ABSTRACT
Massive trajectory data with spatio-temporal information are generated from various devices everyday. Due to a wide variety of devices and data collection methods, these data differ in terms of frequency and precision, which brings challenges to data management and analysis. This paper presents CTDM, a cloud-assisted framework for trajectory data management and analysis, which adds an edge layer between clients and cloud servers. With the assistance of edge devices, CTDM supports fast data acquisition of extensive trajectory data, and reduces the WAN bandwidth and computation overhead for cloud servers by calibrating and compressing data locally. Cloud servers focus on data interpolation based on trajectory similarity to handle data in various frequencies, deriving more accurate trajectory features. To demonstrate the feasibility and performance of CTDM with a preliminary case study of commute behavior analysis using real dataset, which shows 88% of the calculated commute distance has the relative deviation less than 20% and 80% of the calculated commute time deviates less than 10 minutes.

KEYWORDS
cloud-assisted system, trajectory data; trajectory clustering; commute behavior analysis

1 INTRODUCTION
With the rapid development of pervasive smart devices and the popularity of ubiquitous network technologies, massive trajectory data with spatio-temporal information of users are generated. The data could be reported by users actively, like the GPS check-in data; or generated passively, e.g., cellular network base stations usually record the communication events of users, which include the user’s location [10]. These trajectory data have been widely used in understanding human activity [4], optimize personalized recommendation [2], traffic prediction [8], network management [5], etc.

However, due to the variety of devices and location data acquisition methods, the frequency and precision of different trajectory may differ a lot, posing a number of challenges in trajectory data management and analysis. First, it is hard to make unified management with data from different sources without fully utilizing them. For example, a user may carry several devices, each revealing some pattern of the user, while the cellular base station and WiFi access points may offer insights on his geo-location. By integrating the trajectory data accumulated from different resources, more precise profile of a user may be derived. Second, the precision and frequency of positioning data of different devices may vary. Even with the same device, different location positioning methods may generate data with different precision. This heterogeneity makes it difficult to manage and analyze trajectory data. Third, it is costly to send all trajectory data to cloud data centers for processing. The bandwidth requirement and network delay between clients and the cloud data center could be non-trivial. Also, those raw positioning data may contain duplicates and outliers, wasting network resources.

To address these challenges, we propose CTDM, a cloud-assisted framework for trajectory data management and analysis, which includes a client layer, an edge layer and a cloud layer. All reported or sensed positioning data are first preprocessed in the edge servers, which are close to client devices and have smaller network delay [11]. The edge server will combine all data of a same user together, remove duplicates and make best estimation of the user’s location from available data. Lots of network and computation resources can be saved since the data first get preprocessed at the edge before sending to the cloud server. The cloud server would analyze all positioning data, recognize trajectories and interpolate them based on their similarity, thus obtaining more detailed
trajectories of the end users. Based on the interpolated data, activity features can be mined and made available to other applications through open APIs. To prove the feasibility of CTDM, a case study of daily commute behavior is presented. Our experimental results show that CTDM is able to compute the commute metrics quite precisely.

The rest of this paper is organized as follows. Section 2 presents the architecture of our framework. Section 3 introduces a case study of commute behavior analysis. Section 4 gives the experimental results and evaluation of the framework. Finally, Section 5 concludes this paper.

2 CLOUD-ASSISTED FRAMEWORK
The CTDM framework includes a client layer, an edge layer and a cloud layer. The edge devices and servers are close to the end user to reduce network delay and preprocesses positioning data, so that all data of the same user are correlated and best location estimation of user at a certain moment is sent to the cloud servers. Since all the user’s preprocessed data are available in the cloud, data interpolation can be done based on trajectory similarity and more trustful features can be mined. Figure 1 shows the CTDM architecture overview.

