2019 IEEE 35th International Conference on Data Engineering (ICDE)

DRIVEN: a framework for efficient Data Retrieval and clusterIng in VEhicular Networks

Bastian Havers[†*], Romaric Duvignau[†], Hannaneh Najdataei[†],
Vincenzo Gulisano[†], Ashok Chaitanya Koppisetty[*], Marina Papatriantafilou[†]
[†]Chalmers University of Technology
{havers, duvignau, hannajd, vinmas, ptrianta}@chalmers.se
[*]Volvo Cars
{bastian.havers, ashok.chaitanya.koppisetty}@volvocars.com

***Abstract*—Applications for adaptive (sometimes also called smart) Cyber-Physical Systems are blossoming thanks to the large volumes of data, sensed in a continuous fashion, in large distributed systems. The benefits of these applications come nonetheless with a price: the need for jointly addressing challenges in efficient data communication and analysis (among others). The goal of the DRIVEN framework, presented here, is to address these challenges for a data gathering and distance-based clustering tool in the context of vehicular networks. Because of the limited communication bandwidth (compared to the volume of sensed data) of vehicular networks and the monetary costs of data transmission, the intuition behind DRIVEN is to avoid gathering the data to be clustered in a raw format from each vehicle, but rather to allow for a streaming-based error-bounded approximation, through Piecewise Linear Approximation, to compress the volumes of data to be gathered. At the same time, rather than relying on a batch-based clustering algorithm that requires all the data to be first gathered (and then clustered), DRIVEN relies on and extends a streaming-based clustering algorithm that leverages the inherent ordering of the spatial and temporal data being collected, to perform the clustering in an online fashion, while data is being retrieved. As we show, based on our prototype implementation using Apache Flink and our evaluation with real-world data such as GPS and LiDAR, the accuracy loss for the clustering performed on the reconstructed data can be small, even when the raw data is compressed to 10-35% of its original size, and the transferring of data itself can be completed in up to one-tenth of the duration observed when gathering raw data.**

***Index Terms*—compression, streaming data, clustering, edge computing, fog computing**

I. Introduction

Large distributed Cyber-Physical Systems (CPSs) such as vehicular networks [42] (among others) are behind many of the current research threads in computer science. This is because the benefits they potentially bring (e.g., online congestion monitoring, platooning and autonomous driving in the case of vehicular networks) are bounded to many challenges spanning efficient analysis [29], efficient communication [24], [47], security [38] and privacy [18]. An additional challenge in this context is given by the need for solutions that can span and address jointly several such challenges [16], since solutions that focus and/or excel in only one aspect but fall short in others might be impractical in real-world setups.

Challenges: When focusing on aspects such as data communication and analysis, a well known challenge is given by the imbalance between the amounts of data sensed and produced by the sensors deployed in such CPSs (a modern vehicle, for instance, senses more than 20 GB of data per hour) and the infrastructures' capacity of gathering it within small time periods to data centers [8]. This implies that careful design of end-to-end data analysis solutions is crucial for communication not to be a major bottleneck.

A second challenge is given by the inherent limitations of traditional batch and store-then-process (DB) analysis techniques, which on their own cannot sustain the data rates (and required latency bounds) of relevant applications.

Finally, a third challenging aspect gravitates around how to take advantage of the high cumulative computational power of CPSs' edge sensors and devices, since the porting of a given sequential analysis tool (e.g., clustering) to an efficient parallel and distributed implementation and its deployment are not trivial.

Contribution: We present the DRIVEN framework, which copes with the aforementioned challenges for a common problem in vehicular networks' applications, namely clustering of vehicular data. In a nutshell, the DRIVEN framework jointly addresses the challenges of data retrieval, online analysis and leveraging of edge devices' computational power by:

1) reducing the amount of data gathered from vehicles through having the latter forward compact information, by means of Piecewise Linear Approximation - PLA, rather than raw data,
2) leveraging state-of-the-art online clustering techniques such as Lisco [27], which overcome the limitation of batch-based ones, and
3) relying on the data streaming paradigm to transparently achieve distributed and parallel deployments.

As we further elaborate in the remainder, a data analyst interested in gathering and clustering data, sensed by a set of vehicles over a given period of time, can do so by specifying parameters about (i) the type of data to be gathered, (ii) the maximum error that can be introduced while compressing the data to be retrieved to its PLA representation and (iii) the specifications for the clustering of data. The DRIVEN framework then compiles this information into a streaming application that is deployed both at the vehicles providing the data and at the analyst's data center. To support modularity, the framework

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00201

also allows the analyst to define additional components for the resulting application that can be used to process the data before the latter is clustered. We refer to Section V-D for three concrete use cases in which DRIVEN may be employed. An extensive literature exists about clustering, its porting to the streaming protocol and the leveraging of approximation techniques to improve (along certain criteria) the clustering process, as we discuss in Section VI. In this context, our contribution does not aim at surveying all existing solutions nor at comparing them, but rather at providing evidence of how a streaming application that can (i) jointly leverage the computational power of both edge and central components of a CPS and (ii) allow for partial data loss when gathering information is beneficial overall, despite requiring more data processing components (e.g., to compress and decompress the data gathered from the vehicles) than a centralized counterpart which needs all the raw data to be gathered. As we show in our empirical evaluation, based on a prototype implementation using Apache Flink and recent streaming-based PLA and clustering methods, DRIVEN is able to reduce the latency of data transmission by up to 85 % while keeping the quality of the clustering high.

The rest of the paper is organized as follows. We introduce preliminary concepts in Section II and the considered system model and problem statement in Section III. We then present the DRIVEN framework in Section IV and the results of our evaluation in Section V. Finally we discuss related work in Section VI and conclude the paper in Section VII.

II. Preliminaries

We begin this section by discussing preliminary concepts about data streaming, PLA and distance-based clustering. We then present our system model and problem statement.

A. Data Streaming

The data streaming processing paradigm [35] emerged as an alternative to the traditional *store-then-process* one. Thanks to its fast evolution over the last decades, modern Stream Processing Engines (SPEs) allow for distributed, parallel and elastic online analysis [7]. At the same time, efficient designs and methods are in focus in the literature for computationally-expensive streaming analysis [17]. As discussed in [35], the data streaming paradigm has been defined to take into account the challenges proper of large systems gathering data through millions of sensors (as discussed in Section I). Thus, many applications rely on it in many CPSs, including vehicular networks [1], [9], [28].

In data streaming, each sensor produces a *stream* of data, a sequence of *tuples* all sharing the same *schema* composed by *attributes* $\langle ts, A_1, \ldots, A_n \rangle$. Given a tuple t, $t.ts$ represents its creation timestamp while $A_1, \ldots, A_n$ are application-related attributes. We assume that each stream delivers tuples in order as in [4], [22] (or leverages sorting techniques such as [21], [43]). Streaming applications, also referred to as *continuous queries* (or simply queries, in the remainder) are defined as Directed Acyclic Graphs (DAGs) of streams and operators. Each operator defines a function that manipulates its input tuples and potentially produces new output tuples, while streams specify how tuples flow among operators. Modern SPEs such as Apache Flink [7], which we use to implement the DRIVEN framework, provide many operators that can be composed into queries (and also allow for users to define new ad-hoc operators). It should be noted that streaming operators are expected to enforce one-pass analysis [35] and can temporarily maintain a *window* of the most recent tuples when an aggregation function (such as clustering) is to be performed on them [17].

As mentioned in Section I, space and time complexity reduction by means of approximation and/or partial data loss have been discussed in many flavours in the context of streaming applications, proposing solutions such as load shedding, sketches, histograms and wavelets [2], [3], [10], [36]. In the DRIVEN framework we rely on PLA, further discussed in the following section.

B. Piecewise Linear Approximation

Computing a PLA of a time series is a classical problem which aims at representing a series of timestamped records by a sequence of line segments, while keeping the error of the approximation within some acceptable error bound. We consider here the *online* version of the problem, with a prescribed maximum error Δ, i.e. (i) the time series is processed one record at a time, the output line segments are produced along the way, and (ii) the projected points along the compression line segments always fall within Δ from the original tuples. Figure 1 gives an example of a PLA, where original data points are crossed, reconstructed points bulleted, and information about the PLA streamed along the processing of the input stream.

