Time- and Computation-Efficient Data Localization at Vehicular Networks' Edge

Citation for the original published paper (version of record):
Duvignau, R., Havers, B., Gulisano, V. et al (2021)
Time- and Computation-Efficient Data Localization at Vehicular Networks' Edge
IEEE Access, 9: 137714-137732
http://dx.doi.org/10.1109/ACCESS.2021.3118596

(article starts on next page)

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier XXXX/ACCESS-XXXX

Time- and Computation-Efficient Data Localization at Vehicular Networks' Edge

ROMARIC DUVIGNAU[1], BASTIAN HAVERS[1,2], VINCENZO GULISANO[1], MARINA PAPATRIANTAFILOU[1]

[1]Chalmers University of Technology, Gothenburg, Sweden
[2]Volvo Car Corporation, Gothenburg, Sweden

Corresponding author: Romaric Duvignau (duvignau@chalmers.se)

ABSTRACT As Vehicular Networks rely increasingly on sensed data to enhance functionality and safety, efficient and distributed data analysis is needed to effectively leverage new technologies in real-world applications. Considering the tens of GBs per hour sensed by modern connected vehicles, traditional analysis, based on global data accumulation, can rapidly exhaust the capacity of the underlying network, becoming increasingly costly, slow, or even infeasible. Employing the edge processing paradigm, which aims at alleviating this drawback by leveraging vehicles' computational power, we are the first to study how to localize, efficiently and distributively, relevant data in a vehicular fleet for analysis applications. This is achieved by appropriate methods to spread requests across the fleet, while efficiently balancing the time needed to identify relevant vehicles, and the computational overhead induced on the Vehicular Network. We evaluate our techniques using two large sets of real-world data in a realistic environment where vehicles join or leave the fleet during the distributed data localization process. As we show, our algorithms are both efficient and configurable, outperforming the baseline algorithms by up to a $40\times$ speedup while reducing computational overhead by up to $3\times$, while providing good estimates for the fraction of vehicles with relevant data and fairly spreading the workload over the fleet. All code as well as detailed instructions are available at https://github.com/dcs-chalmers/dataloc_vn.

INDEX TERMS Connected vehicles, Data Analysis, Edge computing, Query processing

I. INTRODUCTION

WITH the recent advancements in connected Vehicular Networks [1], often facilitated by Vehicular Ad Hoc Networks (or VANETs) [2], the automotive industry is witnessing an unprecedented growth of possible ways for leveraging the fine-grained data sensed in modern vehicles and enhance drivers' safety and experience. If accessed in a real-time fashion as it is being sensed, such data can lead to fresh, up-to-date insights for analysts and practitioners [3]. Similarly to how Mobile Edge Computing [4] pushes parts of data analysis applications, previously run entirely in the cloud, towards mobile users, to achieve lower latency and bandwidth consumption, Vehicular Edge Computing [5] aims at better utilizing the cumulative computational power of Vehicular Networks while coping with high-mobility networks and the challenges stemming from their dynamic topologies and communications. When focusing on data analysis in the context of Vehicular Networks, a critical challenge is that of data gathering [6], [7] for subsequent analysis. While in the past companies could potentially afford the central gathering of all the data sensed by an entire fleet of vehicles (see [8], [9] for applications and services in VANETs), a modern vehicle can now generate several gigabytes of data per hour [1], making this approach infeasible in terms of infrastructure and costs - which could be alleviated by enforcing the collection of just enough data from relevant vehicles only. Several recent studies in the literature are focusing on how to avoid general central data gathering by transitioning to models in which the data selection processes, or even the analysis itself, are pushed towards the vehicles [10], [11], [12], for efficient and continuous filtering [13], preprocessing through online compression [14], [7], and conversion of raw data into information in Federated Learning [15]. However, these distributed analysis models often lack mechanisms that attempt to involve only valid vehicles into the analysis, to thus avoid unnecessary computational load and data transfers

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

or other hindrances to the analysis. As an example, Federated Learning on vehicles [16] requires the involvement of vehicles that have gathered sufficient suitable data in order not to hinder the learning process [17]. Furthermore exacerbating the issue of vehicle selection are skewed data distributions on vehicles [18], [19] and data minimization directives such as the European GDPR, which dictate to minimize exposure risk and thus overall involvement of customer vehicles. To find vehicles possessing data relevant to an analysis task, one has to overcome the lack of a-priori knowledge about which vehicle has collected which data, without centrally gathering said data first, by leveraging the vehicles' computational power. As vehicles possess only application-specific computational hardware that is not provisioned for more general tasks, it is furthermore paramount to avoid unnecessary computational strain on the fleet.

Contribution

In light of the present challenges concerning the transition from central to distributed, edge data gathering and analysis in large fleets of vehicles, we pose the following question:

How can data residing on the edge nodes of a Vehicular Network be localized efficiently through request-spreading from a central coordinator to the vehicles?

The manner of spreading requests is regulated by a data localization algorithm orchestrated at a central coordinator that has to be aware of the completion time and the computational overheads induced on the fleet of vehicles. Once a request is sent from the coordinator to a vehicle, the latter checks locally whether a set of conditions is satisfied by the stored data (e.g., whether the data spans a given time interval, or whether the data indicates that the vehicle is associated with a specified geographical position, speed, driving mode, etc.), and returns a compact answer indicating whether the conditions hold. When performing traffic flow analysis, for instance, this could be used to efficiently compute a certain statistic (e.g., the average speed) only based on vehicles driving above a certain speed, within a city center, or during rush hour, or to mark these vehicles for a subsequent analysis. With the ultimate goal of collecting a certain amount of answers from vehicles matching a given set of conditions, we propose efficient *data localization algorithms*, that can also cope with dynamic connectivity, and benchmark them against baseline algorithms. Our evaluation, based on realistic queries and also assessing the spreading of requests using real vehicular data, shows that our data localization algorithms provide up to a $40\times$ speedup and less than one-third of the computational overhead, compared with baseline algorithms optimizing only one of the metrics.

The typical characteristics of Vehicular Networks include recurrent topology changes due to vehicles' high-speed mobility and properties of the underlying road network (including communication-challenging environments such as bridges, tunnels, etc.). Though this challenging aspect can impede successful communication between a central coordinator and vehicles, our work accounts for it with algorithms able to react rapidly to dynamic connectivity issues. To the best of our knowledge, we are the first to formulate and analyse the problem of localizing data in a vehicular fleet, as well as to propose algorithms that can tune the key trade-off between resolution time and overhead on the vehicles and the communication network.

