CHALMERS
UNIVERSITY OF TECHNOLOGY

DRIVEN: A framework for efficient Data Retrieval and clustering in Vehicular Networks

Downloaded from: https://research.chalmers.se, 2025-12-04 19:22 UTC

Citation for the original published paper (version of record):
Havers, B., Duvignau, R., Najdataei, H. et al (2020). DRIVEN: A framework for efficient Data Retrieval and clustering in Vehicular Networks. Future Generation Computer Systems, 107: 1-17. http://dx.doi.org/10.1016/j.future.2020.01.050

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

DRIVEN: a framework for efficient Data Retrieval and clusterIng in VEhicular Networks[⋆,⋆⋆]

Bastian Havers[a,b,*], Romaric Duvignau[a], Hannaneh Najdataei[a], Vincenzo Gulisano[a], Marina Papatriantafilou[a], Ashok Chaitanya Koppisetty[b]

[a] *Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden*
[b] *Volvo Cars, Gothenburg, Sweden*

Abstract

The growing interest in data analysis applications for Cyber-Physical Systems stems from the large amounts of data such large distributed systems sense in a continuous fashion. A key research question in this context is how to jointly address the efficiency and effectiveness challenges of such data analysis applications.

DRIVEN proposes a way to jointly address these challenges for a data gathering and distance-based clustering tool in the context of vehicular networks. To cope with the limited communication bandwidth (compared to the sensed data volume) of vehicular networks and data transmission's monetary costs, DRIVEN avoids gathering raw data from vehicles, but rather relies on a streaming-based and error-bounded approximation, through Piecewise Linear Approximation (PLA), to compress the volumes of gathered data. Moreover, a streaming-based approach is also used to cluster the collected data (once the latter is reconstructed from its PLA-approximated form). DRIVEN's clustering algorithm leverages the inherent ordering of the spatial and temporal data being collected to perform clustering in an online fashion, while data is being retrieved. As we show, based on our prototype implementation using Apache Flink and thorough evaluation with real-world data such as GPS, LiDAR and other vehicular signals, the accuracy loss for the clustering performed on the gathered approximated data can be small (below 10 %), even when the raw data is compressed to 5-35 % of its original size, and the transferring of historical data itself can be completed in up to one-tenth of the duration observed when gathering raw data.

Keywords: compression, streaming data, clustering, edge computing, fog computing

1. Introduction

Large distributed Cyber-Physical Systems (CPSs) such as vehicular networks [44] (among others) are behind many of the current research threads in computer science. One of the aspects many of such research threads share has its roots in the large amounts of data sensed continuously in large distributed CPSs. As discussed in the literature, the benefits and possibilities CPSs' data enables (e.g. online congestion monitoring, platooning and autonomous driving in the case of vehicular networks) are bound to many challenges, spanning efficient analysis [31], efficient communication [49, 26], security [40] and privacy [19]. A key aspect in this context is the need for solutions that can jointly address several such challenges [17], since solutions that focus on and/or excel in only one aspect but fall short in others might be impractical in real-world setups.

1.1. 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, on the road today, senses more than 20 GB/h of data [8]) and the infrastructures' capacity of gathering them within small time periods to data centers [12]. Even when data is not to be transmitted continuously, but only for a limited time period and for some selection of sensors, the required bandwidth may far exceed the available one (e.g. a single LiDAR sensor of an autonomous car produces around 7 MB/s, cf. Section 4.1). In this case, solutions focusing on efficient data analysis need to account for communication aspects too, in order for the latter not to result in a major bottleneck. The inherent limitations of traditional batch and store-then-process (DB) analysis techniques, which on

⋆Preliminary results have been presented at the 35th IEEE International Conference on Data Engineering (ICDE 2019) [21].

⋆⋆Accepted manuscript, FGCS, https://doi.org/10.1016/j.future.2020.01.050. Licensed under CC BY-NC-ND 4.0.

*Corresponding author

Email addresses: havers@chalmers.se (Bastian Havers), duvignau@chalmers.se (Romaric Duvignau), hannajd@chalmers.se (Hannaneh Najdataei), vinmas@chalmers.se (Vincenzo Gulisano), ptrianta@chalmers.se (Marina Papatriantafilou), ashok.chaitanya.koppisetty@volvocars.com (Ashok Chaitanya Koppisetty)

their own cannot sustain the data rates of relevant applications, need thus to be overcome by taking into account the end-to-end transformation process of raw data into valuable insights. Specifically, considering which data – as well as how much data – is moved through a certain analysis pipeline. Because of this, a complementary challenge 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.

1.2. Contributions

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

1. leveraging a lossy compression technique, based on Piecewise Linear Approximation (PLA), that significantly reduces the amounts of data to be gathered from vehicles,
2. leveraging state-of-the-art online clustering techniques such as Lisco [29], which overcome the limitations 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 (because of the PLA-based compression) 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 as well as at the analyst's data center. To support modularity, the framework 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.

An extensive literature exists about clustering, its porting to the streaming protocol and the leveraging of approximation techniques to improve (along with certain criteria) the clustering process, as we discuss in Section 6. In this context, our contribution does not aim at surveying all existing solutions nor at comparing them. Rather, the contribution focuses on 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 can provide a healthy tradeoff between data reduction and pipeline speed on the one hand and accuracy loss on the other, 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 recently proposed streaming-based PLA and clustering methods, and four real-world use cases, DRIVEN is able to reduce the duration of data transmission by up to 90 % while incurring a bounded loss on the clustering quality.

The rest of the paper is organized as follows. We introduce preliminary concepts in Section 2 and the considered system model and problem statement in Section 3. We then present the DRIVEN framework in Section 4 and our evaluation in Section 5. Finally, we discuss related work in Section 6 and conclude the paper in Section 7.

2. Preliminaries

We begin this section by discussing preliminary concepts about data streaming, PLA, distance-based clustering and logical latency.

2.1. Data streaming

The data stream processing paradigm (aka data streaming) [37] 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 [18]. As discussed in [37], 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 1). Thus, many applications rely on it in many CPSs, including vehicular networks [1, 9, 30].

In data streaming, each sensor produces a *stream* of data, a sequence of *tuples* that share the same *schema* composed by *attributes* $\langle y^0, y^1, \ldots, y^k \rangle$, where y^0 is a physical or logical timestamp and the other k attributes depend on the sensor producing the stream. We assume that each stream delivers tuples in order based on y^0 as in [4, 24] (or leverages sorting techniques such as [23, 45]). 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 ad-hoc operators). It should be noted that streaming operators are expected to enforce one-pass analysis [37] and can temporarily maintain a *window* of the most recent tuples when an aggregation function (such as clustering) is

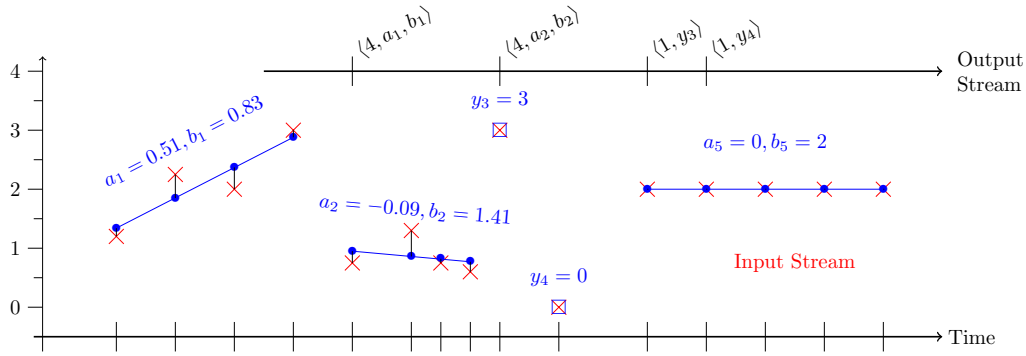

Figure 1: Example of a Piecewise Linear Approximation using maximum error $\Delta = 0.5$.

to be performed on them [18]. As mentioned in Section 1, space and time complexity reduction through approximation and/or partial data loss have been discussed in many flavors in the context of streaming applications. Proposed solutions include load shedding, sketches, histograms and wavelets [2, 3, 10, 38]. In DRIVEN, we rely on PLA, further discussed in the following section.

2.2. Piecewise Linear Approximation

Computing a PLA of a time series is a classical problem that aims at representing a series of timestamped points 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 point 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 points. Figure 1 gives an example of a PLA: original data points (crosses on the figure) from the input stream are compressed and forwarded on the output stream as line segments (solid lines) or singletons (squares), so that a reconstructed stream (bullets and squares) can be generated from the PLA of the original stream (we refer the reader to Section 4 for more details about why both segments and singletons are defined and the conditions upon which they are forwarded).

In the extensive literature dealing with such an approximation (among others [25, 13, 5]), it is clearly stated that the approximation's 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 [43, 28, 16, 11] increasingly place the focus on the streaming aspect of the compression process, and advocate low time/memory consumption as well as small latency 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 together with a streaming output mechanism, both introduced in [11] and briefly described in Section 4.2, balancing trade-offs associated with PLA in a streaming context.

2.3. Distance-based clustering

Clustering is a core problem in data mining; it requires to group 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 arbitrary shapes and form the clusters without a-priori knowledge about their number [20]. For ease of reference we paraphrase the definition of distance-based clustering from [35]:

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 are 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 (rather than noise), its cardinality should be at least a predefined number of points $minPts$.

In a recent work [29], distance-based clustering (for the Euclidean distance case) is studied in the data streaming paradigm to introduce a new approach, named *Lisco*. This approach enables the 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 Section 4.3.

2.4. Logical latency

In data streaming literature [18], the term latency usually refers to the (physical) time difference between the production of an output tuple and the processing of the last input tuple contributing to (or triggering the creation of) the former.

