LEARNING-BASED HIGH-THROUGHPUT DISPATCHING FOR TRAJECTORY STREAMS

Xin Zhang[1,2], Guoqiang Hu[2], Ning Duan[2], Peng Gao[2], Weishan Dong[2],Jun Zhu[2], Rong Chang[3]
[1]Automation Department of Tsinghua University
z-xin07@mails.tsinghua.edu.cn
[2]IBM Research – China
{zxin, hugq, duanning, bjgaop, dongweis, zhujun}@cn.ibm.com
[3]IBM Research, USA & China
rong@us.ibm.com

Abstract
With the development of Internet-of-Things (IoT) technologies, large-volume sensor data streams are sent to cloud in near real-time which raise an important requirement for high performance sensor big data analytics. This paper focuses on the scalability challenges for analyzing large-volume mobile data streams, as mobile data streams convey valuable spatio-temporal information (termed as trajectory or semantic trajectory). A framework for high performance trajectory streams processing is proposed together with a learning method for workload assignment optimization and a dispatching method for high-throughput geo-message dispatching. By taking the spatial connectivity implied by trajectory streams into consideration, a trajectory preserving partitioning method is proposed for improving the quality of geo- partitioning. Based on the optimized geo-partitioning, a novel Geohash Tree based dispatching method is developed for achieving high-throughput geo-message dispatching. Via mobility localization formalism, we demonstrate that an implementation of our geo-spatial partition algorithms could balance workload and minimize cross-node communication. And experimental evaluations using real world and simulated data also validate the performance of the proposed methods.
Keywords: service scalability, mobile data streams, graph partitioning, trajectory, geohash tree

1. INTRODUCTION

Mobile sensors (e.g., built-in sensors on mobile phones or portable devices, and in-vehicle sensors) are pervasively available nowadays. Information such as location, speed, temperature, fuel consumption, and driving operation, can be continuously sensed and uploaded to back-end cloud environment in the form of mobile data streams through mobile network. Hence providing cloud services analyzing large volume mobile data streams becomes a key capability in many application domains including connected vehicles, smart traffic and logistic, mobile commerce, and telematics insurance.

The main payload of mobile data streams is the trajectory information. A trajectory is a sequence of spatio-temporal data records describing a journey of a moving object together with the sensed information along that journey. In recent years, trajectory analytics has got increasingly research interest (Bu, 2009), (Liu 2011), (Davics, 2006) along with the booming volume of available trajectory data. And there is also increasing requirement for analyzing trajectory streams with low latency in many applications. In real-time traffic applications that fuse sensed vehicle speed on the same road to estimate road traffic condition, Map Matching (Newson, 2009) infers a vehicle's traversing roads from a sequence (usually 5 to 20) of sensed GPS records from that vehicle. And in telematics insurance services, driving behavior is expected to be analyzed from a sequence of sensed driving operations (e.g., decelerate-turn-accelerate) immediately for alerting driving risk in-time. And also to enable online trajectory prediction in mobile commerce applications, the recent location records of a moving object, in addition to the current location, shall be considered for accurate prediction. Hence processing trajectory streams in a sliding window approach is a common scheme in many mobility analytics services. And with the growing volume of mobile clients and mobile sensors, throughput and scalability are critical issues for many cloud services analyzing trajectory streams.

In recent years, stream computing platforms (e.g., Storm, InfoSphere Streams (Gedik, 2008), and S4 (Neumeyer, 2010)) become popular for scalable and real-time data stream processing. And there is also solid research outcome on scalable streaming, such as (Andrade, 2009), (Schneider, 2009), (Khandekar, 2009), and stream data mining, such as (Gaber, 2005), (Domingos, 2000), (Chen, 2002). However, to the best of our knowledge, there is little research addressing the scalability challenge of trajectory streams processing. In existing location-based services (LBS), usually the static geo-spatial continuity is considered in geo-partitioning (i.e., partition a geo-space into multiple sub geo-areas). The partitioning result serves as rules for dispatching workload to different servers for parallel processing. While for cloud services analyzing trajectory

streams, geo-partitioning shall also take the geo-spatial continuity implied by the trajectory streams into consideration. Otherwise it would cause large amount cross-server trajectories and incur cross-server data access which will lower parallelization throughput. And service scalability will be impacted.

In this paper, we extend the trajectory preserving partitioning method (Zhang, 2014) into a scalable trajectory streams processing framework. By analyzing the unique requirement in high performance trajectory streams processing, we come up with the scalable framework and major performance measurements. Key algorithms are developed to optimize the workload partitioning and improve the efficiency of service dispatching.

The major research contributions of this paper are:

1. A framework for high performance trajectory streams analysis is proposed which covers both the offline optimal partition learning and online message dispatching.

2. A novel geo-partitioning optimization method is proposed and key algorithms are implemented for optimizing geo-partitioning taking mobility localization and workload into consideration.

3. A Geohash Tree approach and related algorithms are devised to achieve high performance geo-message dispatching.

4. The effectiveness of the proposed framework and methods is validated through experiments on both real-world and simulated data.

The remainder of this paper is organized as follows. Section 2 discusses the trajectory streams features and presents the framework for scalable trajectory streams processing. The mobility localization principle and measurements are also introduced. In section 3, the geo-partitioning optimization method is depicted and related enabling algorithms are introduced. Section 4 presents the devised algorithm for high performance trajectory streams dispatching. Experiments are described in section 5. And related work is given in section 6. Section 7 concludes the paper and highlights the future work.

2. Framework for Scalable Trajectory Streams Processing

A general scalability strategy in location-based services is geo-partitioning that enables dispatching data records (or requests) according to their geo-locations. For the benefit of computational efficiency, spatially near-by data records are dispatched to the same processing node (Jensen, 2007), (Mouratidis, 2005). So partitioning the whole geo-space into multiple geo-areas according to the geo-spatial continuity is a common practice in location-based services. After that, computation nodes are associated with partitioned geo-areas to handle corresponding data workload. Through this approach, geo-spatial message can be dispatched to corresponding computation node according to its geo-location and scalable parallel processing can be achieved on cloud.

While for mobility analytics services, beside the static geo-spatial continuity, geo-partitioning shall also take the geo-spatial relationship and workload implied by the trajectory streams into consideration. Otherwise there could be massive communication overhead caused by cross-server data access and service scalability will be degraded. Figure 1 displays an illustrating sample. Two schemes are applied for handling workload on a geo-space with 3 computation nodes. The dark dot lines are the trajectories of moving objects. The rectangles represent partitioned geo-areas and arrowed lines indicate the assignment between computation nodes and partitioned geo-areas. Intuitively scheme 1 (Upper side of Figure 1.) incurs a lot of cross-server data communication as mobility is not considered during geo-partitioning. And the data workload is unbalanced among partitions. While scheme 2 successfully suppresses cross-server communication as it manages most of trajectories within individual geo-areas. And the workload is evently distributed among partitions. With the capability of reducing cross-node communication and keeping workload balancing, parallelization efficiency will be improved hence scalability can achieved. In real-world services, where there are large volume of moving objects and trajectories and a geo-area usually has irregular shape, determining the optimized partition boundary is not a simple task.

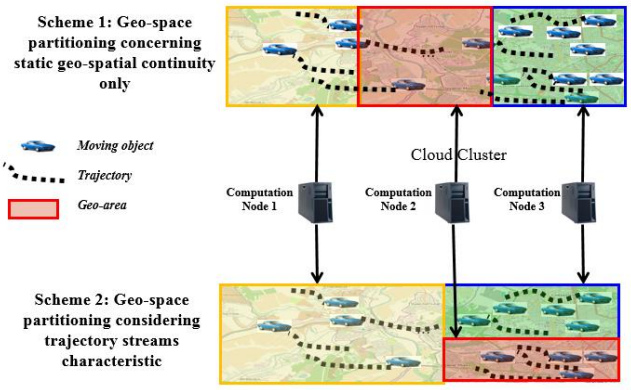


Figure 1. Assignment Schemes.

Motivated by the unique characteristic of trajectory streams, a framework for scalable trajectory streams processing is proposed and shown in Figure 2. In general, the framework consists of the offline part and the online part. The offline part learns from large volume of historical trajectory data and generates optimized geo-partitions as the base for geo-message dispatching rules. The mobility preserved geo-partitioning service associates the trajectory data with map and mines the optimal geo-area boundaries. And the geo-dispatching rule builder processes the partitioned geo-areas and generates rules for geo-message dispatching. The online part has two layers: the messaging layer and the streaming layer. The messaging layer serves as the front-end server handling massive concurrent connections and continuous network messages from mobile

devices. As a core component in the streaming layer, the geo-message dispatching service is responsible for dispatching the mobile data messages to the right computation nodes according to dispatching rules. Throughput is a critical performance measurement for the dispatching service and is one important indicator of service scalability. Modern messaging services usually take the pub-sub scheme to decouple message producers and message consumers. The lightweight MQTT protocol (Locke, 2010) enables efficient message transmitting interface between dispatching service and computation nodes. At the streaming layer, computation nodes are a cluster of processing units connecting to dispatching service with high network bandwidth. Each computation node (or a set of nodes) is responsible for processing data located in a certain geo-area. Obviously preserving trajectories locally within individual geo-areas is much more efficient than splitting trajectories onto multiple geo-areas, which incurs cross-server remote access and would significantly impact the processing latency and throughput. So a good architectural strategy is to localize a node's computation and reduce the cross-node data access or data migration as much as possible. And the more the cross-node communication exists, the less scalable the service is. Therefore geo-partitioning service is critical for service scalability. And the efficiency of dispatching service is also important to the overall runtime performance and to avoid being the bottleneck of cloud services.

