are implemented outlier detection. Whenever the window slides forward, it moves forward by 1 timebin.

It is space-efficient to summarize each trajectory by reserving a small number of sample points. Similar to [7], each trajectory is simplified into a set of characteristic points via trajectory simplification, and every two consecutive characteristic points are connected into a trajectory fragment, denoted as tf. A trajectory thus becomes an ordered sequence of fragments. Let $F = f_1, \cdots, f_L$ denote L features extracted from the attributes of trajectories, and the features are divided into two groups: *Similarity Feature* $(f_1, \cdots, f_b)$, or SF for short, e.g., latitude and longitude coordinates, which is used to find the spatial neighbors for each trajectory fragment, and *Difference Feature* $(f_{b+1}, \cdots, f_L)$, or DF for short, e.g., speed, direction, etc, which is used to identify the trajectory fragment whose motion behavior is obviously distinct from its vicinity. Let $w_1, \cdots, w_L$ denote the weight of L features separately, and $dis_l(tf_i, tf_j)$ denote the distance between tf_i and tf_j by the feature f_l. Here, $dis_l(tf_i, tf_j)$ can be any typical distance metric (e.g., *Euclidean*, *LCSS* and *DTW*, etc.). Furthermore, $Diff_1$ (or $Diff_2$) denotes the distance between tf_i and tf_j by SF (or DF), i.e., $Diff_1(tf_i, tf_j) = \sum_{l=1}^{b} w_l \cdot dis_l(tf_i, tf_j)$, and $Diff_2(tf_i, tf_j) = \sum_{l=b+1}^{L} w_l \cdot dis_l(tf_i, tf_j)$.

Given a proximity threshold d $(d > 0)$, trajectory fragments can be grouped into clusters (denoted as FC) in terms of the distance $(Diff_1)$ between the fragments by SF, and the fragments inside a cluster are neighbors to each other. The aggregated summarization of FC can be maintained using the synopsis data structure, called Fragment Cluster Feature(CF).

Definition 3.2. (Fragment Cluster feature, CF) *CF of a fragment cluster $FC_i = \{tf_1, tf_2, \ldots, tf_n\}$, is of the form $(ls_{cen}, ls_p, ls_{len}, ls_{df}, cor_{bl}, cor_{tr}, n)$.*

- *ls_{cen}: the linear sum of the fragments' center points;*
- *ls_p: the linear sum of the product of the fragments' angle and length;*
- *ls_{len}: the linear sum of the fragments' length;*
- *ls_{df}: the linear sum of the fragments' weighted sum of DF;*
- *cor_{bl}: the bottom left corner of MBR;*
- *cor_{tr}: the top right corner of MBR;*
- *n: the number of fragments;*

Minimum Bounding Rectangle (*MBR*, for short) is leveraged to represent the spatial region of each cluster. The representative fragment rp_i of a fragment cluster FC_i can be derived using the method in [23]. We first obtain the central point and the angle with $\frac{ls_{cen}}{n}$ and $\frac{ls_p}{ls_{len}}$ respectively. Then a line is plotted across the central point, along the angle, and extended to reach the borders of *MBR*. The intersection points are regarded as the starting and ending points of rp_i.

Hereafter, to identify anomaly fragment cluster is to detect anomaly representative fragment. Whether anomaly trajectory fragment or anomaly representative fragment detection at T_c, the key step is to estimate the anomaly degree of a trajectory fragment tf_i (or representative fragment rp_i) with regard to its neighborhood (denoted as $N_{T_c}(tf_i)$ or $N_{T_c}(rp_i)$). $N_{T_c}(tf_i)$ includes the other fragments within the cluster that tf_i belongs to. $N_{T_c}(rp_i)$ includes the representative fragments of FC_i's neighboring clusters in terms of a given proximity threshold d_c $(d_c > d)$ by SF. Here, we employ the concepts of *local difference density* (*ldd*) and *local anomaly factor* (*LAF*) in [7]. *ldd* is defined as the inverse of the average difference of each fragment to its neighbors by DF, i.e., $ldd_{T_c}(tf_i) = \frac{|N_{T_c}(tf_i)|}{\sum_{tf_j \in N_{T_c}(tf_i)} Diff_2(tf_i, tf_j)}$. *LAF* is used to measure the probability of fragment for being an outlier based on *ldd*, i.e., $LAF_{T_c}(tf_i) = \frac{\sum_{tf_j \in N_{T_c}(tf_i)} \frac{ldd_{T_c}(tf_j)}{ldd_{T_c}(tf_i)}}{|N_{T_c}(tf_i)|}$. In general, a higher value of LAF

indicates that a trajectory fragment (or fragment cluster) is more likely to be an outlier.