The remainder of the paper is organized as follows: We introduce the System Model in § II. We present in § III baselines and propose novel algorithms for solving data localization queries by spreading a set of requests over a fleet of vehicles. We lay out our evaluation methodology in § IV and cover the evaluation of the proposed algorithms in § V. We discuss related work in § VI, before concluding the paper with a summary in § VII. In the Appendix, we discuss the relation of the present paper to an earlier conference article that presented first results on a preliminary formulation of the problem for a subset of the system types considered here.

II. SYSTEM MODEL

A. PROBLEM DEFINITION

We consider the following model: a *fleet* V of k vehicles, also referred to as *nodes*, is equipped with different types of sensors $s_1, s_2, \ldots$ from the fixed sensor set S, e.g., $S = \{\text{GPS}, \text{steer}, \text{break}, \ldots\}$. We define the continuous timestamped record sequence recorded by the sensor $s_i \in S$ at vehicle $v \in V$ as

$$s_i(v) = (t_0, x_0), (t_1, x_1), \ldots$$

where x_j is the sensor reading at time point t_j. All vehicles are connected via a two-way communication channel to a central *coordinator* C, e.g., a datacenter. Data analysts require C to process *data localization queries* $q_1, \ldots, q_r$, with each query focusing on some subset of the possible sensors for some time span of recorded data.

A *(data localization) query* here corresponds to the task of identifying n vehicles in the fleet with relevant data. In more detail, a query q carries a specific *condition* P that must be fulfilled by vehicles' data to be *relevant* for q and every query specifies some number of positive *answers* n (responses from distinct vehicles with relevant data, where a "positive" or yes-answer implies that P holds locally) that must be collected to *resolve* q before a potential next analysis step involving only these n vehicles with relevant data can follow. k_q is the number of vehicles in the fleet on which P holds, and $QR = k_q/k$ is the query rate or *average answer rate* of a query q. We assume $k \gg k_q > n$, thus n vehicles with relevant data can indeed be found for a query q. To check if a particular vehicle v fulfills q's condition P for some query q, a *request* $r(q)$ is sent to v and after checking P locally, v responds to C with its yes- or no-answer (and potentially some additional data). If a vehicle v receives a new request $r(q')$ corresponding to another query q' while already processing a previously received one $r(q)$, then $r(q')$ is added to v's local *task queue*; once v has terminated its processing of $r(q)$, v's task queue is then processed in FIFO

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

order. Naturally, not every vehicle can answer positively to every request, because of lack of data or because the data is found not suitable to answer that particular query. The required number of answers is meant to localize a sufficient amount of data from the vehicles to be meaningful for the analysis task at hand, while avoiding excessive participation.

Notice that contacting exactly the number of vehicles given by the analyst is not necessarily enough, as some might not answer or have data that does not satisfy the condition. On the other hand, contacting too many vehicles might result in some of them wasting some of their computational power to inspect data that is not actually needed by the analyst. Thus, we need to require just enough positive answers (e.g., for statistical significance or for reducing the likelihood of identifying individuals in the data), but not too many (because of the time needed to collect all the data [20], [7], the induced computational load, and potential network stress).

We will use the notation $q.P$ for the condition and $q.n$ for the minimum number of answers required to complete the data localization query q. The condition P specifies (i) which sensors are relevant to the query and (ii) an overall condition that local data must satisfy for the vehicle to acknowledge that its data can be part of the desired analysis. In the following: $S_q \subseteq S$ is a subset of sensors which are relevant for the query q (by default $S_q = S$); and (t_{start}, t_{end}) are time bounds spanning q's time interval of interest such that for every participating node v and every sensor of type $s_i \in S_q$, only the portion of local data $\{(t, x) \in s_i(v) \mid t_{start} \leq t \leq t_{end}\}$ is examined (if not specified, the full recorded data is considered).

a: Example query

To interactively check the traffic flow within a certain area A of a city, an analyst wishes to identify $n = 100$ vehicles in a $k = 100,000$ vehicles fleet, with $P =$ "driven within area A in the last hour, with an average speed greater than 50km/h over a 10 minutes window, and with GPS measurements spaced by at most 5s from each other". In this query, assuming the query is created at 9:00, $S_q = \{\text{GPS}\}$, $(t_{start}, t_{end}) = (8{:}00, 9{:}00)$, and A is a bounding box or geofence approximating the area of interest. Notice that, in this example, to check if any period of 10 minutes (representing only 12000 data points for 100 cars with 5 seconds GPS readings) within the last hour fulfills the condition, a centralized solution requires between $n \cdot 3600/5$ (considering at least n vehicles need to answer) and $k \cdot 3600/5$ data points to be transmitted, i.e., 72000 to 7200000 data points; that is, in order to check the existence of any period of 10 minutes of consecutive readings with speed greater than 50km/h, since the coordinator has no information whether they exist (and in which portion of the hour), the entire hour needs to be retrieved and checked. Checking the condition on-board the vehicles alleviates this data transfer and only short yes/no answer messages need to be communicated with the coordinator.

b: Applications

In the aforementioned example query, the mere collection of affirmative or negative answers from the fleet can already provide a good estimation of the fraction of vehicles that satisfy P in the fleet (and thus quantify the traffic flow in the area in question). This defines a first set of applications, in which a query itself gives rise to a statistical insight by providing a population estimate. As hinted in the example, transmitting raw data from a random sample of the fleet in the above example to check at the coordinator whether P holds for a vehicle would incur significantly higher communication costs, while yielding the same insight. In addition to the first set of applications, the coordinator node C can ask the vehicles which answer positively to subsequently perform tasks suitable only to them. These tasks can include transmitting raw data, performing statistical summaries such as averages or other aggregate functions over the local data, or higher-level computations on-board the vehicle over the relevant data, such as training an Artificial Neural Network. However, we do not consider the query post-treatment in this work and concentrate on the aspect of data localization. That is, we focus on finding a suitable set of vehicles that will participate in the query resolution process. One may note that some of the subsequent tasks could be answered alongside P's verification, entailing minor changes in the processing time of the query. The aggregated value could be transmitted with the vehicle's yes/no-answer to C without significantly changing the resolution time. For computationally heavier analysis tasks or those requiring substantially longer time (such as data transfers from the vehicles), we consider that the post-treatment is executed in a way independent of the selection process, i.e., vehicles will always first inform C if they validate P before executing the post-treatment.