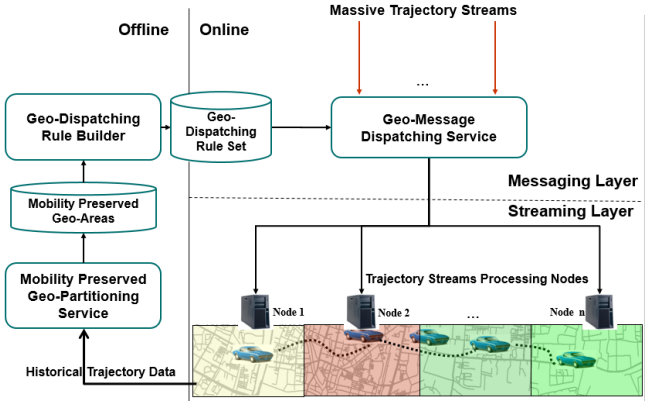


Figure 2. Framework for Scalable Mobile Data Streams Processing.

Before further introducing the methods and algorithms enabling the framework, the key performance measurements are defined in the rest of this section. We term the principle to preserve trajectories within individual partitions as the mobility localization. Technically we measure the mobility localization by Mobility Localization Index (I_{ml}) which is defined as the ratio of within-node trajectory volume to the number of whole trajectories.

$$I_{ml} = \left(\sum_i \left\| \left\{ t_j \middle| t_j \in \{Trj\}, \forall l \in t_j, l \in P_i \right\} \right\|_0 \right) \Big/ \left\| \{Trj\} \right\|_0 \quad (2.1)$$

In (2.1), $\{Trj\}$ is the whole set of trajectories used for partitioning. Usually a big volume of historical trajectory data is used representing the statistical mobility pattern of the geo-space to be partitioned. P_i is the i-th partition and t_j is a trajectory which has all of its discrete location l in one partition. And correspondingly we define the Mobility Delocalization Index as I_{md} , which is closely related with cross server communication.

$$I_{md} = 1 - I_{ml} \quad (2.2)$$

Besides the mobility localization, workload balance is another measurement for partitioning quality. In existing geo-partitioning, usually some static factors are regarded for balancing, e.g., the weighted number of road links or the size of spatial area. In the trajectory streams processing context, trajectory streams exactly represent the computation workload. For example, more vehicles are in cities than rural areas. So it is more reasonable to balance the workload according to vehicle data distribution than to cut the city map into sub areas with even geo-spatial size. So in addition to the mobility localization, trajectory-load-based balance is another measurement for partition performance. For the balance constraint, a ratio of imbalance R (Karypis, 1999) is defined as:

$$R = \max\left\{ W_{sum,i} / (W_{sum} / k) \right\} \quad for\ 1 \le i \le k \quad (2.3)$$

k is the total number of partitions, and $W_{sum,i}$ denotes the sum of node weight in the i-th resulting partition, i.e., weight of P_i . For trajectory-load-based balance, the node weight is specifically modeled according to the distribution of trajectories related with the node. W_{sum} denotes the sum of node weight of the whole graph. So (W_{sum}/k) is the average partition weight, which is constant when the graph and the partition number k are given. A balance constraint can thus be defined as:

$$R \le 1 + \varepsilon \quad (2.4)$$

Where $\varepsilon \ge 0$ is a user-defined parameter indicating the tolerance of imbalance. And $\varepsilon = 1$ implies the weight of any partition shall not exceed twice of the average partition weight. Users can choose a proper ε according to application need to control the imbalance level.

Beside the mobility localization index and ratio of imbalance, the Messaging throughput (T_m) is another important performance measurement. It is defined as the number of geo-messages dispatched to computation nodes per unit time.

$$T_m = n_m / t \quad (2.5)$$

Messaging throughput measures the online geo-message dispatching efficiency and evaluates when the dispatching service may get to be the bottleneck. It is a preferable measurement on general platform performance than the end to end throughput which is application specific. The end to end throughput depends on the computation node configuration and application specific computation complexity, which is beyond the scope of this paper.

3. TRAJECTORY PRESERVING PARTITIONING METHOD AND ALGORITHMS

For achieving mobility localization, the spatial connectivity implied by trajectory data has to be leveraged in geo-partitioning. While some spatial grid based approach and geometric partitioning methods, e.g., (Simon, 1991), (Farhat, 1993), are available for geo-partitioning, the rigid grid shape and the limit of spatial distance metric can significantly impact the partitioning result. A recent enhancement to spatial partitioning is road network distance based partitioning (Ventresque, 2012). The road network-based partitioning can more correctly reflect the localized relationship in many applications. For example, two persons who are 200 meters away spatially actually need more than 1 kilometer to meet each other in terms of road network distance. However the focus of (Ventresque, 2012) is on static network based partitioning, without taking data continuity into consideration. And the connection between road network partitioning and geo-partitioning is not addressed.

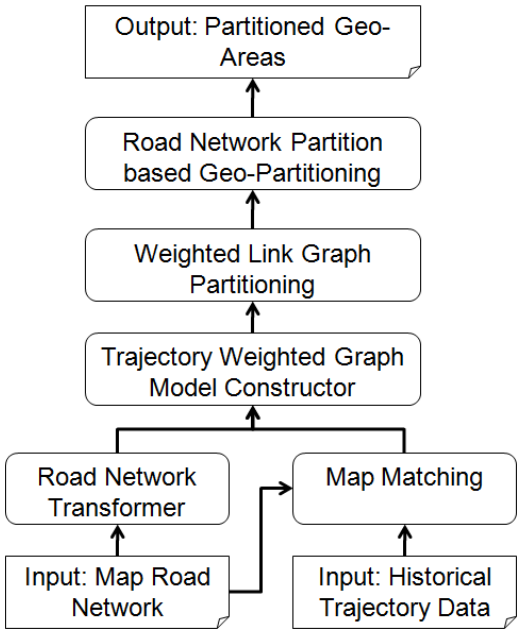


Figure 3. Component Diagram of Trajectory Preserving Partitioning.

In this section a trajectory preserving partitioning method is proposed and corresponding algorithms are developed in following sub-sections. The general idea of trajectory preserving partitioning method is to overlay the trajectory information on top of road network. And then a graph model can be built and graph partitioning method can be applied. After that, the road network partitioning result needs to be projected back into geo-partitions (for the sake that geo-partitions enable much faster dispatching than road network partitions since it's costly to compute the road link of each data record at the point of message dispatching). Figure 3 shows the component diagram of the trajectory preserving partitioning method. The map road network and historical trajectory data are the input of the method. And the partitioned geo-areas are the output. The partitions here imply that the union of the geo-areas covers the whole geo-space and there is no overlap between any pair of geo-areas. As the first step, the Road Network Transformer transforms the map road network from node graph into link graph. Section 3.1 will discuss the transformation process. Map Matching is a well-studied algorithm to associate trajectories with road networks. It takes trajectory data as input and output the sequence of road links that the trajectory traverses. Interested audiences can refer to (Lou, 2009), (Newson, 2009) for Map Matching technology. As a front step before graph partition, the Constructor builds a trajectory weighted graph model according to the transformed road network and trajectory data matched on road network, which will be depicted in section 3.2. Then graph partitioning can be performed. Graph partitioning is a classic mathematical technique with abundant applications. In the domain of traffic network analysis, graph partitioning methods are applied in partitioning large-scale road to speedup shortest path search (Delling, 2009), (Delling, 2011) and distributed transportation simulation (Xu, 2012). In this paper, the METIS toolkit (Karypis, 1999) is adopted for partitioning the weighted road network. METIS takes a heuristic partitioning approach and has been widely referred in research and practice for its proven performance. The trajectory preserving road network partitioning algorithm is described in section 3.3. After that an algorithm is developed to project from road network partitions back to geo-partitions, which will be introduced in section 3.4.

3.1 Road Network Transformation.

In most existing road network partitioning methods, the link cut scheme is applied to partition road network by splitting road links connecting different part of sub road networks. It take a straightforward mapping between road network model and graph model, i.e., each node of road network (e.g., road junction or endpoints) is modeled as a vertex of a graph, and each link(connecting two nodes on road network) is modeled as a graph edge connecting two corresponding vertexes. Thus a graph partitioning problem is defined by minimizing the number of road links cut during the partitioning. At the same time, balance constraint can be imposed on the optimization procedure so that the resulting sub-graphs are balanced in terms of the sum of node weight.

