Thesis for The Degree of Licentiate of Engineering

# Distributed and Communication-Efficient Continuous Data Processing in Vehicular Cyber-Physical Systems

Bastian Havers-Zulka



Division of Networks and Systems
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2020

**Distributed and Communication-Efficient Continuous Data Processing in Vehicular Cyber-Physical Systems**

Bastian Havers-Zulka



Department of Computer Science & Engineering
Division of Networks and Systems
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
If printed: Printed by Chalmers Reproservice,
Gothenburg, Sweden 2020.

*Distributed and Communication-Efficient Continuous Data Processing in Vehicular Cyber-Physical Systems*
Bastian Havers-Zulka

Department of Computer Science & Engineering
Division of Networks and Systems
Chalmers University of Technology and Gothenburg University

# Abstract

Processing the data produced by modern connected vehicles is of increasing interest for vehicle manufacturers to gain knowledge and develop novel functions and applications for the future of mobility. Connected vehicles form Vehicular Cyber-Physical Systems (VCPSs) that continuously sense increasingly large data volumes from high-bandwidth sensors such as LiDARs (an array of laser-based distance sensors that create a 3D map of the surroundings). The straightforward attempt of gathering all raw data from a VCPS to a central location for analysis often fails due to limits imposed by the infrastructure on the communication and storage capacities. In this Licentiate thesis, I present the results from my research that investigates techniques aiming at reducing the data volumes that need to be transmitted from vehicles through online compression and adaptive selection of participating vehicles. As explained in this work, the key to reducing the communication volume is in pushing parts of the necessary processing onto the vehicles' on-board computers, thereby favorably leveraging the available distributed processing infrastructure in a VCPS. The findings highlight that existing analysis workflows can be sped up significantly while reducing their data volume footprint and incurring only modest accuracy decreases. At the same time, the adaptive selection of vehicles for analyses proves to provide a sufficiently large subset of vehicles that have compliant data for further analyses, while balancing the time needed for selection and the induced computational load.

**Keywords**

Vehicular Cyber-Physical Systems, Edge Computing, Data Streaming, Compression, Query Spreading

# Acknowledgments

I would like to sincerely thank my supervisors, colleagues and friends in both academia and industry for having accompanied me thus far. I want to thank especially Vincenzo Gulisano, for taking me on as PhD student, for supervising, aiding and guiding me, and for equally fruitful and vivid discussions and talks; Marina Papatriantafilou, for the continued sharing of experience, ideas and advice that has helped me reach this halfway point; and Ashok Chaitanya Koppisetty, for always supporting me, giving me a perspective and enabling my research to have an impact. I am especially indebted to my close collaborators and co-authors Romaric and Dimitris, and to all the great people at both Chalmers and Volvo Cars that I have the pleasure of working together and being friends with, that have made the first half of my PhD so enjoyable through inspiring collaboration and relaxing coffee breaks (back when physical presence was still a thing). Thank you also to everybody that made possible and participated in the projects OODIDA and AutoSPADA, that provided the frame for conducting my research.

Work would be less pleasant without life, and thus I want to also thank all my friends and family that indirectly or directly contributed to this thesis through their fellowship (or proof-reading). You're the salt of the earth! Lastly, and most importantly, I would like to thank my wonderful wife Linn, who is the reason that I now publish under a different name, and who has supported me through countless years of life and study.

Thanks, y'all!

Bastian Havers-Zulka
Gothenburg, October 15, 2020

This work and the enclosed publications have been supported by the Swedish Government Agency for Innovation Systems VINNOVA, proj. “Onboard/Offboard Distributed Data Analytics (OODIDA)” (grant DNR 2016-04260) and “Automotive Stream Processing and Distributed Analytics (AutoSPADA)” (grant DNR 2019-05884) in the funding program FFI: Strategic Vehicle Research and Innovation; 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).

# List of Publications

## Appended publications

This thesis is based on the following publications:

[A] **B. Havers**, R. Duvignau, H. Najdataei, V. Gulisano, M. Papatriantafilou and A. Koppisetty. "DRIVEN: A Framework For Efficient Data Retrieval And Clustering In Vehicular Networks",
*Future Generation Computer Systems, vol. 107, pp. 1-17, 2020.*

[B] R. Duvignau, **B. Havers**, V. Gulisano and M. Papatriantafilou. "Querying Large Vehicular Networks: How To Balance On-Board Workload And Queries Response Time?",
*Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC 2019), pp. 2604-2611, 2019.*

# Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] **B. Havers**, R. Duvignau, H. Najdataei, V. Gulisano, A. Koppisetty and M. Papatriantafilou. "DRIVEN: A Framework For Efficient Data Retrieval And Clustering In Vehicular Networks",
*Proceedings of the 2019 IEEE International Conference on Data Engineering (ICDE 2019), pp. 1850-1861, 2019.*

[b] V. Gulisano, D. Palyvos-Giannas, **B. Havers**, M. Papatriantafilou. "The role of event-time order in data streaming analysis",
*Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems (DEBS 2020), pp. 214-217, 2020.*

# Contents

# List of Figures

# Chapter 1:

# Overview

## 1.1 Mining Gold from Fleets of Vehicles

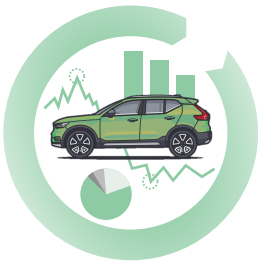
The amounts of data generated by sensors and logged by computers in our daily lives are exploding, with a projected doubling of the overall produced data amounts occurring every three years [2]. The term *Big Data* was coined to describe the resulting enormous, evolving datasets that require novel processing and storing paradigms, but enable the discovery of novel insights about areas ranging from stock market fluctuations to predicting when a car's brake pads will wear down. Due to its vast potential, Big Data is likened to a novel economic asset, "the new oil", or even raw gold [3]. One area in which the transformation towards Big Data will have a large impact on many people's lives is that of personal mobility. In short: how do modern vehicles play into the equation of Big Data and its associated novel value?

#### From Engine Temperature Gauges to LiDARs...

For most of their history, cars have employed only a handful of sensors, and these were discrete and independent (examples include mechanical vehicle speed or engine temperature gauges). Their purpose was to present information to the driver, who could then act based on this input - accelerate, brake, or check the engine. The late 1970s saw eventually the introduction of *connected* sensors and finally the first microcontroller-based control unit in the car: an engine control module to optimize combustion, taking autonomous decisions based on input from connected temperature and exhaust flow sensors [4]. Over time, the number of sensors in all areas of the vehicle has grown steeply, from around 20 in the early 2000s to more than 100 in commercial vehicles today [5]. When looking at the amount of data created in such a modern vehicle, the sheer number of these sensors alone pushes it into the range of several gigabytes per hour [6] - which is a data amount that

in the 1970s could barely fit storage systems the *size of a vehicle*. Today, the impending introduction of semi- and fully autonomous vehicles requires the addition of high-bandwidth sensors such as Radar, multiple stereo cameras and LiDARs (an array of laser-based distance sensors that create a 3D map of the surroundings) to the already complex sensor set carried by a vehicle. These new sensors generate orders of magnitude more data per time than tire-pressure sensors and simple GPS beacons [7], setting the pace for further growth of automotive data amounts. Already, cars have become producers of Big Automotive Data [8,9] - What can all this data be used for?

### ...and from Paper Questionnaires to Autonomous Vehicles



Traditional techniques of gathering insights about the vehicles, and the interaction of vehicle and driver, relied on experimental laboratory scenarios (e.g. *hardware-in-the-loop* [10]), customer clinics, specially-equipped test fleets and similar means; all suffering from various drawbacks such as cost, far-from-live setting and slow response. In contrast, by utilizing data collected automatically and continuously from large parts of the available fleet, the data pool is increased drastically in both size and depth, providing more data from more vehicles and from all real-world driving and usage scenarios. Potential insights are on the level of large-scale behavior, for example in online congestion monitoring, in enabling new cooperative behavior such as platooning and hazard warning systems, in gathering and collectively learning from environment data for the development of Autonomous Driving, in diagnostic applications such as Predictive Maintenance, and also in driver interaction analysis to cater to new driver requirements and preferences (see also [11,12]). With an estimated total value of 1.5 Trillion USD by 2030, car data and corresponding applications are set to mature into a major field of operations for vehicle manufacturers and fleet owners [13]. At the same time, car data is expected to also become essential for observing and managing traffic in increasingly urbanized communities, and for realizing the shared use of potentially autonomous vehicles on a large scale.

### Sharing the Load of Data Processing over the Fleet

While the amount of data generated by vehicles has steadily grown, vehicles have also become more connected to each other and to the internet [6]. Today, almost every new vehicle sold is a *connected car* [14], offering the capability of wirelessly exchanging data with other cars or data centers.

The internal processing of data and the aspect of communication are facilitated by increasingly capable on-board computers. The first simple control units have given way to more complex computers, controlling aspects such as smart lane-keeping assistants or emergency braking systems by processing the continuous input stream originating from the sensors - and a leap in processing capability is required for the transition to semi- and fully-autonomous drive, where large amounts of video and other many-dimensional data require analysis.

Equipped with sensors, wireless connectivity and processing capabilities, connected cars form the edge of a *Vehicular Cyber-Physical System* (VCPS), and

at said edge, vast amounts of data are produced in a continuous fashion. The edge is connected to the hubs of this network, central servers that are operated by the vehicle manufacturers. This setup can enable advanced features based on the utilization of the data produced in it. However, gaining insight from said data also entails manifold difficulties, spanning processing bottlenecks, costs, and especially the coping with large data volumes that need to be transmitted.

To investigate how the processing of data in this distributed setting can be facilitated, I introduce in this work some fundamental notions about VCPSs and present two research questions that surround the title of this thesis: *Distributed and Communication-Efficient Continuous Data Processing in Vehicular Cyber-Physical Systems.* After the Overview chapter, Chapter 2 and Chapter 3 present the main findings of this thesis in the form of two publications.

## 1.2 Vehicular Cyber-Physical Systems

This section gives a short introduction to VCPSs and outlines their defining characteristics in more detail, before highlighting the key challenges for data processing inside a VCPS.

### 1.2.1 Defining Characteristics

A VCPS is a specific type of Cyber-Physical System (CPS). With a plurality of definitions for a CPS (see the appendix of [15] for an extensive collection), one defining aspect is the existence of an interface between the physical world and computer systems that react based on data they receive via said interface. This interface is provided by sensors that measure physical parameters and translate them into machine-readable data, such as a LiDAR sensor that transforms its physical environment into a point cloud of coordinates and distance measurements. Furthermore, these sensors must be connected to the computer system for transmitting their sensed data. Examples for CPSs that fit these definitions are the network of sensors and processing units inside a single vehicle, or a factory with connected sensors that monitor production to continuously adapt the manufacturing process [16]. A VCPS is a specific type of CPS, in which the existence of some computational capacity close to the sensors is assumed - an *on-board computer* that can process the sensor data and/or store it on the on-board computer's disk. In this work, I furthermore assume that a central server exists that is wirelessly connected to the units of sensors and on-board computers via a two-way channel. The sensors are located at the outskirts of the system, physically separated from the central server, and constitute the *edge*. An example sketch of this VCPS is shown in Figure 1.1. In such a VCPS, many applications exist in which insights from data gathered at the edge are required at the central server. As an example application, the analyst in Figure 1.1 at the data center (where the central servers are found) could request to transmit LiDAR point cloud data from the edge to the data center for further processing (e.g. for creating 3D city maps) [17], or the analyst could task the vehicles to report whether they have passed a point of interest in the last day [1].

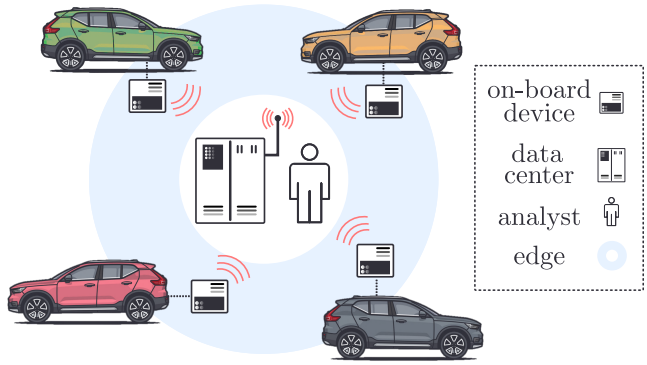



**Figure 1.1:** VCPS: A group of vehicles whose on-board devices form the *edge* of the network. Each such device is wirelessly connected to a data center. The vehicles are constantly gathering data with their on-board devices while they are being driven. At the data center, the analyst can deploy analyses over said data.

### 1.2.2 Key Challenges for Data Processing

With a first overview of VCPSs given, let us in the following go into more detail on two of its defining characteristics, *connectivity* and *processing capabilities*, to highlight the resulting challenges for data processing inside a VCPS.

***Connectivity*** As mentioned, the vehicles and the central server need to be wirelessly connected because of their physical separation, and because they are mobile. For the VCPS assumed in this work, a *Vehicle-to-X (V2X)* [6] connection is used for all matters of communication from vehicles to the central server. This connection relies on third- (3G) or fourth-generation (4G) wireless technology available today[1]. Practically, this type of wireless connection implies that the bandwidth of communication is limited compared to the amount of data that can be sensed (when regarding the data output of modern vehicles, as discussed in Section 1.1), but also that carrier-operated networks are used (in contrast to WiFi-based *Vehicle-to-Vehicle (V2V)* or V2X communication [18] that creates local networks). Two-way connectivity ensures that data can be sent and received. Overall, the assumed technology required for this type of communication in a VCPS is readily deployed in many vehicles that are commercially available today [14].

***Processing capabilities*** To process the data from more than 100 sensors and perform operations such as emergency braking or lane-keeping, modern vehicles possess special-purpose on-board computers whose capabilities can vary depending on their specific use case. However, novel semi- and fully autonomous vehicles (with especially the latter still being in the experimental and concept phase) require the real-time processing of large amounts of camera, Radar and LiDAR data and are usually equipped with a central (and possibly redundantly implemented) computing unit comparable to or exceeding modern consumer hardware (an example is the DRIVE platform by chip manufacturer

[1] Although the fifth generation (5G) is set to replace the aging communication standards of 3G and 4G, 5G is not widely available as of now.

NVIDIA [19]). With power draws in the range of 200 to 2000 W [20], these computing units compete with other components of the vehicle for the total available electric power, an effect that may be exacerbated in battery-electric vehicles. Consequentially, additional and spontaneous analyses deployed onto the on-board computer must have a manageably small impact.

Combining the aspects of connectivity and processing capabilities with the fact that a VCPS produces Big Data (as described in Section 1.1), the question arises: Where should the data be processed?

Data is sensed at the edge, where the available processing power is somewhat limited, while the processing power at the central server is much less constrained and could be orders of magnitude larger than that of a single edge device. Processing the data from one or several vehicles at the central server can thus appear as the right solution to free up resources at the edge that may be needed for base functions. However, as raised in [21], considering the limitations imposed by the wireless network, transmitting raw data from the vehicles to the central server quickly becomes infeasible for analyses that require large amounts of data from a large number of vehicles. In addition to insufficiencies of the network, which cannot sustain transmitting the required data amounts for analysis of VCPS data (as also highlighted in [22]), the costs for moving this much data over carrier-operated networks would be prohibitive. Moreover, with automotive fleets in the range of 100,000s of vehicles, processing and storing the raw data from all vehicles can still exceed the central processing and data management infrastructure [8].

On the other end of the spectrum of distributing the analysis in a VCPS, all computation is done on the vehicle's computing units, and the central server only serves to initiate the data processing. Thereby, costs and bottlenecks imposed by data transmission can be almost completely avoided. However, the demanded analysis may exceed the available local computing budget of an on-board device (as noted in e.g. [23, 24]), or the analysis may need to span the data from several vehicles and thus be impossible when done in a completely siloed fashion.

This continuum of shifting computation between the central server and edge nodes is called *Edge Computing*, with its two extremes outlined above. As recognized in [25], efficient approaches to Edge Computing in a VCPS need to adapt the analysis pipeline in order to utilize the available computational power in the complete network (central server and edge nodes), while minimizing the transferred data amounts. To achieve this, the single stages that make up the analysis pipeline have to be carefully chosen and smartly distributed between the central server and the edge nodes - posing the main challenge for data processing in VCPSs.

With this challenge in mind, I formulate in the following two research questions that explore said challenge more thoroughly.

## 1.3 Research Questions

As explained in the previous Section 1.2.2, the central problems in data analysis in VCPSs gravitate around where to process the data and how to communicate

effectively and efficiently. In that context, this thesis raises two main questions that guide the work in the appended papers.