This latency definition is usually employed to evaluate the processing performance of a given streaming-based

solution. When sequences of multiple input tuples are aggregated together following the processing of a later tuple without an a-priori known size for the length of such sequences (as in a PLA segment, given that the length of each segment depends on the points it approximates), users can also be interested in the *logical latency* introduced by the aggregation mechanism. We refer to the notion of *logical latency* as the number of tuples processed between a given tuple and the first tuple that triggers the aggregation of the sequence to which the former belongs. More concretely, with a sequence of n tuples being aggregated together $\langle y_\ell^0, y_\ell^1, \ldots, y_\ell^k \rangle$, ..., $\langle y_{\ell+n}^0, y_{\ell+n}^1, \ldots, y_{\ell+n}^k \rangle$ and $\langle y_j^0, y_j^1, \ldots, y_j^k \rangle$ the tuple that triggers their aggregation (with $j \geq \ell + n$), the logical latency for any tuple $\langle y_i^0, y_i^1, \ldots, y_i^k \rangle$ is $j - i$ for $i \in [\ell, \ell + n]$.

3. System model and problem statement

We consider systems consisting of a set of many vehicles and one *analysis center*, in which data analysts are interested in gathering data from such a set of vehicles and, subsequently, clustering that data at the analysis center. Each vehicle V_i is equipped with an embedded device which provides limited computational capacity; V_i also mounts a set of sensors, each producing a stream of tuples composed by attributes $\langle y^0, \ldots, y^k \rangle$, i.e., the physical or logical time of each reading and the measurements 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). Notice that lossy data compression techniques such as PLA are not applied to the data before storing it since the allowed error bound is not known before the analyst triggers a data gathering request.

For ease of exposition, we assume in the remainder that (historical) data is stored locally at each vehicle and retrieved only when requested. We nonetheless investigate in Section 5.6 the *logical latency* incurred in a scenario in which live readings are streamed to an analysis center immediately after their compression.

We also assume that 2-way communication exists between the analysis center and each vehicle to deploy queries and to forward the sensed data, respectively. To isolate the effects of DRIVEN on data gathering and analysis from non-deterministic factors such as varying speeds and reliability of the underlying communication layer, we assume this 2-way communication to provide constant upload and download speeds and no packet loss (cf. Section 5.3).

Based on the given system model illustrated in Figure 2, the goal of the DRIVEN framework is to leverage the data streaming paradigm (i.e., to define queries that gather and cluster data as DAGs of operators that can run in a distributed and parallel fashion both at the vehicles

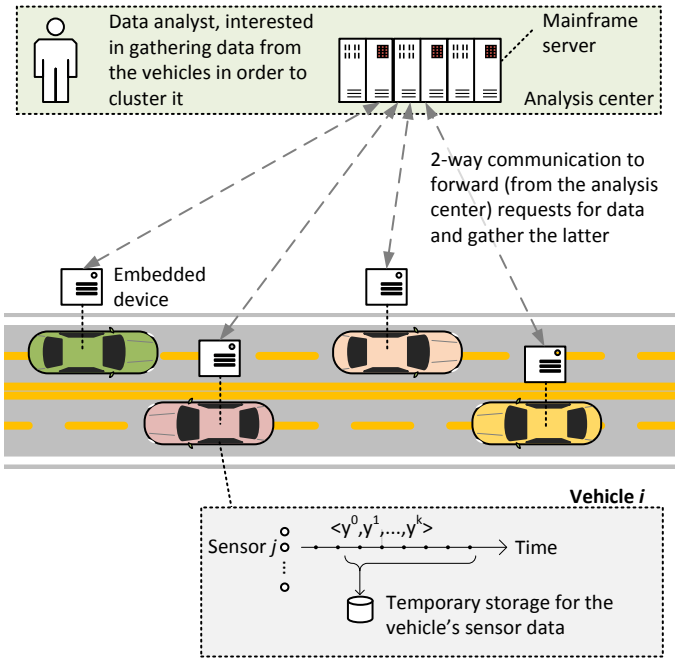

Figure 2: System model overview for DRIVEN.

and the analysis center) while (i) only requiring analysts to provide information about the analysis' semantics (i.e., which data to gather and the distance criteria to cluster it) without composing and deploying the overall streaming query themselves and (ii) allowing for approximations in order to improve the performance (i.e., reduce the time) of retrieving the data sensed by the vehicles.

A query in the DRIVEN framework is expressed as

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

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 or referring to data being sensed live by the vehicle),
- $\mathbb{S}$ 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 the DRIVEN framework, and is further composed of $k+1$ fields, namely $\Delta_1, \Delta_2, \ldots, \Delta_k$ for a sensor with k attributes, plus Δ_0 for the (logical) time attribute,
- q_{pre} is an optional streaming query that defines pre-clustering analysis, and

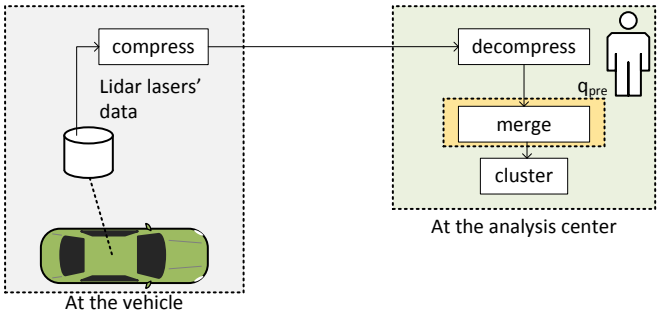

Figure 3: Overview of the modules deployed in the resulting streaming continuous query for the LiDAR use case.

- {clustering parameters} is the set of parameters used by DRIVEN's clustering component (further described in Section 4.3.2).

We refer the reader to Section 5 and Table 1 for concrete examples of queries Q and possible values of the above parameters. 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 rather than the approximated data.

4. Overview of the DRIVEN framework

In this section, we present an overview of 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 5).

As discussed in Section 3, 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.

4.1. Sample use case: study vehicles' surroundings

In our running use case example, the analyst is interested in performing the clustering of LiDAR data of a bounded time interval as a preprocessing step for offboard object detection. This may help, e.g. in better understanding and improving the performance of a resource-constrained onboard object detection algorithm in a certain driving situation, and may be automatically triggered by an event such as a pedestrian crossing the road in front of the vehicle. We assume vehicles equipped with a set of LiDAR (light detection and ranging) sensors such as the ones of a Velodyne HDL-64E [32], which mounts 64 non-crossing lasers on a rotating vertical column and which, at each rotation step, shoots these lasers and produces a stream of distance readings based on the time the reflected light rays take to reach back to the sensors. Each sensor can 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 [29] (around 7 MB/s). For each stream $\langle \alpha, \rho \rangle$ produced by one of the LiDAR sensors, the logical timestamp α allows identifying at which rotation step the distance ρ has been measured (i.e., with which angle in the horizontal plane). Notice that each reading from a sensor can be converted into a 3D point in space based on α, ρ and the elevation angle of the sensor itself.

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 3, the analyst could 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 5), and
- {clustering parameters} is the set of parameters later described in Section 4.3.2.

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).

4.2. Data retrieval and PLA approximation

As discussed in Section 1, DRIVEN relies on streaming PLA to forward a compressed and lossy representation of

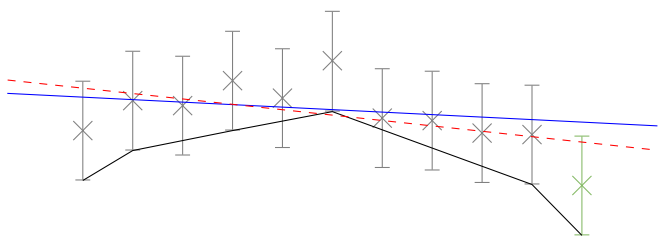

Figure 4: Best-fit lines of a set of points: solid for the first 10 points, dashed for including the 11th point (marked in green).

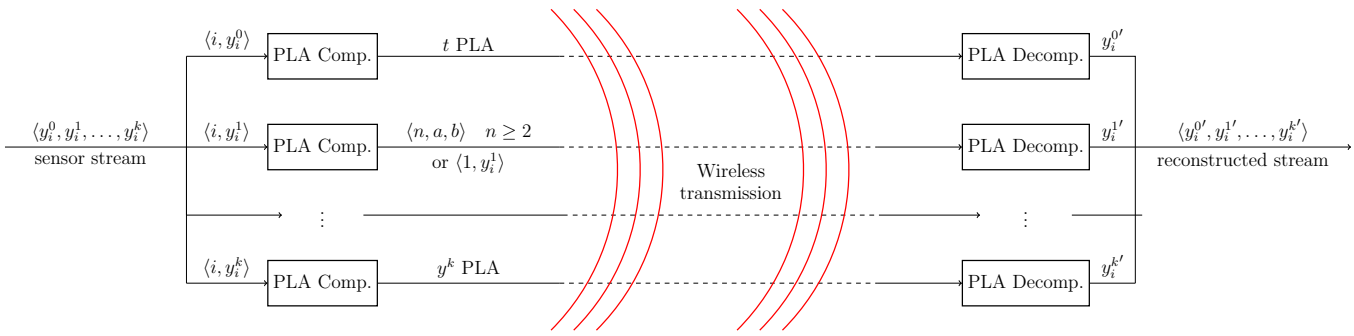

Figure 5: PLA compression / decompression flowchart with y^1's channel detailed.

data. To build the PLA, we use a construction method named *Linear*, introduced in [11], which combines several approaches of previous works on PLA such as using a best-fit line approximation [25, 5, 16] for minimizing errors and maintaining convex hulls [13, 43] 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 latency, and individual errors).

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 (worst-case). However, rather than a naive sequential check that always results in the worst-case cost, 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 (as also observed in our extensive experimental evaluation). Figure 4 illustrates the process where input points are plotted as crosses and tolerated errors around the points are drawn as vertical line segments; the best-fit line for the first 10 points is a plain line that stays within the bounded error. By adding the 11th point (marked green), the best-fit line violates the error bound on the sixth input point (or equivalently, at the sixth input point, the approximation line is below the lower convex hull L depicted on the figure).