In the extensive literature dealing with such an approximation (among others [5], [12], [23]), it is clearly stated that its main intent is to reduce the size of the input time series for both efficiency of storage and (later) processing. This entails a practical trade-off between a bounded precision loss and space saving for the time series representation. Recent works on PLA [11], [15], [26], [41] increasingly place the focus on the streaming aspect of the compression process, and advocate low time/memory consumption as well as small latencies while achieving a high compression, in order for PLA to be feasibly implemented on top of, or close to, a sensor's stream.

In this work, we use a best-fit line approximation associated with a streaming output mechanism, both introduced in [11] and briefly described in subsection IV-B, offering a balance of trade-offs associated with PLA in a streaming context.

C. Distance-based clustering

Clustering is a core problem in data mining; it requires to group the data into sets, known as clusters, so that intra-cluster similarity is maximized. There are various clustering methods that use different similarity metrics. Among them, *distance-based clustering* methods are able to discover clusters with

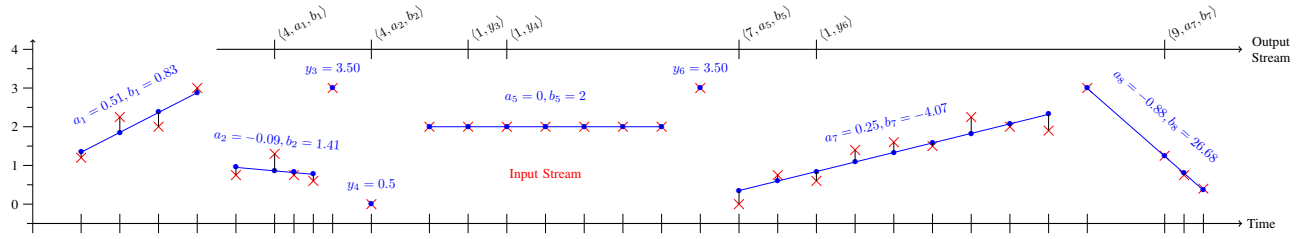

Fig. 1. Example of a Piecewise Linear Approximation using maximum error $\Delta = 0.5$.

arbitrary shapes and form the clusters without knowing the number of them a priori [19].

For ease of reference we paraphrase the definition of distance-based clustering from [33]:

Definition 1. [*Distance-based clustering*] Given n data points, we seek to identify an unknown number of disjoint clusters using a distance metric, so that any two points p_i and p_j be clustered together if they are *neighbors*, i.e. if their distance is within a certain *threshold*. To announce the set of points as a cluster, its cardinality should be at least a predefined number of points $minPts$, otherwise it is considered noise.

In a recent work [27], distance-based clustering (for the Euclidean distance case) is studied in the data streaming paradigm to introduce a new approach, named *Lisco*. The approach enables exploitation of the inner ordering of the data to maximize the analysis pipeline in order to facilitate the extraction of clusters and contribute to real-time processing. In this paper, we use and adapt Lisco as the clustering approach to shape clusters based on distance similarities without knowing the number of clusters in advance. We discuss more details of Lisco and its adaptations in subsection IV-C.

III. System Model and Problem Statement

We consider systems consisting of many vehicles and one *analysis center*, where data analysts are interested in gathering data from a set of vehicles and, subsequently, clustering it.

Each vehicle V_i is equipped with an embedded device which provides limited computational capacity; V_i also mounts a set of sensors producing a stream of tuples composed by attributes $\langle ts, \text{value} \rangle$, i.e. the physical or logical time of each reading and the (possibly multidimensional) measurement itself at that time, respectively. Based on what is found in modern vehicular networks, we assume that each type of sensor produces readings with a given periodicity and that each vehicle is equipped with a storage unit that is used to maintain the sensors' readings (for the sensors deployed in the vehicle) during a given fixed period of time (e.g., during the last month). Finally, we assume that 2-way communication exists between the analysis center and each vehicle, for the former to deploy queries and for the latter to forward the sensed data.

Based on the given system model illustrated by Figure 2, the goal of the DRIVEN framework is to leverage the data streaming paradigm (i.e., to define each query as a DAG of operators that can run in a distributed and parallel fashion both

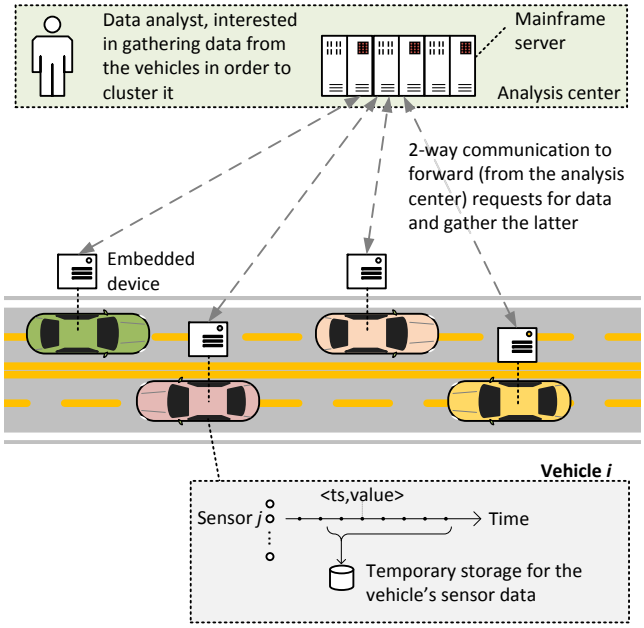

Fig. 2. System model overview for DRIVEN.

at the vehicles and the analysis center) while (i) requesting the analyst to provide information about the analysis' semantics (i.e., which data to gather and the distance criteria to cluster it) while taking care of composing and deploying the overall resulting streaming query and (ii) allowing for approximations, in order to improve the performance (i.e., reduce the time) it takes to retrieve the data sensed by the vehicles.

A query in the DRIVEN framework is expressed as follows:

$$Q(\mathbb{V}, \mathbb{T}, \mathbb{S}, \Delta, [q_{\text{pre}},]\{\text{clustering parameters}\})$$

where:

- $\mathbb{V}$ is a set of vehicles' ids,
- $\mathbb{T}$ is the period of time covered by the data to be gathered (included in the period covered by the vehicles' storage unit),
- $\mathbb{S}$ specifies the set of sensors producing the data (thus allowing the DRIVEN framework to identify the operators needed to gather the stream(s) of data they produce),
- Δ specifies the maximum error that can be introduced during the compression step while retrieving the data by

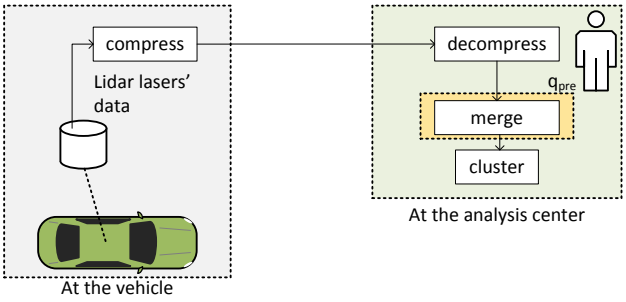

Fig. 3. Overview of the modules deployed in the resulting streaming continuous query for the LiDAR use case.

the DRIVEN framework, and is further composed of k fields $\Delta_1, \Delta_2, \ldots, \Delta_k$ for k-dimensional data,
- q_{pre} is an optional streaming query that defines pre-clustering analysis, and
- $\{$clustering parameters$\}$ is the set of parameters used by DRIVEN's clustering component (further described in Section IV-C2).

Three concrete examples of queries Q and possible values of the parameters are shown in Table I. Notice that, being a streaming query, each DRIVEN application can be extended with additional operators to further process the found clusters (we do not discuss this since it is complementary to our work). In order to quantify the improvement (in terms of efficiency) and the cost (in terms of precision) of the DRIVEN framework, we compare with a baseline that gathers and processes all the raw data rather than the approximated one.

IV. Overview of the DRIVEN framework

In this section, we overview DRIVEN. To facilitate the presentation, we first introduce a use case that serves as a running example in our discussion (we later evaluate it, together with others, in Section V).

As discussed in Section III, each query run by DRIVEN is a streaming continuous query deployed at both the vehicles and the analysis center, with dedicated operators for efficient data retrieval and clustering.