A client can be any device (mobile phone, tablet etc) and may actively send positioning data to the edge servers. The edge servers (e.g., smart wireless routers and cell towers) can sense the position of clients. For example, every time a cell tower accepts a client’s network request, it will log a record that the client is near the tower and estimates the client’s location based on the signal strength. Edge devices and servers integrate the positioning data reported by the clients and those obtained through position sensing, and store them into the raw data storage. The raw data is preprocessed to remove noise and anomalies; further positioning calibration is performed where adjacent points are jointly considered to improve the precision. A client may report the same location multiple times even if it doesn’t move significantly, and the sensed positions may be duplicated with those reported by clients. In CTDM, edge devices and servers remove duplications and data compression to reduce data size. After that, edge device and servers send the compressed data to the cloud servers.

Cloud servers may receive positioning data out of order, so they re-arrange the data and recognize trajectories from it. Due to the unstable frequency of the positioning data, points in some trajectory could be very sparse and even parts of the trajectory may get lost. It’s of great importance to complete the trajectory and thus obtain a more accurate understanding of it. Cloud servers take advantage of the similarity between trajectories to perform data interpolation, insert proper points into them, and save the augmented data into persistent storage. Combining with trajectories and augmented data, cloud servers mine the features of those trajectories and store the mined features into storage. Applications can be built on the mined features based on the open APIs offered by cloud servers.

3 CASE STUDY: COMMUTE BEHAVIOR ANALYSIS
Based on the proposed framework, a prototype system is implemented for commute behavior analysis. We developed a web app to collect positioning data, which runs on laptops and mobile phones. For enhanced user experience, a native mobile app (currently Android version is available) is also developed for end users. All these clients send data to the nearest edge server.

The edge server first performs anomaly detection to eliminate the positioning data with too much deviation from those temporally adjacents. It then merges the data from different sources and chooses the data with the highest precision as the user’s position. Data calibration is performed where adjacent points are jointly considered to improve the precision, which will be detailed in Section 3.2. The edge server may receive the same positioning data multiple times during a relatively long time period, and it compresses the data by resampling and sends them to the cloud servers.

We design and implement an algorithm, so-called similar commute trajectory clustering (SCTC) in the cloud server. It first identifies the user’s home and company locations, and recognizes the commute trajectories. Data interpolation is performed by clustering similar trajectories, so that more
detailed daily trajectory can be obtained. With the help of merged trajectories, important metrics, including commute distance and time, can be calculated.

3.1 Data Collection
Currently, we implement an Android and a web version of data collection application for client devices. In the Android version, captured positioning data are stored in a local SQLite database in case of poor network connection and sent in a batch manner to save energy. We take two strategies to obtain the geo-location: use GPS provider when the user is out of WiFi area and the battery is adequate, and switch to network provider when there is a Wi-Fi connection or the battery is low [3]. We will reduce the sampling frequency when the user doesn’t move and record the user’s location every 5~15 minutes (depending on the type of devices). We use the geolocation SDK of Tencent map [7] in the web version. Due to the privacy and permission issue of the devices, the web version needs to run in foreground. The web app enables users to report their location actively so that users won’t need to keep the web app running in foreground all the time.

We deploy the edge servers in the network of China Telecom, one of the largest ISPs in China and collect 88 volunteers’ data in 5 cities including Beijing, Yangzhou, etc. Volunteers participate in the project for 1 week to 1 month, and there are 601 person-days in total. To help us in ground-truth validation, each volunteer needs to finish an online survey including questions about the departure/arrival time, home/company location, commute mode, etc.

3.2 Positioning data calibration
We find that part of collected position data suffers from drifting, that is, the observed location varies much more than the user actually moves. Usually, it is caused by the switch of surrounding environments, e.g., the user walks through a building, since the mobile device is located based on the communication with positioning satellites while the signal intensity varies when environments change. We tackle this problem by smoothing positioning data in a fixed time window. The calibration is done in the edge server and the calibrated data is sent to the cloud afterwards.

3.3 Commute trajectory recognition and clustering
SCCTC employs the hierarchical clustering algorithm to aggregate positioning data into clusters according to their distance and identify the location of a user’s home and company out of these clusters. We cluster the data in a bottom-up manner and take the complete linkage as the linkage scheme.