For the input sensor stream, composed of tuples of the form $\langle y^0, y^1, \ldots, y^k \rangle$, the different components of the PLA compression, as illustrated[1] in Figure 5, are:

1. **Split:** The sensor stream is split in $k+1$ streams, one for each application-related attribute plus one additional for the timestamps. More precisely, the i-th input tuple $\langle y_i^0, y_i^1, \cdots, y_i^k \rangle$ will generate $\langle i, y_i^\ell \rangle$ on channel ℓ's stream for each $0 \leq \ell \leq k$.
2. **PLA Compression:** Each stream is compressed in parallel by computing its PLA (as depicted in Figure 1, Section 2.2) using its associated error, i.e., channel ℓ uses Δ_ℓ. Each compressor generates a PLA representation as a stream of triplets $\langle n, a, b \rangle$ or singletons $\langle 1, y \rangle$; $\langle n, a, b \rangle$ is generated for compressing n input values into a line segment whose linear coefficients are (a, b), whereas $\langle 1, y \rangle$ is generated to reproduce a single input[2] of value y. In more detail, this is presented in Algorithm 1: the PLA compressor always attempts to first build the longest possible approximation segment $< n_{max}$ (of length 256, in our evaluation, cf. Section 5), but when one such segment has only length $n = 2$, thus covering only two tuples $\langle k-2, y_{k-2} \rangle$ and $\langle k-1, y_{k-1} \rangle$, $\langle k-2, y_{k-2} \rangle$

Algorithm 1 PLA output logic

▷ Receive $\langle i, y_i \rangle$, while maintaining convex hulls U_{i-1}, L_{i-1} & best-fit line l_{i-1} covering $n \geq 2$ points
$l_i \leftarrow$ newBestFitLine$(l_{i-1}, \langle i, y_i \rangle)$
$U_i, L_i \leftarrow$ updateConvexHulls$(U_{i-1}, L_{i-1}, \langle i, y_i \rangle)$
$n \leftarrow n + 1$
if lineBreaksHulls(l_i, U_i, L_i) **then**
  if $n = 3$ **then** ▷ output singleton
    output $\langle 1, y_{i-2} \rangle$
  else ▷ output segment
    output $\langle n-1, \text{slopeOf}(l_{i-1}), \text{interceptOf}(l_{i-1}) \rangle$
  end if
else
  if $n = n_{max}$ **then** ▷ max length reached
    output $\langle n_{max}, \text{slopeOf}(l_i), \text{interceptOf}(l_i) \rangle$
  else ▷ wait for $\langle i+1, y_{i+1} \rangle$
    continue
  end if
end if

[1]For simplicity, we do not draw in the figure the operators in charge of components 1 and 5.

[2]Note that we add 1 in singleton tuples because both singletons and triplets are forwarded using the same channel, and thus require a prefix that distinguishes them when deserializing incoming data.

is then output as a singleton $\langle 1, y_{k-2} \rangle$. The PLA compressor continues the construction of a new approximation line segment beginning with the tuple $\langle k-1, y_{k-1} \rangle$ and the current tuple $\langle k, y_k \rangle$ (this explains the output delay associated with the singletons of Figure 1). This scheme helps to mitigate an inflation phenomenon observed when compression is low (it improves the compression when a single outlier tuple lies between two segment-compressible sequences of tuples).

3. **Diffusion:** The $k+1$ streams are wirelessly transmitted to the analysis center.
4. **PLA Decompression:** All streams are decompressed in parallel. The decompression algorithm is straightforward: after having already reconstructed i values on channel y^ℓ, we either generate n outputs $y'_{i+1}, \ldots, y'_{i+n}$ such that $y'_j = a \cdot j + b$ for $i+1 \leq j \leq i+n$ if the next received record is $\langle n, a, b \rangle$ on y^ℓ's transmitted PLA stream, or alternatively, we produce $y'_{i+1} = y$ if $\langle 1, y \rangle$ was received.
5. **Final Reconstruction:** The final step is to merge the $k+1$ decompressed streams to rebuild records with identical structure as the initial input stream. In particular, to reconstruct the i-th tuple $\langle y_i^{0'}, y_i^{1'}, \ldots, y_i^{k'} \rangle$ on the output stream, we need to wait for the $k+1$ decompressors to have produced at least i reconstructed values on their respective channel.

The compression scheme suggested here compresses all k attributes of a stream as well as the timestamp in the same manner (i.e., by using the tuple counter i for each tuple $\langle i, y_i^\ell \rangle$ on a sensor attribute y^ℓ). This differs from the scheme suggested in [21], where a counter was used only for the timestamp stream $\langle i, y_i^0 \rangle$ while the remaining attributes of the stream were compressed using the original timestamps, and decompressed using the reconstructed timestamps. As explained in [21], this scheme could not guarantee a bounded reconstruction error at all times as errors from timestamp reconstruction could propagate to the reconstruction of other channels. The new scheme proposed here results in similar compression and performance figures on our evaluated data, as discussed later in Section 5, and guarantees a bounded reconstruction error.

4.3. Data clustering with Lisco

As described in Section 2, 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

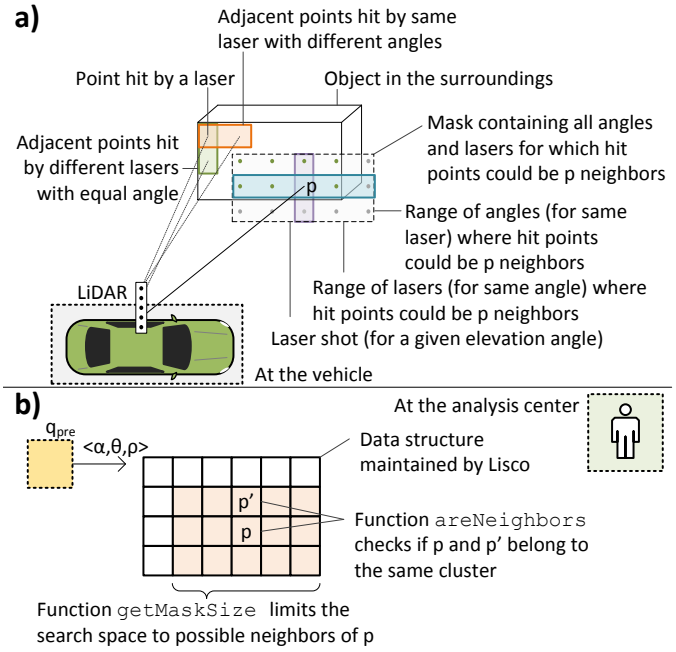

Figure 6: Example of how the search space for a point p (for the LiDAR use case) can be limited to points potentially reported by lasers (and with certain angles) within a mask centered in p (a) and the corresponding 2D matrix maintained by Lisco (b).

(e.g. a kd-tree [15]) in order to organize the collected data before performing the clustering. In this way, a batch-based processing is introduced which results in an average $\mathcal{O}(n \log n)$ cost [42] but requires multiple passes over the data (possibly affecting the performance).

Lisco is a recently proposed method that overcomes the batch processing disadvantages through a single-pass continuous distance-based clustering (Euclidean in the original paper [29]) 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 eliminates 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 the clustering of other data too, as we discuss in the following.

4.3.1. 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.

4.3.2. 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 y^0 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 y^k specified as a dimension by the analyst, the latter must also specify the range of values observable for it, for DRIVEN to setup Lisco's internal data structure.

The second and third clustering parameters are the functions `int[n] getMaskSize(Tuple` τ`)` and `boolean areNeighb(Tuple` τ_1`,Tuple` τ_2`)`. The former function specifies how far (in the sense of indexes) Lisco should explore any of the n dimensions of the array around tuple τ, 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 τ. 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 y^0 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 (Definition 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 consecutive readings from the same laser are assigned to columns and the different lasers are assigned to different rows (as done in the original paper [29]). 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` τ`)` can be implemented to return the neighbor mask of a tuple τ in terms of a limited number of rows and columns around τ. Finally, the `areNeighb(Tuple` τ_1`,Tuple` τ_2`)` can be implemented to check whether the Euclidean distance between the readings of tuples τ_1 and τ_2 is within the threshold defined by the analyst.

5. Evaluation

We evaluate in here the tradeoffs in compression, approximation error, retrieval time and clustering quality for DRIVEN. We first present the datasets used, the software and hardware setup and then discuss four different use cases in which historic data is gathered and clustered. Finally, we gauge PLA's compression performance by comparing it to a lossless, general-purpose compression technique (ZIP) and discuss the concept of inherent logical latency of PLA compression to investigate the impact of DRIVEN on queries gathering live rather than historic data.

5.1. Data

We use three datasets in our evaluation.

1. The *Ford Campus* dataset [32], providing data generated by a VelodyneHDL64E roof-mounted LiDAR (see Section 4.1 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 3), 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.
2. The *GeoLife* dataset, containing GPS data collected in the *GeoLife* project by 182 users over ca. three years [46, 47, 48]. We use a subset of the dataset with vehicular GPS traces in the Beijing area.
3. The *Volvo* dataset, provided by Volvo Cars. This dataset consists of CAN data[3] and GPS traces from 20 hybrid cars and was collected in Sweden in the years 2014 and 2015.

5.2. Software and hardware setup

We implemented Lisco in Python 3.6 and the PLA components (see Section 4.2) 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 (this parameter is adopted from [29]).

We use as a 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

[3]CAN (Controller Area Network) is a vehicular communication bus standard over which sensor data, fault messages, etc. can be transmitted.

the analysis center, we use a server with an 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 the connection bandwidth using the tool *trickle* [14] to simulate four different upload speeds for the ODROID: *slow* (8 KB/s, in the range of 2G), *medium* (500 KB/s, in the range of 3G), *fast* (1000 KB/s, in the range of 4G) and *very fast* (10000 KB/s, in the range of 5G).

5.3. Evaluation metrics

DRIVEN is evaluated for four use cases among four dimensions:

- *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.[4]
- *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.
In addition to the four evaluation dimensions explained above, we also evaluate the *Logical Latency* for the *Ford Campus* and *GeoLife* datasets (we refer the reader to Section 2.4 for a definition of logical latency).
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 from multiple vehicles, 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).