A. Sample use case: study vehicles' surroundings

In our running use case example, the analyst is interested in studying the surroundings of vehicles moving in a certain urban area. We assume that such vehicles are equipped with a set of LiDAR (light detection and ranging) sensors such as the Velodyne HDL-64E [30], which mounts 64 lasers on a rotating vertical column in a non-crossing fashion and that, at each rotation step, shoots these lasers and, based on the time the reflected light rays take to reach back to the sensors, produces a stream of distance readings. Each sensor is able to shoot the laser 4000 times per rotation for up to 5 rotations per second, resulting thus in millions of readings per second for the whole set of sensors [27]. For each stream $\langle ts, distance\rangle = \langle \alpha, \rho\rangle$ produced by one of the LiDAR sensors, the logical timestamp ts allows to identify at which rotation step the distance has been measured (i.e., with which angle α in the x-y plane) while the sensor itself uniquely identifies the elevation angle (we refer the reader to Figure 6 for an example of this).

The analyst is thus interested in the data produced over a certain period of time (e.g., covering a full rotation) by the 64 sensors mounted in each LiDAR deployed in the vehicles moving in the given urban area and relies for the clustering on a function that checks whether the Euclidean distance between any two points is within a certain threshold.

Based on the query description in Section III, the analyst can then run a query $Q(\mathbb{V}, \mathbb{T}, \mathbb{S}, \Delta, q_{\text{pre}}, \{\text{Clustering parameters}\})$ for each vehicle of interest, where:

- $\mathbb{V}$ and $\mathbb{T}$ specify from which vehicle the data should be gathered and which portion of such data should be gathered, respectively,
- $\mathbb{S}$ refers to the sets of LiDAR sensors,
- $\Delta = (\Delta_\alpha, \Delta_\rho)$ defines the maximum approximation error that is allowed when compressing the LiDAR data, bounding the rotation angle error and the distance measurement error, respectively,
- q_{pre} defines an operator merging the data from the different sensors (as further discussed in Section V), and
- $\{$clustering parameters$\}$ is the set of parameters later described in Section IV-C2.

B. Data retrieval and PLA approximation

Figure 3 presents an overview of the modules deployed in the resulting streaming continuous query (each of which will be composed by one or more streaming operators, as also described in the following section).

As discussed in Section I, DRIVEN relies on streaming PLA to forward a compressed and lossy representation of data. In order to build the PLA, we use a construction method, named *Linear*, introduced in [11], by combining several approaches of previous works on PLA such as using a best-fit line approximation [5], [15], [23] for producing small errors and maintaining convex hulls [12], [41] for efficiently checking the violation of the error bound by the approximation. We also use here continuous processing through the *output protocol* proposed in [11] in conjunction with *Linear*, in order to balance the different trade-offs associated with PLA in streaming environments (i.e. compression ratio, reconstruction latencies and individual errors).

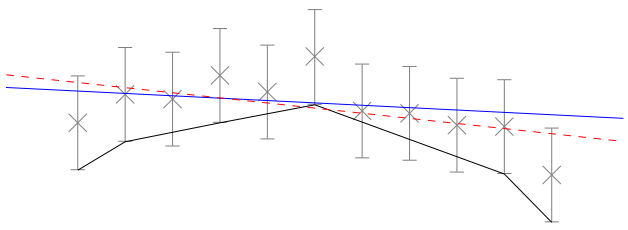

Fig. 4. Best-fit lines of a set of points: solid for 10 points, dashed for 11.

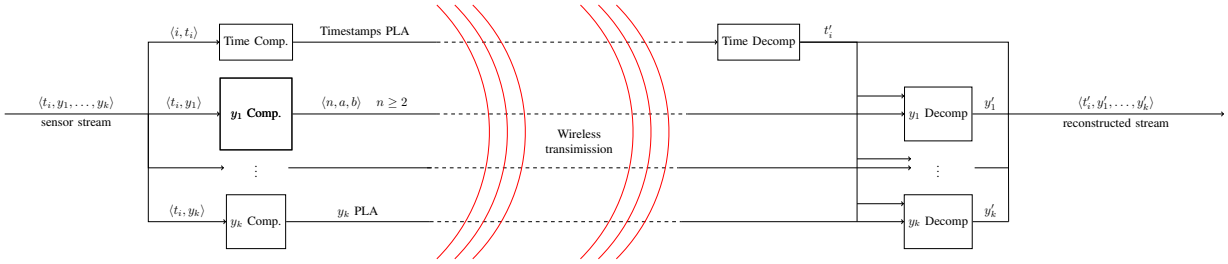

Fig. 5. PLA compression flowchart with y_1's channel detailed.

The *Linear* method successively updates a best-fit line through the latest not yet approximated points, until the maximum error produced by the segment approximation exceeds the tolerated error bound Δ. Updating such an estimate takes $\mathcal{O}(1)$ operations per point, but checking if the line does not violate the error condition can take up to $\mathcal{O}(n)$ if n points are currently being approximated. However, by keeping track of two particular convex hulls U and L along the way (at an extra amortized $\mathcal{O}(1)$ operations per tuple), we can check the error condition in $\mathcal{O}(|U| + |L|)$ by only traversing both hulls, whose sizes are rarely higher than a few units in practice. Figure 4 illustrates the process where input points are plotted as crosses and tolerated errors around the points are depicted as vertical line segments. On the figure, the best-fit line for 10 points is a plain line that stays within the bounded error but for 11 points, the best-fit violates the error bound on the sixth input point (or equivalently the line is below the lower convex hull L shown on the figure).

Here are the different components of the PLA compression as illustrated in Figure 5, used in the framework DRIVEN:

1) **Split** The sensor stream is split in $k+1$ streams, one for each channel plus one additional for the timestamps. More precisely, the i-th input record $\langle t_i, y_1, \cdots, y_k \rangle$ will generate $\langle t_i, y_j \rangle$ on channel j's stream for each $1 \leq j \leq k$ and $\langle i, t_i \rangle$ on the timestamps stream.
2) **PLA compressor** Each stream is compressed in parallel by computing its PLA (as depicted by Figure 1, § II-B) using its associated error, i.e. channel j uses Δ_j; the timestamps stream is considered to be channel no. 0 for that purpose. Each compressor then generates a PLA representation as a stream of triplets $\langle n, a, b \rangle$ where (a, b) are linear coefficients of the next approximation line segment. An optional optimization (discussed in [11]) consists in setting a minimal length of 3 for segments and allowing single values to be output as pairs $\langle 1, y \rangle$ to prevent inflation of the compressed streams.
3) **Diffusion** The $k+1$ streams are wirelessly transmitted to the analysis center.
4) **Time Decompression** The timestamps PLA stream is the first to be processed. The decompression algorithm is straightforward: after having generated i timestamps, we produce n outputs $t'_{i+1}, \cdots, t'_{i+n}$ such that $t'_j = a \cdot j + b$ for $i+1 \leq j \leq i+n$ for each next record $\langle n, a, b \rangle$ of the PLA stream (alternatively, if the non-inflating optimization is in place, we produce $t'_{i+1} = t$ if $\langle 1, t \rangle$ tuples are also forwarded on the PLA stream).
5) **Channel Decompression**[1] Along the time decompression, the k individual PLA streams corresponding to the different input channels can be decompressed in parallel. The decompression procedure here is similar to the one decompressing timestamps, with the addition that it requires a dedicated timestamp stream in connection to the PLA stream. In details, after having reconstructed i values, reading $\langle n, a, b \rangle$ on the PLA stream will trigger the "consumption" of the n subsequent timestamps $t'_{i+1}, \cdots t'_{i+n}$ in order to produce n reconstructed values $y'_j = a \cdot t'_j + b$ for $i+1 \leq j \leq i+n$ (alternatively, if the non-inflating optimization is in place, reading $\langle 1, y \rangle$ generates one single value $y'_{i+1} = y$).
6) **Final Reconstruction** The final step is to merge the $(k+1)$ decompressed streams to rebuild records with identical structure as the initial input stream.