For the trajectory preserving partitioning, it turns out to be a different situation. Cutting on road links leaves the issue of which partitions the cut road links shall join. Since road links have their own geo-shape (could be as long as several kilometers), different joining choices can significantly impact the boundary of geo-areas and could lead to a non-optimum solution. On the contrast it won't be

an issue if partitioning is performed on the nodes of road network: As in road network, nodes are physically modeled as points with no geo-shape. So no matter which partition a cut node joins, it makes no difference to the geo-area boundary. To enable the node cutting, the road network needs to be transformed into a "link graph". A link graph takes each road link on the road network as its vertex (i.e., link vertex) and builds the edges connecting vertexes based on the connectivity between links. In this way, the partitioning on nodes of road network is enabled.

3.2 Trajectory Weighted Graph Construction.

To build the edges connecting vertexes, and more importantly, to define the weights on the edges, both the road network connectivity and the trajectory information needs to be considered. For each adjacent pair of links, the weight on the edge connecting corresponding link vertexes is increased with a certain value w_l [1]. And for each trajectory traversing a sequence of links, each adjacent pair of links in the trajectory will have a certain value w_t [2] contributing to the corresponding edge of the graph. The ratio between w_l and w_t, instead of their absolute value, is important for balancing between the static connectivity and the dynamic connectivity. The static connectivity is more stable with good coverage, while the trajectory is more live through may not cover the full spectrum of network. Merely weighting on static connectivity would lead to non-optimum situation as illustrated in Figure 1. While, in another extreme, purely trajectory based weighting may lead to disconnected graph. As trajectory usually provides many more instances of connectivity than road network (as a pair of links could be traversed by thousands of trajectories), it needs a factor to balance the relative importance of two weights. The principle for weight balance used in this paper is to ensure equal contribution of total weights. For presetting weights w_l and w_t, a tuning factor, r_w, is calculated reflecting the ratio of total static weight to total dynamic weight.

$$r_w = (\sum_{\substack{link \\ pairs}} w_l) / (\sum_{\substack{traversed \\ pairs}} w_t) \quad (3.1)$$

Then the w_t can be tuned by multiplying r_w:

$$w_t^{'} = w_t * r_w \quad (3.2)$$

Static and dynamic weight can be integrated automatically by this approach. One could also adjust r_w manually if prior knowledge is available on application scenario or data quality.

Trajectory data also serves as more precise workload information than static network or geo-spatial elements. As a direct model of workload, each link vertex in the link graph has the weight that equals to the number of trajectories traversing that link.

3.3 Trajectory Preserving Road Network Partitioning Algorithm.

The trajectory preserving partitioning algorithm is presented in Figure 43. It takes road network and trajectory data as input together with two parameters: k is the expected number of partitions and ubR is the upper boundary of imbalance ratio R. It outputs the partitioned sub road networks, each of which contains a list of road links. Step 1- 4 of the algorithm build the link graph and estimate the static edge weight $e.w_s$, the dynamic edge weight $e.w_d$, and the node workload weight $v.w_{wl}$. Step 5 calculates the weight tuning factor r_w. And step 6 integrates the static weight with dynamic weight automatically according to r_w. Step 7 invokes graph partitioning algorithm and gets the partitioned vertex sets. Then in step 8 the partitioned vertex sets are mapped back into sub road networks denoting as lists of road links, which are returned as algorithm output in step 9.

Trajectory Preserving Road Network Partitioning Algorithm

Input:

- $RN(N,L)$: Road network consists of node set N and link set L.
- $Trj = \{t_i\}$: Trajectory data set. ($where\ t_i = (l_x,...l_y)\ \ l_x, l_y \in L$)
- k : Number of partitions to be generated.
- ubR : Upper boundary of imbalance ratio R.

Output:

- $\{subRN_i\}$ **:** each $subRN_i$ is denoted by $L_i = \{l_s | l_s \in L \wedge l_s \in subRN_i\}$ and $\bigcup_1^{nP} L_j = L \wedge \forall i \neq j\ L_i \cap L_j = \varnothing$ holds.

BEGIN

$w_l = w_t = 1$, without loss of generality.

1. Create an empty link graph $G(V,E)$, $V, E = \varnothing$.
2. For each $l_i \in L$
   a) Create a link vertex v_i and add into vertex set $V \leftarrow V \cup v_i$
   b) Initialize the workload weight of v_i : $v_i.w_{wl} \leftarrow 0$
3. For each $n_i \in N$
   a) For each pair of links (l_i, l_j) connected at n_i

[1] w_l can be varying on different pairs of links according to certain extra information, e.g., road level, or intersection level. Here we ignore the detail without loss of generality.

[2] w_t can also be varying, e.g., according to trajectory length.

[3] For clarity of presentation, the multi-link (multiple links between two nodes) handling is omitted.

i. Create an edge e_{ij} and add into edge set $E \leftarrow E \cup e_{ij}$

ii. Set the static weight of e_{ij} : $e_{ij}.w_s \leftarrow w_l$

iii. Initialize the dynamic weight of e_{ij} : $e_{ij}.w_d \leftarrow 0$

4. For each $t_j \in \{Trj\}$

a) Parse each adjacent pair of links (l_x, l_y) of t_j

i. Update dynamic weight of e_{xy} : $e_{xy}.w_d \leftarrow e_{xy}.w_d + w_t$.

ii. Update workload weight of corresponding vertexes $v_i.w_{wl} \leftarrow v_i.w_{wl} + 1$, $i = x, y$

5. $r_w = (\sum_{i,j} e_{ij}.w_s) / (\sum_{i,j} e_{ij}.w_d)$

6. For each $e_{ij} \in E$

a) $e_{ij}.w \leftarrow e_{ij}.w_s + r_w * e_{ij}.w_d$

7. $\{sV_j\} = Metis\_GraphPartition(G, k, ubR)$

8. For each sV_j

a) Create a $L_j \leftarrow \varnothing$

b) For each $v_i \in sV_j$

i. Add corresponding link to L_j : $L_j \leftarrow L_j \cup l_i$

9. Return $\{L_j\}$, $j = 1 \cdots k$

END

Figure 4. Trajectory Preserving Road Network Partitioning Algorithm.

The computational complexity of the algorithm depends on the step 4 and step 7. Step 4 has the time cost $O(n_t)$ (i.e., linear to the number of trajectories in training set n_t). And Step 7 has the approximate complexity of $O(n + m + klog(k))$ according to the author of METIS (Karypis, 1998), where n is the number of nodes, m is the number of edges, and k is the number of expected partitions. So the algorithm's computational complexity is approximately $O(n_t + n + m + klog(k))$.

3.4 Margin Maximized Geo-Partitioning Algorithm.

The trajectory preserving road network partitioning algorithm introduced in section 3.3 generates the road network partitions as output. Each road network partition contains a list of road links. For geo-messages in trajectory streams, the location information is generally presented in terms of geo-spatial coordinates (e.g., longitude and latitude). Since associating coordinates with road links relies on Map Matching which is computationally costly, road network partition is not an ideal structure for online geo-message dispatching. In contrast to a road network partition defined as a list of road links, a geo-partition is defined in terms of a geo-area having a polygon as its boundary. Computationally matching coordinates with polygons is much more effective than associating coordinates with road links. Hence there is a need to transform from road network partitions into geo-spatial partitions for effective geo-message dispatching.

Developing one-one mapping between road network partitions and geo-partitions (represented by a polygon as the boundary) has the following requirements. Basically each polygon shall include all road links allocated to the corresponding road network partition and exclude any road links of other partitions. And there shall be no overlapping or missing coverage between the geo-partitions. Moreover, the position of boundary shall maximize its distance from the data of adjacent partitions as much as possible, so as to be resilient to data noise (if the boundary is close to data, noisy data could float into the other partition and cause wrong dispatching). This raises the need to maximize the margin between polygon boundary and road links near polygon boundary.

To fulfill the above requirements, the Margin Maximized Geo-Partitioning Algorithm is developed and shown in Figure 5. The algorithm determines a compact hull for each road network partition $subRN_j$ and extends those compact hulls into a geo-partition solution in three steps. Firstly, based on the detected compact hulls, the algorithm discovers buffer zones. Then for each buffer zone, it generates separating polylines maximizing the margin in buffer zone. After that the compact hulls are extended into geo-partitions by replacing some lines of compact hull with separating polylines (and corresponding nodes as well). The algorithm assumes that a road network can be modeled as a planar graph, which is valid for most road networks.

Margin Maximized Geo-Partitioning Algorithm

Input:

- $\{subRN_i\}$ **:** each $subRN_i$ is denoted by a list of road links $L_i = \{l_s | l_s \in L \wedge l_s \in subRN_i\}$.
- $\{pn_j\}$: set of partitioned nodes across *subRN* s.

Output:

- $\{GeoArea_j\}$ **:** each *GeoArea* is represented by a polygon as its boundary.