**Research Question 1:**

*"How are analysis speed, precision and costs impacted by distributing the processing stages in a continuous vehicle-to-central server analysis pipeline?"*

This first question has several dimensions. First, the aspect of continuous data processing: Various approaches for adapting the continuous data processing paradigm of *Data Streaming* to the unique challenges of CPSs can be found in the literature [26,27], as well as works on advanced data streaming analytics in a CPS [28]. The topic is more novel to VCPSs: How can one play to the strengths of this system in employing a continuous analysis pipeline, aiming at reducing the latency of the analysis and the communicated data volume? A continuous *streaming* approach is an ideal candidate for not transmitting all required data at once from the vehicle to the central server (e.g., whenever a certain amount of data has been collected, or when a timer expires), but for rather sending fresh data in a continuous stream. The benefits of a continuous analysis enabled by Data Streaming appear evident (namely, constantly obtaining results from fresh data with a low result-latency), but the previously-discussed infeasibility of transmitting raw data from high-bandwidth sensors in the face of limited-bandwidth networks remains and challenges naive implementations of a streaming analysis pipeline.

The other dimension of the question is that of where to deploy the stages of the analysis, as discussed in Section 1.2.2. Which type of processing is best done in which part of the analysis pipeline? While the answer to this question is partly dictated by the analysis at hand, employing additional auxiliary processing stages can help to leverage the benefits of a streaming approach.

**Research Question 2:**

*"How to balance communication, workload and completion time when querying a VCPS for a representative sample?"*

The second question touches deeper on the fact that in many real-life scenarios (see Section 1.4.1), it is not known *which* of the vehicles in a VCPS has the data that is required for analysis. Being able to answer this question can reduce the required communication volumes drastically, by preventing vehicles from transmitting data that would be unfit for the analysis at hand - and by preventing the transmission of too much data. If a specific type of data is needed from a certain number $N$ of vehicles[2], then receiving said data from no more than $N$ vehicles is a large reduction in communication compared to receiving it from much greater parts of the fleet. In the latter case, most of the data possibly has to be discarded from the analysis because it is unfit, and storing and processing it increases the load on the central server. A smart algorithm is required to find such a minimum number of vehicles with data fulfilling certain predicates as quickly as possible - whereby the vehicles

[2]The number of vehicles whose data is required for a certain analysis may be dictated by statistical significance, or by external constraints such as cost.

check their data for compliance themselves. In that way, parts of the work are pushed towards the vehicles, a process that must happen in a balanced fashion, with load spread evenly over the fleet. Lastly, the completion time, that is the time until $N$ vehicles with compliant data have been identified, must be minimized to not stall any subsequent analysis. Varieties of this problem have been approached by assuming V2V communication in several works [29–31]; however, for the specific VCPS presented in this thesis Section 1.2.1, where no direct communication among vehicles exists, the issue merits an investigation.

## 1.4 Preliminaries

In the following sections, I give an overview of some additional topics and techniques that are important for understanding the approaches towards Research Questions 1 and 2 that are outlined in Section 1.5 and ease the understanding of the appended publications (Chapter 2 and Chapter 3).

### 1.4.1 Fleet Data Heterogeneity

In a VCPS, the data that is being sensed at the edge is subject to environmental factors (e.g. the physical location, time of day, or weather) and the behavior of the persons currently controlling the vehicles. As an example, somebody owning a car and commuting with it to work will use it in a different way than somebody that uses a car only for weekend trips. Consequently, while the *types* of signals that are recorded can be identical over a whole fleet, or known parts of a fleet, the events inside those signals and the amount of data collected (which is depending on the length of time a vehicle is in use) can vary widely. This is a further variation of the challenge introduced in Section 1.2.2 - the data is (1) located at the edge of the VCPS, and (2) it may be distributed unevenly. Figure 1.2 shows an example of this heterogeneity between two fleets,

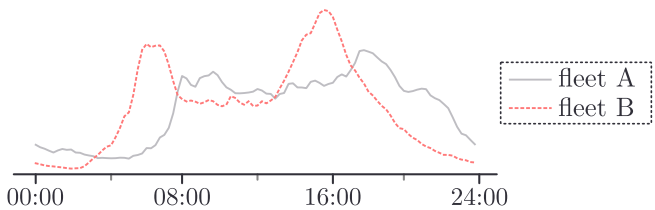



**Figure 1.2:** The distribution of the volume of collected data over one day for two different vehicular fleets *(figure altered and taken from [1]).*

with the x-axis denoting the time of day and the y-axis the share of data collected on average at that time by a vehicle in one of two fleets. Apparently, the vehicles in fleet B collect more data during rush hour in the morning and in the afternoon, while those in fleet A collect more data later in the day and follow a much less pronounced rush hour pattern. From the graph, vehicles from fleet B could be unfit for studying the driving behavior at night, as they are less active during that time.
Generally, there can be insufficient a priori knowledge at the central server about

the exact data that has been collected by a certain vehicle. The previously-discussed difficulty of gathering raw data from vehicles as well as existing privacy laws such as the European GDPR (General Data Protection Regulation) make it furthermore difficult to obtain insights into the nature of the raw collected data on-demand at the central server. As the above example shows, a naïve assumption of equal distribution of data over the fleet can be misleading, and the consequence may be the transmission of data that is unfit for a certain analysis. That way, fleet data heterogeneity can exacerbate the issue of critically high data volumes in a VCPS. On the other hand, by gaining selective knowledge at the central server about where data important to an analysis resides, data volumes that need to be transmitted between edge and center may be reduced.

### 1.4.2 Data Streaming

As described in Section 1.1, Big Data generated in VCPSs is chiefly characterized by two aspects: first, its volume, and second, its continuous nature. As vehicles are used, a constant stream of data is sensed by the numerous on-board sensors, because the processes that are recorded by the sensors are ever-evolving, be it the driving of the car, the behavior of the driver or the vehicle's environment. As the data is continuously evolving, analysis tools are required that can deal with unboundedness.

Classic database design (e.g. SQL, NoSQL) is guided by the *first-the-data-then-the-query* paradigm, which can be unfit for rapidly evolving datasets: First, data is stored before the analysis, which introduces inherent latency and can cause the analysis results to become stale relative to the continuous data stream. Second, it can be computationally expensive to reprocess large chunks of data in a batch fashion whenever updated results are required. An alternative is found in the Data Streaming paradigm [32] that reverses the previously named order into *first-the-query-then-the-data*: Instead of storing data first and deploying *queries* later to generate insights, the query is defined first. Subsequent new data is continuously queried while it is entering the processing pipeline, and incremental aggregations ensure that updated results can be obtained without having to reprocess past data. With Data Streaming, there is the option of only storing compliant data, or of performing aggregations and transformations before data is either stored or forwarded to another system. In fact, by removing the necessity of storing incoming data at all, Data Streaming was inherently designed for dealing with unbounded datasets [32].

#### Streams, Operators and Queries

In Data Streaming, a *Query* is a set of transformations applied to the data entering the data processing system. As an example, such a system (referred to as a *Stream Processing Engine* (SPE)) could be processing in real-time the positional reports of a fleet of vehicles, and a query could answer the question of "which vehicle drove the fastest in the last 15 minutes?". A more abstract query could be the necessary transformations and aggregations on an incoming video feed to forward pre-processed images to a computer vision system. Because of the unboundedness of evolving real datasets (as those produced in a VCPS), such queries typically cover chunks of the input data. These span finite time periods and are called *time windows* [33].

Incoming data to an SPE is designated by its origin, called its *source*. A source may, for example, be a single physical sensor that continuously reports sensor readings, such as a GPS receiver or a wheel speed sensor, or a group of sensors such as a rotating LiDAR (e.g. a Velodyne HDL-64E which consists of 64 individual laser distance sensors [34]).

Said incoming data consists of *tuples*, which in their succession make up a *stream*. Tuples are typically timestamped with the timepoint of entering the SPE or, more commonly, the timestamp at which the underlying data was sensed, and carry one or more fields of data. *Operators* define the operations performed over single tuples or windows of several tuples from one or several streams, and each operator produces one or several output streams itself. In this way, the semantics of a query can be expressed as a directed acyclic graph (DAG) of operators and streams, where the data streams end in a set of *sinks* that define the endpoints of the query. A sink could write the incoming tuples to disk, or forward them to another application [33].

### Data Streaming in VCPSs

Having evolved for distributed, parallel and elastic online analysis (see e.g. [35]), the Data Streaming paradigm is favorably employed in VCPSs [36].

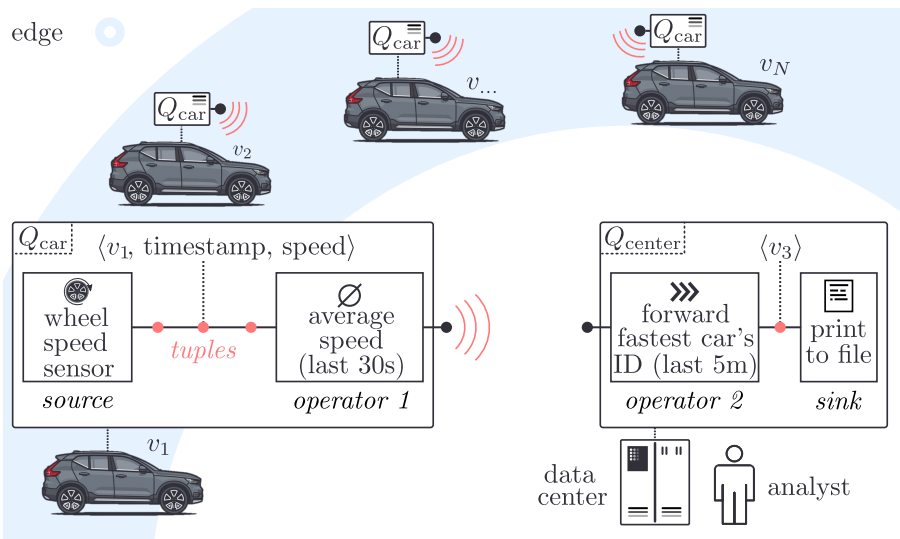



**Figure 1.3:** A sample distributed data streaming application deployed between $N$ vehicles and a data center. An instance of the first part of the query, $Q_{\text{car}}$, is deployed at each vehicle, here detailed for vehicle $v_1$. The second part of the query, $Q_{center}$, is deployed at the data center, to which each vehicle transmits intermediate results.

An example of a data streaming application deployed in a VCPS is shown in Figure 1.3. The application is distributed between the vehicles $v_1, \dots, v_N$, and a data center, from which the application was issued by the analyst. $Q_{\text{car}}$, the first part of the query, is deployed in parallel at each vehicle, here detailed for vehicle $v_1$: The wheel speed sensor is the *source*, sending sensor readings in the form $\langle v_1$, timestamp, speed$\rangle$ to *operator 1*. There, the average of the

sensor readings from the last 30 seconds is calculated using a time window, and then wirelessly transmitted to the data center, where the second part of the query is deployed, $Q_{center}$. All average wheel speed readings from all vehicles are collected here. Every five minutes, the ID of the fastest car is produced by *operator 2* and printed to file by the *sink*.

As seen in the above example, one can observe the following overlaps between processing requirements within a VCPS and capabilities of the Data Streaming paradigm:

***Unboundedness*** In VCPSs, the data created at the edge is of continuous nature, making the Data Streaming paradigm a valid choice for processing the data streams.

***Footprint*** Removing the requirement to store data before processing eases the demands on the available storage, which becomes poignant in the face of resource-constrained embedded systems. Furthermore, incremental aggregation avoids the re-processing of past data.

***Distributed & Parallel Analysis*** A VCPS is inherently a distributed system, with data-producing and data-processing nodes. The required flows of data inside this system can be physically or logically mapped to the structure of a query's DAG, thus matching the architecture of the VCPS and the analysis flow. As shown in the example in Figure 1.3, data sources can reside on the vehicles, data sinks on the central server, and operators that process the data are distributed between both.

### 1.4.3 Time-Series Data Compression

Time-series data is a series of numerical timestamped data describing a $N+1$-dimensional curve, where the $+1$ dimension is time. As such, it can be decomposed into $N$ 2-dimensional curves. One example is the point cloud of a LiDAR (see [17]): A common LiDAR is a rotating column of 64 distance-measuring lasers, whose distance measurements over one full rotation of the column make up the point cloud. Thus, a LiDAR can also be seen as 64 individual time-series of the form (timestamp, distance). To store a time-series such as that of an individual LiDAR laser in a smaller representation than the raw data, time-series data compression can be used. While some techniques exist to achieve compression in a lossless fashion, e.g. Delta Compression [37] for integer values or data-agnostic dictionary compression as used in the `DEFLATE` algorithm of ZIP, many approaches aim towards *approximating* time-series data, e.g. [38,39], incurring a loss in precision of the compressed representation.

One such approach is to calculate a piecewise-linear function that best approximates the original data (see Figure 1.4). The set of all linear *segments* of the approximation results in a (typically) smaller encoding than the uncompressed data, although the viability of the method is subject to the underlying data (some techniques are adaptive and can avoid producing a larger representation [40]). As an example, a piecewise-linear approximation of a LiDAR point cloud may result in a slightly distorted point cloud due to the approximation error. That may mean that the physical objects that appear in

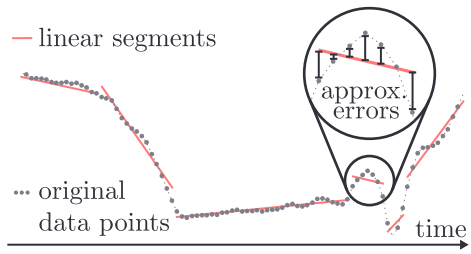



**Figure 1.4:** A piecewise-linear approximation of the dotted grey time-series via the solid red segments. The cut-out shows the approximation errors between the original and linearly approximated data.

the cloud appear as shifted from their true position, while the required storage space for this point cloud is drastically reduced by placing points in the cloud along straight lines where possible [17].
Although different in their approach and their goal (e.g. fast search [41]), many of the techniques in the literature have in common that they allow varying the approximation error or maximum deviation between the original data and the piecewise-linear representation. A larger error typically results in longer segments and thus a reduced size of the encoding. Thus, approximation techniques can typically achieve smaller representations than lossless compression algorithms such as `DEFLATE` (as demonstrated in Chapter 2).
By trading off space and accuracy in the form of approximation errors, time-series data compression via piecewise-linear approximation can be used as a building block in analysis workflows, allowing larger compression (and thus larger error) for less sensitive analyses and smaller compression in more critical situations. In the LiDAR examples, slightly distorting point clouds may be permissible for creating a 3D map of the environment, whereas the precision requirements for collision detection using a point cloud are much stricter.

## 1.5 Thesis Contributions

Having introduced a few key preliminary topics and techniques, I highlight in the following the main contributions of this thesis.

### 1.5.1 Efficient, Continuous Data Retrieval and Clustering

In Chapter 2, Research Question 1 as posed in Section 1.3 is approached: *"How are analysis speed, precision and costs impacted by distributing the processing stages in a continuous vehicle-to-central server analysis pipeline?"* We chose to approach this question by investigating an analysis pipeline that involves the gathering of raw data at the vehicle, wireless transmission to a central server and subsequent clustering of the data. We enable the execution of said pipeline in a streaming fashion, from continuous data transmission at the vehicle to continuous data clustering at the central server. To achieve continuous clustering, we adapt the clustering algorithm in [42] to more general use cases. This scenario now becomes the baseline. In the DRIVEN framework introduced in Chapter 2, we add to said pipeline an additional data compression stage at the vehicle, thereby pushing more than just the recording and transmission of data onto the edge. Like the remainder of the analysis pipeline, the compression

stage in DRIVEN works in a streaming fashion by continuously ingesting data and outputting it in a compressed representation. The level of compression is tunable and can thus be adapted to the use case at hand. We evaluate DRIVEN in several dimensions: (1) for several use cases, varying from a fleet of vehicles transmitting their GPS positions to single vehicles transmitting LiDAR data; (2) for different simulated network speeds ranging from 2G to 5G; and lastly (3) we evaluate the compression stage itself by comparing its effectiveness to ZIP compression. By benchmarking DRIVEN in terms of the volumes of data transmitted between vehicle and central server, analysis duration, and accuracy of the clustering result against a baseline without the compression stage, we give a quantifiable answer to Research Question 1.