In the following, results are presented through violin plots. Violin plots show the distribution of the underlying data along their vertical axis, with the mean marked by a horizontal bar. All the presented plots contain data from at least 55 experiment runs.

5.4. Use cases

5.4.1. Q_1 LiDAR

This is the use case presented in detail in Section 4.1. In accordance with our system model, the data for each of the 64 lasers is stored on-vehicle as a stream of $\langle \alpha, \rho \rangle$ with the azimuth angle α (logical timestamp) and the distance reading ρ. The query for this use case is detailed in Table 1. 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. Query q_{pre} assigns each tuple its laser id and the horizontal angle θ, providing to Lisco the data structured as the 2D matrix (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 [18] based on the logical timestamp α 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 τ defined by `getMaskSizeInRot(`τ`)`, which 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 τ, 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 the 2D matrix, this defines a rectangular area (i.e., mask) in the matrix around τ (cp. Section 4.3.1).

The compression statistics for this use case can be seen in Figure 8 (a) and (b) expressed via violin plots. The angle α is in all cases compressed with a maximum error Δ_α of 1.5×10^{-3} rad, yielding an average error on α of $3.4 \times 10^{-5} \pm 1.0 \times 10^{-5}$ rad (average $\pm$ standard deviation). ρ is compressed for values $[0.1, 0.2, 0.5, 1, 5, 10]$ m. In (a), it appears for larger values of Δ_ρ that the average error is about one order of magnitude smaller than the

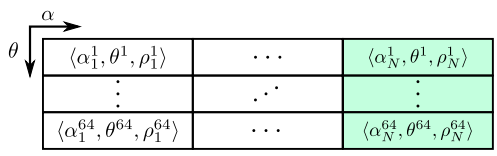

Figure 7: Q_1: Sketch of data structure produced by q_{pre}.

[4]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 [41] for more details).

Table 1: Queries $Q(\mathbb{V}, \mathbb{T}, \mathbb{S}, \Delta, q_{\text{pre}}, \{\text{clustering parameters}\})$ for evaluation. See Section 3 for explanation of query arguments.

Q_1: LiDAR

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

1 vehicle	10 rot.	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` τ`):`
					`return getMaskSizeInRot(`τ`)`
					`areNeighb(Tuple` τ_1`, Tuple` τ_2`):`
					`return euclidDist(`τ_1`,`τ_2`)`$\leq$`0.5m`

Q_2: 1-Vehicle 1-Day

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

1 vehicle	1 day	GPS	$\Delta_t = 1$ s,
			$\Delta_x, \Delta_y \in [1, 2, 5, 10, 20, 50]$ m
				`windowAggr(5s)`
				emit latest tuple
					`getMaskSize(Tuple` τ`):`
					`return 12`
					`areNeighb(Tuple` τ_1`, Tuple` τ_2`):`
					`return euclidDist(`τ_1`,`τ_2`)`$\leq$`50m`

Q_3: 1-Vehicle 14-Day

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

1 vehicle	14 days	GPS	$\Delta_t = 1$ s,
			$\Delta_x, \Delta_y \in [1, 2, 5, 10, 20, 50]$ m
				`windowAggr(10s)`
				add day id
				emit latest tuple
					`getMaskSize(Tuple` τ`):`
					`return 15, 14`
					`areNeighb(Tuple` τ_1`, Tuple` τ_2`):`
					`return euclidDist(`τ_1`,`τ_2`)`$\leq$`100m`

Q_4: Car usage grids

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

20 vehicles	7 days	GPS
		electric RPM
		comb. RPM
			$\Delta_{t^{\text{G}}} = 1$ s, $\Delta_{t^{\text{e}}} = \Delta_{t^{\text{c}}} = 0.005$ s
			$\Delta_x, \Delta_y \in [1, 2, 5, 10, 20, 50]$ m
			$\Delta_{\omega^{\text{e}}}, \Delta_{\omega^{\text{c}}} \in [1, 5, 10, 20, 50, 100]$ Hz
				add day id
				find drive mode
				add to grid
					`getMaskSize(Tuple` τ`):`
					`return 7, 2, 2`
					`areNeighb(Tuple` τ_1`, Tuple` τ_2`):`
					`return counterDist(`τ_1`,`τ_2`)`$\leq$`1)`

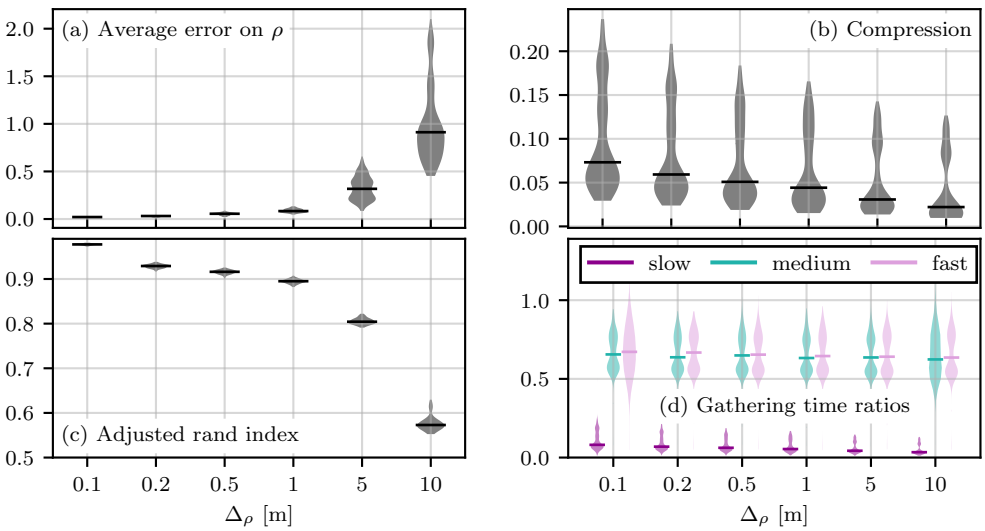

Figure 8: Q_1: (a) - (c) Compression and clustering statistics for various Δ_ρ; (d) gathering time ratio for various Δ_ρ and different network speeds.

maximum error (this is true for small Δ_ρ too, although harder to appreciate in the graph). The compression as a ratio of compressed vs. raw file size in (b) shows that LiDAR data can already for a maximum error $\Delta_\rho = 0.1$ m be compressed below (median) 10 % 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 4.1). For increasing maximum error, the compression ratio decreases only slightly, which indicates that almost maximum compression is reached early. The long tails towards lower compression hint at single files with data that is harder to compress than the average. For $\Delta_\rho = 1$ m, the data transmitted in this use case (10 rotations of the LiDAR, which are completed in 2 s) is around 750 KB. Transmitting this live is possible with network speeds of 3G or larger. The comparison of the resulting clusters 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 resulting clusters between the same rotations, not between the sets of ten rotations). One observes that for increasing Δ_ρ the similarity between the clusters of approximated and baseline data decreases. However, for $\Delta_\rho = 0.1$ m, the median compression ratio is already below

0.10, while the median adjusted rand index is larger than 0.95, indicating that a large compression can be achieved without a large loss of precision in the clustering.

Figure 8 (d) 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 ratio that also decreases only slightly for larger Δ_ρ), for all network speeds data gathering times are reduced to between 60 % for fast networks down to less than 15 % for slow networks.

5.4.2. Q_2 1-Vehicle 1-Day

In this use case, the analyst requests the GPS data of a single day from one vehicle in order 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 data is stored on-vehicle as a stream of tuples $\langle t, x, y\rangle$ with the timestamp as the actual measurement time and the x, y attributes being the coordinates in meters.

The query Q_2 for this use case is described in detail in Table 1. 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 provided to Lisco structured as a 2D matrix is sketched in Figure 9 (the colored 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(`τ`)` 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 `areNeighb(`τ_1`,`τ_2`)` on the basis of the Euclidean distance between the x, y-coordinates of the two tuples.

The compression statistics are given in Figure 10 (a), (b). We choose in this case a fixed error $\Delta_t = 1$ s for the 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 10 (a) shows the average error on both coordinates as a function of the maximum errors, with the average errors being roughly one-third of the allowed maximum error. Figure 10 (b) shows the total compression achieved for each maximum error: assuming a measurement uncertainty for GPS data on the order of few meters, maximum compression errors of less than ten

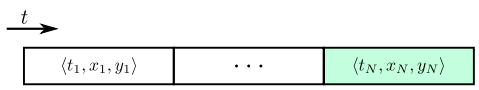

Figure 9: Q_2: Sketch of data structure produced by q_{pre}.

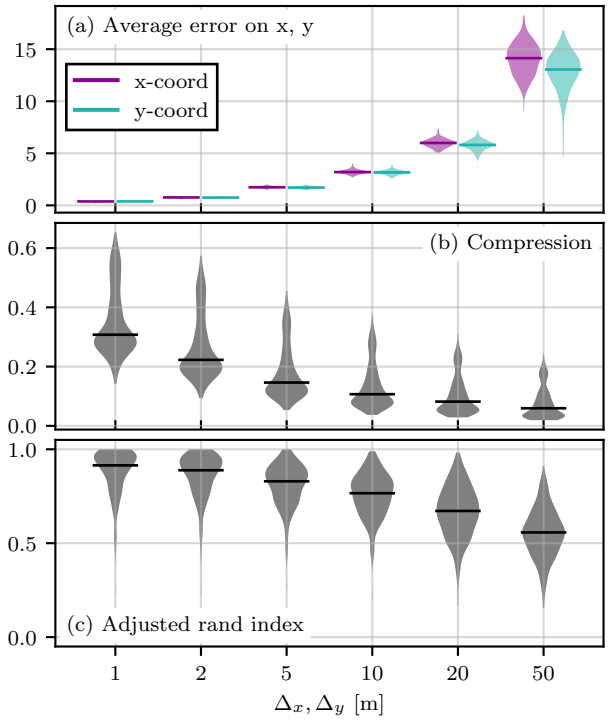

Figure 10: Q_2: (a), (b) Compression statistics; (c): Adjusted rand index.