B. FLEET MODEL

The fleet V of vehicles that can be contacted encompasses the totality of vehicles that are equipped to take part in answering requests incoming from the coordinator C. As vehicles in V may be switched off, we introduce the *active* set of vehicles $V_t \subseteq V$ at a time t as the ones switched on and willing to participate in the data localization queries' resolution process. A realistic fleet is a dynamic entity where vehicles leave and join impromptu (thus, vehicles are being switched on and off). We differentiate two variants of the underlying system model depending on how the fleet V_t evolves during the resolution of a batch of queries.

We first consider a *static fleet* model. In that model, the set of contactable participants is always fixed, hence we do not consider that new vehicles may join the fleet or that vehicles may leave within the time interval spent resolving a particular query. Thus, $V_t = V$ for all t.

In the *dynamic fleet* model, the set V of *all* vehicles (such as all vehicles from a company fleet) is larger than the active set V_t at time t, unlike in the static model. The active set evolves through time with the possibility for new vehicles to join and for old ones to leave. Contrary to the static model,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

a vehicle may be switched off (that is, leaving the active set) during the resolution of a particular query q and may not send its answer for q to C; similarly, new vehicles that were absent at the start of q's resolution may join the active fleet at any time during q's resolution process.

To quantify the amount of vehicles leaving the fleet over time, we rely on the notion of *churn*. Given a period Δ, we define the churn at time t based on the number of vehicles that leave the fleet during the period $[t-\Delta, t)$ but that were part of the fleet during the period $[t-2\Delta, t-\Delta)$. More formally:

Definition 1. *$Churn_\Delta(t)$: The number of vehicles that were part of $V_{t'}$ for all $t' \in [t-2\Delta, t-\Delta)$ but that leave the fleet at any $t'' \in [t-\Delta, t)$, divided by the size of V_t.*

Note that while the churn takes into account only vehicles leaving the fleet, the number of vehicles joining the fleet can be obtained through the size of the active fleet and the value of the churn.

C. COMMUNICATION MODEL

We assume that the coordinator C has no access to the vehicles' local data other than through communication with them; thus, the amount of work needed to test $q.P$ for a query q cannot be estimated before checking P locally on the appropriate vehicle. We further assume that C will always successfully contact any active vehicles and as soon as a vehicle does not communicate for a certain period of time, it is considered inactive.

In the likely event that the local requested data is missing, the involved vehicle does not satisfy $q.P$ and it answers negatively. Similarly, active vehicles unwilling to participate in a query's task (for privacy or other reasons) can be modeled by negative answers.

In the dynamic fleet model, new vehicles signal their presence once they become active and ready to answer potential requests. Since connection to C can be lost at any point in time (e.g., driving through a tunnel or reaching a poorly covered geographical area), we assume that once a vehicle v has become inactive, it cannot be reached by C and drops all currently processed queries; hence, C will not receive any answers from v for the queries it was processing at the time. This allows abstracting the high degree of node mobility and its effect on communication by possibly short interruptions in the active status of the vehicles; here, individual packet losses are neglected as overall they can be compensated by configuring a lower transfer rate and higher latency for the underlying communication channel.

D. PERFORMANCE METRICS

We associate with each query three performance metrics:

1) **Query Resolution Time**, the elapsed time between deploying a particular query q at C and q's resolution, i.e., when $q.n$ positive answers have been collected at C.
2) **Fleet Workload**, the overall computing load on the vehicles defined as the sum of individual processing times (local workloads) of all vehicles involved in processing the received requests associated with the query.
3) **Fairness** of the algorithms, the standard deviation of the cumulative local workloads between the vehicles that received a request, representing how fair the spread of the fleet workload is.

Notice that optimizing for both (1) and (2) at the same time is not straightforward, as they compete. More concretely, query resolution time is minimized by simply asking all vehicles in the fleet and ignoring answers after n positive answers are retrieved (thereby maximizing the Fleet Workload required per query, which is further exacerbated when queries are executed in parallel or are computationally expensive, see § IV-C) while the fleet workload is minimized for instance by asking one vehicle at a time in a round-robin fashion (implying high time overhead).

The amount of uncertainty in the model is an additional challenge: each query requires different amounts of time per vehicle that can hardly be foreseen. The reasons for this are twofold:
(i) *Query semantics.* As it is unknown how much relevant data a vehicle has collected, it can not be estimated beforehand how long it will take for this vehicle to search for the property required by the request. Likewise, some queries may be answered positively as soon as the first matching instance is found in the data, whereas a negative answer requires checking all potentially relevant data.
(ii) *Computing capacity.* A vehicle that is contacted may be performing other processing with higher priority and equally unknown completion time before it can start answering the latest received request at hand.

III. DATA LOCALIZATION ALGORITHMS

We present here algorithms that select a subset of vehicles among the k_q vehicles 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 at C, either sequentially or concurrently, the procedures described hereafter. Without further assumptions on the distribution of nodes satisfying P, it is natural to randomly and uniformly send requests to nodes within the pool of nodes that have not been requested yet. However, other factors (such as the number of requests currently being processed on the vehicle, historical local computation load, etc.) can be used to bias the selection process. Since in our setting, queries are relatively short to solve (from a few seconds to minutes at most), we consider that algorithms do not need to send another request to a vehicle that has answered negatively, in case its newest acquired data now satisfies the property P. We hence assume in our analysis that a vehicle's to a query q does not change over the whole period of the algorithms' execution.

We present in this section four algorithms focusing on different measures:

- BASEEAGER, a baseline approach that optimizes the resolution time needed to answer q,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

- BASELAZY, a baseline approach that optimizes the number of contacted vehicles (hence minimizing required communication and reducing fleet workload), and ensures no more than n positive answers are ever received,
- BALANCEREQUESTS, a new approach that balances the trade-off identified through the two baseline algorithms, in order to quickly collect n answers without inducing excessive load on the vehicular nodes, and
- BALANCELOAD, an approach that extends BALANCEREQUESTS by prioritizing the least-used vehicles during the selection process to balance the workload.