C. Data clustering with Lisco

As described in section II, distance-based clustering approaches form clusters using a given distance metric. Since computing the distances of one tuple from all the other tuples in a certain dataset in order to find the ones within a threshold distance would incur an $\mathcal{O}(n^2)$ complexity when running all-to-all comparisons, it is necessary to prune the search space. For this purpose, several clustering approaches have an intermediate step after data acquisition and before the main clustering algorithm. This additional step builds an extra supporting data structure (e.g. a kd-tree [14]) in order to organize the collected data before performing the clustering. In this way, they introduce a batch-based processing which results in an average $\mathcal{O}(n \log n)$ cost [40] but requires multiple passes over the data (possibly affecting the performance).

Lisco is a recently proposed method that overcomes the batch processing disadvantages by means of a single-pass continuous distance-based clustering (Euclidean in the original

[1]Note that combining approximated times and values implies that the maximum observed error can increase by a factor proportional to the speed of change of the measured values. This can be prevented by compressing timestamps without losses (if, e.g., most timestamps follow regular patterns) or injecting the approximated timestamps before computing the compressed PLA in each channel. As we show in our evaluation, nonetheless, such few increases in errors do not have a dominating effect even when lossy compression is allowed for timestamps.

paper [27]) that exploits inherent orderings of data (when such orderings are present). The intuition behind Lisco is to store the data in a simpler data structure that preserves such inherent ordering and therefore eliminate the need for an extra supporting sorting data structure. In the original paper, it is discussed (and empirically observed) that storing and organizing data tuples using Lisco have $\mathcal{O}(1)$ complexity and can be performed during the data acquisition step which results in an average $\mathcal{O}(n)$ cost.

While we rely on the LiDAR-based use case to overview DRIVEN and its clustering component, our implementation of Lisco within DRIVEN opens up for clustering of other data too, as we discuss in the following.

1) Clustering LiDAR data (intuition): Figure 6 a) shows Lisco's intuition for clustering LiDAR data. As shown, for a certain point p hit by a laser, the search for neighbors within a certain (Euclidean) distance can be limited to a certain set of lasers and angles (based on p's distance and angles). The *neighbor mask* containing possible points hit by such lasers (and for the given angles) specifies the portion of data outside of which neighbors can *not* be found for p. This limits the search space for p's neighbors to the points measured for the given range of angles and lasers. Notice that such points must be checked since not all angles and lasers falling within the given ranges necessarily hit a point that is a neighbor of p, as shown in the figure. Internally, Lisco can then maintain incoming points in a 2D array.

2) Lisco generalization in DRIVEN: The Lisco implementation in DRIVEN maintains data in an n-dimensional array and clusters incoming tuples while they are stored in it. One of the n dimensions is given by the ts attribute while the other *optional* $n-1$ dimensions can be specified as attributes of the tuple's schema. In this way, the analyst can leverage any implicit sorting carried by one or more attributes of the tuples produced by q_{pre} (aside from the timestamp itself) to speed-up Lisco's clustering. To do this, the first clustering parameter defined by the analyst is an optional list of attributes to define the additional $n-1$ dimensions of Lisco's internal multi-dimensional array. It should be noticed that, for each attribute A_i specified as a dimension by the analyst, the latter must also specify the range of values observable for it, in order for DRIVEN to setup Lisco's internal data structure.

The second and third clustering parameters are the functions `int[n] getMaskSize(Tuple t)` and `boolean areNeighbors(Tuple a,Tuple b)`. The former function specifies how far (in the sense of indexes) Lisco should explore any of the n dimensions of the array around tuple t, to search for potential candidates for clustering. Lisco employs the return values of this function to create the neighbor mask and bound the search space around t. Internally, Lisco runs the aggregation over any of the n dimensions as soon as the latter is filled for a given value (e.g., when all the tuples sharing the same ts values are received). The latter function is used to check if two tuples falling into the same neighbor mask should be clustered together or not.

Finally, the analyst must also specify the minimum number of points $minPts$ to differentiate clusters from noise (Def. 1).

Continuing the example in Figure 6 b), the schema of the tuples produced by q_{pre} could in this case carry attributes $\langle \alpha, \theta, \rho \rangle$, where α is the logical timestamp that refers to a certain angle of the LiDAR sensor, θ is the elevation angle (based on the laser producing the reading) and ρ is the measured distance. To store the tuples, Lisco could be instructed to keep data in a 2D matrix where the different lasers are assigned to different rows and consecutive readings from the same laser are assigned to columns (as done in the original paper [27]). In this way, the ordering of two dimensions of the tuples would be kept. Using the 2D matrix to store the data, `int[n] getMaskSize(Tuple t)` can be implemented to return the neighbor mask of a tuple t in terms of a limited number of rows and columns around t. Finally, the `areNeighbors(Tuple a,Tuple b)` can be implemented to check whether the Euclidean distance between a and b is within the threshold defined by the analyst.

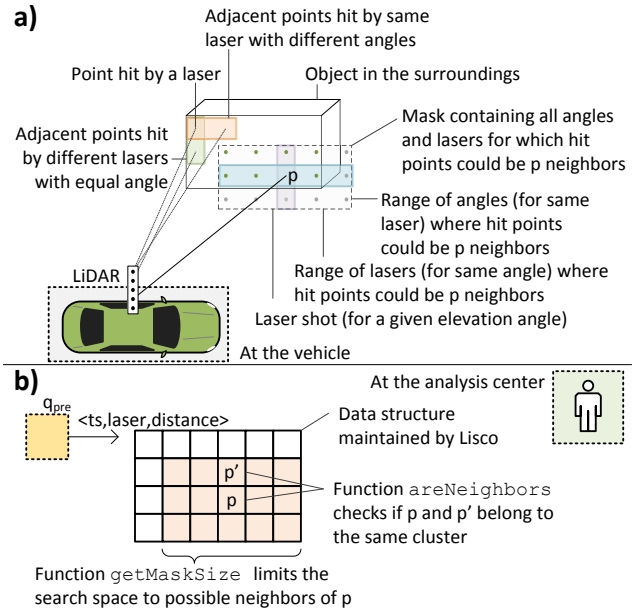

Fig. 6. Example of how the search space for a point p (for the LiDAR use case) can be limited to points potentially reported by lasers (and with certain angles) within a mask centered in p a) and the corresponding 2D matrix maintained by Lisco b).

In Section V, we provide more examples of the clustering attributes for data from other sensors (GPS trackers).

V. Evaluation

We evaluate in here the tradeoffs in compression, introduced error, retrieval time and clustering quality for DRIVEN. We first present the datasets used, the software and hardware setup and then discuss three different use cases.

A. Data

We use two datasets in our evaluation. The first one is the *Ford Campus* dataset [30], providing data generated by a VelodyneHDL64E roof-mounted LiDAR (see Section IV-A for an overview of LiDAR) from one vehicle. Each file in the dataset corresponds to one full rotation of the LiDAR, which consists of 64 individual lasers mounted in a column. According to our system model (Section III), each of the 64 lasers is an individual sensor, with the sensor ID being the sensor's fixed elevation angle. In our implementation, each laser stores its data in a dedicated file. The second dataset contains GPS data in the format *timestamp, latitude, longitude, height* and was collected in the GeoLife project by 182 users over ca. three years [44]–[46]. We use a subset of data with vehicular GPS traces in the Beijing area.

B. Software and hardware setup

We implemented Lisco in Python 3.5 and the PLA components (see Section IV-B) in Apache Flink 1.5.0. The segment length n of the PLA compressor is bounded by 256 to limit its bandwidth consumption to 1 byte, while the parameter *minPts* is set to 10 in all experiments.

We use as stand-in for the vehicle node an ODROID-XU3 single-board computer to approximate the low-power processor of a vehicle, equipped with a Samsung Exynos 5422 Cortex-A15 2.0 Ghz quad core and Cortex-A7 quad core CPU and 2 GB of LPDDR3 RAM at 933 MHz. For the analysis center, we use a server with an Intel Intel(R) Core(TM) i7-4790 3.60 GHz quad-core CPU and 8 GB of RAM. The ODROID and the server are connected via Ethernet. We reduce their bandwidth using the tool *trickle* [13] and simulate three different upload speeds for the ODROID: *slow* (8 KB/s, in the range of 2G), *medium* (500 KB/s, in the range of 3G) and *fast* (1000 KB/s, in the range of 4G). All box-plots presented in the following contain points from at least 60 experiment runs.