BEGIN

1. Generate a compact convex hull $H_{all} = (Hn, Hl)$ for the whole road network $\bigcup subRN_j$. *Hn* and *Hl* are the boundary nodes and boundary lines respectively.
2. Clockwise traverse the boundary polygon and mark the direction of each line on the boundary the same as traversing direction.
3. Generate a compact hull $H_j = (Hsn_j, Hsl_j)$ for each sub road network $\{subRN_j\}$. Hsn_j and Hsl_j are the boundary nodes and boundary lines of $subRN_j$ respectively.

4. Clockwise traverse the boundary polygon of each sub road network and mark the direction of each line on the boundary the same as traversing direction.
5. Initialize buffer zone set $\{Zb_i\} \leftarrow \varnothing$
6. For each node in the partitioning node set $n \in \{pn_j\}$
   a) Discover buffer zones z_j starting from n by searching in all hull elements $H_{all} \bigcup H_j$.
   b) $\{Zb_i\} \leftarrow \{Zb_i\} \cup z_j$
7. Initialize separating polyline set $\{Ps\} \leftarrow \varnothing$
8. For each $z_j \leftarrow \{Zb_i\}$
   a) Find the separating polylines $\{p_k\}$ to maximize the margin in the buffer zone z_j .
   b) $\{Ps\} \leftarrow \{Ps\} \bigcup \{p_k\}$
9. For each $subRN_j$
   a) Extend $subRN_j$'s hull $H_j = (Hsn_j, Hsl_j)$ to margin maximized boundary $B_j = (Bsn_j, Bsl_j)$ by replacing some of its boundary lines with those separating polylines $p_i \in \{Ps\}$ which have both end nodes belong to Hsn_j . Maximized margin geo-partition for $subRN_j$ is bounded by B_j .
10. Return margin maximized boundary set $\{B_j\}$

END

Figure 5. Margin Maximized Geo-Partitioning Algorithm.

To determine the compact hull (i.e., spatial outer borders) of a road network can be realized by constructing a minimal hull to encompass all nodes of the road network. Since each edge of the road network stands for a road segment modeled by a straight line, the resulting hull encompasses all the network edges as well. Algorithms are available for the hull generation (e.g., (DeBerg, 2000)). By representing each edge using its vertex points, hull generation algorithm can output a polygon as the boundary of a list of network edges. And each vertex of the polygon is a boundary node of the road network and each edge stands for an outer border. To integrate the border information into the road network model, each edge of the polygon (i.e., line) is added to the road network as a new edge, if there is no overlap with existing network edge. After compact hulls are identified, all nodes and lines which are not in the compact hulls $H_{all} \bigcup H_j$ are regarded as internal elements and can be ignored in the following operations. And it's obvious that all partitioned nodes are kept in compact hulls' node set $Hn \bigcup Hsn_j$.

A buffer zone can be specified by its border (i.e., a polygon composed by lines in compact hulls $Hl \bigcup Hsl_j$ and with no road network node in it). A sample is shown in Figure 6. The discovery of each buffer zone starts from a boundary node between two sub road networks (i.e., partitioned node), e.g., the boundary node V. And then it traverses on $Hl \bigcup Hsl_j$ following the link direction generated in step 2 and step 4 of Figure 5 (i.e., clockwise direction of each individual polygon). When getting into another boundary node share by two sub network (e.g., boundary node U in Figure 6), it switches to the boundary of another hull and continues the traversing. The process stops when the starting boundary node is revisited. Then the traversed lines form a polygon as the border of a buffer zone.

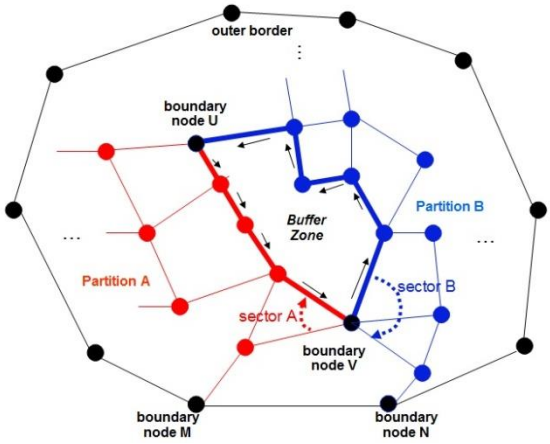


Figure 6. Sample of Discovering Buffer Zone.

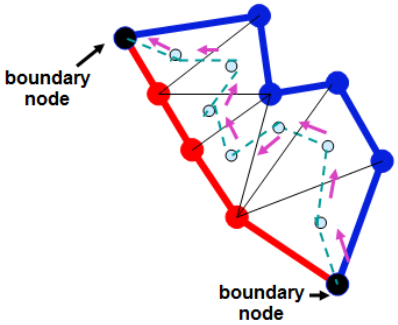


Figure 7. Sample of Finding Separating Polylines Maximizing Margin in Buffer Zone.

For finding the separating polylines to maximize the margin in the buffer zone, a triangulated approach is implemented. The buffer zone is triangulated first and the division lines are obtained by chaining the cendoids of triangle and connecting further to the boundary nodes. The resulting division polylines become the border lines of final geo-spatial partitions. A sample is depicted in Figure 7 which finds the separating polyline in a buffer zone involving two sub road network boundaries. When a buffer zone border involves multiple sub road networks, similar approach can be applied and multiple polylines will be output. For the interest of paper length, the detail is the omitted here. The Margin Maximized Geo-Partitioning Algorithm has the complexity

of $O\left(k*n*log\left(m\right)+n_p*n_h*log\left(m_h\right)\right)$, where n is number of nodes, m is the number of edges, k is the number of partitions, n_p is the size of $\{pn_j\}$, n_h is the size of $Hn\bigcup Hsn_j$ and m_h is the size of $Hl\bigcup Hsl_j$.

With the method proposed in this section, optimized geo-partitions can be generated from road network partitions. On top of the geo-partitions, high performance geo-message dispatching can be enabled, which will be introduced in the next section.

4. HIGH PERFORMANCE GEO-MESSAGE DISPATCHING

For dispatching geo-messages, the dispatching service needs to compare the location information in each geo-message with boundary of geo-areas to identify the right geo-area the message located. Then the message will be routed to the computation node responding for the workload of that geo-area. Comparing to key based or attribute-value based dispatching in ordinary data streams, geo-message dispatching is still heavier in terms of computation: calculating the geo-spatial relationship is in general more complex than value matching. So the dispatching service is prone to scalability bottleneck.

In this section, based on the Geohash technology (Fox, 2013), a Geohash Tree scheme is proposed to codify the dispatching rules for high performance geo-message dispatching. The Geohash Tree gains the performance advantage by transforming geo-spatial matching into less expensive Geohash code matching. And moreover, Geohash Tree takes a hierarchical and flexible layered structure comparing with grid approach: when a node has its bound within a single partitioned geo-area, the node is marked as leaf node and needs not to be split further. Therefore both the storage space can be saved and levels of comparison can be reduced. Geohash Tree has the similar idea of spatial index tree such as R^+-tree (Sellis, 1987). While a major difference in a Geohash Tree is that each node has its Geohash code by inheriting the Geohash code from its parent node and extending with one character. The character uniquely differentiates the node from other nodes with the same parent and maps to a specific sub-area of the parent's area. The number of children that a node can have equals to the cardinality of the Geohash character set. In practice, a Geohash Tree with its character set cardinality of 32 can reach to meter level resolution on its node in the 10^{th} level of depth. On the one hand, Geohash tree shares the same advantage as R+-tree on effective indexing and space saving comparing to grid index scheme (by enabling leaf node on high level of the tree and avoiding massive low level pieces). On the other hand, Geohash tree replaces R+-tree's geo-spatial matching with character matching, which is computationally much more effective.

Figure 8 gives the algorithm pseudo-code for generating a Geohash Tree. And Figure 9(b) illustrates a sample instance of Geohash Tree and Figure 9(a) visualizes the relationship between Geohash codes and spatial areas.

Dispatching Rule Generation Algorithm

Input:

- $\{B_j\}$**:** Boundary polygons of geo-partitions.

Output:

- *GHTree* **:** Geohash Tree. Each node on the *GHTree* has the attributes of $\left(code,isleaf,\{geo\_area\},bound,\{child\}\right)$. *code* is the Geohash code of the tree node; *isleaf* is true if the node is leaf node (with no children nodes); $\{geo\_area\}$ is the areas the node maps to; *bound* is the geo-spatial boundary of the node in terms of up-left limit point and down-right limit point; $\{child\}$ contains the children nodes if it's not a leaf node.