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

The four algorithms can maintain the following sets in each execution:

- $F \subseteq V$, the set of contacted vehicles since the beginning of the algorithm;
- A, the set of all answers from vehicles $v \in F$ that have been received by the coordinator;
- $R \subseteq A$, the subset of positive answers (where each answer contains whether P holds plus metadata identifying the sending vehicle, see § II-A) among all the ones received.

We begin by introducing these algorithms in the context of an idealized model in which the fleet is static and the on-board execution is synchronous, before presenting the algorithms in our complete static (§ III-B) and dynamic model (§ III-C).

A. DATA LOCALIZATION IN THE SYNCHRONOUS STATIC MODEL

To ease the introduction of the algorithms, we consider in this subsection a synchronous model (in the next subsections we present the generalization of the algorithms for the asynchronous model): communications with C are instantaneous and all nodes need a constant amount of time to check the property P, i.e., one "round" is the time to check any request on one vehicle. Hence, after a round of time has elapsed, C has received answers (yes/no) from all nodes that were asked during that round. In this simplified situation, only two aspects have to be considered in order to measure the performance of data localization procedures: (1) the total number of rounds needed at C to receive n answers, and (2) the number of nodes m that have checked if $q.P$ holds (which is equivalent to the fleet workload on the vehicles, since each contacted vehicle has spent exactly one round checking the request). Note that for clarity we discuss the behavior of the algorithms in the case of answering a single data localization query q.

Algorithm 1 BASEEAGER

```
1:  function BASEEAGER(V, q)        ▷ fleet V, query q with n = q.n
2:      R ← ∅                       ▷ set of collected positive answers
3:      for v ∈ V do
4:          send(q, v)              ▷ send request r(q) to vehicle v
5:      while |R| < n do
6:          r ← receive()           ▷ block till receiving next answer
7:          if positive(r) then
8:              R ← R ∪ {r}
9:      return R
```

Algorithm 2 BASELAZY

```
1:  function REQUESTRANDVEHICLE(q, F)
2:      v ← random(V, F)            ▷ random vehicle in V excluding F
3:      send(q, v)
4:      return v

1:  function BASELAZY(V, q)         ▷ fleet V, query q with n = q.n
2:      F ← ∅                       ▷ set of asked vehicles
3:      R ← ∅                       ▷ set of collected positive answers
4:      for 1 ≤ i ≤ n do
5:          F ← F ∪ { REQUESTRANDVEHICLE(q, F) }
6:      while |R| < n do
7:          r ← receive()
8:          if positive(r) then
9:              R ← R ∪ {r}
10:         else
11:             F ← F ∪ { REQUESTRANDVEHICLE(q, F) }
12:     return R
```

BASEEAGER - synchronous, static

Presented in pseudocode in Algorithm 1, BASEEAGER aims to optimize a query's resolution time, by immediately querying all available nodes; this resolves the query in a single round (under our assumption that enough vehicles with relevant data are in the fleet). Indeed, consider any other algorithm $\mathcal{A}$ that does not contact at least one node v during the first round. In the situation that $k_q = n$ and P holds on v, only $k_q - 1$ answers are received after a single round of communication and a second one is required to retrieve all required answers; hence, $\mathcal{A}$ is not optimal in regards of the resolution time. Executing Algorithm 1 to obtain all needed answers leads nonetheless to a large strain on the vehicular nodes. In particular, the number of queried nodes is always k, independently of k_q and n. Thus, all nodes always participate in q's resolution, even though the number of required answers n might be relatively small.

BASELAZY - synchronous, static

This Algorithm is presented in pseudocode in Algorithm 2. The focus of this algorithm is on reducing the computational overhead and communication induced on the fleet. To ensure that only the minimum number of nodes are being requested to check P, one must ask at most as many new nodes as the number of currently missing positive answers. Any algorithm satisfying such an assertion is associated with a minimal fleet workload, and the best algorithm in this category selects randomly as many nodes as possible by asking m "new nodes" for each round where there are m missing answers. Once n positive answers are received, the procedure stops. In

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

an edge case, every vehicle in the fleet has to be contacted to achieve this. Since the algorithm contacts m vehicles per round, the algorithm will proceed the slowest for $m = 1$. When $(n-1)$ yes-answers are collected in the first round and the last yes-answer is obtained after asking every single other vehicle in the subsequent rounds, this results in a resolution time of $k - n + 1$ rounds.

However, on average BASELAZY requires fewer rounds, as claimed by Proposition 1 below. When sending a request to a vehicle that has not participated so far, the probability of receiving a yes-answer is in general $(k_q - r)/(k - f)$ where $f = |F|$ is the number of vehicles already requested and $r = |R| \leq n$ the number of positive answers already received. In the following, we assume for simplicity that the probability of obtaining a yes-answer is constant and equal to $QR = k_q/k$ during the full execution of the algorithm; this corresponds to our typical use-case where n is much smaller in comparison to both k_q and k. The number of rounds used by the algorithm can then intuitively be shown to be logarithmic by the following reasoning: When requesting x vehicles in a round, approximately $x \cdot QR$ positive answers are received, thus if x_i denotes the number of requests sent in round i, we have $x_i \approx x_{i-1} \cdot (1 - QR)$, as vehicles answering negatively trigger another request. Setting $x_1 = n$, we get $x_i \approx n \cdot (1 - QR)^i$ and we obtain $r \approx \log_{1/(1-QR)}(n)$ when requiring $x_r = 1$ (when the algorithm roughly concludes). More formally, we argue that:

Proposition 1. *BASELAZY solves a single query in the static, synchronous case on average in $\mathcal{O}(\log(n))$ rounds.*

Proof. Let us visualize the query resolution as follows. Let $\mathcal{P} = p_1, \ldots, p_n$ be n random processes that aim to retrieve one answer each to the query, and each process will remain active until it acquires a positive answer. BASELAZY can be seen as sending one request per round for each process that is still active and doing nothing for the ones that have already got a yes-answer. Under our assumptions, during a certain round, $p_i \in P$ retrieves a positive answer with constant probability $p = QR$ and a negative answer with probability $1 - p$. Each process, being independent of the others, will need $1/p$ rounds on average to acquire a yes-answer (geometric distribution with parameter p). The number of rounds $M(p, n)$ that is necessary for all processes to stop is thus the maximum of n independent *geometric random variables* of parameter p. The expected value $\mathbb{E}(M(p, n))$ is known [21] to be bounded by

$$\frac{H_n}{\ln \frac{1}{1-p}} \leq \mathbb{E}(M(p, n)) < \frac{H_n}{\ln \frac{1}{1-p}} + 1$$

where H_n is the n-th harmonic number. Using $H_n = \ln n + \mathcal{O}(1)$, one obtains that $\mathbb{E}(M(p, n)) = \log_{\frac{1}{1-QR}}(n) + \mathcal{O}(1)$ for a constant QR.