### 1.5.2 Adaptive, Online Query Spreading Algorithms

In Chapter 3, an approach to Research Question 2 is presented: *"How to balance communication, workload and completion time when querying a VCPS for a representative sample?"*. We investigate the spreading of compliance queries from a central server over a fleet of vehicles, in which each vehicle uses its on-board computing unit to check the compliance of locally stored data with the query - as transmitting data to the central server and checking its compliance later is costly or even infeasible, as explained in Section 1.2.2. A binary answer to the compliance query is returned by each vehicle that was contacted, indicating whether or not compliant data is present on said vehicle. The aim of the query is to obtain at least $N$ positive answers, either to be used (1) as a statistic for the distribution of a certain type of data over the fleet, or (2) as a pre-selection step for involving only cars that have compliant data into further analysis steps. We first evaluate two baseline algorithms that either ask vehicles one-by-one, thereby stretching the time until $N$ vehicles are found while avoiding pushing the workload on too many vehicles (and thus increasing the total workload required for solving the query). On the other end of the spectrum, the second baseline algorithm asks round-based first a random selection of $N$ cars, and in subsequent rounds always the number of missing positive answers. This way, the query is resolved quickly, but the overall workload is high, as many vehicles become involved. We investigate the deployment of not only one such query, but of many that are deployed in parallel and resolved by the vehicles sequentially. Our contribution consists of two adaptive algorithms, one that trades off between the total workload induced on the fleet and the resolution time of the query, and one that additionally respects fairness by spreading the workload evenly over the fleet. We evaluate these two algorithms by comparing them to the baselines by simulating a vast fleet of vehicles moving through two cities using large real-world datasets.

## 1.6 Conclusion and Outlook

This Licentiate thesis explores several avenues for supporting *Distributed and Communication-Efficient Continuous Data Processing in Vehicular Cyber-Physical Systems*. In Chapter 2, the DRIVEN framework demonstrates how an additional, tunable compression stage can deliver analysis speedups and

data transfer savings in a data processing pipeline while keeping analysis accuracy losses bounded, enabling the optimization of the processing pipeline for specific use cases and data types. Chapter 3 shows how to efficiently find those vehicles in a vehicular fleet that carry data that is compliant to a specific query, thereby allowing follow-up analyses that involve just the right vehicles - avoiding the unnecessary transmission of raw data, and reducing the overall workload induced on the fleet. Having vehicles check compliance of their own data smartly leverages the available computational power in the fleet while balancing mechanisms avoid putting too much load on individual vehicles.

In ongoing and future work, we explore further ways of improving the continuous data processing in VCPSs. One promising avenue is presented by techniques for tagging relevant data already inside a data streaming analysis pipeline deployed at the edge, such that the further storage or processing of important pieces of information can be emphasized. In an overarching analysis framework, spanning a complete VCPS, such a technique could be employed in collaborative learning schemes in which vehicles can improve their local analysis prowess by learning, directly or indirectly, from important data of other vehicles. Likewise, an extension of the research on query-spreading mechanisms in VCPSs is currently in the works, to adapt the proposed spreading algorithms to more dynamic fleet behaviors, such as a fluctuating number of vehicles during the query. Lastly, the promising results for the use of a compression stage in a distributed, continuous processing pipeline hint at further possibilities for the use of data reduction schemes in a VCPS that could, for example, be adaptive with respect to the nature of the data: if an interesting pattern inside a data stream is observed, a more faithful compression grade is chosen, whereas less interesting sections of data can be compressed to a higher degree.

# Chapter 2:

## DRIVEN: A Framework For Efficient Data Retrieval And Clustering In Vehicular Networks

---

**B. Havers**, R. Duvignau, H. Najdataei, V. Gulisano,
M. Papatriantafilou and A. Koppisetty

*Future Generation Computer Systems, vol. 107, pp. 1-17, 2020 [17].*

PAPER A

This is a formatted and adapted version of the paper published in *Future Generation Computer Systems, vol. 107, pp. 1-17, 2020.* Any performed changes serve only to retain the consistency of this thesis and to adapt to the layout.

# 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.

## 2.1 Introduction

Large distributed Cyber-Physical Systems (CPSs) such as vehicular networks [43] (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 [44], efficient communication [45, 46], security [47] and privacy [48]. A key aspect in this context is the need for solutions that can jointly address several such challenges [49], since solutions that focus on and/or excel in only one aspect but fall short in others might be impractical in real-world setups.

### 2.1.1 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 [6]) and the infrastructures' capacity of gathering them within small time periods to data centers [1]. 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 2.4.1). 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 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.

### 2.1.2 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:

[a] leveraging a lossy compression technique, based on Piecewise Linear Approximation (PLA), that significantly reduces the amounts of data to be gathered from vehicles,

[b] leveraging state-of-the-art online clustering techniques such as Lisco [42], which overcome the limitations of batch-based ones, and

[c] 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 2.6. 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 2.2 and the considered system model and problem statement in Section 2.3. We then present the DRIVEN framework in Section 2.4 and our evaluation in Section 2.5. Finally, we discuss related work in Section 2.6 and conclude the paper in Section 2.7.

## 2.2 Preliminaries

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

### 2.2.1 Data streaming

The data stream processing paradigm (aka data streaming) [32] 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 [35]. At the same time, efficient designs and methods are in focus in the literature for computationally-expensive streaming analysis [50]. As discussed in [32], 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 2.1). Thus, many applications rely on it in many CPSs, including vehicular networks [27, 36, 51].

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 [52, 53] (or leverages sorting techniques such as [54, 55]). 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 [35], 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 [32] and can temporarily maintain a *window* of the most recent tuples when an aggregation function (such as clustering) is to be performed on them [50]. As mentioned in Section 2.1, 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 [56–59]. In DRIVEN, we rely on PLA, further discussed in the following section.

### 2.2.2 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 $\Delta$, i.e., (i) the time series is processed one point at a time, the output line segments

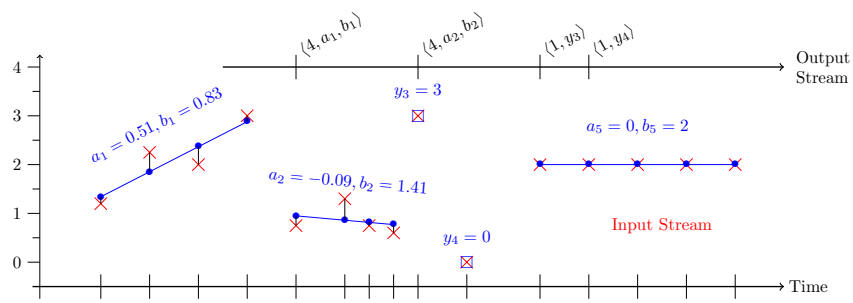



**Figure 2.1:** Example of a Piecewise Linear Approximation using maximum error $\Delta = 0.5$.

are produced along the way, and (ii) the projected points along the compression line segments always fall within $\Delta$ from the original points. Figure 2.1 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 2.4 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 [60–62]), 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 [40, 63–65] 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 [40] and briefly described in Section 2.4.2, balancing trade-offs associated with PLA in a streaming context.

### 2.2.3 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 [66]. For ease of reference we paraphrase the definition of distance-based clustering from [67]:

**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 [42], 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 2.4.3.

### 2.2.4 Logical latency

In data streaming literature [50], 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 $i \in [\ell, \ell + n]$.

## 2.3 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

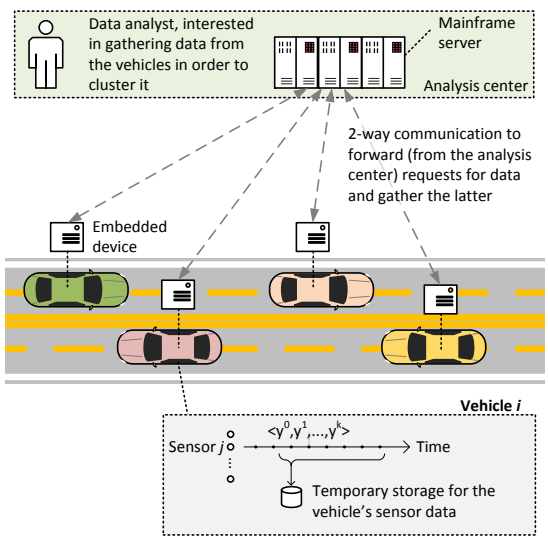



**Figure 2.2:** System model overview for DRIVEN.

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 2.5.6 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 2.5.3).

Based on the given system model illustrated in Figure 2.2, 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 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),
- $\Delta$ 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 $\Delta_0$ for the (logical) time attribute,
- $q_{\text{pre}}$ is an optional streaming query that defines pre-clustering analysis, and
- {clustering parameters} is the set of parameters used by DRIVEN's clustering component (further described in Figure 2.4.3).

We refer the reader to Section 2.5 and Table 2.1 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.

## 2.4 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 2.5).

As discussed in Section 2.3, 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.

### 2.4.1 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 [34], 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 [42] (around 7 MB/s). For each stream $\langle \alpha, \rho \rangle$ produced by one of the LiDAR sensors, the logical timestamp $\alpha$ allows identifying at which rotation step the distance $\rho$ 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 $\alpha$, $\rho$ 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 2.3, 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,
- $q_{\text{pre}}$ defines an operator merging the data from the different sensors (as further discussed in Section 2.5), and
- {clustering parameters} is the set of parameters later described in Figure 2.4.3.

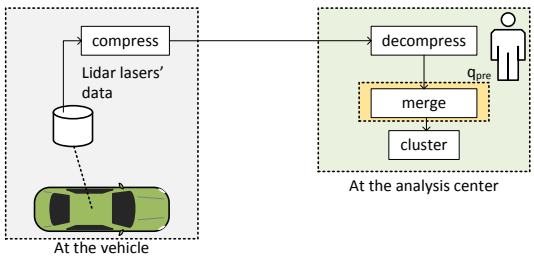



**Figure 2.3:** Overview of the modules deployed in the resulting streaming continuous query for the LiDAR use case.

Figure 2.3 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).

### 2.4.2 Data retrieval and PLA approximation

As discussed in Section 2.1, DRIVEN relies on streaming PLA to forward a compressed and lossy representation of data. To build the PLA, we use a construction method named *Linear*, introduced in [40], which combines several approaches of previous works on PLA such as using a best-fit line approximation [60,62,65] for minimizing errors and maintaining convex hulls [61, 63] for efficiently checking the violation of the error bound by the approximation. We also use here continuous processing through the *output protocol* proposed

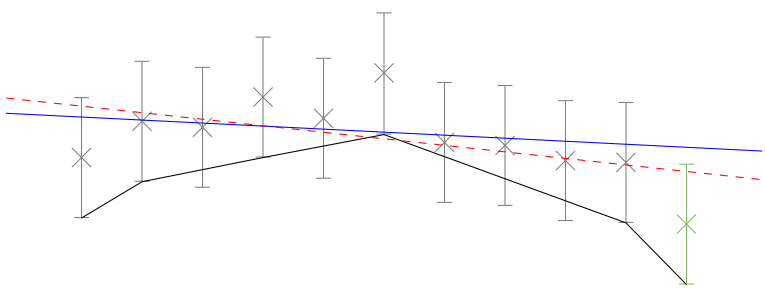


**Figure 2.4:** Best-fit lines of a set of points: solid for the first 10 points, dashed for including the 11th point (marked in green).

in [40] 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 $\Delta$. 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 2.4 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).

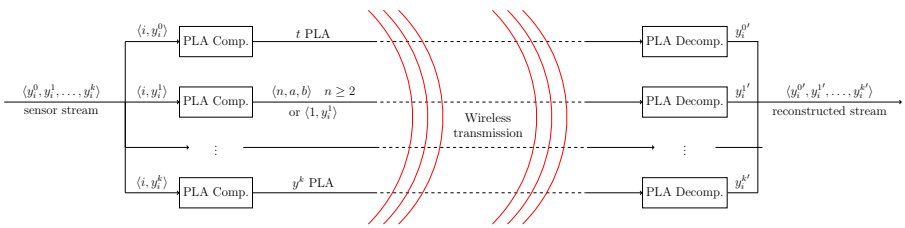



**Figure 2.5:** PLA compression / decompression flowchart with $y^1$'s channel detailed.

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 2.5, are:

[a] **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 $\ell$'s stream for each $0 \leq \ell \leq k$.

[b] **PLA Compression:** Each stream is compressed in parallel by computing its PLA (as depicted in Figure 2.1, Section 2.2.2) using its associated error, i.e., channel $\ell$ uses $\Delta_\ell$. 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

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

**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
$l_i \leftarrow \text{newBestFitLine}(l_{i-1}, \langle i, y_i \rangle)$
$U_i, L_i \leftarrow \text{updateConvexHulls}(U_{i-1}, L_{i-1}, \langle i, y_i \rangle)$
$n \leftarrow n + 1$
**if** $\text{lineBreaksHulls}(l_i, U_i, L_i)$ **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**

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 2.5), but when one such segment has only length $n = 2$, thus covering only two tuples $\langle k-2, y_{k-2} \rangle$ and $\langle k-1, y_{k-1} \rangle$, $\langle k-2, y_{k-2} \rangle$ 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 2.1). 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).

[c] **Diffusion:** The $k + 1$ streams are wirelessly transmitted to the analysis center.

[d] **PLA Decompression:** All streams are decompressed in parallel. The decompression algorithm is straightforward: after having already reconstructed $i$ values on channel $y^\ell$, 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 $y^\ell$'s transmitted PLA stream, or alternatively, we produce $y'_{i+1} = y$ if $\langle 1, y \rangle$ was received.

[e] **Final Reconstruction:** The final step is to merge the $k+1$ decompressed streams to rebuild records with identical structure as the initial input

[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.

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 $y^\ell$). This differs from the scheme suggested in [68], 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 [68], 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 2.5, and guarantees a bounded reconstruction error.

### 2.4.3 Data clustering with Lisco

As described in Section 2.2, 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 (e.g. a kd-tree [69]) 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 [70] 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 [42]) 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.

#### Clustering LiDAR data (intuition)

Figure 2.6 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

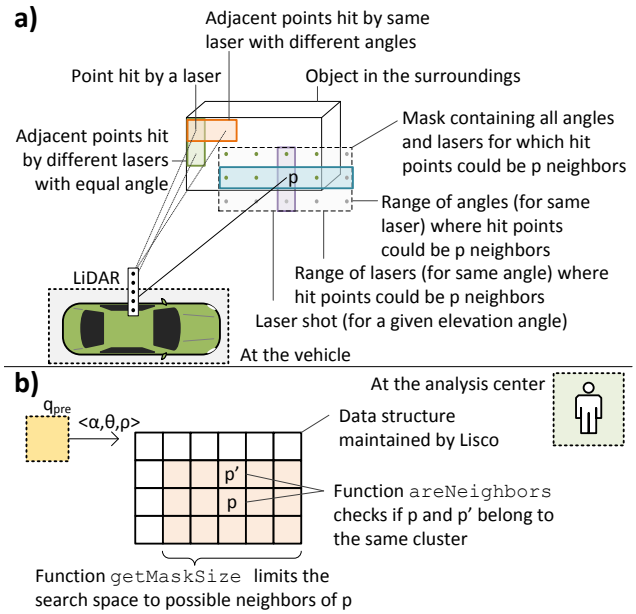



**Figure 2.6:** Example of how the search space for a point $p$ (for the LiDAR use case) can be limited to points potentially reported by lasers (and with certain angles) within a mask centered in $p$ (a) and the corresponding 2D matrix maintained by Lisco (b).

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.

### 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 $q_{\text{pre}}$ (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` $\tau$`)` and `boolean areNeighb(Tuple` $\tau_1$`,Tuple` $\tau_2$`)`. The

former function specifies how far (in the sense of indexes) Lisco should explore any of the $n$ dimensions of the array around tuple $\tau$, 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 $\tau$. 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 2.6 b), the schema of the tuples produced by $q_{\text{pre}}$ could in this case carry attributes $\langle \alpha, \theta, \rho \rangle$, where $\alpha$ is the logical timestamp that refers to a certain angle of the LiDAR sensor, $\theta$ is the elevation angle (based on the laser producing the reading) and $\rho$ 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 [42]). 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` $\tau$`)` can be implemented to return the neighbor mask of a tuple $\tau$ in terms of a limited number of rows and columns around $\tau$. Finally, the `areNeighb(Tuple` $\tau_1$`,Tuple` $\tau_2$`)` can be implemented to check whether the Euclidean distance between the readings of tuples $\tau_1$ and $\tau_2$ is within the threshold defined by the analyst.

## 2.5 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.

### 2.5.1 Data

We use three datasets in our evaluation.

[a] The *Ford Campus* dataset [34], providing data generated by a VelodyneHDL64E roof-mounted LiDAR (see Section 2.4.1 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 2.3), 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.

[b] The *GeoLife* dataset, containing GPS data collected in the *GeoLife* project by 182 users over ca. three years [71–73]. We use a subset of the dataset with vehicular GPS traces in the Beijing area.

[c] 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.

### 2.5.2 Software and hardware setup

We implemented Lisco in Python 3.6 and the PLA components (see Section 2.4.2) 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 [42]).

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 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* [74] 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).

### 2.5.3 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.

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

[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 [75] for more details).