BEGIN

1. Initial a node queue: $q_{node}\leftarrow\varnothing$
2. Calculate *bound* of the whole area and generate the Geohash code *code* to cover the bound.
3. Create a root node: $root\leftarrow\left(code,FALSE,\{ALL\_GEO\_AREAS\},bound,NULL\right)$
4. $GHTree.root\leftarrow root$
5. Push the root node into the node queue: $q_{node}.push(root)$
6. While (q_{node} is not empty)

a) Poll a node from queue: $nd\leftarrow q_{node}.poll()$

b) If ($\|nd.\{geo\_area\}\|_0\le 1$)

i. $nd.isleaf\leftarrow TRUE$

ii. $nd.\{child\}\leftarrow NULL$

c) Else

i. $nd.\{child\}\leftarrow\varnothing$

ii. For each option char *cr* of Geohash code

1. Compute the corresponding sub boundary b_s in $nd.bound$.
2. Find the overlapped areas ga' between b_s and $nd.\{geo\_area\}$.
3. If ($\|ga'\|_0==0$) // out of scope.
   a) Continue;
4. Else if ($\|ga'\|_0==1$)
   a) New a leaf node: $nd_s\leftarrow\left(code+cr,TRUE,ga',b_s,NULL\right)$
   b) Add nd_s to *nd* 's children list.
5. Else // may need to break down
   a) If reaches max tree level
      i. $nd.isleaf\leftarrow TRUE$
      ii. $nd.\{child\}\leftarrow NULL$
   b) Else

i. New a node: $nd_s \leftarrow (code+cr, FALSE, ga', b_s, NULL)$

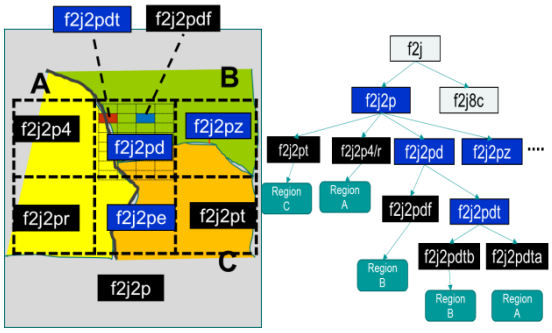
ii. Add nd_s to nd 's children list.

iii. Add to queue: $q_{node}.push(nd_s)$

7. Return *GHTree* .

END

Figure 8. Dispatching Rule Generation Algorithm.



(a) (b)

Figure 9. Geohash Tree.

The Dispatching Rule Generation Algorithm adopts a breadth-first approach to explore and build the tree. It initializes a root node and sets its Geohash code covering the whole geo-area of interest. Then the root node is pushed into queue as the starting seed. Step 6 is the main body of the algorithm. In the loop of step 6, nodes are polled out of the queue in the same sequence as they enter the queue. By evaluating the relationship between the polled node's boundary and the geo-areas, actions are taken on the node to either mark the node as leaf or explore the children of the node further. In the action of further exploration of children nodes (i.e., step 6.c) of Figure 8), a child node could be either added to tree as leaf node (if it maps to a single geo-area), or discarded (if it maps to no geo-area of interest), or pushed into queue for further exploration (if it maps to multiple geo-areas). In the example in Figure 9, we explore the space from very beginning to 'f2j' and then explore its grandchildren 'f2j2p', 'f2j8c', etc. The 'f2j2p' node needs to be explored further since it covers 3 geo-areas (as Figure 9(a) shows). So 'f2j2p' is further split into nodes 'f2j2p4', 'f2j2pd', 'f2j2pz', 'f2j2pr', 'f2j2pe', 'f2j2pt', etc.. Since each node of 'f2j2pt', 'f2j2p4' and 'f2j2pt' covers only one partition area, they are marked as leaf node and will not be further drilled down. On the contrast, 'f2j2pd', 'f2j2ppe' and 'f2j2pz' has to be explored further as each of them covers multiple geo-areas. Ideally each leaf node on the resulting Geohash Tree shall map to exactly one geo-area, if the depth of tree is unrestricted. While in practice, the depth of tree stops at a level (mostly between 6 and 11) for storage and precision consideration. So there would be a small portion of leaf nodes across multiple geo-areas when these leaf nodes are over the partition boundaries. Only in those cases, geo-spatial calculation is needed for determine the right geo-area for a geo-message. The Dispatching Rule Generation Algorithm is relatively time consuming. In worst case it takes $c^{d+1} * k$ times of geo-relationship calculation if all leaf nodes are at the most depth of the tree and there are no pre-ending branches in the intermediate levels of the tree. Here c is the cardinality of code, d is the maximum levels of tree, and k is the number of partitions. Fortunately that will not happen as Geohash Tree can always function and cut branches radically at early stage. In practice, it takes less than a minute to generate a Geohash Tree for a 100,000 km^2 and 6 partitions, which is affordable for offline processing.

Figure 10 gives the detail of Geo-message dispatching algorithm which takes the Geohash Tree as its dispatching rule.

Geo-Message Dispatching Algorithm

Input:

- m : a geo-message which is a tuple with the following attributes $(id, lon, lat, ts, message\_body\cdots)$. *id* is the moving object's id; *lon* and *lat* are the location information in longitude and latitude; *ts* is the timestamp of the message; *message_body* has information that needs to be dispatched to computation node.

Output:

- *GeoArea id* : the *GeoArea* id (eventually, the responsible computation node id) that the geo-message shall be dispatched to.

BEGIN

1. Get Geohash Tree Root Node $nd \leftarrow GHTree.root$.
2. Generate Geohash code for m : $m.code \leftarrow geohash.gen(m.lon, m.lat)$
3. If m is within the area of interest: $has\_prefix(m.code, nd.code)$.
   a) While (not $nd.isleaf$)
      i. Find nd 's child node nd_s that $has\_prefix(m.code, nd_s.code)$
      ii. Get down a level: $nd \leftarrow nd_s$
   b) If ($\|nd.\{geo\_area\}\|_0 == 1$)
      i. Return $nd.geo\_area.id$
   c) Else
      i. Do geo-spatial comparison between m 's location with the geo-areas nd covers.
      ii. Return the matched geo-area id.
4. Else // m is out of the scope.
   a) Return Out-Of-Map-Scope error.

END

Figure 10. Geo-message Dispatching Algorithm.

The computational complexity of Geo-Message Dispatching Algorithm is linear to the number of geo-messages. In most cases, it only takes a Geohash code

generation operation and several operators on tree node access for dispatching a geo-message, which is very efficient.

5. EXPERIMENT

In this section, the performance of proposed methods and accompany algorithms are evaluated by experiments using both real world trajectory data collected from GPS equipped vehicles and simulated trajectory data streams. The performance measurements are the Mobility Localization Index (I_{ml}), the ratio of imbalance (R), and the Messaging Throughput (T_m) which are introduced in section 2. A set of experiments are conducted on partition performance for validating the soundness of trajectory preserving partition method. The experiment context and result will be introduced in section 5.1. Another set of experiments are perform for real-time messaging dispatching performance, which will be introduced in section 5.2.

Table 1. Experiment Real-World Datasets

Dataset	CityWideU[a]	UrbanWideB[b]
Road network size (number of links)	62,810	31,434
Geo-message volume (number of records)	80,297,139	57,102,570
Trajectory data volume (number of trajectories)	416,840	315,636
Average Length of Trajectory (number of records) [c]	192.6	180.9

Dataset characteristics:

[a] *CityWideU is on a larger map with unbalanced trajectory distribution (trajectories on certain areas of the road network are much dense than those on other areas).*

[b] *UrbanWideB is on a small map with near balanced trajectory distribution.*

[c] *Mobile Data was sampled every 25 seconds/record in average.*

Two real-world datasets were collected and used for experiments, as shown in Table 1. The trajectory data was generated from 3000 GPS equipped vehicles travelling in a large city in 7 days. Each raw trajectory is a sequence of GPS data records (and each record contains longitude, latitude, velocity, timestamp, and vehicle ID). After the preprocessing of associating GPS with road network, a trajectory of a vehicle can be expressed as a sequence of road links that the vehicle traverses. The first dataset, CityWideU, is a relatively large dataset which contains imbalance workload: central urban traffic is much heavier than that of sub-urban areas. From this perspective, the trajectory density is a good reflection of the urban roads popularity, but is biased when the suburban roads are concerned. The second dataset, UrbanWideB, has a smaller scale and a more balanced trajectory distribution. For evaluating geo-message dispatching throughput, simulated data was generated beside the real-world datasets. The detail will be introduced in section 5.2.

The experiments were conducted on a cloud environment using SoftLayer machines. The server for geo-spatial partition and message dispatching is a dedicated server. Its configuration is 4-CPU x86 Server with 16 G memory and Linux OS. The computation nodes are virtual machines with 4-CPU x86 Server with 8 G memory running map matching tasks. As formerly noted, since this paper focuses mainly on the message dispatching throughput instead of application dependent workload throughput, we ensure there are sufficient computation nodes and they would not be a source of bottleneck in the experiments.

5.1 Experiments on Partition Performance.

In the sub section, the Mobility Localization Index (I_{ml}) and the ratio of imbalance (R) are evaluated for the partition performance using datasets described in Table. 1. We denote the proposed trajectory preserving partition method and algorithms as TPP. And the road network-based partitioning method (Ventresque, 2012), denoting as RNP, is used as the baseline scheme for performance comparison.

Since the geo-partitioning is a learning process, the experiments follow the cross validation scheme to avoid over-fitting. Specifically ten-fold cross validation is applied and the final result is consolidated from the ten running output.