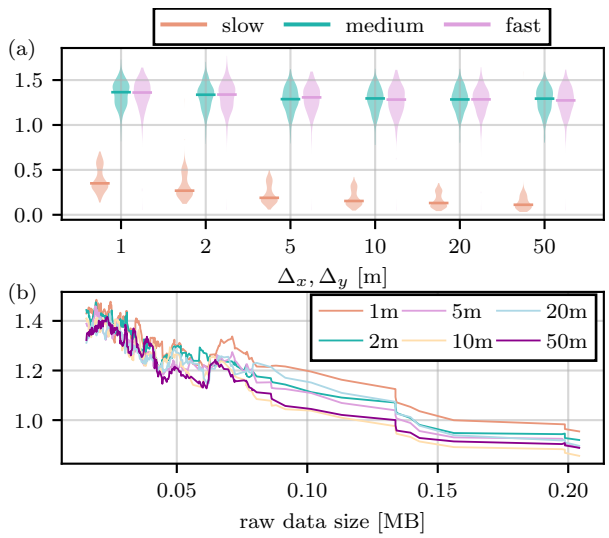

Figure 11: Q_2: Gathering time ratios for various (a) maximal errors for different network speeds and (b) raw data sizes (rolling average over 13 values, different colors are used for distinct values of Δ_x, Δ_y) for a medium speed network.

meters may be assumed to be small. Still, these result in compression ratios that can be lower than 0.2. This may be explained with straight roads, resulting in long, linear segments in the GPS data, as well as regularity of the timestamps. The violin plots' long upward tails hint at individual files with lower compressibility. The results for the comparison of the resulting clusters from approximated and raw data are shown in Figure 10 (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 11 (a). The median is around 1.3 for faster networks and does not decrease for increasing compression. This shows that DRIVEN in this use case is only beneficial for a slow network. More insight is gained from Figure 11 (b), showing the gathering time ratios as a function of the baseline data size for a medium speed network. For small data sizes, the additional time overhead of the compression and decompression procedure increases the gathering duration over directly transmitting the raw sample. For larger sample sizes, the gathering time ratio approaches 1 for all 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 we further show in the remainder (we stress nonetheless that our evaluation setup favors raw data gathering, as explained in Section 5.3).

5.4.3. Q_3 1-Vehicle 14-Day

In this use case, the analyst requests the GPS data from one specific vehicle from the last 14 days, to possibly identify routes that one vehicle follows regularly. The query Q_3, described in the query overview in Table 1, differs from Q_2 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 12 (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" (14 ensures that all days are encompassed), for tuples within a Euclidean distance of 150 m.

The compression statistics may be found in Figure 13 (a), (b) and are similar to those seen in Figure 10 (a), (b), as 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 (0.093 ± 0.081) s. The evaluation of clustering qualities is shown in Figure 13 (c), also with similar results to the previous use case. The addition of the attribute "days" for Lisco seemingly has only a small

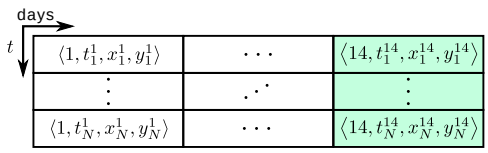

Figure 12: Q_3: Sketch of data structure produced by q_{pre}.

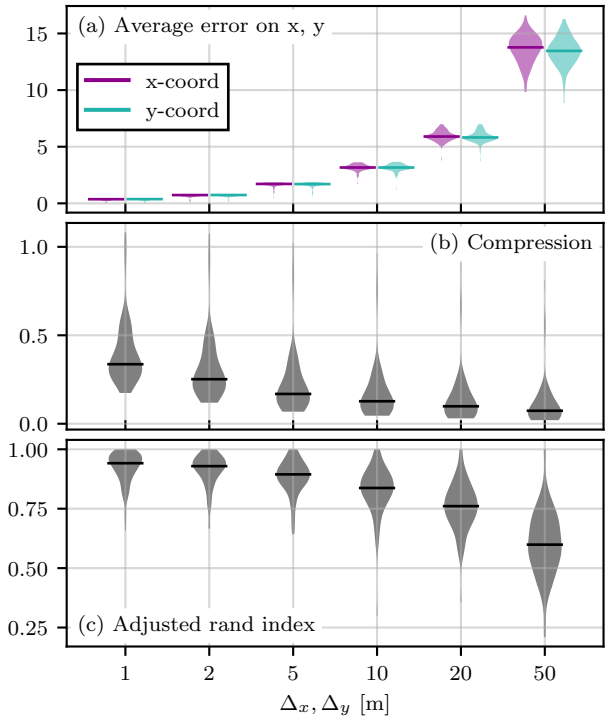

Figure 13: Q_3: (a), (b) Compression statistics; (c) Adjusted rand index.

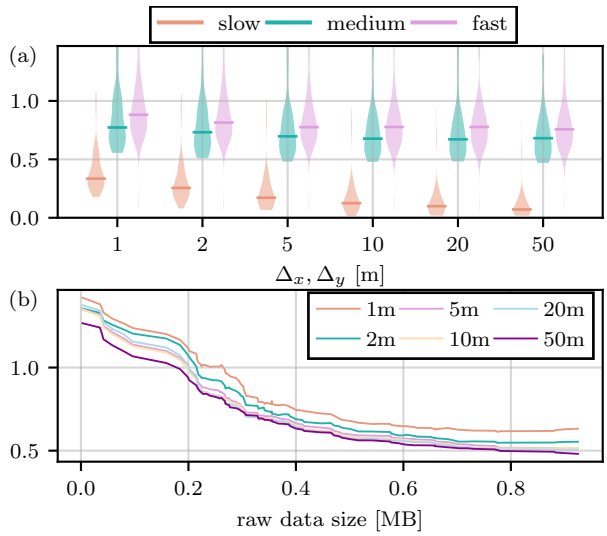

Figure 14: Q_3: Gathering time ratios for various (a) maximal errors for different network speeds and (b) raw data sizes (rolling average over 13 values, different colors are used for distinct values of Δ_x, Δ_y) for a medium speed network.

influence, suggesting that in the majority of samples there is no significant number of inter-day clusters.

Figure 14 (a) shows the measured gathering time ratios. The median gathering time ratios are below 0.35 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 14 (b) (for medium network speeds), this is due

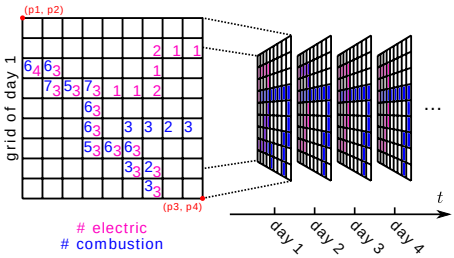

Figure 15: Q_4: Sketch of data structure produced by q_{pre}.

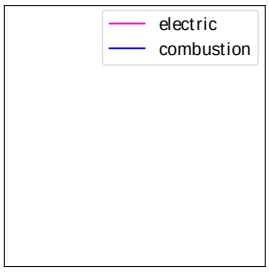

Figure 16: Q_4: Example of one grid (instead of showing the number of vehicles per drive mode and cell, only colors indicate the cell occupation).

to samples with raw data size smaller than 200 KB (at medium network speeds). For samples larger than 200 KB, gathering time ratios for all values of Δ_x, Δ_y are smaller than 1.

5.4.4. Q_4 Car usage grids

In this use case, the analyst wants to investigate if a fleet of hybrid cars uses the same drive mode (electric/traditional) on the same routes at similar times of the day. To perform this query, the analyst requests GPS data as well as the combustion engine and electric rear axle engine (ERAD) RPMs (rotation per minute) time series, requiring three different time series from three different sensors, for one week from 20 hybrid cars. The three time series in tuple notation are $\langle t^{\text{G}}, x, y\rangle$ for GPS (physical timestamp [s], x-coordinate [m] and y-coordinate [m]), $\langle t^{\text{e}}, \omega^{\text{e}}\rangle$ for the ERAD RPM (physical timestamp [s], RPM [Hz]) and $\langle t^{\text{c}}, \omega^{\text{c}}\rangle$ for the combustion engine RPM (physical timestamp [s], RPM [Hz]). Using this data, a map is created for each day, and clusters of identical drive mode (electric/combustion engine use) between different days and different locations on the map are created to identify routes for which a certain drive mode is preferred.

Over a rectangular geographic grid of 150×150 cells, the GPS trace of a car V_i during each day of the requested week is discretized. For each cell, characterized by the time period T during which V_i was present within the cell's boundaries, the combustion and electric engine RPMs during T are regarded and a decision is taken whether the car was in combustion or electric mode during T. Each cell contains a counter for each mode, and if V_i is found to be in a certain mode while in that cell then the corresponding

#	$[\Delta_x = \Delta_y, \Delta_{\omega^{\text{e}}}, \Delta_{\omega^{\text{c}}}]$
1	[1 m, 1 Hz, 1 Hz]
2	[2 m, 5 Hz, 5 Hz]
3	[5 m, 10 Hz, 10 Hz]
4	[10 m, 20 Hz, 20 Hz]
5	[20 m, 50 Hz, 50 Hz]
6	[50 m, 100 Hz, 100 Hz]

Table 2: Q_4: Maximum-error sets used. As three different time series are requested, three different error parameters are given (for GPS, $\Delta_x = \Delta_y$).

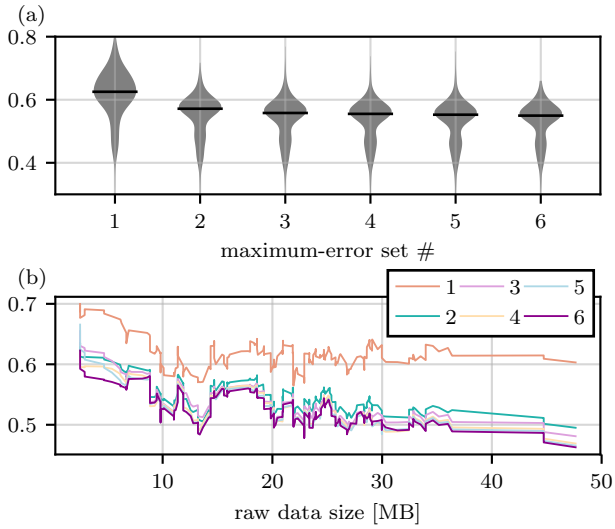

Figure 17: Q_4: Gathering time ratios for (a) a very fast network speed and (b) for various raw data sizes (rolling average over 13 values, different colors are used for different maximum-error sets) for a very fast network.

mode counter is increased. For each day, each V_i can only contribute to each cell's counter once. This is repeated for all $V_i, i \in \{1, \ldots, 20\}$, such that a map is created for each day of the week containing the cells visited (including drive mode) by all vehicles.