C. Evaluation metrics

DRIVEN is evaluated for three use cases among four dimensions as a function of the maximum errors Δ_{ts}, Δ_A:

- *Average error*: The average error $\bar{Y} = \frac{1}{N}\sum_{i=1}^{N}|y_i - y'_i|$ between original values y_i and reconstructed ones y'_i.
- *Compression ratio*: The size of the compressed data divided by the raw data size.
- *Adjusted rand index*: The clustering obtained from the approximated data compared to the clustering obtained from the original data via the adjusted rand index.[2]
- *Gathering time ratio*: The time needed to gather the approximated data (including compression/decompression overheads) divided by that taken to gather the raw data.

For all use cases, we use the term *baseline* to refer to a setup in which raw data (i.e, with no compression) is gathered and clustered. Since simulating the communication behavior of a large vehicular network is beyond the scope of this work (and based also on the observation that real behavior would depend on factors we cannot predict, such as the position of a certain vehicle), experiments studying the gathering time are set up to favor the baseline over DRIVEN and thus avoid bias. More concretely, gathering time is measured for the collection of each sensor's data without concurrent or parallel transfers, thus avoiding overheads (e.g., packet losses) proportional to the size of the transmitted information (i.e., higher for the baseline, given that raw data is larger in size than the compressed one, as we show in the following).

[2]The rand index of two partitions (or sets of clusters) A, B is a symmetric measure that counts how many pairs of elements in partition B are clustered exactly as in partition A. The adjusted rand index extension takes into account accidental random clusterings (see [39] for more details).

D. Use cases

1) Q_1 LiDAR: This is the use case presented in detail in Section IV-A. In accordance with our system model, the data for each of the 64 lasers is stored on-vehicle as a stream of $\langle ts, A\rangle = \langle \alpha, \rho\rangle$ with the azimuth angle α (logical timestamp) and the distance reading ρ. The query for this use case is detailed in Table I. Based on the query, all the sensor reading streams from the last ten rotations from each of the 64 LiDAR lasers from one vehicle are successively compressed on-vehicle with some maximum errors $\Delta_\alpha, \Delta_\rho$ on the logical timestamps α and the distance readings ρ. Each compressed stream of laser readings is then successively sent to the analysis center, where the streams are decompressed. q_{pre} assigns each tuple its laser id and the horizontal angle θ, providing the data structure shown in Figure 7 to Lisco. The tuples are added column-wise (see colored column) by q_{pre} with decreasing laser id θ^i (the id of the i-th laser) from top to bottom and increasing rotation angle α^i_j (the j-th rotation step of the i-th laser) from left to right. This merging of data from different sensors is performed deterministically [17] based on the ts attribute carried by the tuples. As clustering parameters, Lisco is instructed to check the Euclidean distance between pairs of tuples; to accomplish that, it searches for candidates in a maximum α, θ - area around tuple `t` defined by the function `getMaskSizeInRotation(t)`, which simply calculates the angles α, θ of the horizontal and vertical laser beams who could hit points that are within distance $\epsilon = 0.5$ m from the sensor reading corresponding to t, as they bound the ϵ-neighborhood of the latter, while also ensuring that only points within the same rotation are part of the mask. Through the way that data is maintained in a 2D matrix, this defines a rectangular area (i.e. mask) in the matrix around t (cp. Section IV-C1). The compression statistics for this use case can be seen in Figure 8 (a) and (b) expressed via boxplots. The angle α is in all cases compressed with a maximum error Δ_α of 5×10^{-4} rad, yielding an average error on α of $5\times10^{-5} \pm 2.6\times10^{-5}$

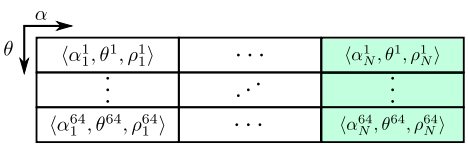

Fig. 7. Sketch of data structure produced by q_{pre} (Q_1).

TABLE I
QUERIES $Q(\mathbb{V}, \mathbb{T}, \mathbb{S}, \Delta, \{\text{clustering parameters}\}, q_{\text{pre}})$ FOR EVALUATION. SEE SECTION III FOR EXPLANATION OF QUERY ARGUMENTS.

Q_1: LiDAR. $\langle ts, A \rangle = \langle \alpha, \rho \rangle$

$\mathbb{V}$	$\mathbb{T}$	$\mathbb{S}$	Δ	q_{pre}	{clustering parameters}

1 specific vehicle	10 full rotations	LiDAR 64 lasers	$\Delta_\alpha = 0.0005$ rad, $\Delta_\rho \in [0.1, 0.2, 0.5, 1, 5, 10]$ m	merge 64 lasers
				add laser id
					`getMaskSize(Tuple t):`
					`  return getMaskSizeInRotation(t)`
					`areNeighbors(Tuple a, Tuple b):`
					`  return euclideanDist(a,b)≤0.5 m`

Q_2^a: 1-Vehicle 1-Day. $\langle ts, A \rangle = \langle t, [x, y] \rangle$

$\mathbb{V}$	$\mathbb{T}$	$\mathbb{S}$	Δ	q_{pre}	{clustering parameters}

1 specific vehicle	1 day	GPS	$\Delta_t = 1$ s, $\Delta_x, \Delta_y \in [1, 2, 5, 10, 20, 50]$ m	`windowAggregate(5 s)`
				emit latest tuple
					`getMaskSize(Tuple t):`
					`  return 12`
					`areNeighbors(Tuple a, Tuple b):`
					`  return euclideanDist(a,b)≤50 m`

Q_2^b: 1-Vehicle 7-Day. $\langle ts, A \rangle = \langle t, [x, y] \rangle$

$\mathbb{V}$	$\mathbb{T}$	$\mathbb{S}$	Δ	q_{pre}	{clustering parameters}

1 specific vehicle	7 days	GPS	$\Delta_t = 1$ s, $\Delta_x, \Delta_y \in [1, 2, 5, 10, 20, 50]$ m	`windowAggregate(10 s)`
				add day id
				emit latest tuple
					`getMaskSize(Tuple t):`
					`  return 15, 7`
					`areNeighbors(Tuple a, Tuple b):`
					`  return euclideanDist(a,b)≤100 m`

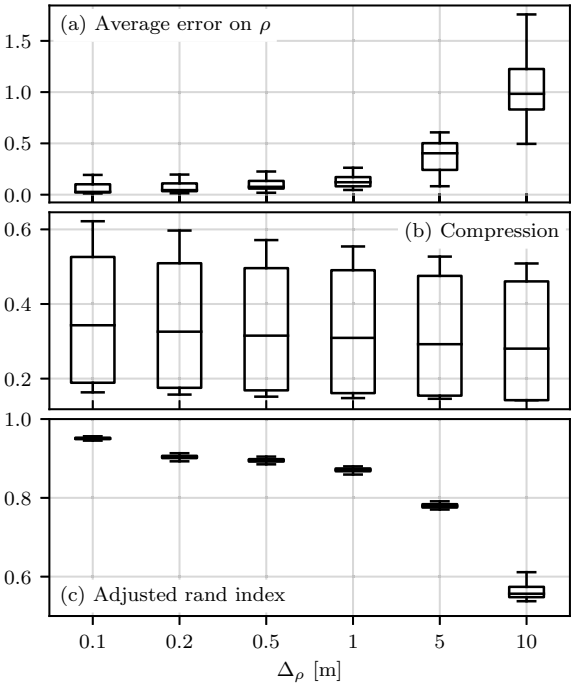

Fig. 8. Compression and clustering statistics (Q_1) for various Δ_ρ.