The number of expected partitions is the major parameter of the partition algorithms. In the experiments, different numbers of partitions (4, 6, 8, 10, 12 partitions respectively) are tested on both datasets. In the following the experiment result of both datasets will be presented.

Experiment Result on CityWideU. Table 2 summaries the overall partitioning result of both TPP and RNP on the CityWideU dataset with different setting of partition numbers. And the overall ratio of improvement is presented in the last column. In Table 2, the average number of cross-partition trajectories is significantly reduced from 70,200 to 27,496 which contributes to the remarkable improvement of Mobility Localization Index (12.32% taking the I_{ml} of RNP as the base). Also TPP provides more balanced result and reduces the Ratio of Imbalance (R) by 67.11% comparing with RNP.

Figure 11 displays the detail partitioning result on the number of cross-partition trajectories with various setting of partition numbers. And Figure 12 is the detail result on the Mobility Localization Index (I_{ml}). TPP gets higher I_{ml} than RNP in every tested setting of partition numbers. And it stably keeps the I_{ml} beyond 90% while RNP drops down to less than 80% when the partition number gets larger.

Table 2. Summary Result of Experiment on CityWideU

Measurements	TPP	RNP	Ratio of Improvement
Number of Cross-Partition Trajectories	27,496	70,200	60.83%
Mobility Localization Index (I_{ml})	93.40 %	83.16 %	12.32%
Average Ratio of Imbalance(R)	1.159	3.524	67.11%

The ratio of imbalance detail of CityWideU is shown in Figure 13. Considering the CityWideU dataset is characterized as the geospatially imbalanced distribution of trajectory workload, the ratio of imbalance would have challenge to control. The result of RNP algorithm with different numbers of partitions verified this. The ratio of imbalance always approaches or exceeds 3.0, indicating the workload of some partition is 3 times the average workload of all the partitions. While the TPP remarkably keeps the ratio of imbalance in the range of [1.03, 1.25], which means a nearly even workload distribution among partitions (i.e., the partition with the heaviest workload is less than 25% higher to the average workload).

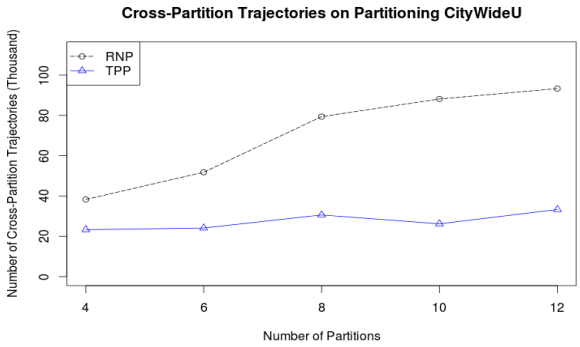


Figure 11. Cross-Partition Trajectories on Partitioning CityWideU

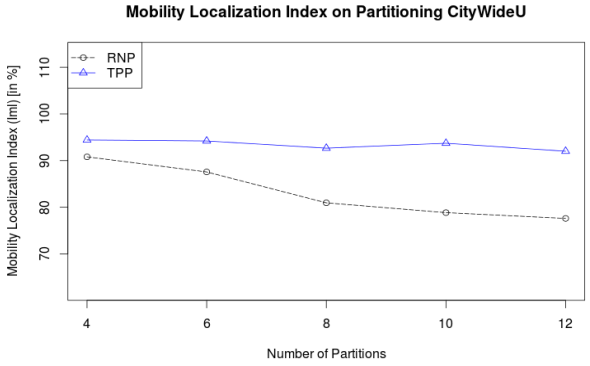


Figure 12. Mobility Localization Index on Partitioning CityWideU.

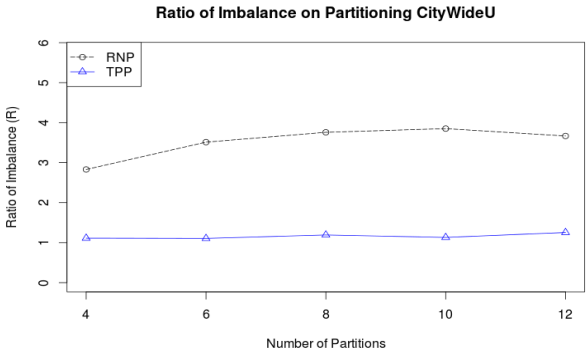


Figure 13. Ratio of Imbalance on Partitioning CityWideU.

The experiment on the CityWideU dataset with different settings of partition numbers shows the stable performance of TPP method, which demonstrates the advantage of mobility localization and balanced partitioning.

Experiment Result on UrbanWideB. Table 3 summarizes the partitioning result on the UrbanWideB dataset. The UrbanWideB dataset is a smaller dataset than CityWideU, and is more connected with dense trajectories. In general it is more challenging to sustain the mobility localization. So it costs more in partitioning. This can be observed by comparing Table 3 with Table 2: both the average number of cut trajectories and the average mobility localization index of Table 3 are lower than those of Table 2. On the UrbanWideB dataset, TPP gets in average 19.74% of improvement on the I_{ml} gets remarkably 44.54% improvement on the Ratio of Imbalance comparing to RNP. The details of the cross-partition trajectories and mobility localization comparison result on the UrbanWideB dataset are shown in Figure 14 and Figure 15 respectively. The I_{ml} of partition result by TPP are mostly more than 90% (except the case that partition number is 12). And in each setting of partition number, TPP gets more than 10% higher on I_{ml} than that of RNP.

Table 3. Summary Result of Experiment on UrbanWideB

Measurements	TPP	RNP	Ratio of Improvement
Number of Cross-Partition Trajectories	30,001	93,765	68.00%
Mobility Localization Index (I_{ml})	92.80%	77.51%	19.74%
Average Ratio of Imbalance(R)	1.195	2.154	44.54%

Figure 16 is the detail result of Ratio of Imbalance. RNP improves its performance since the UrbanWideB dataset is much more balanced than the CityWideU dataset. But it has still at least 0.8 higher than TPP on each of the cases.

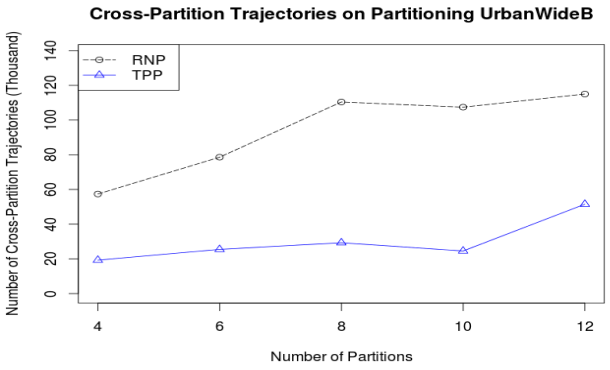


Figure 14. Cross-Partition Trajectories on Partitioning UrbanWideB.

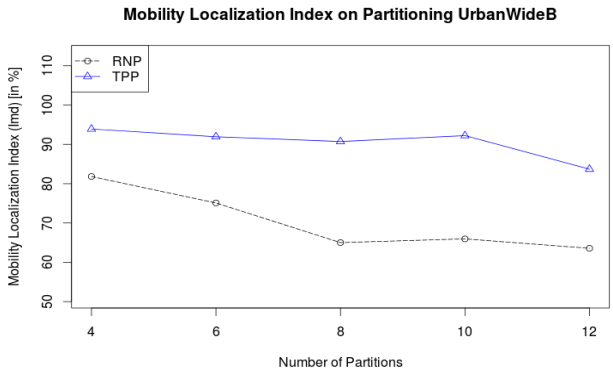


Figure 15. Mobility Localization Index on Partitioning UrbanWideB.

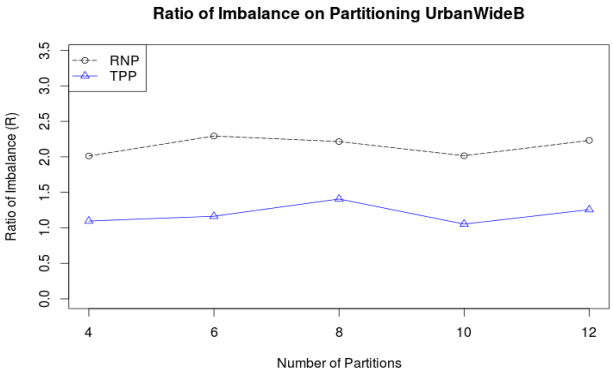


Figure 16. Ratio of Imbalance on Partitioning UrbanWideB.

TPP has its computation time range from 52 seconds to 5 minutes generating an experiment result. RNP requires much less time for partitioning. It generates result within 30 seconds in any of the experiment cases. Since the partition is a learning process that can be done offline, the time is not a big concern. When there is a need to perform partitioning in online mode to reflect the trajectory pattern changes timely, the computation time will be an important measurement.