**Table 2.1:** Queries $Q(\mathbb{V}, \mathbb{T}, \mathbb{S}, \Delta, q_{\text{pre}}, \{\text{clustering parameters}\})$ for evaluation. See Section 2.3 for explanation of query arguments.

$Q_1$: **LiDAR**

| $\mathbb{V}$ | $\mathbb{T}$ | $\mathbb{S}$ | $\Delta$ | $q_{\text{pre}}$ | 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` $\tau$`):` `return getMaskSizeInRot(`$\tau$`)` `areNeighb(Tuple` $\tau_1$`, Tuple` $\tau_2$`):` `return euclidDist(`$\tau_1$`,`$\tau_2$`)≤0.5m` |

$Q_2$: **1-Vehicle 1-Day**

| $\mathbb{V}$ | $\mathbb{T}$ | $\mathbb{S}$ | $\Delta$ | $q_{\text{pre}}$ | 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` $\tau$`):` `return 12` `areNeighb(Tuple` $\tau_1$`, Tuple` $\tau_2$`):` `return euclidDist(`$\tau_1$`,`$\tau_2$`)≤50m` |

$Q_3$: **1-Vehicle 14-Day**

| $\mathbb{V}$ | $\mathbb{T}$ | $\mathbb{S}$ | $\Delta$ | $q_{\text{pre}}$ | 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` $\tau$`):` `return 15, 14` `areNeighb(Tuple` $\tau_1$`, Tuple` $\tau_2$`):` `return euclidDist(`$\tau_1$`,`$\tau_2$`)≤100m` |

$Q_4$: **Car usage grids**

| $\mathbb{V}$ | $\mathbb{T}$ | $\mathbb{S}$ | $\Delta$ | $q_{\text{pre}}$ | clustering parameters |
|---|---|---|---|---|---|
| 20 vehicles | 7 days | GPS electric RPM comb. RPM | $\Delta_t G = 1$ s, $\Delta_t e = \Delta_t c = 0.005$ s $\Delta_x, \Delta_y \in [1, 2, 5, 10, 20, 50]$ m $\Delta_\omega e, \Delta_\omega c \in [1, 5, 10, 20, 50, 100]$ Hz | add day id find drive mode add to grid | `getMaskSize(Tuple` $\tau$`):` `return 7, 2, 2` `areNeighb(Tuple` $\tau_1$`, Tuple` $\tau_2$`):` `return counterDist(`$\tau_1$`,`$\tau_2$`)≤1)` |

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 2.2.4 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.

### 2.5.4 Use cases

#### $Q_1$ LiDAR

This is the use case presented in detail in Section 2.4.1. 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 $\alpha$ (logical timestamp) and the distance reading $\rho$. The query for this use case is detailed in Table 2.1. 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 $\alpha$ and the distance readings $\rho$. Each compressed stream of laser readings is then successively sent to the analysis center, where the streams are decompressed. Query $q_{\text{pre}}$ assigns each tuple its laser id and the horizontal angle $\theta$, providing to Lisco the data structured as the 2D matrix (Figure 2.7) to Lisco. The tuples are added column-wise (see colored column) by $q_{\text{pre}}$ with decreasing laser id $\theta^i$ (the id of the $i$-th laser) from top to bottom and increasing rotation angle $\alpha^i_j$ (the $j$-th rotation step of the $i$-th laser) from left to right. This merging of data from different sensors is performed deterministically [50] based on the logical timestamp $\alpha$ 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 $\alpha, \theta$ - area around tuple $\tau$ defined by `getMaskSizeInRot(`$\tau$`)`, which calculates the angles $\alpha, \theta$ 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 $\tau$, as they bound the $\epsilon$-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 $\tau$ (cp. Figure 2.4.3).

The compression statistics for this use case can be seen in Figure 2.8 (a) and (b) expressed via violin plots. The angle $\alpha$ is in all cases compressed with

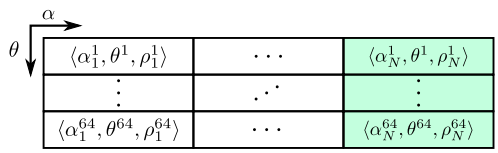



**Figure 2.7:** $Q_1$: Sketch of data structure produced by $q_{\text{pre}}$.

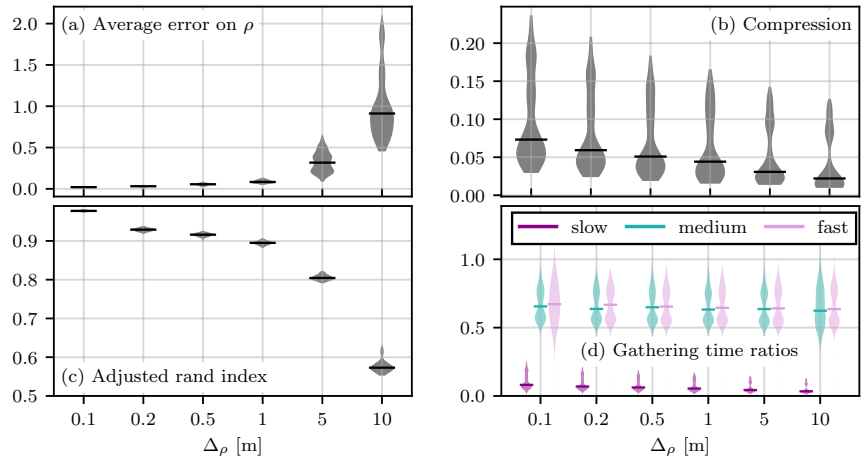



**Figure 2.8:** $Q_1$: (a) - (c) Compression and clustering statistics for various $\Delta_\rho$; (d) gathering time ratio for various $\Delta_\rho$ and different network speeds.

a maximum error $\Delta_\alpha$ of $1.5 \times 10^{-3}$ rad, yielding an average error on $\alpha$ of $3.4 \times 10^{-5} \pm 1.0 \times 10^{-5}$ rad (average $\pm$ standard deviation). $\rho$ is compressed for values $[0.1, 0.2, 0.5, 1, 5, 10]$ m. In (a), it appears for larger values of $\Delta_\rho$ that the average error is about one order of magnitude smaller than the maximum error (this is true for small $\Delta_\rho$ 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 $\alpha$ 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 2.4.1). 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 2.8 (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 $\Delta_\rho$ 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 2.8 (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 $\Delta_\rho$), for all network speeds data gathering times are reduced to between 60 % for fast networks down to less than 15 % for slow networks.

### $Q_2$ 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 2.1. The GPS position data stream from one day and one specific vehicle is compressed on-vehicle with some error $\Delta_t$ 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 $q_{\text{pre}}$ 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 2.9 (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(`$\tau$`)` 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(`$\tau_1$`,`$\tau_2$`)` on the basis of the Euclidean distance between the $x, y$-coordinates of the two tuples.

The compression statistics are given in Figure 2.10 (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 $(0.095 \pm 0.090)$ s. $\Delta_x$ and $\Delta_y$ are chosen to be equal and $\in [1, 2, 5, 10, 20, 50]$ m. Figure 2.10 (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 2.10 (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 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 2.10 (c): for small maximum errors $\Delta_x, \Delta_y$, the adjusted rand index is close to 1, but it decreases for larger maximum errors.

The gathering time ratios are shown in Figure 2.11 (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.

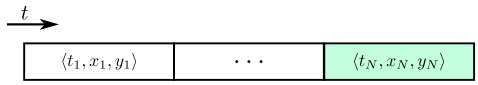



**Figure 2.9:** $Q_2$: Sketch of data structure produced by $q_{\text{pre}}$.

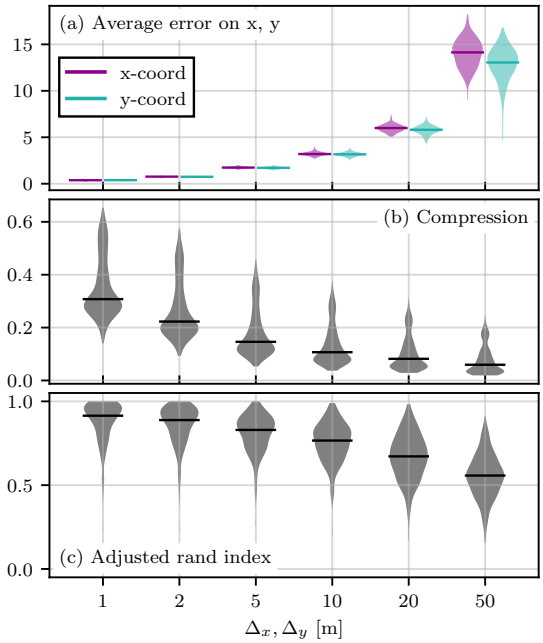



**Figure 2.10:** $Q_2$: (a), (b) Compression statistics; (c): Adjusted rand index.

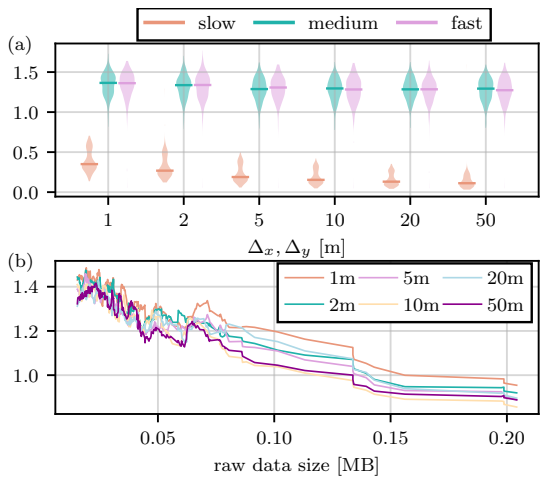



**Figure 2.11:** $Q_2$: Gathering time ratios for various (a) maximal errors for different network speeds and (b) raw data sizes (rolling average over 13 values, different colors are used for distinct values of $\Delta_x, \Delta_y$) for a medium speed network.

More insight is gained from Figure 2.11 (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 $\Delta_x, \Delta_y$. 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 2.5.3).

### $Q_3$ 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 2.1, differs from $Q_2$ in the period covered by the data and in the task of $q_{\text{pre}}$: 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 2.12 (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 2.13 (a), (b) and are similar to those seen in Figure 2.10 (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 $(0.093 \pm 0.081)$ s. The evaluation of clustering qualities is shown in Figure 2.13 (c), also with similar results to the previous use case. The addition of the attribute "days" for Lisco seemingly has only a small influence, suggesting that in the majority of samples there is no significant number of inter-day clusters.

Figure 2.14 (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 2.14 (b) (for medium network speeds), this is due 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 $\Delta_x, \Delta_y$ are smaller than 1.

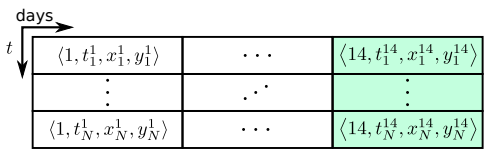



**Figure 2.12:** $Q_3$: Sketch of data structure produced by $q_{\text{pre}}$.

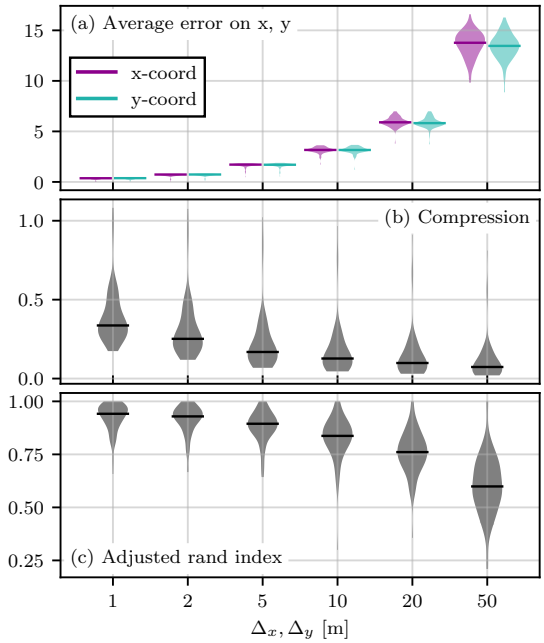



**Figure 2.13:** $Q_3$: (a), (b) Compression statistics; (c) Adjusted rand index.

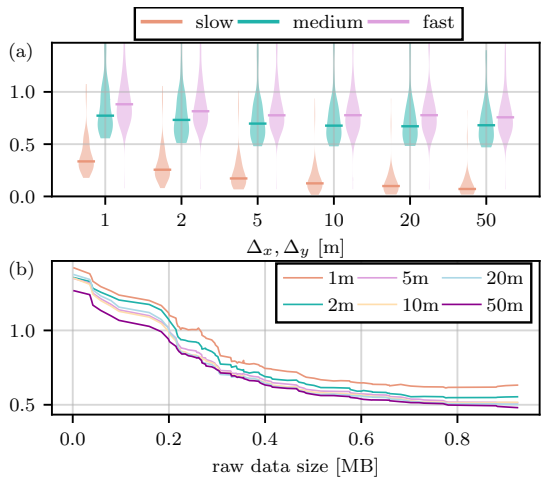



**Figure 2.14:** $Q_3$: Gathering time ratios for various (a) maximal errors for different network speeds and (b) raw data sizes (rolling average over 13 values, different colors are used for distinct values of $\Delta_x, \Delta_y$) for a medium speed network.

### $Q_4$ 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

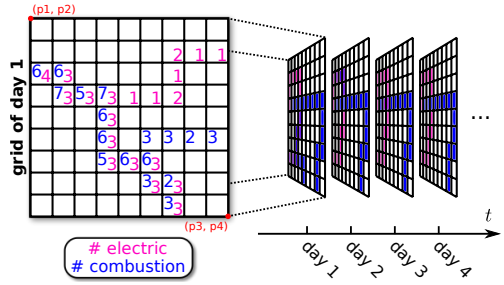



**Figure 2.15:** $Q_4$: Sketch of data structure produced by $q_{\text{pre}}$.

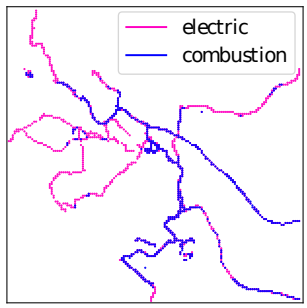



**Figure 2.16:** $Q_4$: Example of one grid (instead of showing the number of vehicles per drive mode and cell, only colors indicate the cell occupation).

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 $150 \times 150$ 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 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 $q_{\text{pre}}$, and a sketch of the data structured as a 3D matrix that is passed to Lisco is visualized in Figure 2.15: The coordinates (p1,p2) and (p3,p4) are located at the corners of the geographical grid (which in this sketch is a $9 \times 9$ grid). The first dimension of the 3D matrix

| # | $[\Delta_x = \Delta_y, \Delta_{\omega^e}, \Delta_{\omega^c}]$ |
|---|---|
| 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 2.2:** $Q_4$: Maximum-error sets used. As three different time series are requested, three different error parameters are given (for GPS, $\Delta_x = \Delta_y$).

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 2.16 (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 2.1. Two grid cells, represented by tuples $\tau_1, \tau_2$, become clustered in accordance with `counterDist(`$\tau_1, \tau_2$`)` if the difference in both cells' electric or both cells' combustion mode counter is smaller than or equal to 1.

The time channels $t^G, t^e, t^c$ (for GPS, electric engine RPM and combustion engine RPM) are compressed with $\Delta_{t^G} = 1$ s; $\Delta_{t^e} = \Delta_{t^c} = 0.005$ Hz, resulting in average errors of $(0.49 \pm 0.06)$ 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 2.2.

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 2.18: (a) is the average error on the $x, y$-coordinate of the GPS time

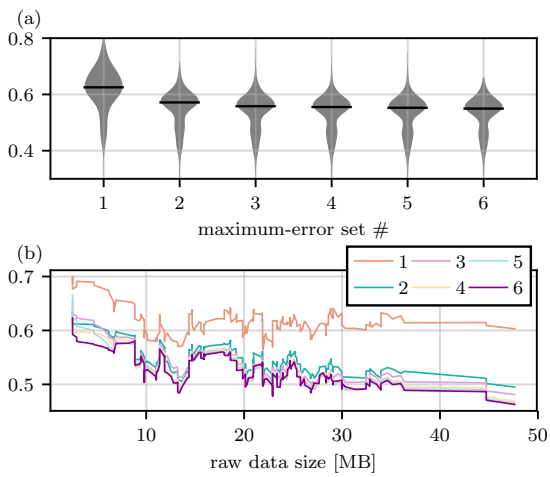



**Figure 2.17:** $Q_4$: Gathering time ratios for (a) a very fast network speed and (b) for various raw data sizes (rolling average over 13 values, different colors are used for different maximum-error sets) for a very fast network.