Definition 3.3. (Anomaly Trajectory Fragment) *Given a local outlier threshold ρ ($\rho > 1$), trajectory fragment tf_i at timebin T_c is called a fragment outlier (or* F-outlier*, for short), iff $LAF_{T_c}(tf_i) > \rho$.*

Definition 3.4. (Anomaly Fragment Cluster) *Given a local cluster outlier threshold ρ_c ($\rho_c \geq \rho$), a fragment cluster FC_i and its representative fragment rp_i, FC_i at timebin T_c is identified as a fragment cluster outlier (or* FC-outlier*, for short), iff $LAF_{T_c}(rp_i) > \rho_c$.*

Considering the evolving nature of streaming trajectories, the object with abnormal fragments in several timebins (may be inconsecutive) is intrinsically an abnormal moving object. To verify the abnormal moving property of the object, we shall continuously observe the objects that have anomaly trajectory fragments in current time window. As the objects usually move across various regions, the trajectories of the objects would arrive at several remote sites. To identify the evolutionary anomaly object, the anomaly trajectory fragments detected by the remote sites need to be transferred to the coordinator at per timebin. The coordinator sets a list (denoted as $list_{O_i}$) of anomaly timebins for the object O_i that has anomaly fragment, and conducts object outlier detection by judging whether the size of $list_{O_i}$ (denoted as $|list_{O_i}|$) reaches the given threshold thr_a.

Definition 3.5. (Evolutionary Anomaly Object) *Given an anomaly timebin count threshold thr_a ($\frac{N}{3} \leq thr_a < N$), an object O_i at timebin T_c is an evolutionary object outlier (or* EO-outlier*, for short), iff $|list_{O_i}| \geq thr_a$.*

Problem Statement. Given local outlier threshold ρ and ρ_c, anomaly timebin count threshold thr_a, and a union of local trajectory streams within a sliding window of size N that distributed in multiple remote sites, our goal is to continuously detect *F-outlier*, *FC-outlier* and *EO-outlier* over distributed trajectory streams.

4 Framework: Outlier Detection over Distributed Trajectory Streams

In this section, we propose a distributed framework, termed as *ODDTS*, to continuously identify trajectory outliers over distributed streams. Our distributed solution is inherently parallel and can perform on any modern distributed infrastructure. In subsequent experiments, we use Spark, which performs analysis by organizing incoming trajectory stream within each time window into micro-batches, and the size of micro-batch needs to be as small as possible to guarantee low latency.

Algorithm 1: *ODDTS* (Outlier Detection over Distributed Trajectory Streams)

Input: S: a trajectory stream which consists of M subsets in current time window
Output: (1)A_F: a set of *F-outlier*s; (2)A_{FC}: a set of *FC-outlier*s; (3)A_O: a set of *EO-outlier*s;

1 **foreach** *remote site* **do**
2 Upon arrival of new trajectory data, detect *F-outlier*s (A_F) and *FC-outlier*s (A_{FC}) by implementing FCD algorithm;
3 Send A_F to the coordinator;
4 **for** *the coordinator* **do**
5 Upon receiving A_F from any remote site, detect *EO-outlier*s (A_O) by implementing EOD algorithm;
6 **return** A_F, A_{FC} *and* A_O;

ODDTS consists of remote site processing and coordinator processing, the detailed description is outlined in Algorithm 1. At each timebin, the workflow is summarized as below: (i) Remote site processing (Lines 1-3), including (a) Trajectory simplifying and clustering: incoming trajectory data is simplified into fragments and then grouped into clusters. and (b)*F-outlier* and *FC-outlier* detection: *F-outlier* and *FC-outlier* are identified according to behavior dissimilarities among trajectory fragments (or representative fragments) and their respective neighborhoods. (ii)Coordinator processing, *EO-outlier* detection: on the basis of *F-outlier*s transferred by the remote sites, the timebin for identifying *F-outlier* is inserted into the anomaly timebin list of the object that *F-outlier* belongs to. Then the size of anomaly timebin list of the influenced object is updated and compared with the given threshold thr_a to identify *EO-outlier* (Lines 4-5).

In our distributed environment, the remote sites only communicate with the coordinator, and the concern is to implement efficient outlier detection in parallel while minimizing the communication cost between the coordinator and the remote sites during detection process.