□

Algorithm 3 BALANCE*-skeleton

```
1:  function BALANCE*(V, q, α, β)        ▷ fleet V, query q with n = q.n, α > 0, β ∈ (0, 1]
2:      p ← 1                            ▷ estimation of probability to answer yes
3:      F ← ∅                            ▷ set of requested vehicles
4:      A ← ∅                            ▷ set of collected answers
5:      R ← ∅                            ▷ set of collected yes-answers
6:      while |R| < n do                 ▷ until n answers are collected
7:          ASKNEWBATCH(α, p)
8:          while WAITFORANSWERS() do
9:              RECEIVEANDUPDATE()
10:         p ← max{|R|/|A|, 1/(|A|+1)}  ▷ update probability
11:     return R
```

Similarly, BASELAZY sends on average requests to significantly fewer vehicles than in the extreme case, as shown by the following Proposition 2:

Proposition 2. *BASELAZY solves a single query in the static, synchronous case asking on average n/QR vehicles.*

Proof. Following the presentation of the previous proof, the number of requested vehicles is obtained as the sum of n independent geometric random variables, each of which has an expected value of $1/p = 1/QR$. By linearity of expectation, we obtain that n/QR vehicles will receive a request. □

BALANCEREQUESTS - synchronous, static

We introduce now an efficient scheme to achieve a low fleet workload while resolving queries within few processing rounds, balancing the tradeoffs of BASEEAGER (high workload) and BASELAZY (slow query resolution time). The main idea behind BALANCEREQUESTS is to employ QR, the share of vehicles in the fleet on which $q.P$ holds, to scale the number of vehicles contacted in each round such that the expected number of positive answers is equal to the number of total outstanding positive answers. As QR is unknown during the execution of the query, we replace it with the running estimate $p = |R|/|A|$, and show in § V-D that p gives a reasonable estimation of QR.

We will present various implementations of the Balance* algorithms based on the skeleton algorithm shown in Algorithm 3, which proceeds as follows: Keeping track of p, F, A, R, the algorithm concludes by returning R, the set of yes-answers. To achieve this, the algorithm begins by contacting a new batch of vehicles in askNewBatch(), and then proceeds to receive answers from the contacted vehicles using receiveAndUpdate until waitForAnswers() evaluates to $false$. At this point, the algorithm updates the value of p using the answers received so far, and loops back to the beginning; the loop is continued until $|R| = n$, i.e., a sufficient number of yes-answers has been acquired.

In the synchronous model and with a static fleet, algorithm BALANCEREQUESTS employs those variants of askNewBatch() and waitForAnswers() described in Algorithm 4. askNewBatch() has as input the running estimate p and a parameter α, and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

Algorithm 4 BALANCEREQUESTS - synchronous, static

```
1: procedure ASKNEWBATCHAUX(α, p, m)
2:     ℓ ← ⌈α · m/p⌉                          ▷ new vehicles to contact
3:     for 1 ≤ i ≤ ℓ do
4:         if |F| < k then                     ▷ if fleet not exhausted yet
5:             F ← F ∪ { REQUESTRANDVEHICLE(q, F) }

1: procedure ASKNEWBATCH(α,p)
2:     m ← n − |R|                            ▷ remaining yes-answers to collect
3:     ASKNEWBATCHAUX(α, p, m)

1: procedure RECEIVEANDUPDATE()
2:     r ← receive()
3:     A ← A ∪ {r}
4:     if positive(r) then
5:         R ← R ∪ {r}

1: function WAITFORANSWERS()
2:     return |A| ≠ |F|
```

proceeds as follows: m marks the currently missing yes-answers $n - |R|$. We denote by

$$\ell = \left\lceil \alpha \cdot \frac{m}{p} \right\rceil \tag{1}$$

the adjusted expected number of vehicles to contact to receive the outstanding m answers, employing the running estimate p. The parameter $\alpha > 0$ allows the algorithm to depart from the estimated expected number of vehicles to contact to get the remaining answers, by sampling more or fewer vehicles. This allows to either shorten (when $\alpha > 1$) the average number of rounds needed to resolve q while potentially increasing the fleet workload, or on the contrary (when $\alpha < 1$) to slow down q's processing by being more prudent and avoiding requesting more vehicles than necessary (and thus getting closer to receiving exactly n answers at the end). Having contacted ℓ vehicles or exhausted the fleet of vehicles, the function returns. BALANCEREQUESTS then enters a loop of receiving answers until waitForAnswers() returns $true$, i.e., until all contacted vehicles have sent an answer.

Following the general logic of the Balance* algorithms described in Algorithm 3, the value of p is then updated as

$$p = \max\left\{ \frac{|R|}{|A|}, \frac{1}{|A|+1} \right\}, \tag{2}$$

where the second case is used when no positive answers have been received during the first round(s), i.e., $|R| = 0$. A next batch of vehicles is then contacted, until n yes-answers are received.

BALANCELOAD - synchronous, static

This algorithm does not have an equivalent in the synchronous model, as it attempts to balance the individual workloads of each vehicle. In the synchronous model, the workload of each vehicle answering a request is identical by assumption (as all vehicles answer a request synchronously). We thus defer introducing this algorithm to the following section.

B. DATA LOCALIZATION IN THE ASYNCHRONOUS STATIC MODEL

We now generalize the algorithms presented in the previous section to the asynchronous data localization model, i.e., when request processing time is both vehicle- and context-dependent. In a typical vehicular environment, one cannot generally assume bounds on neither the time a vehicle needs to process a request nor on the communication delays in the network. Consequently, the algorithms have to adapt to the following scenarios: (1) how to avoid being blocked by the slowest-answering vehicles; and (2) how to deal with vehicles that answer late? While BASEEAGER and BASELAZY achieve their respective goals without adaptations in the asynchronous model, we tune BALANCEREQUESTS to the asynchronicity and furthermore extend it to yield BALANCELOAD. In addition to asynchronicity, we now also extend to the more general case of more than a single data localization query deployed simultaneously. As a reminder, vehicles possess a FIFO task queue (see § II-A) in which incoming requests are stored and processed sequentially.