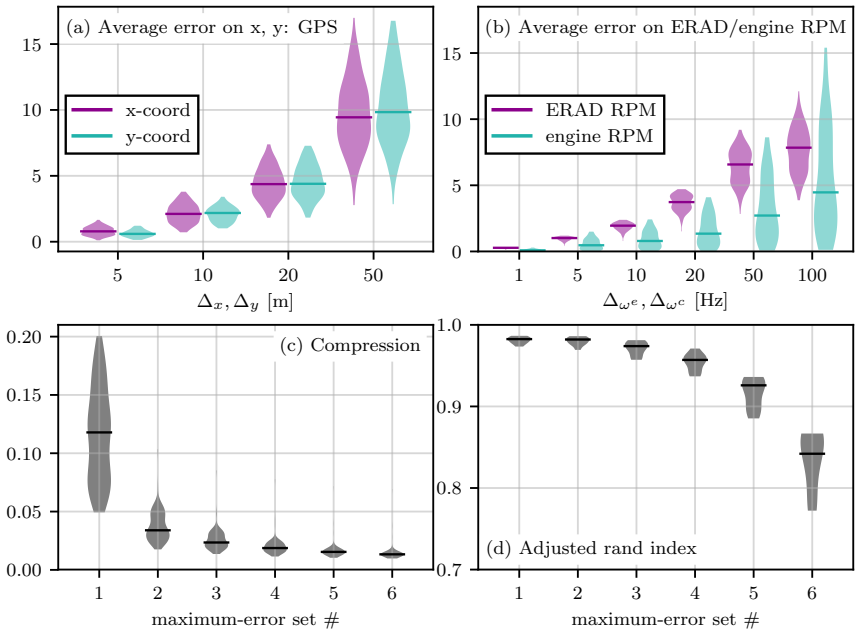



**Figure 2.18:** $Q_4$: (a) Average error on the $x$- and $y$-coordinate for several values of the maximum compression errors $\Delta_x, \Delta_y$ for GPS data; (b) average error on the ERAD and engine RPM for several values of the maximum compression errors $\Delta_{\omega^e}, \Delta_{\omega^c}$; (c) compression statistics for different maximum-error sets (see Table 2.2); (d) adjusted rand index.

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 2.17 (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 2.18 (c). Figure 2.17 (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.

### 2.5.5 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 2.19, with "$\times$" 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 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 2.8 (a), 2.10 (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.

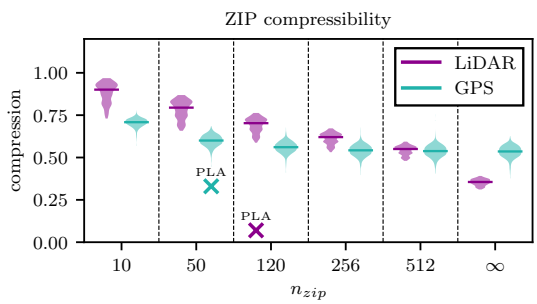



**Figure 2.19:** Compression ratios using ZIP for varying segment lengths $n_{zip}$. "$\times$" marks the compression achieved with PLA for equivalent *average* $n$ for smallest maximum errors (almost lossless).

### 2.5.6 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 2.2.4), the logical latency incurred by DRIVEN can be modeled as follows. For the stream of values $y$, the logical latency 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 [40] 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 latencies 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 2.20 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.

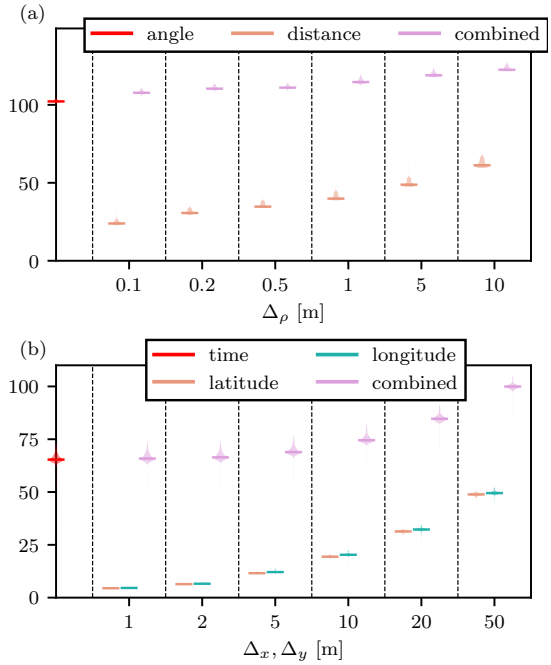



**Figure 2.20:** Distribution of logical latencies in number of tuples for (a) the LiDAR (*Ford Campus*) and (b) the Beijing GPS (*GeoLife*) dataset as a function of the respective maximum errors. The logical latency for the angle/time coordinate is displayed over the y-axis (red), as their corresponding maximum errors are constant over each of the two datasets.

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 $100 \times 5$ 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 $\alpha$ per second (see Section 2.4.1). 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.

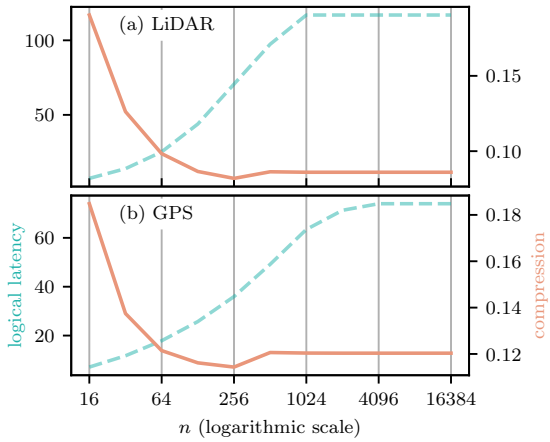



**Figure 2.21:** Average logical latency (over all channels) and compression for different maximum segment lengths $n$ ($n = 256$ is the maximum segment length chosen in this evaluation).

Figure 2.21 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.

### 2.5.7 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.

## 2.6 Related work

Clustering, as a core problem in data mining, has been extensively studied in the last decades (see e.g. the survey [76] 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 [69]-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 [77], 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 [77]. In [78], the authors design a fully streaming clustering algorithm (as the streaming version of a recently proposed clustering algorithm [79]), 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 [80] 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 [81]. 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 [82]. 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. [60, 61, 63, 64]) 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 [62] (a modification of the well-known SWAB [60] 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 [65], 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 [68], 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.

## 2.7 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.

# Chapter 3:

## Querying Large Vehicular Networks: How To Balance On-Board Workload And Queries Response Time?

---

R. Duvignau, **B. Havers**, V. Gulisano, M. Papatriantafilou

*Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC 2019), pp. 2604-2611, 2019 [1].*

PAPER B

This is a formatted and adapted version of the paper published in *Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC 2019), pp. 2604-2611, 2019*. Any performed changes serve only to retain the consistency of this thesis and to adapt to the layout.

# Abstract

Data analysis plays a key role in designing today's Intelligent Transportation Systems (ITS) and is expected to become even more important in the future. Connected vehicles, one of the main instantiations of ITS, produce large volumes of data that are difficult to gather by centralized analysis tools. The even larger volumes of data expected from autonomous driving will further exacerbate the bottleneck problem of data retrieval. When analysts issue queries that seek data from vehicles satisfying certain criteria (e.g. those driving above a certain speed or in a certain area), the problem can partially be overcome by pushing to vehicles themselves the job of checking and reporting the compliance of their local data (e.g. recorded GPS positions or CAN data), hence avoiding a costly data retrieval phase. The problem we tackle in this work consists in spreading a set of such queries over a vehicular fleet while balancing the time needed to resolve the queries and the computational overhead induced on the vehicular network. We present in this work efficient and configurable query-spreading algorithms tailored for vehicular networks. Our tunable algorithms, which we evaluate on two large sets of real-world vehicular data, outperform baseline solutions and are able to balance the trade-off between the overall on-board workload and the response time needed to receive all answers for a set of queries.

## 3.1 Introduction

One of the key triggers for further deployment of Intelligent Transportation Systems (ITS) relies on their communication capacity and on an efficient use of their available bandwidth. For instance, in the automotive industry and in academic research, Vehicular Ad Hoc Networks (VANETs) [83] have been established in the recent decade as a key-enabling technology for providing a wide range of applications such as vehicle road safety, enhanced traffic, entertainment, infotainment, and improved comfort for drivers and passengers. A central component of many applications in vehicular networks is data gathering (see [11, 12] for applications and services in VANETs), i.e., the process of collecting sensed data from a fleet of vehicles to a central point (e.g. a company's data center). Since evolving vehicles can produce several gigabytes of data per hour [6], analysis of data sensed from a large fleet becomes practically infeasible if raw data is transferred to the analysis center. Hence there is a strong need to leverage processing power on-board the vehicles [44, 68, 84, 85], or use some form of compression mechanisms (e.g. [40]) in order to decrease the volume of data on the communication network (hence reducing monetary cost associated with the analysis) and utilize possibilities for continuous, stream processing [36].

Two types of communication are most often discussed in association with VANETs: namely Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I). V2V relies on some form of short-range communication whereas V2I can be further subdivided into either communicating with *roadside units* [86] or to some *base stations* via mobile broadband. Short-range communication is a promising technology that has been greatly exploited to efficiently solve a wide range of information gathering in vehicular networks, but even though the technology is mature enough it is not yet readily available [87]. Cellular connectivity, on the other hand, is becoming standard on modern cars (now using 4G/LTE mobile infrastructure and soon 5G [88]) and all vehicles are projected to be equipped with it by 2020, while 5G connectivity is expected to be widely used in vehicles in the next couple of years. 5G will allow huge amounts of data to be transferred but at predictably higher monetary costs [87].

In this context, pushing the analysis on-board participating vehicles, thus leveraging their distributed computational power, can reduce communication overheads by avoiding central data gathering, which is of utmost importance for big data analysis to be feasible within large fleets of vehicles. This work challenges a recurring assumption in the literature about VANETs, differentiating them from other sensor networks [89, 90]: it is assumed that vehicles have "no strict limitation for processing power and storage capabilitities" contrary to other typical sensor networks made of small computing units. As more computationally heavy on-board tasks will be performed on *intelligent* vehicles and considerable amounts of data must be stored or transferred, we cannot assume any longer that those vehicles have unlimited computing power and storage space; moreover, the need for smart vehicles to run priority security services and applications [83] other than the analysis task, is an additional reason why we cannot assume unbounded computational power. Hence, solutions tailored to reducing computing resources will soon be of key importance.

**Contribution:** In this exploratory work, we present algorithms relying on broadband communication to spread a set of *queries*, a particular instance of pre-processing for big data analysis, on a fleet of vehicles. The aim in this context is to minimize the time and computational overhead needed to check whether the queries can be fulfilled. In our work, queries consist of performing some task locally on a subset of the fleet satisfying some criteria (e.g. having available data on the requested time interval, or restricted geographical position, speed, driving mode, etc.) and collecting to a centralized point a certain amount of answers from vehicles matching the query's many criteria. For example, to perform some traffic flow analysis, one can retrieve 100 vehicles' average speed among those in the fleet driving above 50 km/h within a city center during rush hour. We present here algorithms that distribute a batch of such queries to vehicles in a fashion that balances the response time to resolve all queries and the total workload induced on the vehicular network. We perform an extensive evaluation on two large real-world sets of vehicular data that allows us to simulate both the time needed to resolve different realistic ad-hoc queries and the processing time needed on-board all participating vehicles. Our main contribution is a set of tuned algorithms that produce significantly improved trade-offs compared to baseline solutions.

The paper is organized as follows. Section 3.2 introduces the system model we are using and provides definitions for queries and the vehicular network. In Section 3.3, different algorithms are provided for spreading a set of queries over a fleet of vehicles. Section 3.4 is dedicated to a thorough evaluation of the proposed algorithms on real vehicular datasets. Related work is discussed in Section 3.5, while Section 3.6 presents our conclusions and future research directions.

## 3.2 System Model and Problem Definition

### 3.2.1 System Model

In this study we consider the following model: a *fleet* $V$ of $k$ vehicles, referred to also as nodes, is equipped with multiple sensors $S = \{\text{GPS}, \text{steer}, \text{break}, \dots\}$ where each sensor $s_i \in S$ records a timestamped (multi-valued) stream of data $s_i(v) = (t_0, x_0), (t_1, x_1), \dots$ for each vehicle $v \in V$. Furthermore, all vehicles are connected to a central entity $C$, thought of as a datacenter, via a two-way communication channel. From data analysts, $C$ receives queries $q_1, \dots, q_r$, each focusing on some subset of the possible sensors for some time span of recorded data.

In more details, queries carry a specific *condition* that needs to be fulfilled by vehicles to participate in the analysis and every query specifies some number of *answers* (response to the query from distinct vehicles) that need to be collected to complete the analysis task. Hence, not every vehicle may answer every query because of lack of data or the data is found not suitable to answer that particular query; the required number of answers is meant to gather a sufficient amount of data from the vehicles to be meaningful for the analysis task at hand. In this context, we need a big enough data sample (e.g. for statistical significance or to decrease possible privacy leaks) but also small enough (because of the time to collect all the data [68], and network/CPU

time). Once the query's condition has been checked locally on a vehicle, a yes/no answer is transmitted to $C$.

A query $q$ in our model is represented by a pair $(P, n)$ where $P$ is a general condition ranging over the local sensed data and $n$ is the minimum number of positive[1] answers to collect in order to complete $q$; we will use the notation $q.P$ for the condition and $q.n$ for the minimum number of answers required to complete query $q$. The condition $P$ encodes which sensors are affected by the query and an overall condition for all concerned portions of streams to satisfy to be able to participate in answering the query. In the following: $S_q \subseteq S$ is a subset of sensors on which the query $q$ is focusing (by default $S_q = S$); $(t_{start}, t_{end})$ are time bounds spanning $q$'s time interval of interest such that only the portion of stream $\{(t, x) \in s_i \mid t_{start} \leq t \leq t_{end}\}$ for sensors $s_i \in S_q$ is examined (if non-specified, the full recorded data is considered); $(\tau, \delta)$ defines a minimum sensor sampling frequency requiring at least $\tau$ units of time of data with a maximum $\delta$ units of time between consecutive records (imposing at least $\tau/\delta$ records within the imposed timespan).

**Example Query**

To estimate traffic flow in a city center during the morning rush hour, one can ask to select $n = 100$ vehicles in a $k = 10,000$ vehicles fleet, with $P =$ "driven within area $A$ with an average speed more than 50km/h between 8 and 9 o'clock with GPS measurements spaced by at most 5s from each other". In this query, $S_q = \{\text{GPS}\}$, $(t_{start}, t_{end}) = (8{:}00, 9{:}00)$, $(\tau, \delta) = (1\text{h}, 5\text{s})$, and $A$ delimits a bounding box approximating the city center.

### 3.2.2 Problem Definition

The collection of a large enough number of answers provides an estimation of the fraction of vehicles that satisfy $q.P$ in the fleet. Such a query mechanism can be used as is in many scenarios, e.g to estimate the proportion of electric cars driving close to a potential location for a new electrical charging station. Once $C$ is aware of a positive answer, it can subsequently ask that vehicle to perform a query-dependent task. For instance, the expected outcome might be the result of an aggregate function computed locally over the portion of data that has been checked (e.g compute the average speed) or the data itself. In this work, we do not consider the query post-treatment but rather concentrate on the dissemination part, i.e., securing a set of vehicles that will participate in the query analysis. In practice, we can also carry out simple aggregation tasks (min/max, mean, etc.) during $q.P$'s processing on the vehicle. This will entail minor changes in the time needed to process $q.P$ and the aggregated value can be transmitted along the vehicle's yes/no-answer to $C$.

We suppose $C$ has no access to the vehicles' data other than through communication with them, so that the amount of work needed to test $q.P$ cannot be estimated before processing the query locally on the appropriate vehicle. Furthermore, $C$ knows exactly how to contact all vehicles in the fleet. In the likely event that data is missing for the requested time period, it is equivalent to considering that the involved node does not satisfy $q.P$ and cannot answer the

[1] "positive" or yes-answer implies that $P$ holds locally.

query. In a similar fashion, vehicles unwilling to participate in a query's task (for privacy or other reasons) are modelled by negative answers. For simplicity, the set of participants is considered constant here, hence we do not consider that new vehicles may join the network or that vehicles may leave within the time interval spent resolving a particular query (in our case study, the query processing time is low enough to entail a negligible churn of the vehicles during the execution of the algorithms). Finally, we ignore here the transmission time of the answer as we assume it to be constant and very small (one bit for a yes/no answer and other $\mathcal{O}(1)$ additional information as the vehicle id, the time it took for the processing, etc.).