4.1 Remote Site Processing The main tasks of the remote sites are to implement outlier detection on respective incoming trajectory data in parallel and return detection results to the coordinator timely. The detailed description of parallel outlier detection algorithm (or FCD, for short) is presented in Algorithm 2. Initially, incoming trajectories are split into a set of consecutive trajectory fragments with least information loss. Through trajectory simplifying (denoted by $TraSimp$),

Algorithm 2: *FCD* (*F-outlier* and *FC-outlier* Detection)

Input: S_k: a local trajectory stream at T_c; ρ, ρ_c: the local outlier thresholds
Output: (1)A_F: a set of *F-outlier*s; (2)A_{FC}: a set of *FC-outlier*s;

1 **foreach** *trajectory* Tr *in* S_k **do**
2 $\quad Tr_{simp} \leftarrow TraSimp(Tr)$;
3 $\quad Tr_{all} \leftarrow Tr_{all} \cup Tr_{simp}$;
4 Generate a set of trajectory fragments TF_c from Tr_{all};
5 Group the trajectory fragments of TF_c into FCs according to d;
6 **foreach** *trajectory fragment* tf_i *in each* FC **do**
7 $\quad LAF_{T_c}(tf_i) \leftarrow \frac{\sum_{tf_j \in N_{T_c}(tf_i)} \frac{ldd_{T_c}(tf_j)}{ldd_{T_c}(tf_i)}}{|N_{T_c}(tf_i)|}$;
8 $\quad$ **if** $LAF_{T_c}(tf_i) > \rho$ **then**
9 $\quad\quad A_F \leftarrow A_F \cup \{tf_i\}$;
10 **foreach** *fragment cluster* FC_i **do**
11 $\quad$ Derive its representative fragment rp_i ;
12 **foreach** *representative fragment* rp_i **do**
13 $\quad$ Find its spatial neighbors $N_{T_c}(rp_i)$ according to d_c;
14 $\quad LAF_{T_c}(rp_i) \leftarrow \frac{\sum_{rp_j \in N_{T_c}(rp_i)} \frac{ldd_{T_c}(rp_j)}{ldd_{T_c}(rp_i)}}{|N_{T_c}(rp_i)|}$;
15 $\quad$ **if** $LAF_{T_c}(rp_i) > \rho_c$ **then**
16 $\quad\quad A_{FC} \leftarrow A_{FC} \cup \{FC_i\}$;
17 **return** A_F *and* A_{FC};

a small number of characteristic points are derived from raw trajectories using the method in [7], and every two consecutive characteristic points are connected into a trajectory fragment (Lines 1-4). To identify the outlier with respect to its spatial neighborhood, the trajectory fragments are grouped into clusters (FCs) via hierarchical clustering according to the proximity threshold d (Line 5). Here, a *STR-tree* index technique is leveraged to accelerate clustering.

For *F-outlier* detection, each trajectory fragment shall be measured the abnormality degree of its motion behavior relative to the others in the same cluster, which involves the calculations of *ldd* and LAF(Lines 6-7). Through comparing the values of *ldd* between each trajectory fragment and its vicinity by DF, we derive the value of LAF for each fragment at T_c. The trajectory fragments with LAF's values exceeding ρ are reported as *F-outlier*s (Lines 8-9). Then, the object that *F-outlier* belongs to is treated as *EO-outlier* candidate and transferred to the coordinator. For *FC-outlier* detection, to identify anomaly fragment cluster is essentially to detect anomaly representative fragment as relative to its neighboring representative fragments. Specifically, for per fragment cluster, we obtain its representative fragment (Lines 10-11), and search for its neighboring representative fragments that are within distance d_c (Lines 12-13) by SF from it. Then the value of *ldd* of each representative fragment is derived by comparing the distance between it and its neighboring representative fragments by DF. LAF's value of each representative fragment is also derived and used to detect *FC-outlier* in terms of local anomaly threshold ρ_c (Lines 14-16).