This pre-processing task is performed by q_{pre}, and a sketch of the data structured as a 3D matrix that is passed to Lisco is visualized in Figure 15: The coordinates (p1,p2) and (p3,p4) are located at the corners of the geographical grid (which in this sketch is a 9×9 grid). The first dimension of the 3D matrix is time (day of the week), the other two are geographic x, y-coordinates. A concrete example grid for one day is shown in Figure 16 (for visibility, the counters for each cell are represented with only a color marker).

The query Q_4 is formally described in the query overview in Table 1. Two grid cells, represented by tuples τ_1, τ_2, become clustered in accordance with `counterDist`(τ_1, τ_2) if the difference in both cells' electric or both cells' combustion mode counter is smaller than or equal to 1.

The time channels $t^{\text{G}}, t^{\text{e}}, t^{\text{c}}$ (for GPS, electric engine RPM and combustion engine RPM) are compressed with $\Delta_{t^{\text{G}}} = 1$ s; $\Delta_{t^{\text{e}}} = \Delta_{t^{\text{c}}} = 0.005$ Hz, resulting in average

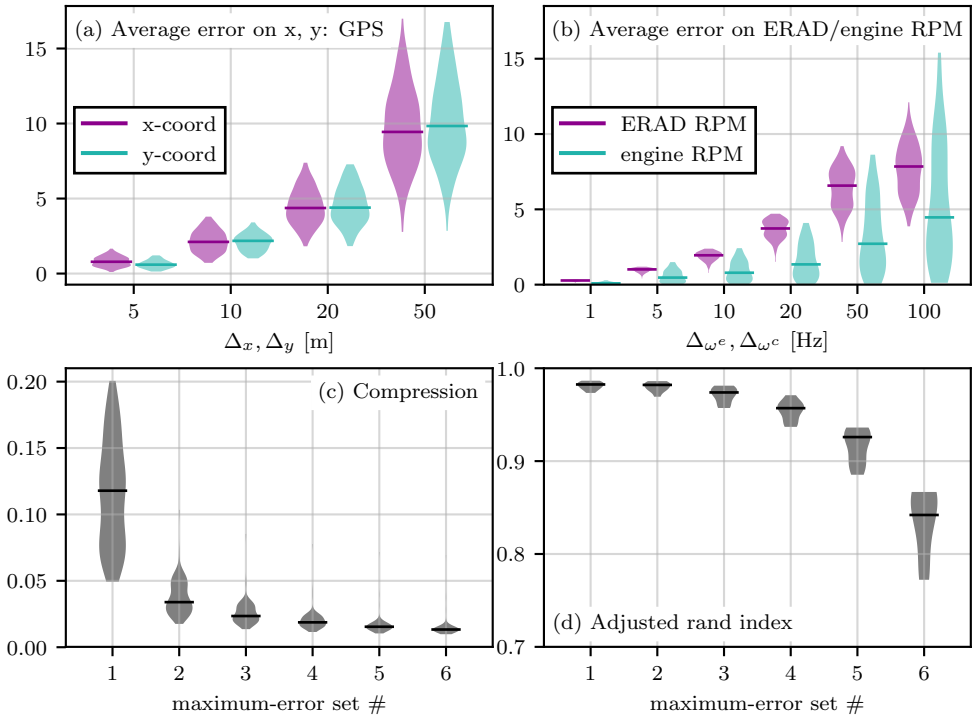

Figure 18: Q_4: (a) Average error on the x- and y-coordinate for several values of the maximum compression errors Δ_x, Δ_y for GPS data; (b) average error on the ERAD and engine RPM for several values of the maximum compression errors $\Delta_{\omega^e}, \Delta_{\omega^c}$; (c) compression statistics for different maximum-error sets (see Table 2); (d) adjusted rand index.

errors of (0.49 ± 0.06) s, $(142 \pm 72)\,\mu$s and $(921 \pm 161)\,\mu$s, respectively. The remaining channels are compressed for six sets of maximum errors shown in Table 2.

We assume for the evaluation that there is no change in network and analysis center performance for gathering the data from 20 vehicles at once, and thus simulate the query on one vehicle only (i.e., utilizing one ODROID as in the other use cases).

The compression and clustering statistics for this use case are shown in Figure 18: (a) is the average error on the x, y-coordinate of the GPS time series for different values of $\Delta_x = \Delta_y$. The average errors for the first two error sets are not displayed, due to the low geospatial precision of the GPS time series the average error on the GPS coordinates is on the order of 10^{-5} m for $\Delta_x = \Delta_y \in \{1\text{ m}, 2\text{ m}\}$. (b) is the average error on both the ERAD and the combustion engine RPM, which are roughly one order of magnitude smaller than the allowed maximum errors $\Delta_{\omega^e}, \Delta_{\omega^c}$. (c) shows that already for the maximum-error set #1 a median compression of 12 % can be achieved, down to 2 % for #6. This is explained by long stretches of inactivity of either the ERAD or the combustion engine, resulting in long stretches of constant zero readings in their respective time series. These stretches can be compressed well with PLA. The adjusted rand indices in (d) remain in the median above 0.9 until maximum-error set #6, indicating that the analysis accuracy in this use case is quite robust towards compression.

Gathering time ratios for a very fast network are shown in Figure 17 (a) for the six different maximum-error sets. For the smallest individual maximum errors at maximum-error set #1, the gathering time ratio is below 0.65, and for increasing individual maximum errors the gathering time ratio decreases slightly more to 0.55, but is almost constant. This may be due to the almost-constant compression for higher maximum-error sets, see Figure 18 (c). Figure 17 (b) shows the gathering time ratios for the same very fast network speed for various raw data sizes. For all maximum-error sets, the gathering time ratio tends to decrease for increasing raw data size. The noisy behavior of the curves, which is almost identical for each maximum-error set, may hint at individual files that are harder or easier to compress than other files of similar raw data size.

5.5. Compression evaluation

To gauge the performance of our PLA compression technique, we compare it with the `DEFLATE` compression algorithm used for ZIP compression. We choose ZIP because of its general-purpose nature, widespread use and lossless compression. Here, we show a comparison for the data used in Q_1 (LiDAR) and Q_2 (GPS). In our experiments, we zip for each separate channel n_{zip} consecutive points (thus $n_{zip} \cdot \text{size}(float)$ bytes) and take the average over all channels per file. The results are plotted in Figure 19, with "×" marking the compression achieved with PLA plotted in n_{zip}'s column corresponding to the average segment length obtained through DRIVEN (be reminded that the segment length with our PLA method varies and

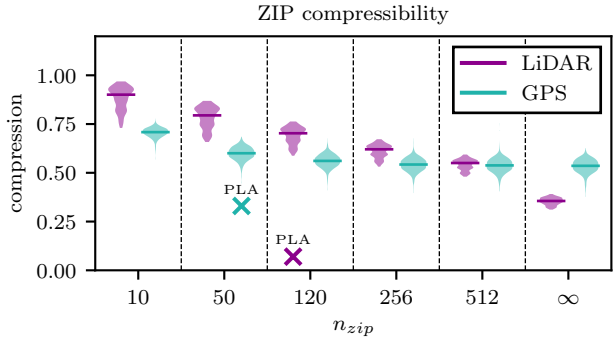

Figure 19: Compression ratios using ZIP for varying segment lengths n_{zip}. "×" marks the compression achieved with PLA for equivalent *average* n for smallest maximum errors (almost lossless).

depends on the underlying data; only the maximum segment length is specified and set to 256 points). We set here the maximum tolerated errors to minimal-loss values, *i.e.* $\Delta_\rho = 0.01$ m for LiDAR, and $\Delta_x = \Delta_y = 1$ m for GPS, cf. Figures 8 (a), 10 (a). For comparable segment lengths, the ZIP representation is 2-10 times larger, indicating the advantages of lossy, piecewise linear compression for this type of data and scenario. Even when zipping all the available data ($n_{zip} = \infty$), the gap remains stark, which further hints at the validity of our PLA implementation. Moreover, allowing for larger segment lengths would limit the usefulness in a live data gathering scenario, as shown in the following subsection.

5.6. Logical latency

Lastly, we evaluate DRIVEN by studying the logical latency observed when compressing data from the *Ford Campus* and *GeoLife* datasets. By doing this, we can thus estimate the live gathering time incurred when performing PLA compression on live data (which can be approximated by the average segment length multiplied with the sampling period of data being clustered, since this is orders of magnitude larger than emission time, as well as transmission and reconstruction delays).

Notice that we do not present results for the *Volvo* dataset in this case, since the different order of magnitude of sampling period between GPS and ERAD/engine data (seconds versus milliseconds) results in GPS data (already discussed for the *GeoLife* dataset) being the one dominating the live gathering time delay for compression of live sensed data. More concretely, given the GPS' data sampling period of 5 s and the ERAD/engine RPMs sampling period of 40 ms, the shortest possible segment of GPS' data (approximating 3 points) results in an average live gathering time of 10 seconds while the longest possible segment of ERAD/engine RPMs data (approximating 256 points) results in an average live gathering time of approximately 5 seconds.