BASEEAGER - asynchronous, static

Shown in pseudocode in Algorithm 1, BASEEAGER optimizes the time required to answer a single query q by contacting all vehicles upon receiving it. In our asynchronous model, the query resolution time then needed for a single q is the best possible and corresponds to the n-th fastest positive answer received at C. In contrast to that, the fleet workload is also the highest possible, as all k vehicles have processed $r(q)$. Note that the guarantee on fastest resolution does not hold in the case of multiple simultaneous queries: Let us assume that vehicle v gives the n-th fastest yes-answer to query q in the single-query case. However, when $r(q)$ is received by v, v is busy processing another request $r(q')$; thus v will wait before answering $r(q)$, which would conclude the query q. Thus, the execution time of q is dependent on the presence and order of other concurrent queries on the requested vehicles.

BASELAZY - asynchronous, static

Shown in pseudocode in Algorithm 2, BASELAZY optimizes the number of requested vehicles (hence minimizing needed communication to spread all requests) by contacting a new vehicle only when strictly needed. Since in our asynchronous model it is not guaranteed nor assumed that vehicles will have similar answer times (only that they will answer at some point), this algorithm does not necessarily imply a minimum load on the network. Indeed, it might be the case that requesting more vehicles that require shorter processing times to answer will use fewer resources overall.

BALANCEREQUESTS - asynchronous, static

The asynchronous variant of BALANCEREQUESTS (Algorithm 5) is similar in essence to its round-based version described in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

Algorithm 5 BALANCEREQUESTS - asynchronous, static

```
1: procedure ASKNEWBATCH(α,p)
2:     m ← n − |R| − p · (|F| − |A|)   ▷ remaining yes-answers, corrected by late ones
3:     ASKNEWBATCHAUX(α, p, m)          ▷ cf. Algorithm 4

1: procedure RECEIVEANDUPDATE()         ▷ same as Algorithm 4

1: function WAITFORANSWERS(β)
2:     return |A| < β · |F|             ▷ share β of contacted vehicles has answered
```

§ III-A but needs to take into account that not all contacted nodes reply at the same time (as they do in the synchronous model). To do so, we change the behavior of the function waitForAnswers(), as shown in Algorithm 5; it accepts a new parameter $\beta \in (0, 1]$: a certain proportion of answers over all requested vehicles that we will wait to receive before re-evaluating the running estimate of yes-answer share p and proceeding to the next batch of selection (see Algorithm 3). When $\beta = 1$, the algorithm waits for the reception of all answers before continuing; this is effectively the case in the synchronous model, § III-A. Setting a lower value for β allows us to make 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 starts. Based on previously received answers, we estimate that a fraction p of the $|F| - |A|$ requested vehicles that have not yet answered, will eventually answer positively while the next batch of vehicles is already being sent requests. This allows to reduce the number of vehicles asked in the next iteration, and thus reduce excessive participation. This change is applied in askNewBatch() (Algorithm 4): we adjust the number of vehicles to contact next, ℓ, by

$$\ell = \left\lceil \alpha \cdot \left(\frac{n - |R|}{p} - (|F| - |A|) \right) \right\rceil \tag{3}$$

Those $\lceil p \cdot (|F| - |A|) \rceil$ vehicles are hence counted as *expected* answers when calculating ℓ.

BALANCELOAD - asynchronous, static

This is a variation of the previous algorithm that presents a further refinement of vehicle selection, differing in how the ℓ vehicles are selected during each batch in askNewBatch(), as shown in Algorithm 6. Instead of randomly selecting new nodes to request, vehicles having low local workload or involved in only a few concurrent data localization queries are picked first in the selection phase. The main difference with Algorithm 5 is that instead of requesting a random vehicle using requestRandVehicle() among the not yet requested ones (see requestRandVehicle() in Algorithm 2), vehicles are selected in the order of their lowest *local workload* measured as (1) number of simultaneous requests being processed on the vehicle (for the concurrent execution of several data localization queries) and (2) reported local processing time since the start. As shown in Algorithm 6, vehicles are for that purpose stored in an updatable priority queue W (initialized

Algorithm 6 BALANCELOAD - asynchronous, static

```
1: procedure INIT()
2:     global W                          ▷ priority queue
3:     W.insertAll(V, [0, 0])            ▷ initially, all vehicles have same priority

1: procedure ASKNEWBATCH(α,p)
2:     m ← n − |R| − p · (|F| − |A|)
3:     ℓ ← ⌈α · m/p⌉
4:     for 1 ≤ i ≤ ℓ do
5:         if |F| < k then
6:             v → W.getLowestPriority()   ▷ get vehicle with fewest parallel queries and lowest workload
7:             send(q, v)
8:             W.updatePriority(v,[+1, +0])   ▷ increase no. of parallel queries of v

1: procedure RECEIVEANDUPDATE()
2:     r ← receive()
3:     A ← A ∪ {r}
4:     if positive(r) then
5:         R ← R ∪ {r}
6:     W.updatePriority(r.v, [−1, +r.workload])   ▷ decrease no. of parallel queries of r.v (sender of r), increase r.v's workload

1: function WAITFORANSWERS(β)            ▷ same as Algorithm 5
```

in a call to init()) where vehicle v's priority is defined as a tuple of [no. of parallel queries, total local workload]. As shown in line 6 of askNewBatch() in Algorithm 6, the vehicle v with the lowest priority is contacted first. After sending a request to v, its priority is updated by increasing the first field of the priority tuple, no. of parallel requests, by one. Likewise at line 6 of receiveAndUpdate(), upon reception of an answer r from vehicle v, v's priority is updated by reducing the number of its parallel requests, and increasing the total local workload registered in W for v by the workload transmitted alongside r.

C. DATA LOCALIZATION IN THE ASYNCHRONOUS DYNAMIC MODEL

The dynamic fleet model introduces vehicles dynamically joining the fleet (which is detected at C) and leaving the fleet (undetected). We describe here adaptations in the presented data localization algorithms to handle both types of events, i.e., vehicles joining and leaving the fleet.