Pruning step The most time-consuming part of this phase is *ldd* calculation, which involves pairwise difference calculations among trajectory fragments within a cluster, and that among neighboring representative fragments. Costs of *ldd* calculations would become comparatively expensive especially when massive amount of trajectories arrive at one timebin. To speed up *F-outlier* and *FC-outlier* detection, unnecessary *ldd* calculations need to be pruned. After clustering trajectory fragments, the mean of the fragments' weighted sum of DF within a cluster is derived by $\frac{ls_{df}}{n}$, denoted as AVG_{DF}. During *F-outlier* detection, only the trajectory fragment whose weighted sum of DF higher (or lower) than μ times of AVG_{DF} ($\mu > 1$), shall compute the values of *ldd* and LAF. Also, the mean of the representative fragments' weighted sum of DF within similar neighborhood can be derived, and on the basis of which the representative fragments with weighted sum of DF higher (or lower) than μ times of that mean, need *ldd* and LAF calculation. Obviously, such pruning step can quickly eliminate the need of examining a large number of fragments, the operational overhead of *ldd* calculation is thus sharply reduced and parallel outlier detection process achieves the significant performance gain.

4.2 ***Coordinator Processing*** With the evolution of trajectories, the moving object who has anomaly trajectory fragments in several timebins (maybe inconsecutive) is intrinsically an object outlier, e.g., the speeding car. It is desirable to exploit a procedure to track the evolving abnormal behaviors of objects from the starting timebin to the recent timebin in current time window. At each timebin, upon receiving the detected results (i.e.,*EO-outlier* candidates) from the remote sites, the coordinator would implement *EO-outlier* detection, called *EOD*, the detailed description of which is given in Algorithm 3. In current time window, the timebin for identifying the object O_i's fragment as *F-outlier* is viewed as an anomaly timebin and inserted into $list_{O_i}$ (Lines 1-10). $list_{O_i}$ would be maintained

Algorithm 3: *EOD* (Evolutionary Moving Object Detection)

Input: A_F: a set of *F-outlier*s; thr_a: the anomaly timebin count threshold;
Output: A_O: a set of *EO-outlier*s;

```
1  foreach F-outlier in A_F do
2      Obtain the object ID (denoted as O_i) that F-outlier belongs to;
3      if list_{O_i} not exists then
4          Create list_{O_i};
5          list_{O_i} ← ∅;
6      else
7          /* T_o denotes the oldest timebin of list_{O_i} */
8          while T_o < T_c − N + 1 do
9              list_{O_i} ← list_{O_i} − T_o;
10         list_{O_i} ← list_{O_i} ∪ T_c;
11         if |list_{O_i}| ≥ thr_a then
12             A_O ← A_O ∪ {O_i};
13 return A_O;
```

with the slide of time window, i.e., the obsolete anomaly timebins shall be eliminated as new one is inserted into $list_{O_i}$ (Lines 8-10). Once the size of $list_{O_i}$ grows beyond the anomaly timebin count threshold thr_a, the object O_i would be reported as an *EO-outlier* (Lines 11-12).

4.3 Time Complexity Analysis For each timebin, given the maximum number of incoming trajectories n and the window size N, execution overhead of *ODDTS* algorithm is composed of two parts: (i) the operational overheads of parallel outlier detection on the remote sites and *EO-outlier* detection on the coordinator, and (ii) the time to transfer detection results from the remote sites to the coordinator. Let n_p ($n_p < n$) denote the maximum number of the incoming trajectory data on the individual remote site at a certain timebin, *F-outlier* and *FC-outlier* detection incurs a complexity of $O(n_p \log n_p)$ using *STR-tree* index. Let n_O denote the maximum number of moving objects in current time window. When all of the objects have *F-outlier*s at a certain timebin, the worst computational cost of coordinator is $O(n_O)$, which scarcely happens. Let Com_k denote the cost of transmitting *EO-outlier* candidates from one remote site to the coordinator, the whole transferring overhead Com is $\sum_{k=1}^{M} Com_k$. Let n_{EO} denote the maximum number of *EO-outlier* candidates at some timebin, Com at most requires $O(M(n_{EO}))$, and the worst-case cost is $O(n_O)$. Actually, the communications can even be avoided when none of *F-outlier*s is detected by any remote site. Since $n_p \gg n_O$, *ODDTS* algorithm approximately takes $O(n_p \log n_p)$, and even an ideal complexity of $O(\frac{n}{M} \log \frac{n}{M})$. That the execution overhead is in the linear order of M indicates that *ODDTS* is well scalable with regard to the number of remote sites. As compared to the computational cost of the centralized version (i.e., $M = 1$, $O(n \log n)$), the execution overhead of *ODDTS* is significantly reduced owing to parallel outlier detections of the remote sites with minimal communication cost.