5.2 Experiments on Dispatching Performance.

Messaging throughput is evaluated in this sub section. A grid based dispatching method is implemented for comparison. Grid index is a single layer structure. It's very effective to locate a spatial point to a cell of the grid. The size of the grid cell is the major parameter that could affect indexing performance. Here we choose two grid sizes: Grid_DA50 is the grid dispatching algorithm with 50 meter cell configuration. And Grid_DA200 is the grid dispatching algorithm with 200 meter cell configuration. GHT_DA represents the Geohash tree based dispatching algorithm we proposed in section 4 (i.e., the Dispatching Rule Generation Algorithm and the Geo-Message Dispatching algorithm).

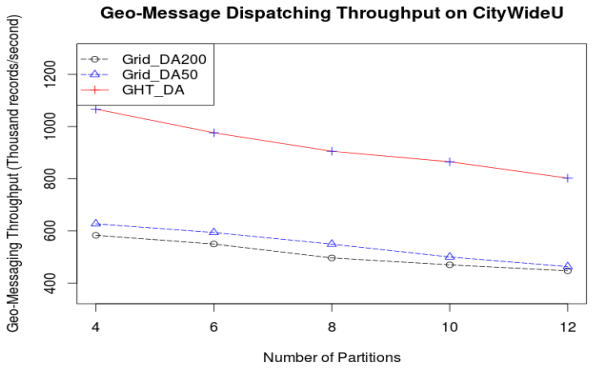


Figure 17. Geo-Message Dispatching Throughput on CityWideU.

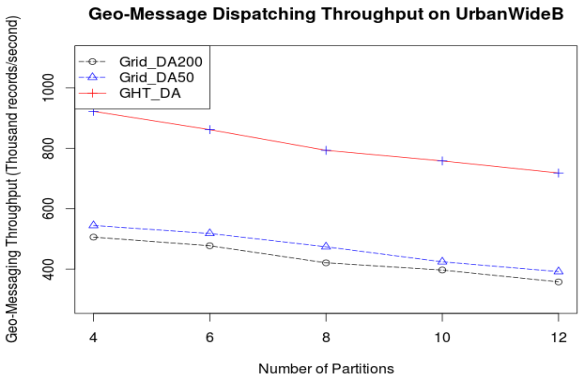


Figure 18. Geo-Message Dispatching Throughput on UrbanWideB.

Figure 17 and Figure 18 are the messaging throughput result on CityWideU and UrbanWideB respectively. In the case of CityWideU, the throughputs of GHT_DA range from 0.8~1 million records/second in different configurations of partition numbers. The throughputs of Grid_DA50 and Grid_DA200 are in the range of [390k, 630k]. Grid_DA50 gets slightly better result than Grid_DA200. This attributes to the smaller grid size which reduces the probability for calculating geo-shape

relationship. The Figure 18 shows the similar result. In experiments on both datasets, GHT_DA has more than 30% improvement of the message dispatching performance.

For further validating the performance on large-scale map with more choices of partition numbers, a simulation is implemented on a map with 420,633 road segments. 6,955,046,200 geo-messages are generated and sent to server by 8 client machines in UDP protocol. The partition numbers are set to 4, 10, 20, 40, and 60 respectively. The dataset is denoted as Simu6B and the result is shown in Figure 19. The result is consistent with those on CityWideU and UrbanWideB and shows the promising generalization capability to apply the technology to large-scale map.

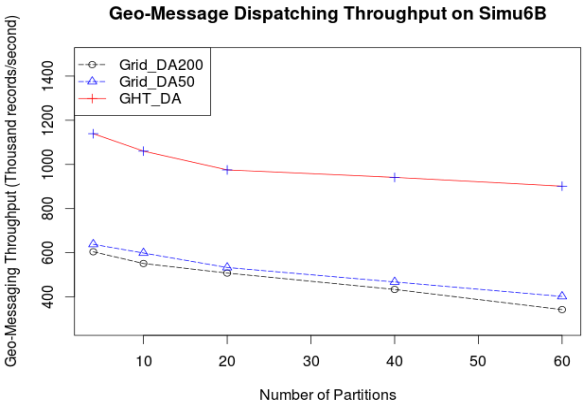


Figure 19. Geo-Message Dispatching Throughput on

6. Related Work

Providing high quality, scalable and real time analytics is one unprecedented challenge for the Internet-of-Things (Aggarwal, 2013). There have been research interests enabling scalable cloud services for IoT applications from resource virtualization, networking and architecture angles (Kovatsch, 2014), (Mukherjee, 2014), (Li, 2013). The research on scalable data stream processing has two main threads. The first thread takes the architectural perspective to enhance the programming model and develop optimization technologies for platform scalability. The second thread focuses on algorithm improvement for high performance distributed data stream analytics.

(Andrade, 2009) defined a streaming architectural pattern for the sensor-and-response application domains. Scalable application can be generated following the architectural pattern and corresponding programming model. (Khandekar, 2009) solved the assignment problem of processing elements (PEs) to processing hosts (PHs) in the context of high scalable distributed stream processing. It employed a graph partitioning method to minimize the inter-PH network communication while simultaneously balancing load across the PEs. In light of stream workload dynamics, (Schneider, 2009) proposed a technique to dynamically adjust the amount of computation of an operator. (Cherniack, 2003) introduced two stream processing systems and discussed the load (re-)partitioning issue in the context of generic workload. (Pietzuch, 2006) considered the dynamic network condition to relocate operator for higher stream processing performance. The existing work regarded mostly generic data streams and did not address the mobile characteristics.

There is also solid research outcome on scalable data streams analytics focusing on algorithm improvement (Gaber, 2005). (Domingos, 2000) proposed a general method, called VFML (Very Fast Machine Learning), for scaling up machine learning algorithms. In more recent years, the research on trajectory streams has it applications on anomaly detection (Bu, 2009), spatio-temporal causal interactions discovering (Liu, 2011), and map generation (Davics, 2006). This category of work forms the advanced analytics workload for scalable streams processing platform.

Beside stream processing platform, graph partitioning technologies have been well studied and applied in parallel processing. Interest audiences can refer (Buluc, 2013) for recent advance of graph partitioning technologies as well as their applications in parallel processing.

7. Conclusions and Future Work

In the IoT era, large-volume mobile data streams are generated from pervasive mobile sensors. This provides opportunities for precise mobility awareness and timely mobility insight analysis. This paper focuses on the scalability challenge of cloud services analyzing trajectory streams. With the proposed trajectory preserving partition method and Geohash Tree based dispatching method, high throughput streams processing cloud services can be achieved with high quality of balanced workload and reduced cross-server communication. The algorithms are developed and validated through experiments.

There are two interesting topics we would like to further pursue. Currently under the assumption that the workload pattern is stable and identical to that of historical data, the paper addresses the offline partitioning problem. While real world workload could dynamically shifts along with time. This raises the further requirement of online (re-)partitioning in face of the dynamics. Online repartitioning needs to take partition adjustment cost into consideration and requires incremental design on algorithm to avoid dramatic fluctuation. Besides that, the computational efficiency of online partitioning algorithm is also of interest. And this paper puts its focus mainly on the geo-message dispatching efficiency. In the future, with the maturity of the mobile data streams analytics workload patterns, end-to-end measurements of scalability performance with typical application workload patterns can be further investigated and validated.

8. REFERENCES

C.C. Aggarwal, N. Ashish, and A. Sheth, (2013). "The Internet of Things: A Survey From the Data-Centric Perspective," Managing and mining sensor data, pp. 383-428, 2013.

H. Andrade, B. Gedik, K. Wu, and P.S. Yu, (2009). "Scale-Up Strategies for Processing High-Rate Data Streams in System S," Proc. IEEE 25th Inter-national Conference on Data Engineering, pp. 1375-1378, 2009.

Y. Bu, L. Chen, A.W. Fu, and D. Liu, (2009) "Efficient Anomaly Monitoring Over Moving Object Trajectory Streams," Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 159-168, 2009.

A. n Buluç H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, (2013). "Recent Advances in Graph Partitioning ," preprint, 2013.

M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S.B. Zdonik, (2003). "Scalable Distributed Stream Processing," Proc. CIDR, pp. 257--268, 2003.

Y. Chen, G. Dong, J. Han, B.W. Wah, and J. Wang, (2002). "Multi-Dimensional Regression Analysis of Time-Series Data Streams," Proc. 28th international conference on Very Large Data Bases, pp. 323-334, 2002.

J.J. Davics, A.R. Beresford, and A. Hopper, (2006). "Scalable, Distributed, Real-Time Map Generation," IEEE Pervasive Computing, vol. 5, no.4, pp. 47-54, 2006.

M. De Berg, M. Van Kreveld, M. Overmars, O. Schwarzkopf, and M.H. Overmars, (2000). "Computational Geometry: Algorithms and Applications," Springer, , 2000.

D. Delling, M. Holzer, K. Müller, F. Schulz, and D. Wagner, (2009). "High-Performance Multi-Level Routing," The Shortest Path Problem: Ninth DI-MACS Implementation Challenge, vol. 74, pp. 73-92, 2009.

D. Delling, A.V. Goldberg, I. Razenshteyn, and R.F. Werneck, (2011). "Graph Partitioning with Natural Cuts," Proc. 2011 IEEE International on Parallel & Distributed Processing Symposium (IPDPS), pp. 1135-1146, 2011.