### 3.2.3 Performance Metrics

We associate with each query two performance measures: (i) **Query (response) Time:** the elapsed time between receiving a query at $C$ and having collected the appropriate number of positive answers at $C$; (ii) **Analysis Cost (or total workload):** the overall computing load on the vehicles defined as the sum of individual processing times of all nodes involved in processing the query.

Notice that satisfying the 2 measures at the same time is not trivial as they go in opposite directions. More concretely, query time is minimized by simply asking all vehicles in the fleet and ignoring answers after $n$ positive answers are retrieved (requiring maximum resources) while the workload is balanced by asking 1 car at the time in a round-robin fashion (requiring maximum time). Furthermore, the amount of uncertainty in the model is challenging: each query requires different amounts of time per node that cannot be predicted[2]. For instance, on the one hand, a query of the form "has the vehicle driven within $x$ meters from a parking lot between 12:00 and 18:00?" induces that positive answers may be sent as soon as a matching GPS record is found whereas negative answers need to scan first all records within the time interval. On the other hand, a large number of negative answers for $q$ may be received at $C$ much quicker than any positive answers due to missing on-board data.

## 3.3 Query Spreading Algorithms

The challenges described in the preceding section lead us to design different strategies to disseminate a set of queries over the fleet. We present here algorithms that select a subset of nodes among the $k_q$ nodes satisfying $P$ for a *single* query $q = (P, n)$ assuming $k_q \geq n$. For a set of queries $q_1, \ldots, q_r$, each query can be resolved by executing the procedures of this section at $C$, either in parallel or sequentially.

Without further assumptions on the distribution of nodes satisfying $P$, it is natural to randomly select nodes to question in a uniform way within the pool of nodes that have not been interrogated yet. However, other factors (as the number of queries currently running on the vehicle, local computation time so

[2]One may consider here the time interval on which the query is focused as a good indicator of the amount of work needed to answer, but this would undoubtedly ignore that some properties can quickly be resolved (in case of missing data for instance) whereas some other properties might need more than linear time (e.g. when sorting is needed).

far, etc.) can be used to bias the selection process. We present in this section four algorithms focusing on different measures:

- BASELINE1 optimizes the time needed to answer $q$,
- BASELINE2 optimizes communication by interrogating new vehicles only when needed (hence no more than $n$ positive answers are received),
- BALANCED-ALGO balances the two baseline algorithms in order to collect quickly $n$ answers without inducing a too high load on the vehicular nodes, and
- FAIR-ALGO extends BALANCED-ALGO by prioritizing the least-used vehicles in the selection.

The *baseline* options introduced in this work are meant to benchmark the balanced algorithms against edge-cases (i.e., optimizing only one aspect) of the spectrum of possible trade-offs.

### 3.3.1 Simple Model Description

For the ease of presentation, we first briefly describe the first three algorithms assuming a synchronous model of query-checking so as to provide the intuition which is behind each presented algorithm. Let us assume in this section that no time is spent sending query messages from $C$ and that nodes need a constant amount of time to check the property $P$, so that after a "round" of time has elapsed, $C$ has received answers (yes/no) from all nodes that were interrogated during that round. In this simplified situation, only two aspects have to be considered in order to measure the performance of query-spreading procedures: (1) the number of rounds needed at $C$ to receive $n$ answers; (2) the number of nodes $m$ that have checked their capability to answer $q$ (which is equivalent to the total workload on the vehicles as each interrogated vehicle has spent 1 round checking the query).

#### Baseline1 (Optimal Time)

In order to optimize the processing time, all nodes have to be contacted upon $q$'s arrival; this process then solves the query in a single round. Indeed, with any other algorithm that leaves at least one node $v$ un-contacted, if $k_q = n$ and if $P$ holds on $v$, then only $k_q - 1$ answers are received after one round and a second round of communication is required to get all answers. Following this simple procedure to obtain all needed answers leads nonetheless to a potentially huge effort on the nodes. In particular, the number of interrogated nodes is always $k$ independently of $k_q$ and $n$. Hence all nodes always participate in order to resolve $q$ even though the number of required answers $n$ might be relatively small.

#### Baseline2 (Optimal Load)

In the opposite direction of potential solutions to our problem, there is an algorithm focusing on the computational overhead induced on the nodes. To

ensure that only the minimum number of nodes are being asked to process $q$, we should only question at most as many new nodes as the number of missing answers. Any algorithm satisfying such an assertion is associated with a minimal analysis cost as interrogating more nodes might lead to the reception of more than $n$ answers, hence involving a higher than necessary analysis cost. The best algorithm in this category selects randomly as many nodes as possible by interrogating $m \leq n$ "new nodes" at each round when there are $m$ missing answers. When $n$ positive nodes are found, the procedure stops. The worst processing time is $k - n + 1$ rounds where $(n - 1)$ positive answers are collected in the first round and the last answer is obtained after questioning every other vehicle in the subsequent rounds. However, the average is much lower (in the order of $\log_{k/(k-k_q)}(n)$ rounds) and the average number of interrogated nodes in $\mathcal{O}(n \log(n))$.

**Balanced-Algo**

BASELINE1 needs only one round but all nodes end up being questioned to process $q$, whereas BASELINE2 communicates with a minimum number of nodes but requires a longer time to collect all answers. We present here an efficient scheme to achieve a reduced computing load on the network within a few processing rounds. The main idea behind BALANCED-ALGO is to evaluate the proportion of vehicles that are able to answer positively to a query by measuring it through the execution of the algorithm. A first sample probability of $p = a/n$ is obtained by randomly questioning $n$ vehicles and getting $a$ positive answers; subsequently, $\ell = \alpha \cdot (n - a)/p$ vehicles are randomly interrogated so that if $p$ was a good estimation, very few vehicles will remain to interrogate during next round (when $\alpha = 1$) and we repeat this process until all $n$ answers have been collected. The parameter $\alpha$ allows us to depart from the estimated expected number $(n - a)/p$ of vehicles to interrogate to get the remaining answers, by sampling more or fewer vehicles. This parameter allows us to either shorten (when $\alpha > 1$) the average number of rounds needed to resolve $q$ while potentially increasing the analysis cost, or on the contrary (when $\alpha < 1$) to slow down $q$'s processing by being more prudent and avoiding questioning more vehicles than necessary (and by this way getting closer to receiving exactly $n$ answers at the end). In the case that no positive answers have yet been received, we evaluate the probability of positive answers to $p = 1/(m + 1)$ where $m$ is the number of nodes interrogated so far.

### 3.3.2 General Model

We now generalize the algorithms of the previous section to our general query-answering model, i.e., when query processing time is vehicle- and context-dependent. In our typical vehicular setting, we cannot in general assume bounds on the time a vehicle needs to process a query and on the communication network.

**Baseline1**

optimizes the time needed to answer a single query $q$ by interrogating all vehicles upon receiving it. In our general model, the query time then needed

for $q$ is the best possible and corresponds to the $n$-th fastest positive answer received at $C$. In contrast to that, the analysis cost is the highest possible, as all $k$ vehicles have processed $q$.

**Baseline2**

optimizes the number of interrogated vehicles (hence minimizing needed communication to spread queries) by questioning a new vehicle only if strictly needed. This is achieved by interrogating $n$ vehicles and asking new vehicles to check $P$ only upon receiving a negative answer. Since in our general model it is not guaranteed that vehicles will have similar answer times (and it is even expected the opposite), this algorithm does not necessary imply a minimum load on the network (i.e., it might be that questioning more vehicles that need a short processing time to answer will require less resources overall).

**Balanced-Algo**

**Algorithm 2** BALANCED-ALGO

**Input:** parameters $\alpha, \beta > 0$ and query $q$ with $n = q.n$
**Output:** $n$ answers to query $q$

$p \leftarrow 1$ ▷ estimation of probability to answer yes
$S \leftarrow \emptyset$ ▷ set of interrogated vehicles
$A \leftarrow \emptyset$ ▷ set of answers collected
$R \leftarrow \emptyset$ ▷ set of collected positive answers
**while** $|R| < n$ **do**
  $\ell \leftarrow \lceil \alpha \cdot (n - |R| - (|S| - |A|) \cdot p) \cdot p^{-1} \rceil$
  i $\leftarrow 0$
  **while** $i < \ell$ and $|S| < k$ **do**
    $v \leftarrow \text{random}(S)$ ▷ random vehicle excluding $S$
    $\text{send}(q, v)$ ▷ send query $q$ to vehicle $v$
    $S \leftarrow S \cup \{v\}$
    $i \leftarrow i + 1$
  **end while**
  **if** $|S| = k$ **then**
    **output** $R$ ▷ set of possible vehicles exhausted
  **end if**
  **repeat**
    $r \leftarrow \text{receive}()$ ▷ Block till receiving next answer
    $A \leftarrow A \cup \{r\}$
    **if** $\text{positive}(r)$ **then**
      $R \leftarrow R \cup \{r\}$
    **end if**
  **until** $\frac{|A|}{|S|} \geq \beta$
  $p \leftarrow \max\{\frac{|R|}{|A|}, \frac{1}{|A|+1}\}$
**end while**
**output** $R$

Algorithm 2 is similar in essence to its round-based version described in Section 3.3.1 but needs to be tuned to take into account that all nodes do not reply at the same time (after 1 round in the aforementioned version). To do

so, we wait to receive a certain proportion $\beta$ of answers over all interrogated vehicles before proceeding to the next batch of selection (and re-evaluating the probability to answer positively). When $\beta = 1$, the algorithm waits for the reception of all answers before continuing; setting a lower value for $\beta$ allows us to take a decision without having to wait for the slowest vehicles. Another change is about taking into account vehicles that have not yet answered when a new iteration has started. Based on previous answers, we evaluate that a fraction of $(|S| - |A|) \cdot p$ will answer positively, where $|S| - |A|$ counts the number of queried vehicles that have not answered yet, so we remove those *expected* answers when calculating the number of remaining answers to collect.

#### Fair-Algo

We suggest a variation of the previous algorithm, differing on how we select vehicles once we know the number $\ell$ of vehicles to interrogate in the next batch. Instead of randomly selecting new nodes to interrogate, vehicles having low local workload are picked first in the selection phase. The main difference with Algorithm 2 is the function `random(S)` that selects a vehicle among the not yet interrogated ones. Instead of selecting randomly, vehicles are selected in the order of their lowest *local workload* measured as (1) number of simultaneous queries running on the vehicle and (2) reported local processing time since the start. Vehicles are for that purpose stored in an updatable priority queue where vehicle $v$'s priority is updated upon sending a new query to $v$ and receiving $v$'s answer (containing $v$'s local processing time).

## 3.4 Evaluation

To attest to the performance of the proposed algorithms, we evaluate them on two large real-world sets of vehicular data. In this section, we first describe the datasets and how the data has been pre-processed, then present a set of common queries that will serve to benchmark the different algorithms, and finally show the experiments' results.

### 3.4.1 Experimental Setup

#### Beijing Dataset

The first dataset consists of trajectories collected within the scope of the (Microsoft Research Asia) *Geolife* (version 1.3) project by 182 users over approximately four years [72]. The trajectories were collected from diverse users using different mobile devices and feature predominantly vehicular usage (by car, taxi, or bus). The original dataset consists of 18,670 GPS traces containing between 50 and 92,645 records of the form timestamp (s), latitude (deg), longitude (deg). After pre-processing the data, we used 9,772 files each for one daily usage of one user.

#### Volvo Dataset

The second dataset is made of CAN data and GPS traces from 20 hybrid cars collected in Sweden in the years 2014 and 2015 by *Volvo Cars Corporation* [68].

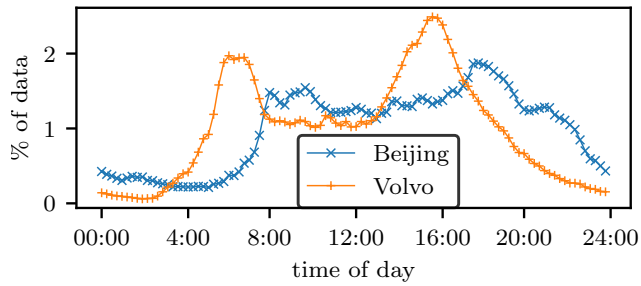



**Figure 3.1:** Distribution of records in both datasets during 1 day.

After pre-processing, we use 3246 traces corresponding each to a daily usage of one vehicle. Among the large quantity of CAN data, we have concentrated on two signals, the combustion engine rotation and electric engine rotation. These can be combined, leading to three possible driving modes: electric (*elec.*), combustion (*comb.*), hybrid (*hybr.*).

### Simulation Setup

In our experiments, we will simulate a fleet consisting of 9772 (for Beijing) and 3246 (for Volvo) vehicles; each simulated vehicle will have one day of data stored on it. The distribution of data over a day for both datasets is depicted in Figure 3.1. To evaluate our algorithms, we define 15 queries to be run locally on the vehicles (presented in the following section). The queries are programs written in Python that are transferred to the vehicle via mobile broadband communication, then executed *on-board* the vehicle over their already stored data (1 day each). We note size($q$) the amount of code and data[3] that needs to be transferred from $C$ to each vehicle in order for the latter to be able to process $q$ on-board. The elapsed time $T$ (in milliseconds) needed between the coordinator sending a query message to a vehicle $v$ and the reception of the corresponding answer is approximated as $T = T_l + \text{size}(q)/R + T_p(v, q)$ where $T_l$ is a fixed round-trip latency for wireless communication, $R$ is a constant standing for the wireless link data rate, and finally $T_p(v, q)$ is the time needed by the vehicle to decide if it can answer $q$ or not; recall here, the transfer of the response message (yes/no answer and the small pieces of information about the vehicle) is neglected (cf. Section 3.2.2). In our set of experiments, we have set $T_l = 50$ms and $T_d = 10$Mb/s which are within current 4G/LTE latency and download rates (similar results are obtained using 5G parameters). To have a fair estimation $T_p(v, q)$, we have computed all queries on a vehicular processing unit representative [68]: an ODROID-XU3 single-board computer to approximate the low-power processor of a vehicle, equipped with a Samsung Exynos 5422 (Cortex-A15 2.1GHz Quad-Core and 1.4GHz Quad-Core CPUs) and 2 GB of LPDDR3 RAM at 933 MHz. We then use the computed time measured on the vehicular stand-in to simulate $T_p(v, q)$ for every possible vehicle $v$ and query $q$. Based on the measured transfer time (through an Ethernet link with software-capped bandwidth to $T_d = 10$Mb/s), size($q$)/$T_d$ expressed in ms is very well approximated by the size of data to transfer expressed in Kb.

[3] e.g. GPS positions of stationary elements as parking lots or fuel stations.

The coordinator receives a certain number of queries in a random uniform order, and starts the batch of interrogations in the same order as the queries arrival's one. The queries are then answered in parallel by the vehicles and the coordinator reacts to each message arrival by either just updating its internal statistics for the corresponding query or by sending a new batch of interrogations to a new set of vehicles. If a vehicle $v$ receives a new query task $q'$ while already processing a previously received query $q$, then $q'$ is added to $v$'s local *task queue*; once $v$ has terminated its processing with $q$, $v$'s task queue is then processed in FIFO order. This approach simplifies the vehicles' internal computing architecture and is well suited in situations for which the remaining computing resources on-board the vehicles (if any) can be used to process security-sensitive applications.

### 3.4.2 Selected Queries

We introduce here a set of 15 queries, representative of possible vehicular analysis tasks. Table 3.1 presents (cf. Section 3.2 for notations) the query $q$ key ($Q_1$ to $Q_{15}$), the time interval $t_{start} - t_{end}$ given in hours, size(q) given in Kb, the description of the condition $q.P$, and the average answer rate (rounded to closest percentage) for Beijing (B%) and Volvo (V%) datasets. We use three possible values for $(\tau, \delta)$: $\Delta_0 = (60, 5)$, $\Delta_1 = (45, 10)$ and $\Delta_2 = (80, 10)$, all values being in seconds.