5 Empirical Evaluation

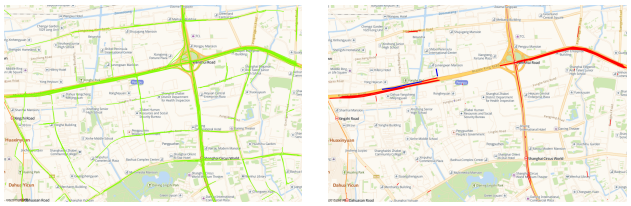
In this section, we conduct extensive experiments on two real datasets to assess effectiveness and efficiency of *ODDTS* over distributed streams. It is worth noting that none of the existing solutions is tailored to trajectory outlier detection over distributed streams. Thus, they are not suitable to compare with *ODDTS*.

All the experiments are conducted on a cluster of 5 nodes running *Spark-2.1.0-bin-hadoop 2.7* on *centos 7.2*. Each node consists of 20 physical cores *Intel 2.2GHz* processors, and the nodes are interconnected with 10Gbps Ethernet. The values of parameters are set for each dataset based on our experimental tuning. Unless mentioned otherwise, the window size N is set to 30 minutes, and *timebin* is set to 2 minutes.

5.1 Datasets We evaluate our proposed method on two real datasets: taxi trajectory dataset of 2013 (or $Taxi13$, for short), and taxi operational dataset of 2015 (or $Taxi15$, for short). During the experiments, we simulate the trajectory stream via sending data from the disk files to the compute cluster with a specified transfer rate. The transfer rate can be adjusted by changing the number of streaming elements at each timebin.

Taxi13 contains more than 400GB of trajectory data generated by the taxis of Shanghai and Beijing in the fourth quarter of 2013. It includes four attributes of timestamp, velocity, longitude and latitude coordinates. To evaluate the robustness of *ODDTS* for trajectory data in different scale of regions, we derive two datasets of Shanghai from $Taxi13$: *Taxi13-1* with 1.89 million records, and *Taxi13-2* with 3.69 million records.

Taxi15 contains more than 200GB of trajectory data derived by 13,600 taxis of Shanghai in April 2015. It has about 114 million points per day with six attributes of Vehicle ID, Time, Longitude and Latitude, Speed, and Taxi Status (free/occupied), etc. We select a test area consisting of about 315 road segments, and each road segment has at least one lane in each direction. The ground truth outlier set is manually verified through comparative velocity analysis for per



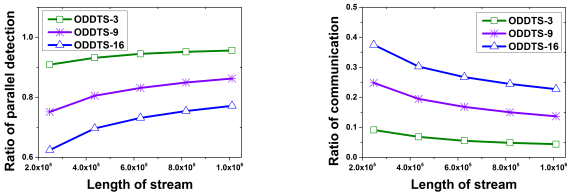
(a) Movement distribution (b) Outlier detection result

Figure 2: Outlier detection on *Taxi15*

trajectory fragment (or road segment) and its neighbors at each timebin by the volunteers. Labeling of the outliers is determined by majority of volunteers' voting.

5.2 Effectiveness Evaluation For effectiveness validation purpose, we conduct *ODDTS* (setting with 16 remote sites) on $Taxi15$. We aim to identify *F-outlier*s, *FC-outlier*s and *EO-outlier*s according to velocity feature. We choose longitude and latitude coordinate as SF, and regard velocity as DF. Local outlier thresholds are empirically set as follows: $\rho = 2.5$, $\rho_c = 3.5$. The portions of test area and the outlier detection results are visualized in Fig.2. Fig.2 (a) shows the movement distribution of taxis' traces (in light green) within [8:00, 8:30] a.m. on April 15. The average speed of most roads is not beyond $30km/h$. Outliers at timebin ([8:29-8:30]) detected by *ODDTS* are illustrated in Fig.2 (b). *F-outlier*s with thin red lines indicate that only a few abnormal trajectories (with speed of $83km/h$) occur on parts of roads. *EO-outlier* (in blue) represents an overspeed car (with mean velocity of $155km/h$ during multiple timebins). *FC-outlier*s (in thick red) show that the roads where a majority of taxis travel on have the significantly different speed ($68km/h$) from their neighbors (with mean speed of $25km/h$). Outlier detection result is coincide with the ground truth outliers. During peak hours, such information like the roads with high speed is pretty useful to help the drivers for optimal route planning in real time.