BASEEAGER - asynchronous, dynamic

The algorithm is a straightforward extension of BASEEAGER defined for the dynamic model: all active vehicles get asked upon starting processing a new query at C. Vehicles leaving the fleet will not provide any answer whereas vehicles arriving receive all unresolved queries upon becoming available.

BASELAZY - asynchronous, dynamic

This algorithm could be blocked indefinitely if any of the involved vehicles leave the fleet before the moment when all answers are collected: indeed, the algorithm waits for receiving a negative answer before asking a new vehicle. To

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

deal with leaving vehicles, we introduce a timer set upon sending a request. A timeout is then considered equivalent to a negative answer and triggers requesting one of the remaining available vehicles; if the timeout vehicle answers later than its corresponding timer, the answer is accepted in case of a yes-answer and ignored in case of a negative one. Contrary to BASEEAGER, new vehicles may get requested (upon receiving a negative answer or timeout event at C) some time after they become active. However, a new arrival by itself will not trigger directly the transmission of a request, except in the particular case of unresolved queries that have already exhausted the pool of known active vehicles.

BALANCEREQUESTS and BALANCELOAD - asynchronous, dynamic

Contrary to BASELAZY, the Balance* algorithms as designed for the *static* setting are not at risk of becoming blocked by vehicles exiting the fleet. Indeed, they both already have a mechanism to ask a new batch of vehicles before having answers from all the vehicles that had been requested earlier (through the β parameter, cf. Algorithm 5) and can hence deal with a dynamic fleet where some vehicles leave the fleet. However, over the long run, the estimation of the probability p of answering positively will be less accurate, as vehicles that have left will be excluded from the estimation (only received answers are taken into account); also, if a proportion greater than $1-\beta$ of the vehicles currently checking requests associated with a particular query leaves the fleet during the algorithm execution, no new batch of vehicles will ever get contacted even though there might be plenty of available vehicles. To circumvent these issues, we also introduce timers in those algorithms: a timeout is equivalent to receiving a negative answer, i.e., a negative answer is added to the set of received answers A, which is used for p's computation and for testing when the β threshold has been crossed. If a vehicle answers positively later than its timer, it is added to the set of known positive answers R; this has no further effect on A, but slightly modifies the calculated value for p as

$$p = \max\left\{\frac{|R|}{|A|}, \frac{1}{|A|+1}\right\}.$$

We note that timers help the estimation p to take into account both the positive answering rate and the fleet churn rate when computing the size of the next vehicle batch to request.

IV. METHODOLOGY OF THE EXPERIMENT STUDY

To investigate the performance of the proposed algorithms, we evaluate them on two large real-world sets of vehicular data. In this section, we first describe in detail the datasets and the induced churn in each of them (see Definition 1) and the experiment setup used for our study. We then present a set of common queries that will serve to benchmark the different algorithms, including longer-running versions of such queries for our dynamic fleet model. Finally, we show the distribution of data over the fleet and the query answer rates in the studied datasets.

A. DATASETS

Our evaluation encompasses two datasets (one public, one proprietary) that differ in the number of active vehicles and the rate of churn (see Figure 1), the distribution of data per vehicle (see Figure 3), as well as the types of data included in the dataset.

Geolife 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 [22]. 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 18670 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 10528 files, each for one day of usage of one user (cf. Figure 1a for the number of vehicles over the course of 24h).

Volvo Dataset

The second dataset consists of CAN data and GPS traces from 20 hybrid cars internally collected by *Volvo Car Corporation* [20], [7] in the year 2015. After pre-processing, we generate 3462 trace files, each corresponding to a daily usage of one vehicle (cf. Figure 1b). 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*, *combustion*, and *hybrid*. Each trace in this dataset hence contains records of the form *timestamp (s), latitude (deg), longitude (deg), driving mode (e/c/h)* (cf. Figure 1b for the vehicle number over the course of 24h).

Vehicles leaving and joining the fleet in the datasets

$Churn_{\Delta}(t)$, measuring the fraction of vehicles leaving the fleet within a predefined time interval Δ, influences how fast queries get resolved (see Definition 1). In the studied datasets, the churn is evaluated to be between 2% (for $\Delta =$ 30 seconds) and 38% ($\Delta = 15$ minutes), see Figures 1a and 1b. In a general sense, churn not only describes vehicles leaving the fleet while the latter is processing requests but also associates with communication issues due to the high node mobility, with many vehicles featuring intermittent short activeness periods, typical of dense urban driving. A non-negligible churn causes problems to data localization algorithms as explained in § III-C. In the majority of our experiments (§ IV-D to § V-D), there is negligible churn in the fleet during query execution when regarding the timescale for query resolution (with queries lasting only up to 30s, and $0.02 \leq \text{Churn}_{30s} \leq 0.08$ as shown in Figures 1a,1b). Longer queries, subject to longer churn intervals, are studied in § V-E.

Dynamic changes to the active fleet pool are also based on arriving vehicles. While vehicles joining the pool do not alter the execution of the requests being currently processed by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

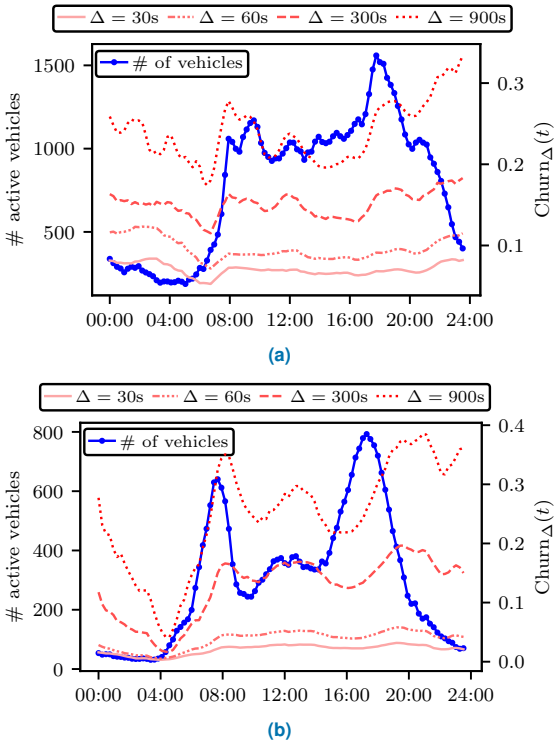

FIGURE 1: Number of active vehicles and churn during one day in the (a) *Geolife* and (b) *Volvo* dataset (see Definition 1 for a formal definition of *churn*).

the fleet, new vehicles support the execution of the running queries when the number of vehicles with positive answers to a query is scarce or declining due to non-negligible churn (exacerbated for the *Volvo* dataset with its lower active vehicles count).