Of the queries, 10 are run over both datasets whereas 5 additional queries focus on signals only contained within the *Volvo* dataset. The queries were chosen to represent different requirements (on time interval, queried sensors,

| **Key** | **Time** | **Size (Kb)** | **Condition for *Beijing*** | ***Volvo*** | **B%** | **V%** |
|---|---|---|---|---|---|---|
| $Q_1$ | 8-12 | 0.3 | at least 1 record | | 55 | 47 |
| $Q_2$ | 0-24 | 7.5 \| 19.9 | one parking within 50m | 25m | 43 | |
| $Q_3$ | 17-18 | 1.3 | $\Delta_0$ | $\Delta_1$ | 28 | |
| $Q_4$ | 0-24 | 0.5 | Passed by City | $\Delta_2$ | 19 | |
| $Q_5$ | 17-18 | 1.1 | Max speed $\geq$ 80km/h | 68km/h | 14 | |
| $Q_6$ | 0-24 | 1.6 | Speed $\geq$42km/h for 15min | 22min | 6 | |
| $Q_7$ | 0-24 | 1.4 | Driven in Downtown, $\Delta_0$ | $\Delta_2$ | 5 | |
| $Q_8$ | 17-18 | 0.8 | Passed by City | | 3 | 2 |
| $Q_9$ | 12-13 | 0.7 | Stayed in City | | 1 | |
| $Q_{10}$ | 0-24 | 3.8 \| 7.7 | *Stopped* at some Gas stations* | | 1 | |
| $Q_{11}$ | 0-24 | 4.8 | *comb.* (alone) used 50% of the time | | 15 | |
| $Q_{12}$ | 0-24 | 5 | Driven on *elec.* only outside City | | 13 | |
| $Q_{13}$ | 0-24 | 4.4 | Have used *hybr.* mode for 10min | | 10 | |
| $Q_{14}$ | 0-24 | 6.6 | *elec.* speed reached over 100km/h | | 7 | |
| $Q_{15}$ | 0-24 | 4 | 3 diff. charging stations on *elec.* | | 4 | |

* In *Beijing* dataset, we require $\Delta_0$ records within 50m of a fuel station, and in the *Volvo* dataset we require that the vehicle stopped (speed=0km/h) for 10min within 20m of a fuel station.

**Table 3.1:** Selected query conditions and their parameters.

geographic constraints, sampling constraints, etc), have distinct positive answer rates ranging from about 50% to about 1%, and finally give meaningful insights into the fleet's behavior. The parameters of the first 10 queries have been slightly tuned between the two datasets (in Table 3.1 the *Volvo* column indicates differing parameter in the query's condition) so that each query in both datasets has a similar fraction of positive answers. Two geographical zones are defined for both datasets: *City* is the area of a large city chosen within the dataset and *Downtown* is a sub-area within *City* thought of as its heart. In our experiments, all queries require $n = 50$ answers to get resolved.

### 3.4.3 Experiments

#### Query Answer Rates

The average answer rates over all queries as well as the distribution of the data volume is presented in Figure 3.2 for the *Volvo* (solid) and *Beijing* dataset (dashed lines); 100% on the $x$-axis corresponds to the vehicle with the largest data volume among all vehicles in the respective dataset. For *Beijing*, the average query answer rate (blue, dashed) appears to be positively linked to the data volume; thus, vehicles with larger amount of data will have a higher chance to answer queries. For *Volvo* (blue, solid), the average query answer rate is almost flat, which indicates that vehicles with a large amount of data are roughly as likely to answer a query as vehicles with only little data. Concerning the distribution of data volume among the fleet, *Beijing* dataset shows an exponential distribution whereas *Volvo* presents a logistic distribution with a long tail.

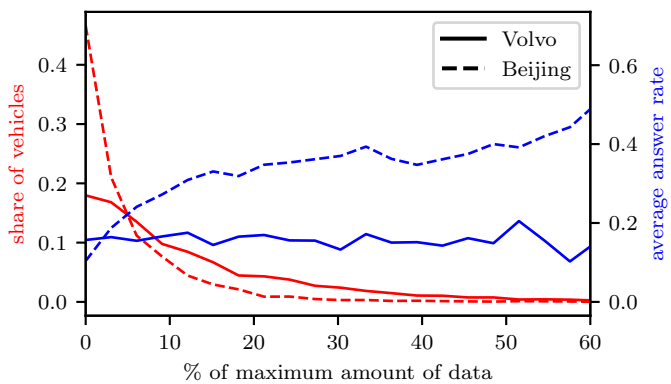



**Figure 3.2:** Distribution of data volumes (left vertical axis) and average query answer rates (right vertical axis) for *Volvo* (solid) and *Beijing* (dashed line) dataset.

#### Parameter Exploration $\alpha, \beta$

To choose well-fitted parameters for our evaluation, we explore the parameter space for Balanced-Algo in the *Beijing* dataset. We run 1000 times the query set in our query simulator with different values for the parameter $\alpha$ (proportion of vehicles to ask, higher value translates to interrogating more

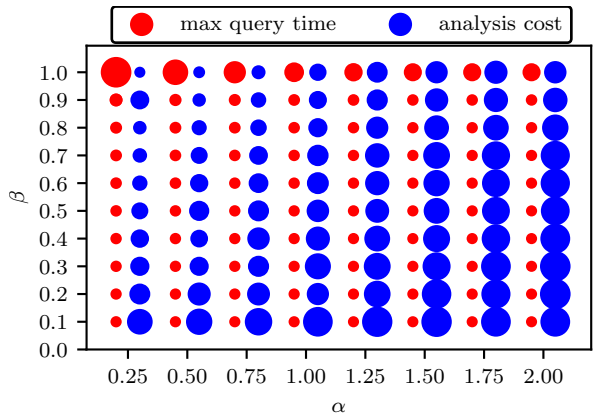


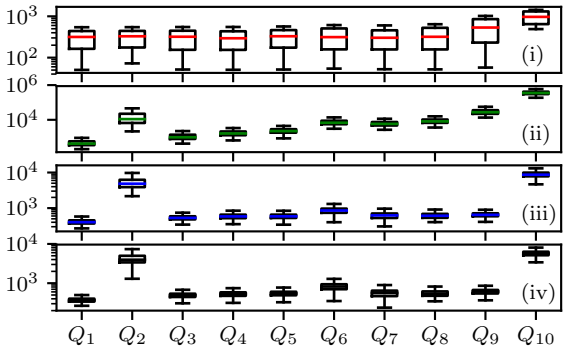

**Figure 3.3:** Maximum query response time and analysis cost needed to resolve $Q_1 - Q_{10}$ over the *Beijing* dataset for BALANCED-ALGO for different $\alpha, \beta$. Circle size scales with maximum time (red) and total workload (blue), respectively.

vehicles) and $\beta$ (fraction of vehicles to wait before sending next batch, higher fraction translates to longer waiting time between two query batches). For each run, we measure the time needed to answer all queries (i.e., the maximum query resolution time among the query set) and the analysis cost (total workload on the system measured as the sum of the processing time for each participating vehicle) and present them on a 2D plot in Figure 3.3 (note absolute values are given in Figure 3.4).

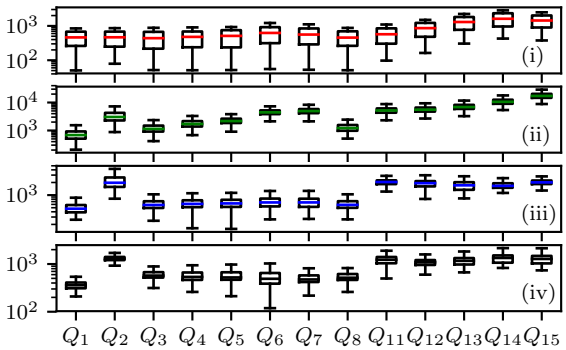
This exploration shows that for most values of $\beta$ up to 0.9, the maximum time is similar (independently of $\alpha$) but increases significantly for $\beta = 1$ due to waiting for the slowest vehicles to answer before asking a new set of vehicles (here $\alpha$ decreases the maximum query time when set to higher values). The total workload is, as expected, minimized when the algorithm asks few vehicles and waits for all answers before the next batch ($\alpha = 0.25$ and $\beta = 1$), whereas it increases if either $\beta$ decreases (waiting fewer vehicles might lead to overestimate the proportion of negative answers) or $\alpha$ increases (questioning more vehicles will lead to receiving more than the required amount of answers and some vehicles might have thus processed unnecessarily the query). We set in the following $\alpha = 0.25$, $\beta = 0.9$ for BALANCED-ALGO and FAIR-ALGO, resulting in optimal trade-offs.

### Comparison of the Different Algorithms

To give a general idea of the response time for the different queries, we present in Figure 3.4 the query response time measured over 1000 experiments consisting in resolving all 10/15 queries arriving in a random order; for the *Volvo* dataset, $Q_9$ and $Q_{10}$ have been removed as all vehicles end up being asked (there are fewer than 50 positive answers in this case). The interesting facts to note are: BASELINE1's response time is short and varies the least among queries, BASELINE2 has the largest response times, clearly depending on the queries' answer rate (lower rate is associated with larger response times), BALANCED-ALGO and FAIR-ALGO present similar query times that do not vary significantly with the queries' answer rate. We also note that computationally heavier queries ($Q_2$,




**(a)** *Beijing* dataset




**(b)** *Volvo* dataset

**Figure 3.4:** Query response time (in ms) for all valid queries executed over the (a) *Beijing* and (b) *Volvo* dataset for (i) BASELINE1, (ii) BASELINE2, (iii) BALANCED-ALGO, (iv) FAIR-ALGO.

$Q_{10}$, and $Q_{11} - Q_{15}$) get resolved much slower than lightweight queries. As a summary, Figure 3.5 presents the average time (left side) and total work (right side) over all queries for the four algorithms relative to the average response time of BASELINE1 (over all queries) and the average workload of BASELINE2, respectively. The *average time* to answer queries is up to 2 orders of magnitude higher in BASELINE2 (in the *Beijing* dataset) compared to BASELINE1, whereas the proposed algorithms are about 1-4 times slower than BASELINE1; FAIR-ALGO performs really well in the *Volvo* dataset being close to the optimal solution. The *average analysis* cost is in the order of 3-5 times larger in BASELINE1 compared to BASELINE2. BALANCED-ALGO performs well on both datasets, having an average cost very close to the baseline. Finally, FAIR-ALGO's analysis cost is dependent on the distribution of data in the fleet: with skewed data (as in the *Beijing* dataset), it completely outperforms BALANCED-ALGO, whereas in a uniformly spread dataset (e.g. *Volvo* one) FAIR-ALGO presents a relatively poor choice (with performance varying between the two baseline algorithms).

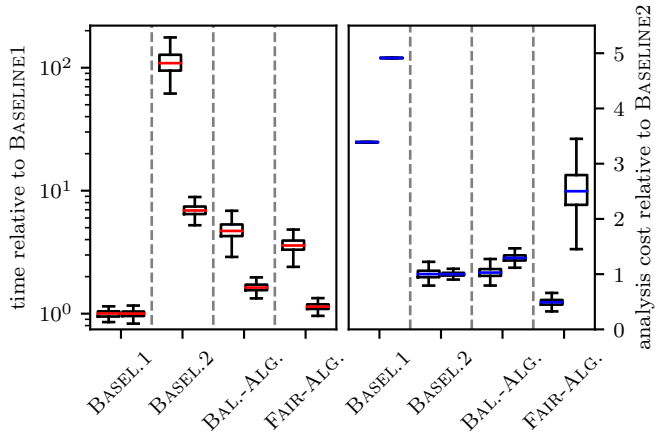



**Figure 3.5:** The average relative time (left) and total work (right) between the four algorithms for the *Beijing* dataset (left boxplots in each column) and *Volvo* dataset (right boxplots).

### Summary of the Results

The presented solutions (well-tuned BALANCED-ALGO and FAIR-ALGO) provide large improvements in the trade-off of query response time versus on-board workload compared to baseline solutions. Furthermore, a query's response time in the proposed algorithms is not negatively impacted by a low positive answer rate among the fleet. The new solutions perform also better on different data distributions among the fleet: BALANCED-ALGO is more suited to a uniform distribution of positive answers (*Volvo* dataset) whereas FAIR-ALGO for a skewed distribution of positive answers (*Beijing* dataset).

## 3.5 Related Work

The traditional approach to query a set of vehicles has been through SQL-inspired languages [91–93] to process continuous queries on live vehicular sensors' data. Two main differences with the current work are that in previous works (i) "queries" were usually initiated by vehicles themselves (see e.g. [94–97]) and (ii) the full fleet was queried upon receiving new queries (as in [98]), contrary to our work where a known and fixed set of *general* queries is deployed from the coordinator to the fleet.

Query-answering mechanisms for vehicular networks in the literature also predominantly concentrate on using the architecture of the network (for instance using pre-existing P2P approaches, as in [29–31] or 2-tier architectures [99,100]) to resolve the query. In this work, we do not presume any connections between vehicles; that positions our work in readily deployable technologies on modern vehicles.

In vehicle data analysis, privacy aspects are important when dealing with e.g. location-based services (see e.g. [101–103]). We suggest that our work, by allowing to check whether a certain number (chosen by the analyst) of

vehicles meets a given condition, can complement applications where privacy is supported by aggregating data from many sources.

## 3.6 Conclusions

We presented algorithms able to spread a batch of queries over a set of vehicles, in a balanced fashion in terms of the time needed to resolve the queries and the on-board workload on the vehicles. We further presented a simulation environment, that we used to evaluate our algorithms, using real traces of accumulated data and realistic query processing times for vehicles. One future direction is to consider computationally heavier queries that necessitate longer processing times on the vehicles; this will require taking into consideration the churn rate of the vehicle set (since some vehicles might be turned off during the query resolution time). It will also be interesting to investigate the benefits of embedding V2V-related methodology, such as [104]. Finally, this work lays a first stone on the path to produce a complete on-board query simulation (producing simulated query answers and simulated query response times) that could be integrated with traffic simulations to be able to produce a fully simulated environment for vehicular data analysis benchmarking.

# Bibliography


[1] R. Duvignau, B. Havers, V. Gulisano, and M. Papatriantafilou, "Querying Large Vehicular Networks: How to Balance On-Board Workload and Queries Response Time?" in Proceedings of the IEEE Intelligent Transportation Systems Conference. IEEE, 2019, pp. 2604–2611.

[2] D. Reinsel, J. Gantz, and Rydning. (2018, November) The Digitization of the World - From Edge to Core. Accessed 2020-09-01. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[3] World Economic Forum. (2012, January) Big Data, Big Impact: New Possibilities for International Development. Accessed 2020-09-01. [Online]. Available: http://www3.weforum.org/docs/WEF_TC_MFS_BigDataBigImpact_Briefing_2012.pdf

[4] W. J. Fleming, "Overview of automotive sensors," IEEE Sensors Journal, vol. 1, no. 4, pp. 296–308, 2001.

[5] W. J. Fleming, "New Automotive Sensors—A Review," IEEE Sensors Journal, vol. 8, no. 11, pp. 1900–1921, 2008.

[6] R. Coppola and M. Morisio, "Connected Car: Technologies, Issues, Future Trends," ACM Computing Surveys, vol. 49, no. 3, pp. 1–36, 2016.

[7] W. Xu, H. Zhou, N. Cheng, F. Lyu, W. Shi, J. Chen, and X. Shen, "Internet of vehicles in big data era," IEEE/CAA Journal of Automatica Sinica, vol. 5, no. 1, pp. 19–35, 2018.

[8] M. Johanson, S. Belenki, J. Jalminger, M. Fant, and M. Gjertz, "Big Automotive Data: Leveraging large volumes of data for knowledge-driven product development," in Proceedings of the IEEE International Conference on Big Data, 2014, pp. 736–741.

[9] P. Bedi and V. Jindal, "Use of Big Data technology in Vehicular Ad-hoc Networks," in Proceedings of the International Conference on Advances in Computing, Communications and Informatics. IEEE, 2014, pp. 1677–1683.

[10] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, "Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations," Vehicle System Dynamics, vol. 44, no. 7, pp. 569–590, 2006.

[11] E. Schoch, F. Kargl, M. Weber, and T. Leinmuller, "Communication patterns in VANETs," IEEE Communications Magazine, vol. 46, no. 11, pp. 119–125, 2008.

[12] M. Gerla and L. Kleinrock, "Vehicular networks and the future of the mobile internet," Computer Networks, vol. 55, no. 2, pp. 457–469, 2011.

[13] McKinsey. (2016, March) Car data: paving the way to value-creating mobility. Accessed 2020-08-29. [Online]. Available: https://www.mckinsey.com/~/media/McKinsey/Industries/Automotive%20and%20Assembly/Our%20Insights/Creating%20value%20from%20car%20data/Creating%20value%20from%20car%20data.pdf

[14] PWC. (2017, Sep) The 2017 Strategy& Digital Auto Report. Accessed 2020-08-29. [Online]. Available: https://www.strategyand.pwc.com/gx/en/insights/2017/fast-and-furious/2017-strategyand-digital-auto-report.pdf

[15] C. Greer, M. Burns, D. Wollman, and E. Griffor. (2019, March) NIST Special Publication: Cyber-Physical Systems and Internet of Things. Accessed 2020-09-01. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1900-202.pdf

[16] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in Proceedings of the Design Automation Conference, 2010, pp. 731–736.

[17] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, M. Papatriantafilou, and A. C. Koppisetty, ""DRIVEN: A framework for efficient data retrieval and clustering in vehicular networks"," Future Generation Computer Systems, vol. 107, pp. 1 – 17, 2020.