Define cluster as $C = \{R\}$, where $R$ is the positioning record, and each record has timestamp $R.time$, longitude $R.lng$ and latitude $R.lat$. Let $c_j$ denote the geographic center of $C_i$, and its longitude and latitude can be approximated as the average of all points in $C_i$.

We define the period from 9:00 p.m. to 6:00 a.m. as night, and take the cluster including most records at night as the user’s home cluster $c_h$. Its geographic center is taken as the user’s home address $c_h$. For each remaining cluster, we calculate the total distance of records in it to $c_h$, and choose the one with largest total distance as the user’s company cluster $C_w$. The location of user’s company is $C_w$’s geographic center $c_w$.

The positioning points in the commute process can be identified between $c_w$ and $c_h$, and we can get the commute trajectory by arranging them. However, there are two challenges: 1) the positioning data collected in the commute process are sparse and irregular. 2) Due to factors like weather or traffic control, a user may choose different commute modes and routes.

SCCTC addresses these two challenges by merging similar trajectories to obtain a reference trajectory approximating the real one. Section 3.3.1 defines the distance metric $merge distance$ between two trajectories and Sections 3.3.2 and 3.3.3 show the steps to merge all similar trajectories.

3.3.1 Distance between trajectories. There are several distance metrics between two trajectories such as Euclidean distance, edit distance and DTW distance[9]. However, these metrics are susceptible to outliers. For two commute trajectories generated by a user taking the same route, if one trajectory misses some part of the route, the calculated distance would be very large and therefore may not be suitable for measuring the similarity between two trajectories following the same route.

If we merge two trajectories following the same route, the merged trajectory should be spatially similar to the original ones, thus the length of merged trajectory should not increase much. To this end, we propose $merge distance$ as the distance between two commute trajectories. Let $L_T$ be the length of trajectory $T$, $M$ be the generated trajectory after merging trajectory $A$ and $B$, their merge distance can be calculated as $d_{merge} = L_M\max(L_A, L_B)$.

3.3.2 Spatio-temporal merging of two trajectories. There are two dimensions of trajectory, i.e., temporal and spatial ones. The temporal dimension determines how much time spent to travel between two consecutive points, while the spatial dimension indicates the shape of a trajectory. Different from most existing methods which considers only one dimension, we merge trajectories both in temporal and spatial dimensions.

**Merge in spatial dimension:** Sort the positioning data of trajectory $T_a$ and $T_b$ in chronological order, and extract the longitude and latitude array of each trajectory as $P_a$ and
Algorithm 1: SpatialMerge

Input: Positioning arrays $P_a$, $P_b$, user’s home location $c_h$ and company location $c_w$
Output: The merged position array $P_m$

$P_m = \emptyset$
insert $c_h$ into $P_m$
insert all points of $P_a$ into $P_m$
insert $c_w$ into $P_m$
$pos = 0$
foreach $r_b$ in $P_b$
do
$pos_m$ = the insertion index making minimum length increasement of $P_m$ after $pos$
insert $r_b$ into $P_m$ at $pos$
end
$pos = pos_m + 1$
remove $c_h$ and $c_w$ from $P_m$

Algorithm 2: TemporalMerge

Input: Positioning arrays $P_a$, $P_b$, traveling time array $D_a$, $D_b$ of trajectory $T_a$ and $T_b$, positioning data array $P_m$ of the merged trajectory $T_m$
Output: The traveling time array $D_m$ of the merged trajectory $T_m$

initialize $D_m$
$D_m'[i] = AmortizeTime(P_a, D_a, P_m)$
$D_m'[i] = AmortizeTime(P_b, D_b, P_m)$
foreach $i$ in $[0, size(P_m) - 1]$ do
$D_m[i] = D_m'[i] + D_b'[i]$
end
Function $AmortizeTime(P_a, D_a, P_m)$:
initialize $D_m'$
foreach $t$ in $D_k$ do
find the start point $p_{start}$ and end point $p_{end}$ in $P_m$
corresponding to $t$
find the corresponding index $index_{start}$ and $index_{end}$ of $p_{start}$ and $p_{end}$ in $P_m$
calculate the total distance $d$ among points of $P_m$
between $p_{start}$ and $p_{end}$
for $i$ in $[index_{start}, index_{end}]$ do
$D_m'[i] = distance(P_m[i], P_m[i + 1])/d \times t$
end
end
return $D_m'$

where trajectories is all the trajectories of this cluster, and trajectory$_m$ is the generated trajectory after merging all trajectories in the cluster: trajectory$_m = \{P, D\}$, where $P$ is the positioning array and $D$ is the traveling time array. Algorithm 3 presents the procedure of clustering trajectory.