Based on our description of logical latency (Section 2.4), the logical latency incurred by DRIVEN can be modeled as follows. For the stream of values y, the logical latency

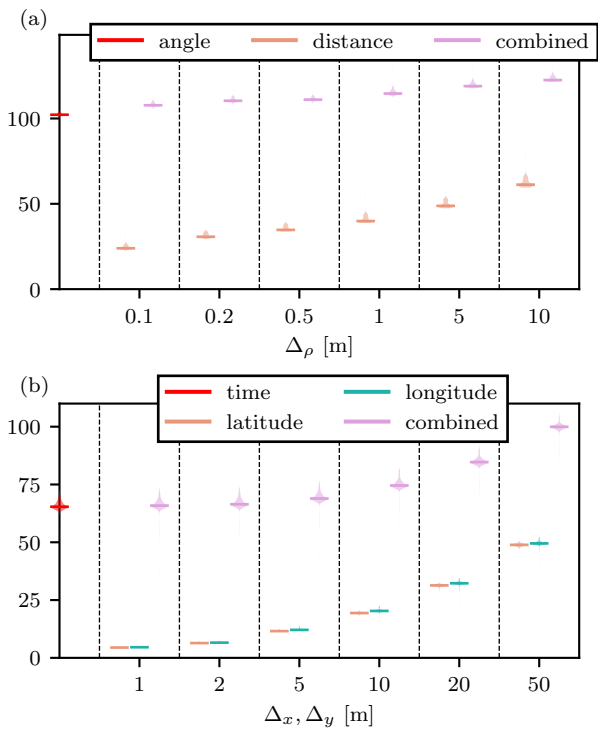

Figure 20: Distribution of logical latencies in number of tuples for (a) the LiDAR (*Ford Campus*) and (b) the Beijing GPS (*GeoLife*) dataset as a function of the respective maximum errors. The logical latency for the angle/time coordinate is displayed over the y-axis (red), as their corresponding maximum errors are constant over each of the two datasets.

is obtained as the difference $j - i$ where $\langle j, y_j \rangle$ is the last tuple read before the compressor sends information that triggers the i-th tuple's reconstruction on the decompressor side; two situations are then possible: either processing $\langle j, y_j \rangle$ triggers the emission of a line segment $\langle n, a, b \rangle$ and in this case $\langle i, y'_i \rangle$, with $j - n - 1 \leq i \leq j - 1$, is reconstructed among $n - 1$ other tuples using the segment's information, or $\langle j, y_j \rangle$ triggers the emission of a singleton $\langle 1, y'_i \rangle$ where then $i = j - 2$ (since 3 tuples are read before emitting a singleton, the logical latency is always 2 in this case). Logical latencies are bounded by the maximum segment length (this occurs for the first tuple on a maximum-length segment), and the average logical latency corresponds (when omitting singletons) to half the average segment length. When no compression is performed, logical latencies are 0. For an input tuple $\langle y_i^0, \ldots, y_i^k \rangle$ (which is split into $\langle i, y_i^0 \rangle, \ldots, \langle i, y_i^k \rangle$), the *combined* logical latency is the maximum logical latency of the individual tuples $\langle i, y_i^0 \rangle, \ldots, \langle i, y_i^k \rangle$, as the original tuple can only be reconstructed as soon as all its attributes have been individually reconstructed. The compression scheme used in DRIVEN, i.e., the PLA construction method *Linear* coupled with a streaming-based protocol, has been shown in [11] to produce logical latencies one to two orders of magnitude smaller than other state-of-the-art PLA construction algorithms.

In addition to calculating the average logical laten-

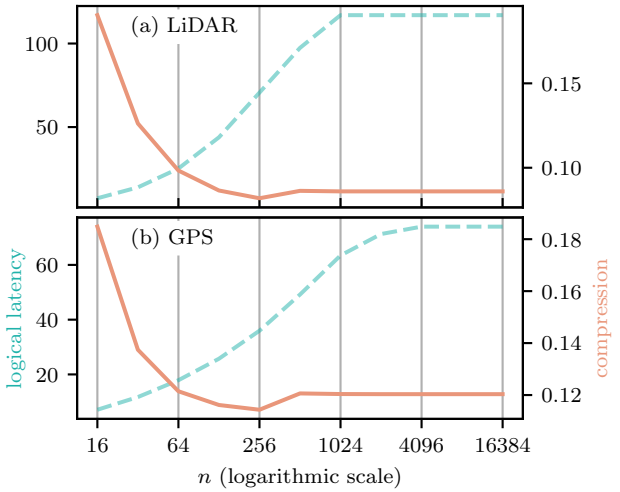

Figure 21: Average logical latency (over all channels) and compression for different maximum segment lengths n ($n = 256$ is the maximum segment length chosen in this evaluation).

cies of the *Ford Campus* and *GeoLife* datasets, we further study to which extent the logical latencies can be reduced by reducing one parameter of our PLA compression scheme: the maximum segment length (set to 256 tuples in our evaluation).

Figure 20 shows the individual and combined average logical latencies as violin plots for different compressions measured over the (a) LiDAR (*Ford Campus*) and (b) Beijing GPS (*GeoLife*) datasets. The red violin plot on the left of both (a) and (b) displays the distribution of average logical latencies for the (logical) timestamps. As the (logical) timestamps are only compressed with a constant maximum error ($\Delta_\alpha = 0.0015$ rad for LiDAR, $\Delta_t = 1$ s for GPS), only one violin plot is shown per dataset for the (logical) time channel.

The span of the violin plots for different compressions is small for both (a) and (b), meaning that the logical latency depends more on the type of data than on the specific file of a certain datatype. Second, the combined logical latencies for both datasets are dominated by the latency of the timestamp channel. As this channel is quite linear, and thus easily compressible, we expect the longest segments for the timestamp channel and thus a large logical latency. Concretely, this means that other channels have to wait for the time channel to be decompressed before an original tuple can be reconstructed.

For the *GeoLife* GPS dataset, used in $Q_2 - Q_3$, the average difference between two timestamps in the original data is 5 s. Thus, neglecting transmission time, it takes on the order of 100×5 s $= 500$ s to reconstruct an original input tuple for GPS data with the aforementioned sampling rate.

For the *Ford Campus* LiDAR dataset, used in Q_1, there are 20000 readings of the logical timestamp channel α per second (see Section 4.1). Combined with a logical latency on the order of 100 tuples, this results in an average reconstruction time of at least 0.005 s.

In the use cases investigated in this evaluation, those latencies carry little significance, as historic data is gathered. In these cases, where the data is replayed at much higher than live speeds, the transmission duration is dominant for the data gathering time. When live data is requested, however, the logical latency can lead to significant delays, but this is inevitable for PLA compression. The logical latency is strictly linked to the segment lengths of the PLA and can be reduced either via a smaller maximum segment length or via a smaller maximum error threshold, resulting in a PLA with shorter segments. For the combined logical latency, these changes will have greater effects if applied on the time channel, which due to its compressibility is dominant overall in our evaluation.

Figure 21 shows the logical latency and the compression for the LiDAR and GPS dataset (each averaged over all contained channels) for different values of the maximum segment length using constant error bounds (LiDAR: $\Delta_\rho = 1$ m, $\Delta_\alpha = 0.0015$ rad; GPS: $\Delta_x = \Delta_y = 10$ m, $\Delta_t = 1$ s). This figure motivates the choice of $n = 256$ for the maximum segment length, as for this value the compression is maximal. For higher n, the compression does not increase further, as the maximal length of segments is an inherent characteristic of the data used (for a given maximum compressor error). The compression even becomes slightly worse as more data must be allocated for transmitting the segment length (i.e., two bytes are needed for $n > 256$). The logical latency increases with increasing segment length, and becomes stationary as the maximum inherent segment length is reached.
If lower logical latency is desired for a query, this figure shows that in turn lower compression will be achieved. However, depending on the region within the plots, large gains in logical latency can be achieved with comparatively smaller losses in compression, especially noticeable in the $n = 64 \ldots 256$ region.

5.7. Summary of evaluation results

The evaluation shows that, compared to the baseline, DRIVEN can maintain an adjusted rand index greater than or equal to 0.9 for the clustering of approximated data while compressing the raw data to less than 5 %, 20 % and 2.5 % for LiDAR, GPS and a combination of GPS and other vehicular signals, respectively - outperforming lossless ZIP compression by factors of $2 - 10$ for LiDAR and GPS. Also, DRIVEN affords speed ups exceeding $\times 10$ in data gathering times for large-enough amounts of data (at least 200 KB per sensor in our setup). Still, the logical latency inherent to PLA must be considered when working with live data. This logical latency can also be significantly decreased using an appropriate maximum segment length.

6. Related work

Clustering, as a core problem in data mining, has been extensively studied in the last decades (see e.g. the survey [22] 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 [15]-style). Other features that can distinguish existing clustering algorithms include their sensitivity to outliers (not interesting data that should be ignored in some applications), their ability to work with any distance function or the required level of parametrization.

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 [36], 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 windows, or assigning decreasing weights to older data) of points [36]. In [39], the authors design a fully streaming clustering algorithm (as the streaming version of a recently proposed clustering algorithm [34]), computing the exact same clustering of its batch-based counterpart. 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 [33]. 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 [27]. In our work, the objects being clustered are not the time series but the input points themselves and PLA is used to gather data efficiently (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. [25, 13, 43, 28]) 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 [25] 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, thus reducing energy consumption. The authors note that the abstraction size is crucial in wireless sensor networks, thus motivating the study of 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 [16], the authors devise a PLA algorithm with a small memory footprint and average instruction count for resource-constrained wireless sensor nodes. They use a best-line approximation (similarly to us) but with no intercept (so more segments are produced), and the approximation error is bounded by segment instead of by point.

In an earlier work [21], a preliminary demonstration of DRIVEN can be found. In contrast to that work, we here propose a new algorithmic implementation of our multi-channels PLA compression that enables exact error guarantees through completely independent processing of time and other channels of a sensor and additionally resulting in better compression ratios. Moreover, we provide a more extensive evaluation that extends previous results to larger data volumes and higher network speeds. Furthermore, the present work investigates the effects and limitations of applying DRIVEN in live data gathering scenarios and introduces an additional experiment advocating for the benefits of using PLA versus standard lossless ZIP compression.