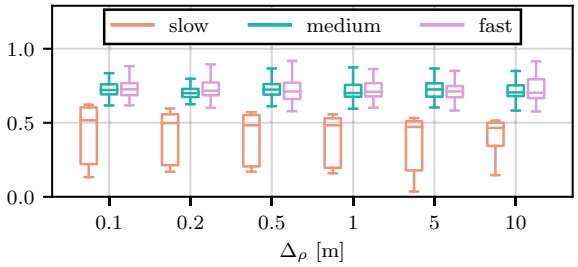

Fig. 9. Gathering time ratio (Q_1) for various Δ_ρ and network speeds.

rad (average $\pm$ standard deviation). ρ is compressed for values $[0.1, 0.2, 0.5, 1, 5, 10]$ m. In (a), it appears that the average error is about one order of magnitude smaller than the maximum error. The compression as a ratio of compressed vs. raw file size in (b) shows that LiDAR data can already for maximum errors $\Delta_\rho \leq 0.1$ m be compressed below (median) 35 % of the raw size, which we attribute to the regularity of the logical timestamps α as well as to the existence of stretches of $\rho = 0$ in the raw data (these occur when the laser is not reflected, cp. Section IV-A). For increasing maximum error, the compression ratio decreases only slightly, which indicates that almost maximum compression is reached early. The comparison of the clusterings from query Q_1 with the baseline is shown in Figure 8 (c) (as only points within the same rotation may be clustered, we compare the clusterings between the same rotations, not between the sets of ten rotations). One observes that for increasing Δ_ρ the similarity between the clustering of approximated and baseline data decreases. However, for $\Delta_\rho = 0.1$ m, the median compression ratio is already below 0.35, while the median adjusted rand index is 0.95, hinting that a large compression can be achieved without a large loss of precision in the clustering. Figure 9 shows the end-to-end gathering time ratio for the three network speeds. While there is no significant decrease for increasing maximum error (according to the compression rate), for all network speeds data gathering times are reduced to 90 % for fast networks down to less than 25 % for slow ones.

2) Q_2^a 1-Vehicle 1-Day: In this use case, the analyst requests the GPS data of a single day from one vehicle to cluster all points within a predefined distance and timespan. This could e.g. serve to identify areas of slow traffic or areas where the vehicle stopped. Based on our system model, the

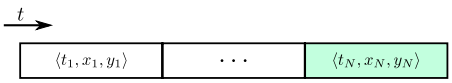

Fig. 10. Sketch of data structure produced by q_{pre} (Q_2^a).

data is stored on-vehicle as a stream of tuples $\langle t, [x, y] \rangle$ with the logical timestamp being the actual measurement time and the x, y attributes being the coordinates in meters.

The query Q_2^a for this use case is described in detail in Table I. The GPS position data stream from one day and one specific vehicle is compressed on-vehicle with some error Δ_t on the timestamps and errors $\Delta_x = \Delta_y$ on the vehicle's GPS coordinates and sent to the analysis center. There, the stream of decompressed tuples is aggregated by q_{pre} in tumbling windows of 5 seconds, and for each window only the latest tuple is returned as soon as the window completes. The data structure provided to Lisco is sketched in Figure 10 (the coloured field contains the last added tuple). If a window is empty because no data exists for the corresponding time period, the field in the data structure will also remain empty. As clustering parameters, Lisco is instructed via `getMaskSize(t)` to check the last 6 indexes of the 1D array, reducing the search space for neighbors and ensuring the clustering only of points that are also close in time. The clustering decision is taken by the function `areNeighbors(a,b)` on the basis of the Euclidean distance between the x, y-coordinates of the two tuples. The compression statistics are given in Figure 11 (a), (b). We choose in this case a fixed error $\Delta_t = 1$ s for the

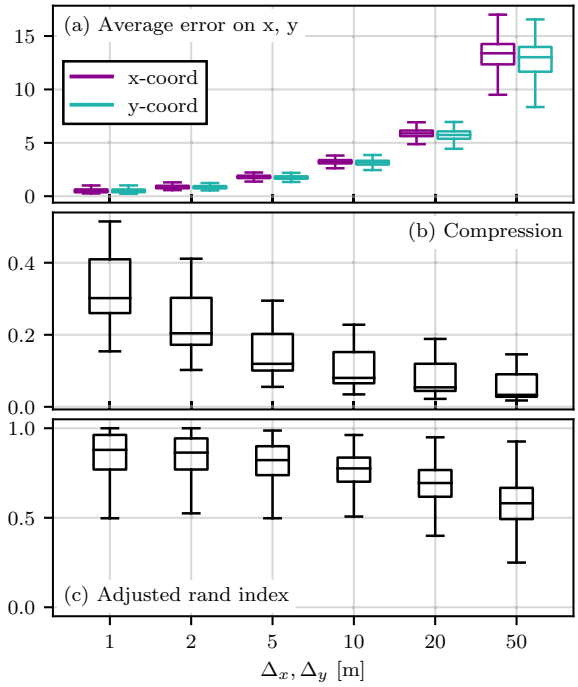

Fig. 11. (a), (b): Compression statistics (Q_2^a); (c): Adjusted rand index (Q_2^a).

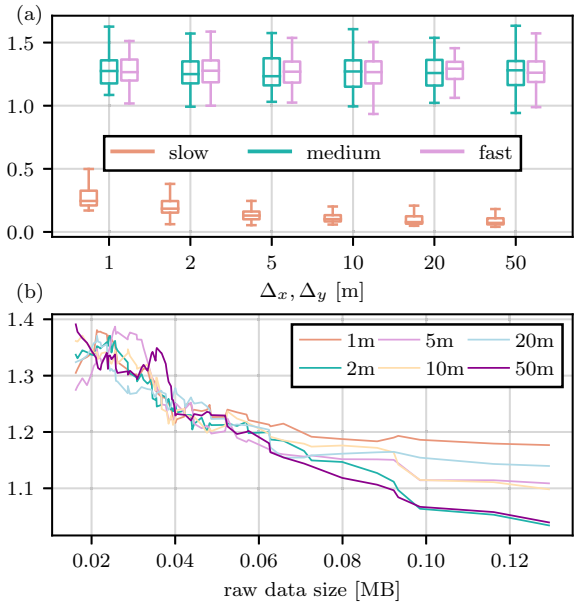

Fig. 12. Gathering time ratios (Q_2^a) for various (a) maximal errors for different network speeds and (b) raw data sizes (rolling average over 13 values, different colors are used for distinct values of Δ_x, Δ_y) for a 3G network.

compression of the timestamps, resulting in an average error of (0.095 ± 0.090) s. Δ_x and Δ_y are chosen to be equal and $\in [1, 2, 5, 10, 20, 50]$ m. Figure 11 (a) shows the average error on both coordinates as a function of the maximum errors, with the average errors as in query Q_1 being roughly one order of magnitude smaller than the allowed maximum error, although for $\Delta_x = \Delta_y = 1$ m the average error is sometimes greater than 1 m (see Section IV-B, *Channel Decompression* for a discussion). Figure 11 (b) shows the total compression achieved for each maximum error: assuming a measurement uncertainty of GPS on the order of a few meters, maximum compression errors of less than ten meters may be assumed small. Still, these result in compression ratios that can be lower than 0.4. This may be explained with straight roads, resulting in long, linear segments in the GPS data. The results for the comparison of the clustering of approximated and raw data is shown in Figure 11 (c): for small maximum errors Δ_x, Δ_y, the adjusted rand index is close to 1, but it decreases for larger maximum errors. The gathering time ratios are shown in Figure 12 (a). The median is greater than or close to 1 for faster networks. This shows that DRIVEN in this use case is only beneficial for a slow network. More insight is gained from Figure 12 (b), showing the gathering time ratios as a function of the baseline data size for a medium speed (3G) network. For small data sizes, the additional time overhead of the compression-decompression procedure increases the gathering duration over directly transmitting the raw sample. For larger sample sizes, the latency ratio approaches 1 for large maximum errors Δ_x, Δ_y. This gives an approximation for the minimum size of data to be collected given the network bandwidth and the compression / decompression overheads, as

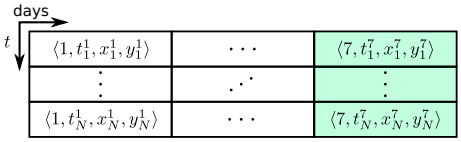

Fig. 13. Sketch of data structure produced by q_{pre} (Q_2^b).

we further show in the remainder (we stress nonetheless our evaluation setup is optimal for raw data gathering).