3.4 Feature Analysis

Section 3.3.3 groups similar commute trajectories into one cluster and gets a merged trajectory after merging all the trajectories in the same cluster in both temporal and spatial dimensions. This section takes the merged trajectory as a reference for missing part completion, based on which we calculate the key spatio-temporal metrics.

3.4.1 Spatial Metric. Commute distance is an important spatial metric. We can find the containing cluster for each commute trajectory, and take the corresponding merged trajectory as a reference of that day. Note that the user’s home and company are not included in the reference trajectory, so the commute distance is the sum of the distance from home to the first point of the reference trajectory, its length and the distance from its last point to the company.

3.4.2 Temporal Metric. A user’s commute behavior has 3 important temporal metrics: departure time, arrival time and commute time. Although we could take the merged
ALGORITHM 3: TrajectoryClustering

Input: Trajectory set \( S \), the user’s home location \( c_h \), the user’s company location \( c_w \), maximum distance threshold \( d_{\text{max}} \)

Output: Trajectory cluster set \( \text{clusters} \)

foreach \( T \) in \( S \) do

extract the position array \( P \) and calculate the traveling time array \( D \) of \( T \)

insert \( (P, D, \{T\}) \) into clusters

end

while size (clusters) \( > \)1 do

find the closest cluster \( C_i \) and \( C_j \) in clusters

\( d_{\text{min}} = \) the shortest distance in clusters

if \( d_{\text{min}} > d_{\text{max}} \) then

break

end

get the merged trajectories \( (P_i, D_i) \) and \( (P_j, D_j) \) from \( C_i \) and \( C_j \)

\( P_m = \) SpatialMerge\( (P_i, P_j, c_h, c_w) \)

\( D_m = \) TemporalMerge\( (P_i, P_j, D_i, D_j, P_m) \)

use \( (P_m, D_m) \) as the merged trajectory to initialize cluster \( C_m \) and insert all the trajectories in \( C_i \) and \( C_j \) into \( C_m \)

insert \( C_m \) into clusters remove \( C_i \) and \( C_j \) from clusters

end

trajectory as a reference for trajectory completion, it doesn’t contain the user’s home and company information, thus we cannot obtain the departure and arrival time directly from it.

A user may have different commute modes on different days, thus the commute speeds may differ. Here we first calculate the average commute speed and then use it to estimate the user’s traveling time from home to the first point on the merged trajectory and that from the last point to company.

Let \( p_{\text{start}} \) and \( p_{\text{end}} \) denote the first and last point of the trajectory on a certain day, and let \( t_{\text{start}} \) and \( t_{\text{end}} \) be their corresponding timestamp. Segment the merged trajectory into three parts: \( \text{trajectory}_{\text{start}} \), \( \text{trajectory}_{\text{mid}} \) and \( \text{trajectory}_{\text{end}} \) with \( p_{\text{start}} \) and \( p_{\text{end}} \), as shown in Figure 2.

Calculate the length of \( \text{trajectory}_{\text{mid}} \) as \( l_{\text{mid}} \), then the average speed on that day can be calculated as \( v = l_{\text{mid}} / (t_{\text{end}} - t_{\text{start}}) \). Calculate the distance from home to \( p_{\text{start}} \) as \( d_{\text{home}} \), from \( p_{\text{end}} \) to company as \( d_{\text{comp}} \), then the corresponding traveling time can be calculated as: \( d_{\text{home}} = l_{\text{home}} / v \), \( d_{\text{comp}} = l_{\text{comp}} / v \). The corresponding traveling time of \( \text{trajectory}_{\text{start}} \) and \( \text{trajectory}_{\text{end}} \) can be obtained from the traveling array \( D \), and thus the departure time \( t_{\text{departure}} \), arrival time \( t_{\text{arrival}} \) and commute time \( t_{\text{commute}} \) can be calculated as:

\[
\begin{align*}
    t_{\text{departure}} &= t_{\text{start}} - d_{\text{start}} - d_{\text{home}} \\
    t_{\text{arrival}} &= t_{\text{end}} + d_{\text{end}} + d_{\text{comp}} \\
    t_{\text{commute}} &= t_{\text{arrival}} - t_{\text{departure}}
\end{align*}
\]

4 PRELIMINARY EVALUATION

This section first compares \( \text{SCTC} \)’s accuracy with existing methods, and then presents evaluation of the cloud-assisted framework in terms of resource saving.

Existing methods estimate the commute distance in two ways. One of them takes the crow-fly distance \([4]\), i.e., great circle distance, to estimate the real commute distance; another calculates the commute distance based on the filtered commute trajectory, we mark it as filter \([1]\).

In terms of temporal metrics estimation, we compare the accuracy between \( \text{SCTC} \) with \( \text{diff} \) \([6]\), which calculates the time difference between first not-home point and last not-company point in the morning as the commute time.

4.1 Spatial Metric

Figure 3 shows the CDF of relative distance deviation of the 3 algorithms. Due to the sparsity of the positioning data and incomplete commute trajectory, both of the commute distances calculated by filter and crow-fly are small and their average relative deviations are -0.13 and -0.30 respectively. \( \text{SCTC} \) merges the similar trajectories to get a more precise trajectory, thus overcomes the data sparsity problem. The average relative deviation of \( \text{SCTC} \) is only -0.06.

4.2 Temporal Metrics

Figure 4 shows the departure deviation of \( \text{SCTC} \) and \( \text{diff} \). As shown in Figure 4, taking the timestamp of the first not-home point as departure time is not precise, with an average deviation of \(-1011\) seconds (about 17 minutes earlier). \( \text{SCTC} \)
is more accurate since it uses the merged trajectory to calculate the speed and then estimate the departure time. The average deviation of SCTC is only −148 seconds (about 2.5 minutes ahead of time).

Figure 5 shows the frequency distribution of commute time deviation. Because of more accurate arrival time estimation, SCTC outperforms diff in commute time estimation, with 80% time deviation kept in 10 minutes.

4.3 Resource Saving
CTDM significantly reduces the volume of network traffic to be sent the cloud servers and also minimize the use of computation resources in mobile devices. In the real dataset of the case study, the average amount of raw positioning data a user generates per day is 164 kB. The edge servers reduce 66.89% of the data through preprocessing, anomaly value elimination and deduplication. If the proposed framework gets fully deployed in China Telecom, it will save 3.3 TB network traffic for 1 million users per month. Based on our experiments in 64-core 2.2 GHz high-performance server, the average time spent for behavior analysis is 48.77 seconds per user. CTDMy also offloads some computation task to cloud servers, and it will save 1855.89 seconds (around 31 minutes) of computation in the mobile devices like iPhone 6S, which has two cores and a 2.2 GHz processor.

5 CONCLUSION AND FUTURE WORK
This paper investigated the difficulties of massive positioning data processing and proposed CTDMy, a cloud-assisted framework for trajectory data management and analysis. Through separation of the roles of edge and cloud servers, CTDMy can support massive data acquisition and handle trajectory data in a wide variety of frequency and precision. A case study of commute behavior analysis is conducted under this framework and extensive experiments show its feasibility and effectiveness. Based on the calculated commute metrics, more applications can be implemented such as smart home, traffic control prediction, etc. Currently, CTDMy has been partially deployed in one of the largest teleco operators in China; more applications and results are expected with more data from more volunteering users in the near future.

ACKNOWLEDGMENTS
This work is partially supported by Tsinghua University Initiative Scientific Research Program (Grant No. 20161080066) and EU FP7 MobileCloud project (Grant No. 612212).

REFERENCES