7. Conclusion and future work

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 based on an analyst's queries. Information retrieval is achieved using PLA for compressing the input stream in a streaming fashion. Once uncompressed, the approximated stream is fed to a distance-based streaming clustering algorithm. Both the approximation and the clustering are parameterizable for allowing different applications to be run by the framework. Through thorough experimentation using real-world GPS and LiDAR data as well as other vehicular signals, we show the versatility of the framework in being able to answer different types of queries of historical data involving various

clustering requests for vehicular networks, and also show 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 $5 - 35$ % of its raw size, reducing drastically the duration of the gathering phase for large volumes of data, with only a small accuracy loss on the clustering.
We furthermore have evaluated the application of DRIVEN in a live-data scenario and studied the additional (and inherent) latency from the PLA compression, which can nevertheless be reduced for a predictable loss in compression, and gauged the compression capabilities of our PLA implementation using ZIP compression.

The idea behind DRIVEN is to leverage the cumulative power of edge devices to improve data analysis applications that are traditionally deployed entirely at data centers and that require all input raw data to be first gathered centrally. The solution we propose in this paper can be enhanced along different dimensions in future work. First, other techniques (e.g. Symbolic Aggregate Approximation - SAX) can be leveraged at the vehicles to reduce the amount of data to be forwarded, and it is thus interesting to study how such techniques would perform along with the performance metrics we take into account in this paper. Second, given that many other machine learning techniques are commonly used in cyber-physical systems' data analysis, their integration (and possible porting to the streaming paradigm) within DRIVEN is also of interest. Finally, it is also important to notice that the computational power of each edge device (be it a vehicle or something else) can be used in conjunction with the data center's one in several ways. While we study a solution that leverages the edge device's computational power to approximate and compress raw data, we also believe that distribution of machine learning tasks (e.g. learning over different subsets) is a way to leverage such computational power that is worth exploring and can enable efficient and effective solutions.

Acknowledgments. 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] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (Toronto, Canada) *(VLDB '04)*. VLDB Endowment, 480–491. `http://dl.acm.org/citation.cfm?id=1316689.1316732`

[2] Brian Babcock, Shivnath Babu, Rajeev Motwani, and Mayur Datar. 2003. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (San Diego, California) *(SIGMOD '03)*. ACM, New York, NY, USA, 253–264. `https://doi.org/10.1145/872757.872789`

[3] B. Babcock, M. Datar, and R. Motwani. 2004. Load shedding for aggregation queries over data streams. In *Proceedings. 20th International Conference on Data Engineering*. IEEE, 350–361. `https://doi.org/10.1109/ICDE.2004.1320010`

[4] Magdalena Balazinska, Hari Balakrishnan, Samuel Madden, and Michael Stonebraker. 2005. Fault-tolerance in the Borealis Distributed Stream Processing System. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (Baltimore, Maryland) *(SIGMOD '05)*. ACM, New York, NY, USA, 13–24. `https://doi.org/10.1145/1066157.1066160`

[5] E. Berlin and K. Van Laerhoven. 2010. An on-line piecewise linear approximation technique for wireless sensor networks. In *IEEE Local Computer Network Conference*. 905–912. `https://doi.org/10.1109/LCN.2010.5735832`

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

[7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. 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] Riccardo Coppola and Maurizio Morisio. 2016. Connected car: technologies, issues, future trends. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 46.

[9] Stefania Costache, Vincenzo Gulisano, and Marina Papatriantafilou. 2016. Understanding the data-processing challenges in Intelligent Vehicular Systems. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*. IEEE, 611–618.

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

[11] Romaric Duvignau, Vincenzo Gulisano, Marina Papatriantafilou, and Vladimir Savic. 2019. Streaming Piecewise Linear Approximation for Efficient Data Management in Edge Computing. In *34th ACM/SIGAPP Symposium On Applied Computing SAC'19*. to appear.

[12] Romaric Duvignau, Bastian Havers, Vincenzo Gulisano, and Marina Papatriantafilou. 2019. Querying Large Vehicular Networks: How to Balance On-Board Workload and Queries Response Time?. In *The 22nd IEEE Intelligent Transportation Systems Conference (ITSC2019)*. IEEE. to appear.

[13] Hazem Elmeleegy, Ahmed K Elmagarmid, Emmanuel Cecchet, Walid G Aref, and Willy Zwaenepoel. 2009. Online piece-wise linear approximation of numerical streams with precision guarantees. *Proceedings of the VLDB Endowment* 2, 1 (2009), 145–156.

[14] Marius A Eriksen. 2005. Trickle: A Userland Bandwidth Shaper for UNIX-like Systems.. In *USENIX Annual Technical Conference, FREENIX Track*. 61–70.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*. 226–231.

[16] Florian Grützmacher, Benjamin Beichler, Albert Hein, Thomas Kirste, and Christian Haubelt. 2018. Time and Memory Efficient Online Piecewise Linear Approximation of Sensor Signals. *Sensors* 18, 6 (2018), 1672.

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

[18] Vincenzo Gulisano, Yiannis Nikolakopoulos, Daniel Cederman, Marina Papatriantafilou, and Philippas Tsigas. 2017. Efficient Data Streaming Multiway Aggregation Through Concurrent Algorithmic Designs and New Abstract Data Types. *ACM Transactions on Parallel Computing (TOPC)* 4, 2, Article 11 (2017), 28 pages.

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

[20] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[21] Bastian Havers, Romaric Duvignau, Hannaneh Najdataei, Vincenzo Gulisano, Ashok Chaitanya Koppisetty, and Marina Papatriantafilou. 2019. DRIVEN: a framework for efficient Data Retrieval and clusterIng in VEhicular Networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1850–1861.

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

[23] Yuanzhen Ji, Hongjin Zhou, Zbigniew Jerzak, Anisoara Nica, Gregor Hackenbroich, and Christof Fetzer. 2015. Quality-Driven Continuous Query Execution over Out-of-Order Data Streams. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. ACM, New York, NY, USA, 889–894. `https://doi.org/10.1145/2723372.2735371`

[24] Evangelia Kalyvianaki, Marco Fiscato, Theodoros Salonidis, and Peter Pietzuch. 2016. THEMIS: Fairness in Federated Stream Processing Under Overload. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. ACM, New York, NY, USA, 541–553. `https://doi.org/10.1145/2882903.2882943`

[25] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2001. An online algorithm for segmenting time series. In *Proceedings of 2001 IEEE International Conference on Data Mining*. IEEE, 289–296.

[26] Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, Philippas Tsigas, and Yiannis Nikolakopoulos. 2018. MAD-C: Multi-stage Approximate Distributed Cluster-Combining for Obstacle Detection and Localization. In *European Conference on Parallel Processing*. Springer, 312–324.

[27] Jessica Lin, Michai Vlachos, Eamonn Keogh, Dimitrios Gunopulos, Jianwei Liu, Shoujian Yu, and Jiajin Le. 2005. 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*. Springer, 333–342.

[28] Ge Luo, Ke Yi, Siu-Wing Cheng, Zhenguo Li, Wei Fan, Cheng He, and Yadong Mu. 2015. Piecewise linear approximation of streaming time series data with max-error guarantees. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 173–184.

[29] Hannaneh Najdataei, Yiannis Nikolakopoulos, Vincenzo Gulisano, and Marina Papatriantafilou. 2018. Continuous and Parallel LiDAR Point-Cloud Clustering. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 671–684.

[30] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. 2014. MCEP: a mobility-aware complex event processing system. *ACM Transactions on Internet Technology (TOIT)* 14, 1 (2014), 6.

[31] Dimitris Palyvos-Giannas, Vincenzo Gulisano, and Marina Papatriantafilou. 2018. GeneaLog: Fine-Grained Data Streaming Provenance at the Edge. In *Proceedings of the 19th International Middleware Conference*. ACM, 227–238.

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

[33] Chotirat Ann Ralanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. 2005. Mining time series data. In *Data mining and knowledge discovery handbook*. Springer, 1069–1103.

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

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

[36] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, Andre CPLF De Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)* 46, 1 (2013), 13.

[37] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. 2005. The 8 requirements of real-time stream processing. *ACM Sigmod Record* 34, 4 (2005), 42–47.

[38] Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. 2003. Load shedding in a data stream manager. In *Proceedings of the 29th international Conference on Very Large Data Bases-Volume 29*. VLDB Endowment, 309–320.

[39] Liudmila Ulanova, Nurjahan Begum, Mohammad Shokoohi-Yekta, and Eamonn Keogh. 2016. Clustering in the Face of Fast Changing Streams. In *Proceedings of 2016 SIAM International Conference on Data Mining*. SIAM, 1–9.

[40] Joris van Rooij, Vincenzo Gulisano, and Marina Papatriantafilou. 2018. 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*. ACM, 171–182.

[41] Silke Wagner and Dorothea Wagner. 2007. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe.

[42] Ingo Wald and Vlastimil Havran. 2006. 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*. IEEE, 61–69.

[43] Qing Xie, Chaoyi Pang, Xiaofang Zhou, Xiangliang Zhang, and Ke Deng. 2014. Maximum error-bounded Piecewise Linear Representation for online stream approximation. *The VLDB Journal* 23, 6 (2014), 915–937.

[44] Saleh Yousefi, Mahmoud Siadat Mousavi, and Mahmood Fathy. 2006. Vehicular ad hoc networks (VANETs): challenges and perspectives. In *Proceedings of 2006 6th International Conference on ITS Telecommunications*. IEEE, 761–766.

[45] Nikos Zacheilas, Vana Kalogeraki, Yiannis Nikolakopoulos, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2017. Maximizing determinism in stream processing under latency constraints. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM, 112–123.

[46] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *Proceedings of the 10th International Conference on Ubiquitous Computing*. ACM, 312–321.

[47] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.

[48] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th International Conference on World Wide Web*. ACM, 791–800.

[49] Jiazhen Zhou, Rose Qingyang Hu, and Yi Qian. 2012. Scalable distributed communication architectures to support advanced metering infrastructure in smart grid. *IEEE Transactions on Parallel and Distributed Systems* 23, 9 (2012), 1632–1642.