3) Q_2^b 1-Vehicle 7-Day: In this use case, the analyst requests the GPS data from one specific vehicle from the last 7 days, to possibly identify routes that one vehicle is driving on a regular basis. The query Q_2^b, described in the query overview in Table I, differs from Q_2^a in the period covered by the data and in the task of q_{pre}: upon consecutively receiving the GPS streams for each day and windowing (with 10 second windows) as in the previous use case, each tuple is also assigned an identifier for the day. As soon as the stream for one day is processed, it is added column-wise to a data structure as shown in Figure 13 (the first entry of each tuple is the day identifier id, t is the number of seconds from midnight on day id). Lisco can thus process the GPS stream of each day as soon as it is received. In contrast to the previous use case, Lisco is now instructed to search the last 15 cells in the direction t, and all cells in the direction "days" (at most 7), for tuples within a Euclidean distance of 150 m.

The compression statistics may be found in Figure 14 (a), (b) and are similar to those seen in Figure 11 (a), (b), as

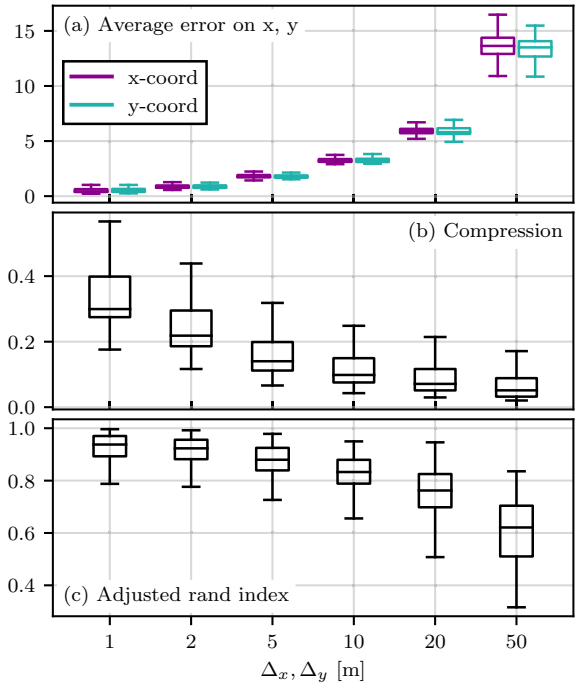

Fig. 14. (a), (b): Compression statistics (Q_2^b); (c): Adjusted rand index (Q_2^b).

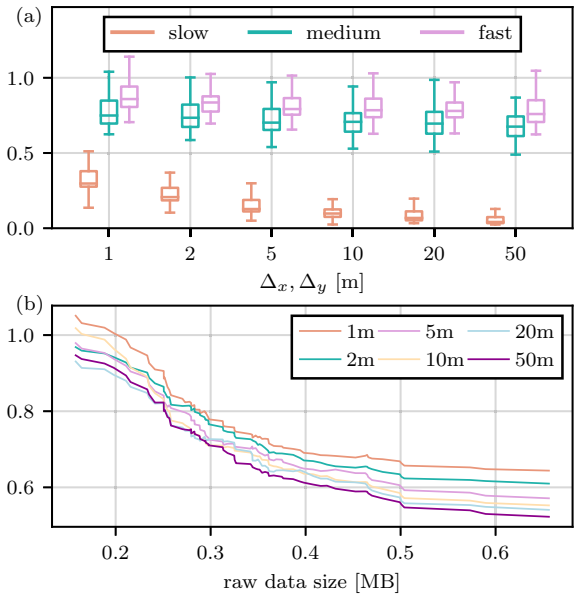

Fig. 15. Gathering time ratios (Q_2^b) for various (a) maximal errors for different network speeds and (b) raw data sizes (rolling average over 13 values, different colors are used for distinct values of Δ_x, Δ_y) for a 3G network.

the same data type with only increased sample size is used. Here the constant maximum error $\Delta_t = 1$ s results in an average error of (0.082 ± 0.080) s. The evaluation of clustering qualities is shown in Figure 14 (c), also with similar results to the previous use case. The addition of the attribute "days" for Lisco has little influence, suggesting that in the majority of samples there is no significant number of inter-day clusters. Figure 15 (a) shows the measured gathering time ratios. The median gathering time ratios are below 0.5 for all values of the maximum errors for a slow network, and for faster networks around 0.75, although also samples with ratios greater than 1 are present. As shown in Figure 15 (b) (for medium network speeds), this is due to samples with raw data size smaller than 200 KB. For samples larger than 250 KB the gathering time ratios for all values of Δ_x, Δ_y are less than 1.

Summary: The evaluation shows that GPS and LiDAR compression rates below 40 % can be achieved with small impact on the quality of approximated data clustering. Also, DRIVEN affords $\times 4$ speedup in data gathering time for large-enough data sets (at least 200 KB per sensor in our setup).

VI. Related Work

Clustering, as a core problem in data mining, has been extensively studied in the last decades (see e.g. the survey [20] and the references therein). The two main trends in clustering algorithms differ on what should be considered as a cluster, either privileging well-balanced ball-like clusters (as in the widely-studied k-means approach) or rather focusing on local density leading to arbitrarily shaped clusters (e.g. DBSCAN [14]-style). Further granularity in the many existing clustering algorithms features their sensitivity to outliers (not

interesting data that should be ignored in some applications), their ability to work with any distance function and the level of parametrization that is required to make them cluster properly.

Research on data streaming has also investigated how traditional batch-based clustering can be ported to the continuous domain. Clustering for large fast-coming streaming data has been widely studied in the last decade [34], focusing on producing approximations of the batch-clustering algorithm. Facing high rate data streams, attention has indeed been paid to maintaining statistical summaries of the streamed data in order to generate on-demand clustering. Focus on recent data is captured by clustering only a *recent window* (using either landmarks, sliding window, or assigning decreasing weights to older data) of points [34]. In [37], the authors design a fully streaming clustering algorithm (as the streaming version of a recently proposed clustering algorithm [32]), computing the exact same clustering as the batch version of the algorithm. Similar to the clustering algorithm described here, the clustering is density-based (hence for arbitrarily shaped clusters), works with any distance function but uses a different notion of dissimilarity between objects. However, contrary to our work, the ordering of data is not exploited resulting in $\mathcal{O}(n)$ time for the processing of a single point (while clustering n points).

Various solutions in the literature use approximation techniques together with streaming-based clustering methods to improve the performance, in one or more dimensions, of different clustering problems. Replacing time series by shorter representations [6] such as Discrete Wavelet/Fourier Transforms or Symbolic Aggregate Approximation to facilitate the processing and enhance the performance of several data mining algorithms (including clustering) has been a long trend in time series data mining [31]. Differently from our work, PLA or similar techniques (such as piecewise aggregate or piecewise constant approximation) are used to replace a time series by a lighter version to be later processed, as for clustering of time series in [25]. In our work, the objects being clustered are not the time series but the input points themselves, PLA is used only to efficiently gather data (i.e., the data stream eventually clustered has the same length as the original one). To the best of our knowledge, this joint leveraging of streaming and PLA was not discussed before.

Concerning the generation of the PLA of a time series, there is an extensive literature covering it (e.g. [12], [23], [26], [41]) while focusing on different aspects of the approximation (errors, number of segments, processing time, etc). Among recent works targeting sensor streams, we note the Embedded SWAB algorithm [5] (a modification of the well-known SWAB [23] segmentation) dedicated to the compression of wireless sensor raw data before transmission. The experimental study measuring power consumption shows that using PLA pays off in embedded devices by balancing out the computation overhead with reduced communication, and thus less energy is spent overall. The authors note that the abstraction size is crucial in wireless sensor networks, thus motivating studying trade-offs between small errors and high compression, which is one of the focal points of this work, in the context of the considered applications relevant in industrial settings. We also measure the time spent in the decompression process in our work and advocate that information retrieval from the measured data is also faster with PLA than with raw data transmission. In another recent work [15], the authors devise a PLA algorithm with small memory footprint and average instruction count for resource constrained wireless sensor nodes. They use a best-line approximation (similar to us) but with no intercept (so more segments are produced), and the error is bounded by segment instead of by point (hence producing a PLA with a higher number of segments for the same threshold).