Metrics. In a bid to further verify the effectiveness of *ODDTS*, we use *F-measure* as the criteria measurement. It is defined as $F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$, where $Precision = \frac{|R \bigcap D|}{|D|}$ and $Recall = \frac{|R \bigcap D|}{|R|}$. R denotes the manually labeled trajectory outlier set, and D denotes the detected outlier set by our proposal. *F-measure* reaches a high value only when both *Precision* and *Recall* are high. To understand how the parameters impact on outlier detection, we implement *ODDTS* with different thresholds (including d, d_c, ρ, ρ_c, thr_a) setting to evaluate the value of *F-measure*, as illustrated in Fig.3. Note that d uses the variation of latitude and longitude, the distance of 0.0001 is corresponding to about 11 meters. We observe that *ODDTS* obtain almost the high *F-measure* for *F-outlier* detection when $\rho \geq 3$ and $d \geq 0.003$, for *FC-outlier* detection when $\rho_c \geq 3.5$ and $d_c \geq 0.008$, and for *EO-outlier* detection when $d > 0.004$ and $thr_a \geq 7$. It can be concluded that *ODDTS* attains better detecting validity as long as the thresholds are set appropriately.

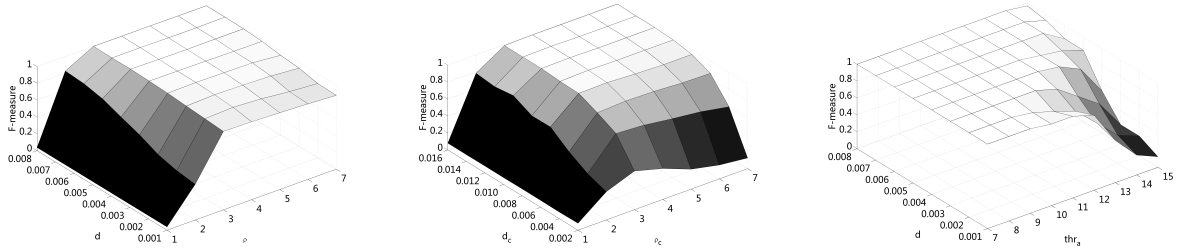


(a) parallel detection cost (b) communication overhead

Figure 4: Ratio of detection and communication overhead to total execution overhead

5.3 Efficiency Evaluation We proceed to assess the efficiency of *ODDTS*. The whole execution overhead is broken down into time spent on key phases: the parallel detection cost of the remote sites, the detection cost of the coordinator, and the communication overhead for transferring outlier detection results. Since the detection cost of the coordinator is a tiny proportion of the overall execution cost, we mainly estimate the parallel detection cost and the communication overhead by implementing *ODDTS* on *Taxi13-2*. The number of trajectories gradually grows from $250k$ to $1,010k$. Noted that parallel outlier detection phase would finish after every remote site has completed its detection, the maximal parallel detection cost is chosen as the evaluation criterion. Fig.4 (a) reports the ratio of the maximal parallel detection cost to the total execution time. We can see that the parallel detection overheads of *ODDTS* with three different number of the remote sites (abbreviated as *ODDTS-3*, *ODDTS-9*, *ODDTS-16* respectively) increase as the trajectory data continue to flow in, while *ODDTS-16* guarantees the greatest time savings. It can be concluded that the overall execution cost is mainly determined by the time spent to implement outlier detection in parallel for all the remote sites. Owing to the high efficiency of parallel detection phase, less detection cost would be spent as more remote sites participate in the parallel detection procedure.

As far as communication overhead is concerned, Fig.4 (b) shows the ratio of communication time to the total execution time. We observe that the ratio reduces with data size, and communication overhead grows slightly with the increase of number of remote sites. Fortunately, even though the amount of trajectory data reaches $1,010k$, communication cost of *ODDTS-16* is at most the ratio of 22% of the total execution over-



(a) *F-outlier* detection result (b) *FC-outlier* detection result (c) *EO-outlier* detection result

Figure 3: *F-measure* ($Taxi15$)

head. As more remote sites execute outlier detection in parallel, a little more time is spent to transfer the outlier detection results from the remote sites to the coordinator site during detection process. However, instead of transferring all the data to the coordinator for detecting, *ODDTS* simply needs to transmit the detection results on local streams. The communication cost of *ODDTS* is far less than the time savings that parallel detections bring about, and hence is always a small portion of the whole execution overhead.

To verify the effectiveness of pruning process (depicted in Section 4), we compare the running time of *ODDTS* with that of its unpruned version (denoted as $ODDTS_{no}$) by implementing them on *Taxi15*. As shown in Fig.5 (a), the execution overheads of *ODDTS-9* and *ODDTS-16* are significantly improved, relative to the solutions ($ODDTS_{no}$*-9* and $ODDTS_{no}$*-16*) without pruning process. In addition, *ODDTS-16* saves more time for parallel detection. This is due to that a great deal of unnecessary *ldd* calculations are pruned and thus the cost of parallel detection procedure are greatly reduced.