P. Domingos and G. Hulten, (2000). "Mining High-Speed Data Streams," Proc. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71-80, 2000.

C. Farhat and M. Lesoinne, (1993). "Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics," International Journal for Numerical Methods in Engineering, vol. 36, no.5, pp. 745-764, 1993.

A. Fox, C. Eichelberger, J. Hughes, and S. Lyon, (2013). "Spatio-temporal indexing in non-relational distributed databases," Proc. IEEE International Conference on Big Data, pp. 291--299, 2013.

M.M. Gaber, A. Zaslavsky, and S. Krishnaswamy, (2005). "Mining Data Streams: A Review," ACM Sigmod Record, vol. 34, no.2, pp. 18-26, 2005.

B. Gedik, H. Andrade, K. Wu, P.S. Yu, and M. Doo, (2008). "SPADE: The System S declarative stream processing engine," Proc. ACM SIGMOD International Conference on Management of Data, pp. 1123-1134, 2008.

C.S. Jensen and S. Pakalnis, (2007). "Trax: Real-World Tracking of Moving Objects," Proc. 33rd International Conference on Very Large Databases, pp. 1362-1365, 2007.

G. Karypis and V. Kumar, (1998). "A fast and high quality multilevel scheme for partitioning irregular graphs," vol. 20, no.1, pp. 359-392, 1998.

G. Karypis and V. Kumar, (1999). "Parallel Multilevel Series K-Way Partitioning Scheme for Irregular Graphs," Siam Review, SIAM, vol. 41, no.2, pp. 278-300, 1999.

R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K. Wu, H. Andrade, and B. Gedik, (2009). "COLA: Optimizing Stream Processing Applications via Graph Partitioning," Lecture Notes in Computer Science, vol. 5896, pp. 308-327, 2009.

M. Kovatsch, M. Lanter, and Z. Shelby, (2014). "Californium: Scalable cloud services for the internet of things with coap," Proc. 4th International Conference on the Internet of Things, 2014.

F. Li, V. Michael, M. Claesens, and S. Dustdar, (2013). "Efficient and Scalable IoT Service Delivery on Cloud," Proc. IEEE Sixth International Conference on Cloud Computing (CLOUD), pp. 740--747, 2013.

W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing, (2011). "Discovering Spatio-Temporal Causal Interactions in Traffic Data Streams," Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1010-1018, 2011.

D. Locke, (2010). "Mq telemetry transport (mqtt) v3. 1 protocol specification," http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html, 2010.

Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, (2009). "Map-Matching for Low-Sampling-Rate GPS Trajectories," Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic In-formation Sys-tems, pp. 352-361, 2009.

K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, (2005). "Conceptual Partitioning: an Efficient Method for Continuous Nearest Neighbor Monitoring," Proc. ACM SIGMOD International Conference on Man-agement of Data, pp. 634-645, 2005.

A. Mukherjee, H.S. Paul, S. Dey, and A. Banerjee, (2014). "Angels for Distributed Analytics in IoT," Proc. 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 565-570, 2014.

L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, (2010). "S4: Distributed Stream Computing Platform," Proc. IEEE International Conference on Data Mining Workshops (ICDMW), pp. 170-177, 2010.

P. Newson and J. Krumm, (2009). "Hidden Markov Map Matching through Noise and Sparseness," Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 336-343, 2009.

P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, (2006). "Network-Aware Operator Placement for Stream-Processing Systems," Proc. 22nd IEEE International Conference on Data Engineering, pp. 49--60, 2006.

S. Schneider, H. Andrade, B. Gedik, A. Biem, and K. Wu, (2009). "Elastic Scaling of Data Parallel Operators in Stream Processing," Proc. IEEE International Symposium on Parallel & Distributed Processing, pp. 1-12, 2009.

T. Sellis, N. Roussopoulos, and C. Faloutsos, (1987). "The R1-tree: A Dynamic Index for Multi-Dimensional Objects," Proc. Thirteenth International Conference on Very Large Data Bases, pp. 507–518, 1987.

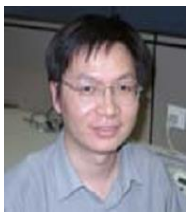
H.D. Simon, (1991). "Partitioning of Unstructured Problems for Parallel Processing," Computing Systems in Engineering, vol. 2, no.2, pp. 135-148, 1991.

A. Ventresque, Q. Bragard, E.S. Liu, D. Nowak, L. Murphy, G. Theodo-ropoulos, and Q. Liu, (2012). "SParTSim: A Space Partitioning Guided by Road Network for Distributed Traffic Simulations," Proc. 16th IEEE/ACM In-ternational Symposium on Distributed Simulation and Real Time Ap-plications, pp. 202-209, 2012.

Y. Xu and G. Tan, (2012). "An Offline Road Network Partitioning Solution in Distributed Transportation Simulation," Proc. IEEE/ACM 16th Interna-tional Symposium on Distributed Simulation and Real Time Applications, pp. 210-217, 2012.

X. Zhang, G. Hu, N. Duan, P. Gao, W. Dong, and J. Zhu, (2014). "Scalable Mobile Data Streaming with Trajectory Preserving Partitioning," Proc. IEEE International Conference on Mobile Services, pp. 16-23, 2014.

AUTHORS



Xin Zhang is with IBM Research – China and Automation Department of Tsinghua University, Beijing, China. He got his master degree in computer science and technology from BeiHang University, and joined IBM Research in 2000. His main research interest includes mobility big data analytics, mining data streams, graph model and graph-based optimization technology. In recent years, he led multiple projects in the domain of asset preventive maintenance analytics, railway transportation schedule optimization, IoT-based intelligent traffic systems, scalable dynamic map analysis, and big data analytics.

Guoqiang Hu received his B.S. Degree from the Shanghai Jiao-Tong University, China, in 1997 and the M.S. Degree in Information Technology from the University of Stuttgart, Germany, in 2002. From 2002 to 2007, he was with the Institute of Communication Networks and Computer Engineering, Stuttgart, Germany as a research staff and Ph.D. candidate. From 2008 to 2010, he worked as a research fellow in the Centre for Quantifiable Quality of Service in Communication Systems, Trondheim, Norway. He joined IBM Research - China in 2010 and researched in the domain of Internet-of-Things and Connected Vehicles. His research interests include distributed system architecture design, performance evaluation and large-scale data mining technologies.

Ning Duan is a research staff member of IBM Research, China. His research mainly focuses on the big moving object data mining and management. He has been in the area of data mining for almost 6 years. And apply big data mining and machine learning technology on connected vehicle area for more than 3 years. He received his Master degree in 2006 from XiDian University of China.

Peng Gao is a staff researcher of IBM Research, China. He received Ph.D. from Tongji University in 2010, and joined IBM subsequently. With a research experience of 10 years in Traffic & Transportation domain, he once served as a trustee of Traffic & Transportation Engineering Society of Tongji University. His research interests include: Complex Science, Intelligent Transportation System, Multi-agent System & Simulation, and Analytics & Optimization. He has over 10 publications and 12 patents.

Weishan Dong is a Research Staff Member from IBM Research, China. He acts as a research team leader working on big data processing, spatiotemporal analytics, and mobile computing. He received his B.E. degree in computer science and technology from the University of Science and Technology of China (USTC) in 2004, and his Ph.D. degree in pattern recognition and intelligent system from the Institute of Automation, Chinese Academy of Sciences in 2009. He joined IBM Research – China in 2009. His current research mainly focuses on data mining, especially mining big spatiotemporal data with addressing large scale and low latency. He is also interested in evolutionary computation and computer vision topics. Dr. Dong has more than 30 refereed publications in international journals and conferences and over 20 patents.

Jun Zhu is a Senior Technical Staff Member at the IBM Research, China in Shanghai and Senior Manager driving the Smarter Mobility research project. He joined IBM after graduation from Shanghai JiaoTong University in 2001, and has been involved in several research projects including model-driven business process analytics, cloud-based service delivery platform, data-driven testing planning & optimization, and connected vehicle service platform & data analytics solutions. Mr. Zhu was an IBM Master Inventor with more than 50 patents filed. He published 30+ papers.

Rong N. Chang is with IBM Research, USA & China, leading a global team creating innovative IoT cloud services technologies. He received his PhD degree in computer science and engineering from the University of Michigan at Ann Arbor in 1990 and his B.S. degree in computer engineering with honors from the National Chiao Tung University in Taiwan in 1982. Before joining IBM in 1993, he was with Bellcore researching on B-ISDN realization. He has won one IEEE Best Paper Award, received four IBM corporate-level Outstanding Technical Achievement Awards, held 30+ patents and published 40+ papers. He is Member of IBM Academy of Technology, ACM Distinguished Engineer, Chair of IEEE Computer Society Technical Committee on Services Computing (TCSVC), Chair of 2015 IEEE World Congress on Services, EIC of the Int. J. of Cloud Computing, and Associate Editor of IEEE Trans. on Services Computing and the Int. J. of Services Computing.