VII. Conclusion

We have presented here the DRIVEN framework for data retrieval and clustering in vehicular networks. The framework, implemented in a state-of-the-art SPE, provides simultaneously an efficient way for gathering data and performing clustering on said data upon an analyst's queries. Information retrieval is achieved using PLA for compressing the input stream in a streaming fashion. The lossy stream is then fed to a distance-based clustering algorithm. Both the approximation and the clustering are parameterizable for allowing different applications of the framework. Through thorough experimentation using real-world GPS and LiDAR data, we show the versatility of the framework in being able to answer different types of queries involving various clustering requests for vehicular networks and that compression in data retrieval speeds up the transmission of gathered data while being able to preserve a very similar clustering quality compared to raw data transmission. Data can be reduced to $10-35$ % of its raw size, reducing drastically the gathering phase for large volumes of data, with only a small accuracy loss on the clustering.

Acknowledgment

Work supported by VINNOVA, the Swedish Government Agency for Innovation Systems, proj. "Onboard/Offboard Distributed Data Analytics (OODIDA)" in the funding program FFI: Strategic Vehicle Research and Innovation (DNR 2016-04260); the Swedish Foundation for Strategic Research, proj. "Future factories in the cloud (FiC)" (grant GMT14-0032), and the Swedish Research Council (Vetenskapsrådet), proj. "HARE: Self-deploying and Adaptive Data Streaming Analytics in Fog Architectures" (grant 2016-03800).

References

[1] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *Proceedings of the 30th International Conference on Very Large Data Bases-Volume 30*, pages 480–491. VLDB Endowment, 2004.

[2] B. Babcock, S. Babu, R. Motwani, and M. Datar. Chain: Operator scheduling for memory minimization in data stream systems. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 253–264. ACM, 2003.

[3] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *Proceedings of the 20th International Conference on Data Engineering*, pages 350–361. IEEE, 2004.

[4] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 13–24, New York, NY, USA, 2005. ACM.

[5] E. Berlin and K. Van Laerhoven. An on-line piecewise linear approximation technique for wireless sensor networks. In *Proceedings of IEEE 35th Conference on Local Computer Networks*, pages 905–912. IEEE, 2010.

[6] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh. Beyond one billion time series: indexing and mining very large time series collections with i sax2+. *Knowledge and information systems*, 39(1):123–151, 2014.

[7] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[8] R. Coppola and M. Morisio. Connected car: technologies, issues, future trends. *ACM Computing Surveys (CSUR)*, 49(3):46, 2016.

[9] S. Costache, V. Gulisano, and M. Papatriantafilou. Understanding the data-processing challenges in intelligent vehicular systems. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 611–618. IEEE, 2016.

[10] M. Datar and R. Motwani. The sliding-window computation model and results. In *Data Streams*, pages 149–167. Springer, 2007.

[11] R. Duvignau, V. Gulisano, M. Papatriantafilou, and V. Savic. Streaming piecewise linear approximation for efficient data management in edge computing. In *34th ACM/SIGAPP Symposium On Applied Computing SAC'19*, 2019. to appear.

[12] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref, and W. Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. *Proceedings of the VLDB Endowment*, 2(1):145–156, 2009.

[13] M. A. Eriksen. Trickle: A userland bandwidth shaper for unix-like systems. In *USENIX Annual Technical Conference, FREENIX Track*, pages 61–70, 2005.

[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, pages 226–231, 1996.

[15] F. Grützmacher, B. Beichler, A. Hein, T. Kirste, and C. Haubelt. Time and memory efficient online piecewise linear approximation of sensor signals. *Sensors*, 18(6):1672, 2018.

[16] V. Gulisano, M. Almgren, and M. Papatriantafilou. Metis: a two-tier intrusion detection system for advanced metering infrastructures. In *International Conference on Security and Privacy in Communication Systems*, pages 51–68. Springer, 2014.

[17] V. Gulisano, Y. Nikolakopoulos, D. Cederman, M. Papatriantafilou, and P. Tsigas. Efficient data streaming multiway aggregation through concurrent algorithmic designs and new abstract data types. *ACM Transactions on Parallel Computing (TOPC)*, 4(2):11:1–11:28, 2017.

[18] V. Gulisano, V. Tudor, M. Almgren, and M. Papatriantafilou. Bes: Differentially private and distributed event aggregation in advanced metering infrastructures. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 59–69. ACM, 2016.

[19] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[20] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[21] Y. Ji, H. Zhou, Z. Jerzak, A. Nica, G. Hackenbroich, and C. Fetzer. Quality-driven continuous query execution over out-of-order data streams. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 889–894, New York, NY, USA, 2015. ACM.

[22] E. Kalyvianaki, M. Fiscato, T. Salonidis, and P. Pietzuch. Themis: Fairness in federated stream processing under overload. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 541–553, New York, NY, USA, 2016. ACM.

[23] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings of 2001 IEEE International Conference on Data Mining*, pages 289–296. IEEE, 2001.

[24] A. Keramatian, V. Gulisano, M. Papatriantafilou, P. Tsigas, and Y. Nikolakopoulos. Mad-c: Multi-stage approximate distributed cluster-combining for obstacle detection and localization. In *European Conference on Parallel Processing*, pages 312–324. Springer, 2018.

[25] J. Lin, M. Vlachos, E. Keogh, D. Gunopulos, J. Liu, S. Yu, and J. Le. A mpaa-based iterative clustering algorithm augmented by nearest neighbors search for time-series data streams. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 333–342. Springer, 2005.

[26] G. Luo, K. Yi, S.-W. Cheng, Z. Li, W. Fan, C. He, and Y. Mu. Piecewise linear approximation of streaming time series data with max-error guarantees. In *2015 IEEE 31st International Conference on Data Engineering*, pages 173–184. IEEE, 2015.

[27] H. Najdataei, Y. Nikolakopoulos, V. Gulisano, and M. Papatriantafilou. Continuous and parallel lidar point-cloud clustering. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 671–684. IEEE, 2018.

[28] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, and U. Ramachandran. Mcep: a mobility-aware complex event processing system. *ACM Transactions on Internet Technology (TOIT)*, 14(1):6, 2014.

[29] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou. Genealog: Fine-grained data streaming provenance at the edge. In *Proceedings of the 19th International Middleware Conference*, pages 227–238. ACM, 2018.

[30] G. Pandey, J. R. McBride, and R. M. Eustice. Ford campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552, 2011.

[31] C. A. Ralanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das. Mining time series data. In *Data mining and knowledge discovery handbook*, pages 1069–1103. Springer, 2005.

[32] A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.

[33] R. B. Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010.

[34] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, 46(1):13, 2013.

[35] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.

[36] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th international Conference on Very Large Data Bases-Volume 29*, pages 309–320. VLDB Endowment, 2003.

[37] L. Ulanova, N. Begum, M. Shokoohi-Yekta, and E. Keogh. Clustering in the face of fast changing streams. In *Proceedings of 2016 SIAM International Conference on Data Mining*, pages 1–9. SIAM, 2016.

[38] J. van Rooij, V. Gulisano, and M. Papatriantafilou. Locovolt: Distributed detection of broken meters in smart grids through stream processing. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*, pages 171–182. ACM, 2018.

[39] S. Wagner and D. Wagner. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.

[40] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in o (n log n). In *2006 IEEE Symposium on Interactive Ray Tracing*, pages 61–69. IEEE, 2006.

[41] Q. Xie, C. Pang, X. Zhou, X. Zhang, and K. Deng. Maximum error-bounded piecewise linear representation for online stream approximation. *The VLDB Journal*, 23(6):915–937, 2014.

[42] S. Yousefi, M. S. Mousavi, and M. Fathy. Vehicular ad hoc networks (vanets): challenges and perspectives. In *Proceedings of 2006 6th International Conference on ITS Telecommunications*, pages 761–766. IEEE, 2006.

[43] N. Zacheilas, V. Kalogeraki, Y. Nikolakopoulos, V. Gulisano, M. Papatriantafilou, and P. Tsigas. Maximizing determinism in stream processing under latency constraints. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 112–123. ACM, 2017.

[44] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. Understanding mobility based on gps data. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 312–321. ACM, 2008.

[45] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.

[46] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th International Conference on World Wide Web*, pages 791–800. ACM, 2009.

[47] J. Zhou, R. Q. Hu, and Y. Qian. Scalable distributed communication architectures to support advanced metering infrastructure in smart grid. *IEEE Transactions on Parallel and Distributed Systems*, 23(9):1632–1642, 2012.