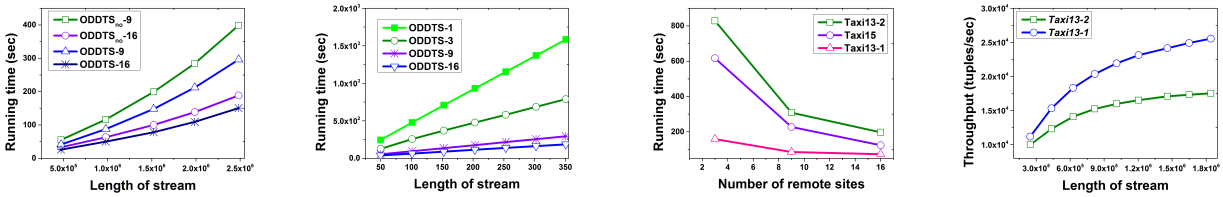
Then, we evaluate the impact of data scale on execution overhead of *ODDTS* by comparing it with its centralized version (abbreviated as *ODDTS-1*). We implement *ODDTS* on *Taxi13-2* using different number of remote sites ($M = 1, 3, 9, 16$ respectively). As illustrated in Fig.5 (b), when the trajectory data continue to flow in (from $490k$ to $3,500k$), the costs of all the approaches scale linearly with data size. But the distributed versions perform much better than the centralized version, especially *ODDTS-16* shows the best performance. As discussed earlier, a little more time is required to transfer the outlier detection result when using more remote sites, but in the meantime, the total execution overhead is sharply reduced owing to vast time savings of parallel detection procedure. Therefore, the performance of the distributed solution is obviously improved as compared to the centralized version.

Also, we studied the scalability of *ODDTS* with respect to the number of remote sites. We conduct *ODDTS* on *Taxi13-1*, *Taxi13-2* and *Taxi15* separately. As shown in Fig.5 (c), as more remote sites are used, the total time consumptions of *ODDTS* on three datasets are appreciably reduced. It also demonstrates the advantage of *ODDTS* contributed by the efficient parallel detection phase with minimal communication cost. Transmission overhead is much smaller than the time savings of parallel detections and less sensitive to the variation of number of the remote sites. Therefore, *ODDTS* provides significant scalability advantages when a large number of remote sites are available.

Further, to verify the performance gain of *ODDTS*, we conduct a comparative analysis on the throughput of *ODDTS-16* on *Taxi13-1* and *Taxi13-2*. Throughput is defined as the average amount of tuples processed each second. As can be observed in Fig.5 (d), as compared to larger scale dataset (*Taxi13-2*), the smaller one (*Taxi13-1*) provides higher throughput owing to less data processed in every timebin. Moreover, both the throughputs of *ODDTS* on two datasets scale linearly with the increase of data. This also confirms the scalability of *ODDTS* with respect to increasing amount of data. Such significant performance gain benefits from parallel detection procedure. The above experiments establish that *ODDTS* can handle distributed trajectory streams in a promising efficiency.

6 Conclusion

With the substantial increment of distributed trajectory stream data, it is indispensable to exploit a distributed trajectory outlier detection solution to satisfy the needs of emerging big data applications. To this end, on the basis of distributed computing platform, we first propose a two-phase distributed *feature-grouping based* outlier detection method, termed as *ODDTS*, to identify three types of trajectory outliers (*F-outlier*, *FC-outlier* and *EO-outlier*) over distributed streams. Our proposal can achieve obvious performance gains through parallel outlier detection mechanism with the minimal transmission overhead among the nodes. The evalua-



(a) Running time versus pruning optimization (b) Running time versus data scale (c) Running time versus number of remote sites (d) Throughput versus data scale

Figure 5: Execution overhead

tive experiments on real-world data proved that *ODDTS* could obtain high detection precision and scale well for ever-growing data volumes and increasing number of the remote sites. Moreover, our distributed solution is generic and can be applied in most of parallel and distributed trajectory processing scenarios.

7 Acknowledgements

This work is supported by the National Key Research and Development Program of China (No.2016YFB1000905), NSFC (Nos.61702423, 61370101, 61532021, U1501252, U1401256 and 61402180), Natural Science Foundation of the education department of Sichuan Province (No.17ZA0381), Special Fundation of National Programme Cultivation (No.16C005) and Meritocracy Research Funds of China West Normal University (No.17YC158).