[18] IEEE. (2010) IEEE 802.11p-2010 - IEEE Standard for Information technology - Wireless access in vehicular environments. Accessed 2020-10-02. [Online]. Available: https://standards.ieee.org/standard/802_11p-2010.html

[19] NVIDIA. (2020) NVIDIA DRIVE AGX Developer Kit. Accessed 2020-09-02. [Online]. Available: https://developer.nvidia.com/drive/drive-agx

[20] J. H. Gawron, G. A. Keoleian, R. D. De Kleine, T. J. Wallington, and H. C. Kim, "Life cycle assessment of connected and automated vehicles: sensing and computing subsystem and vehicle level effects," Environmental Science & Technology, vol. 52, no. 5, pp. 3249–3256, 2018.

[21] N. Cheng, F. Lyu, J. Chen, W. Xu, H. Zhou, S. Zhang, and X. Shen, "Big Data Driven Vehicular Networks," IEEE Network, vol. 32, no. 6, pp. 160–167, 2018.

[22] L. Carafoli, F. Mandreoli, R. Martoglia, and W. Penzo, "Evaluation of Data Reduction Techniques for Vehicle to Infrastructure Communication Saving Purposes," in Proceedings of the International Database Engineering & Applications Symposium. ACM, 2012, p. 61–70.

[23] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," Mobile Networks and Applications, pp. 1–24, 2020.

[24] K. Zhang, Y. Mao, S. Leng, A. Vinel, and Y. Zhang, "Delay constrained offloading for Mobile Edge Computing in cloud-enabled vehicular networks," in Proceedings of the International Workshop on Resilient Networks Design and Modeling. IEEE, 2016, pp. 288–294.

[25] Max Peterson et. al. (2020) BADA –On-board Off-board Distributed Data Analytics. Accessed 2020-10-03. [Online]. Available: https://www.vinnova.se/globalassets/mikrosajter/ffi/dokument/slutrapporter-ffi/effektiva-och-uppkopplade-transporter-rapporter/2016-04260eng.pdf

[26] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in Proceedings of the IEEE International Conference on Data Engineering. IEEE, 2006, pp. 49–49.

[27] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, and U. Ramachandran, "MCEP: A mobility-aware complex event processing system," ACM Transactions on Internet Technology, vol. 14, no. 1, pp. 1–24, 2014.

[28] X. Fei, N. Shah, N. Verba, K.-M. Chao, V. Sanchez-Anguix, J. Lewandowski, A. James, and Z. Usman, "CPS data streams analytics based on machine learning for Cloud and Fog Computing: A survey," Future Generation Computer Systems, vol. 90, pp. 435–450, 2019.

[29] C.-L. Liu, C.-Y. Wang, and H.-Y. Wei, "Cross-layer mobile chord P2P protocol design for VANET," International Journal of Ad Hoc and Ubiquitous Computing, vol. 6, no. 3, pp. 150–163, 2010.

[30] T. Wang, L. Song, and Z. Han, "Collaborative data dissemination in cognitive vanets with sensing-throughput tradeoff," in Proceedings of the IEEE International Conference on Communications in China. IEEE, 2012, pp. 41–45.

[31] N. Kumar and J.-H. Lee, "Peer-to-peer cooperative caching for data dissemination in urban vehicular communications," IEEE Systems Journal, vol. 8, no. 4, pp. 1136–1144, 2013.

[32] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," ACM Sigmod Record, vol. 34, no. 4, pp. 42–47, 2005.

[33] V. Gulisano, D. Palyvos-Giannas, B. Havers, and M. Papatriantafilou, "The Role of Event-Time Order in Data Streaming Analysis," in

Proceedings of the ACM International Conference on Distributed and Event-based Systems. ACM, 2020, p. 214–217.

[34] G. Pandey, J. R. McBride, and R. M. Eustice, "Ford campus vision and lidar data set," The International Journal of Robotics Research, vol. 30, no. 13, pp. 1543–1552, 2011.

[35] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 36, no. 4, 2015.

[36] S. Costache, V. Gulisano, and M. Papatriantafilou, "Understanding the data-processing challenges in Intelligent Vehicular Systems," in Proceedings of the IEEE Intelligent Vehicles Symposium. IEEE, 2016, pp. 611–618.

[37] T. Suel, "Delta Compression Techniques". Springer International Publishing, 2018, pp. 1–8.

[38] K.-P. Chan and A. W.-C. Fu, "Efficient time series matching by wavelets," in Proceedings of the IEEE International Conference on Data Engineering. IEEE, 1999, pp. 126–133.

[39] F. Eichinger, P. Efros, S. Karnouskos, and K. Böhm, "A time-series compression technique and its application to the smart grid," The VLDB Journal, vol. 24, no. 2, pp. 193–218, 2015.

[40] R. Duvignau, V. Gulisano, M. Papatriantafilou, and V. Savic, "Streaming Piecewise Linear Approximation for Efficient Data Management in Edge Computing," in Proceedings of the ACM/SIGAPP Symposium On Applied Computing. ACM, 2019, pp. 593–596.

[41] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: a novel symbolic representation of time series," Data Mining and knowledge discovery, vol. 15, no. 2, pp. 107–144, 2007.

[42] H. Najdataei, Y. Nikolakopoulos, V. Gulisano, and M. Papatriantafilou, "Continuous and Parallel LiDAR Point-Cloud Clustering," in Proceedings of the IEEE International Conference on Distributed Computing Systems. IEEE, 2018, pp. 671–684.

[43] S. Yousefi, M. S. Mousavi, and M. Fathy, "Vehicular ad hoc networks (VANETs): challenges and perspectives," in Proceedings of the IEEE International Conference on ITS Telecommunications. IEEE, 2006, pp. 761–766.

[44] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "GeneaLog: Fine-Grained Data Streaming Provenance at the Edge," in Proceedings of the ACM Middleware Conference. ACM, 2018, pp. 227–238.

[45] J. Zhou, R. Q. Hu, and Y. Qian, "Scalable distributed communication architectures to support advanced metering infrastructure in smart grid," IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 9, pp. 1632–1642, 2012.

[46] A. Keramatian, V. Gulisano, M. Papatriantafilou, P. Tsigas, and Y. Nikolakopoulos, "MAD-C: Multi-stage Approximate Distributed Cluster-Combining for Obstacle Detection and Localization," in Proceedings of the European Conference on Parallel Processing. Springer, 2018, pp. 312–324.

[47] J. van Rooij, V. Gulisano, and M. Papatriantafilou, "LoCoVolt: Distributed Detection of Broken Meters in Smart Grids through Stream Processing," in Proceedings of the ACM International Conference on Distributed and Event-based Systems. ACM, 2018, pp. 171–182.

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

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

[50] V. Gulisano, Y. Nikolakopoulos, D. Cederman, M. Papatriantafilou, and P. Tsigas, "Efficient Data Streaming Multiway Aggregation Through Concurrent Algorithmic Designs and New Abstract Data Types," ACM Transactions on Parallel Computing (TOPC), vol. 4, no. 2, pp. 11:1–11:28, 2017.

[51] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear Road: A Stream Data Management Benchmark," in Proceedings of the VLDB Endowment. VLDB Endowment, 2004, pp. 480–491.

[52] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker, "Fault-tolerance in the Borealis Distributed Stream Processing System," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2005, pp. 13–24.

[53] E. Kalyvianaki, M. Fiscato, T. Salonidis, and P. Pietzuch, "THEMIS: Fairness in Federated Stream Processing Under Overload," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2016, pp. 541–553.

[54] Y. Ji, H. Zhou, Z. Jerzak, A. Nica, G. Hackenbroich, and C. Fetzer, "Quality-Driven Continuous Query Execution over Out-of-Order Data Streams," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2015, pp. 889–894.

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

[56] B. Babcock, S. Babu, R. Motwani, and M. Datar, "Chain: Operator Scheduling for Memory Minimization in Data Stream Systems," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2003, pp. 253–264.

[57] B. Babcock, M. Datar, and R. Motwani, "Load shedding for aggregation queries over data streams," in Proceedings of the IEEE International Conference on Data Engineering. IEEE, April 2004, pp. 350–361.

[58] M. Datar and R. Motwani, "The sliding-window computation model and results," in Data Streams. Springer, 2007, pp. 149–167.

[59] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in Proceedings of the VLDB Endowment. VLDB Endowment, 2003, pp. 309–320.

[60] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in Proceedings of the IEEE International Conference on Data Mining. IEEE, 2001, pp. 289–296.

[61] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref, and W. Zwaenepoel, "Online piece-wise linear approximation of numerical streams with precision guarantees," in Proceedings of the VLDB Endowment. VLDB Endowment, 2009, pp. 145–156.

[62] E. Berlin and K. Van Laerhoven, "An on-line piecewise linear approximation technique for wireless sensor networks," in Proceedings of the IEEE Local Computer Network Conference. IEEE, 2010, pp. 905–912.

[63] Q. Xie, C. Pang, X. Zhou, X. Zhang, and K. Deng, "Maximum error-bounded Piecewise Linear Representation for online stream approximation," The VLDB Journal, vol. 23, no. 6, pp. 915–937, 2014.

[64] G. Luo, K. Yi, S.-W. Cheng, Z. Li, W. Fan, C. He, and Y. Mu, "Piecewise linear approximation of streaming time series data with max-error guarantees," in Proceedings of the IEEE International Conference on Data Engineering. IEEE, 2015, pp. 173–184.

[65] F. Grützmacher, B. Beichler, A. Hein, T. Kirste, and C. Haubelt, "Time and Memory Efficient Online Piecewise Linear Approximation of Sensor Signals," Sensors, vol. 18, no. 6, p. 1672, 2018.

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

[67] R. B. Rusu, "Semantic 3D object maps for everyday manipulation in human living environments," KI-Künstliche Intelligenz, vol. 24, no. 4, pp. 345–348, 2010.

[68] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, A. C. Koppisetty, and M. Papatriantafilou, "DRIVEN: a framework for efficient Data Retrieval and clusterIng in VEhicular Networks," in Proceedings of the IEEE International Conference on Data Engineering. IEEE, 2019, pp. 1850–1861.

[69] M. Ester, H.-P. Kriegel, J. Sander, X. Xu et al., "A density-based algorithm for discovering clusters in large spatial databases with noise." in SIGKDD Conference on Knowledge Discovery and Data Mining. ACM, 1996, pp. 226–231.

[70] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in O(N log N)," in Proceedings of the IEEE Symposium on Interactive Ray Tracing. IEEE, 2006, pp. 61–69.

[71] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on GPS data," in Proceedings of the International Conference on Ubiquitous Computing. ACM, 2008, pp. 312–321.

[72] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory." IEEE Data Engineering Bulletin, vol. 33, no. 2, pp. 32–39, 2010.

[73] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in Proceedings of the International Conference on World Wide Web. ACM, 2009, pp. 791–800.

[74] M. A. Eriksen, "Trickle: A Userland Bandwidth Shaper for UNIX-like Systems." in Proceedings of the USENIX Annual Technical Conference, FREENIX Track. USENIX, 2005, pp. 61–70.

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

[76] A. K. Jain, "Data clustering: 50 years beyond K-means," Pattern Recognition Letters, vol. 31, no. 8, pp. 651–666, 2010.

[77] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama, "Data stream clustering: A survey," ACM Computing Surveys (CSUR), vol. 46, no. 1, p. 13, 2013.

[78] L. Ulanova, N. Begum, M. Shokoohi-Yekta, and E. Keogh, "Clustering in the Face of Fast Changing Streams," in Proceedings of the SIAM International Conference on Data Mining. SIAM, 2016, pp. 1–9.

[79] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," Science, vol. 344, no. 6191, pp. 1492–1496, 2014.

[80] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh, "Beyond one billion time series: indexing and mining very large time series collections with $i$ sax2+," Knowledge and information systems, vol. 39, no. 1, pp. 123–151, 2014.

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

[82] J. Lin, M. Vlachos, E. Keogh, D. Gunopulos, J. Liu, S. Yu, and J. Le, "A MPAA-based iterative clustering algorithm augmented by nearest neighbors search for time-series data streams," in Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2005, pp. 333–342.

[83] M. Kakkasageri and S. Manvi, "Information management in vehicular ad hoc networks: A review," Journal of network and computer Applications, vol. 39, pp. 334–350, 2014.

[84] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "Haren: A Framework for Ad-Hoc Thread Scheduling Policies for Data Streaming Applications," in Proceedings of the ACM International Conference on Distributed and Event-based Systems. ACM, 06 2019, pp. 19–30.

[85] G. Ulm, E. Gustavsson, and M. Jirstrand, "Active-Code Replacement in the OODIDA Data Analytics Platform," in Proceedings of the European Conference on Parallel Processing. Springer, 2019, pp. 715–719.

[86] I. Ku, Y. Lu, M. Gerla, R. L. Gomes, F. Ongaro, E. Cerqueira et al., "Towards software-defined VANET: Architecture and services." in Proceedings of the Annual Mediterranean Ad Hoc Networking Workshop, 2014, pp. 103–110.

[87] K. Wevers and M. Lu, "V2X Communication for ITS-from IEEE 802.11 p Towards 5G," IEEE 5G Tech Focus, vol. 1, no. 2, 2017.

[88] A. Fernandez and M. Fallgren, "5GCAR Scenarios, Use cases, Requirements and KPIs," Fifth Generation Communication Automotive Research and innovation, Tech. Rep. D, vol. 2, 2017.

[89] F. Kong and J. Tan, "A collaboration-based hybrid vehicular sensor network architecture," in Proceedings of the International Conference on Information and Automation. IEEE, 2008, pp. 584–589.

[90] U. Lee, E. Magistretti, M. Gerla, P. Bellavista, and A. Corradi, "Dissemination and harvesting of urban data using vehicular sensing platforms," IEEE Transactions on vehicular technology, vol. 58, no. 2, pp. 882–901, 2009.

[91] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," ACM Sigmod Record, vol. 31, no. 3, pp. 9–18, 2002.

[92] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "CarTel: a distributed mobile sensor computing system," in Proceedings of the International Conference on Embedded Networked Sensor Systems. ACM, 2006, pp. 125–138.

[93] Y. Zhang, B. Hull, H. Balakrishnan, and S. Madden, "ICEDB: Intermittently-connected continuous query processing," in Proceedings of the IEEE International Conference on Data Engineering. IEEE, 2007, pp. 166–175.

[94] U. Lee, J. Lee, J.-S. Park, and M. Gerla, "FleaNet: A virtual market place on vehicular networks," IEEE Transactions on Vehicular Technology, vol. 59, no. 1, pp. 344–355, 2010.

[95] Y. Zhang, J. Zhao, and G. Cao, "Roadcast: a popularity aware content sharing scheme in vanets," ACM SIGMOBILE Mobile Computing and Communications Review, vol. 13, no. 4, pp. 1–14, 2010.

[96] T. Delot, N. Mitton, S. Ilarri, and T. Hien, "Decentralized pull-based information gathering in vehicular networks using GeoVanet," in Proceedings of the IEEE International Conference on Mobile Data Management. IEEE, 2011, pp. 174–183.

[97] G. M. N. Ali, E. Chan, and W. Li, "Supporting real-time multiple data items query in multi-RSU vehicular ad hoc networks (VANETs)," Journal of Systems and Software, vol. 86, no. 8, pp. 2127–2142, 2013.

[98] Y. Lai, L. Zheng, T. Wang, F. Yang, and Q. Zhou, "Cloud-assisted data storage and query processing at vehicular ad-hoc sensor networks," in Proceedings of the Interntional Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. Springer, 2017, pp. 692–702.

[99] S.-L. Tsao and C.-M. Cheng, "Design and evaluation of a two-tier peer-to-peer traffic information system," IEEE Communications, vol. 49, no. 5, pp. 165–172, 2011.

[100] C.-M. Cheng and S.-L. Tsao, "Adaptive lookup protocol for two-tier VANET/P2P information retrieval services," IEEE Transactions on Vehicular Technology, vol. 64, no. 3, pp. 1051–1064, 2015.

[101] J. Xu, Z. Jin, M. Xu, and N. Zheng, "Mobile-aware anonymous peer selecting algorithm for enhancing privacy and connectivity in location-based service," in Proceedings of the International Conference on E-Business Engineering. IEEE, 2010, pp. 172–177.

[102] X. Huang, R. Yu, J. Kang, and Y. Zhang, "Distributed reputation management for secure and efficient vehicular edge computing and networks," IEEE Access, vol. 5, pp. 25 408–25 420, 2017.

[103] D. Christin, A. Reinhardt, S. S. Kanhere, and M. Hollick, "A survey on privacy in mobile participatory sensing applications," Journal of Systems and Software, vol. 84, no. 11, pp. 1928–1946, 2011.

[104] A. Gidenstam, B. Koldehofe, M. Papatriantafilou, and P. Tsigas, "Scalable group communication supporting configurable levels of consistency," Concurrency and Computation: Practice and Experience, vol. 25, no. 5, pp. 649–671, 2013.