References

[1] H. Liu, C. Jin, and A. Zhou, "Popular route planning with travel cost estimation," in *DASFAA*, 2016, pp. 403–418.

[2] X. Duan, C. Jin, X. Wang, A. Zhou, and K. Yue, "Real-time personalized taxi-sharing," in *DASFAA*, 2016, pp. 451–465.

[3] T. Wang, J. Mao, and C. Jin, "Hymu: A hybrid map updating framework," in *DASFAA*, 2017, pp. 19–33.

[4] P. Lei, "A framework for anomaly detection in maritime trajectory behavior," *KIS*, vol. 47, no. 1, pp. 189–214, 2016.

[5] J. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *ICDE*, 2008, pp. 140–149.

[6] Y. Ge, H. Xiong, C. Liu, and Z. Zhou, "A taxi driving fraud detection system," in *ICDM*, 2011, pp. 181–190.

[7] J. Mao, T. Wang, C. Jin, and A. Zhou, "Feature grouping-based outlier detection upon streaming trajectories," *TKDE*, vol. 29, no. 12, pp. 2696–2709, 2017.

[8] X. Li, J. Han, S. Kim, and H. Gonzalez, "ROAM: rule- and motif-based anomaly detection in massive moving object data sets," in *SDM*, 2007, pp. 273–284.

[9] X. Li, Z. Li, J. Han, and J. Lee, "Temporal outlier detection in vehicle traffic data," in *ICDE*, 2009, pp. 1319–1322.

[10] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing, "Discovering spatio-temporal causal interactions in traffic data streams," in *SIGKDD*, 2011, pp. 1010–1018.

[11] S. Chawla, Y. Zheng, and J. Hu, "Inferring the root cause in road traffic anomalies," in *ICDM*, 2012, pp. 141–150.

[12] B. Pan, Y. Zheng, D. Wilkie, and C. Shahabi, "Crowd sensing of traffic anomalies based on human mobility and social media," in *SIGSPATIAL*, 2013, pp. 334–343.

[13] Y. Bu, L. Chen, A. W. Fu, and D. Liu, "Efficient anomaly monitoring over moving object trajectory streams," in *SIGKDD*, 2009, pp. 159–168.

[14] Y. Yu, L. Cao, E. A. Rundensteiner, and Q. Wang, "Detecting moving object outliers in massive-scale trajectory streams," in *SIGKDD*, 2014, pp. 422–431.

[15] Y. Ge, H. Xiong, Z. Zhou, H. T. Ozdemir, J. Yu, and K. C. Lee, "Top-eye: top-k evolving trajectory outlier detection," in *CIKM*, 2010, pp. 1733–1736.

[16] C. Chen, D. Zhang, P. S. Castro, N. Li, L. Sun, S. Li, and Z. Wang, "iboat: Isolation-based online anomalous trajectory detection," *TITS*, vol. 14, no. 2, pp. 806–818, 2013.

[17] J. Mao, C. Jin, Z. Zhang, and A. Zhou, "Anomaly detection for trajectory big data: Advancements and framework," *Journal of Software*, vol. 28, no. 1, pp. 17–34, 2017.

[18] L. Cao, Y. Yan, C. Kuhlman, Q. Wang, E. A. Rundensteiner, and M. Y. Eltabakh, "Multi-tactic distance-based outlier detection," in *ICDE*, 2017, pp. 959–970.

[19] Y. Yan, L. Cao, C. Kuhlman, and E. A. Rundensteiner, "Distributed local outlier detection in big data," in *SIGKDD*, 2017, pp. 1225–1234.

[20] Q. Ma, B. Yang, W. Qian, and A. Zhou, "Query processing of massive trajectory data based on mapreduce," in *CIKM*, 2009, pp. 9–16.

[21] D. Zeinalipour-Yazti, C. Laoudias, C. Costa, M. Vlachos, M. I. Andreou, and D. Gunopulos, "Crowdsourced trace similarity with smartphones," *TKDE*, vol. 25, no. 6, pp. 1240–1253, 2013.

[22] Z. Zhang, Y. Wang, J. Mao, S. Qiao, C. Jin, and A. Zhou, "DT-KST: distributed top-k similarity query on big trajectory streams," in *DASFAA*, 2017, pp. 199–214.

[23] J. Mao, Q. Song, C. Jin, Z. Zhang, and A. Zhou, "Tscluwin: Trajectory stream clustering over sliding window," in *DASFAA*, 2016, pp. 133–148.