B. EXPERIMENT SETUP

We will present here the components and key settings of the evaluation of our proposed algorithms, involving the adaptation of real-world datasets and parameters.

Query response time calculation

To evaluate our algorithms, we define 15 queries to be run locally on the vehicles (presented in § IV-C). The requests 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); size(q) denotes the amount of code and extra data[1] that needs to be transferred from C to each vehicle in order for the latter to be able to process $r(q)$ on-board. The elapsed time $R(v, q)$ (in milliseconds) needed between the coordinator sending a request message $r(q)$ for query q to a vehicle v and the reception of the corresponding answer is approximated as

$$R(v,q) = T_l + \frac{\text{size}(q)}{T_d} + T_p(v,q) \tag{4}$$

[1]For example, GPS positions of Points of Interests (POIs) such as parking lots or fuel stations.

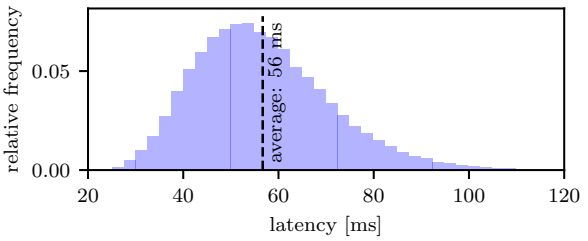

FIGURE 2: Distribution of wireless round-trip latencies T_l (modeled after [23]).

where T_l is a round-trip latency for wireless communication, T_d is the wireless link data rate, and finally $T_p(v, q)$ is the time needed by the vehicle to decide if it can answer positively to $r(q)$ or not. The transmission time of the answer, considering that the answer is of constant and small size (for a yes/no reply and a constant amount of additional information such as the vehicle id, the time it took for the processing, etc.), is neglected here (it can be accounted as part of T_l).

Resolution time of concurrent queries

The experiment process is done as follows. The coordinator node C receives a certain number of queries in a random uniform order and starts the batch of sending requests to vehicles in the same order as the queries' arrival times. The queries are then resolved in parallel by the vehicles and C reacts to each message reception by either just updating its internal statistics for the corresponding query q or by spreading the request $r(q)$ over the fleet to a new set of vehicles. As introduced in § II-A, vehicles possess a *task queue* 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. A vehicle v hence starts processing a request as soon as v is done with the processing of its already queued tasks. When considering multiple queries concurrently processed at C, the reception time $R'(v, q_k)$ of v's answer to the request $r(q_k)$ corresponding to the k-th received query q_k at v is obtained as

$$R'(v,q_k) = \max\{R'(v,q_{k-1}) + T_p(v,q),\ t_k + R(v,q)\} \tag{5}$$

where $t_1 < t_2 < \cdots < t_k$ indicate the sending times of requests $r(q_1), \ldots, r(q_k)$ to vehicle v, and with $R'(v, q_1) = R(v, q_1)$ where $R(v, q)$ is calculated using equation 4.

Real-world values used for the parameters

In our set of experiments, we have set $T_d = 10$Mb/s, which is within current 4G/LTE download rates[2] (similar results are obtained using 5G parameters). To model a non-deterministic but realistic 4G round-trip latency T_l, we sample T_l randomly from the Gamma distribution shown in Figure 2, as modeled after the results from a study of 4G latencies across several

[2]To take into account packet losses, T_d is chosen inferior to typical broadband bandwidth.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

TABLE 1: Selected query conditions and their parameters (QR = share of positive answers over the dataset, G = *Geolife* and V = *Volvo*).

Key	Time span	Size (Kb) in G | V	Condition to fulfill "$q.P$" for G	for V	QR in G (%)	V (%)
Q_1	8-12	0.3	At least 1 record during the time span		56	60
Q_2	0-24	7.5 | 19.9	Driven within *50m* of any parking lot	*25m*	42	43
Q_3	17-18	1.3	Continuous records *with* ζ_G [a]	*with* ζ_V [a]	29	28
Q_4	0-24	0.5	Driven through City area	*with* ζ_V	18	
Q_5	17-18	1.1	Maximum speed reached *over 89km/h*	*over 99km/h*	12	
Q_6	0-24	1.6	Instant speed over 42km/h *for 10min*	*for 18min*	8	
Q_7	0-24	1.4	Driven in Downtown *with* ζ_G consecutive records	*with* $\zeta_{V'}$ [a]	5	
Q_8	17-18	0.8	Passed by *City area*	*Downtown area*	4	
Q_9	12-13	0.7	Stayed in City area for all time span		1	
Q_{10}	0-24	3.8 | 7.7	*Stopped* at any gas station for a short duration [b]		1	
			The queries below are only defined for the *Volvo* dataset.			
Q_{11}	0-24	4.8	*Combustion* engine used less than 10% of the time		34	
Q_{12}	0-24	5	Driven on *electric* mode only outside City area		25	
Q_{13}	0-24	4.4	Driven using *hybrid* mode for 10min		10	
Q_{14}	0-24	6.6	Instant speed on *electric* mode reached over 100km/h		7	
Q_{15}	0-24	4	Passed by 3 distinct electric vehicle charging stations on *electric*		4	

[a] $\zeta_G = (80, 5)$, $\zeta_V = (85, 10)$, $\zeta_{V'} = (300, 10)$, where $\zeta = (\tau, \delta)$ requires at least τ consecutive measurements spaced at least δ seconds apart.

[b] For *Geolife*, we require ζ_G records within 50m of a gas station; for *Volvo* the vehicle must stop (speed = 0km/h) for 10min within 20m of it.

mobile carriers in the UK [23]. To have a fair estimation of $T_p(v, q)$, we have computed all queries on a vehicular processing unit representative [20], [7]: an ODROID-XU4 single-board computer to approximate the limited processing headroom 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 as $T_p(v, q)$ 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.

C. SELECTED DATA LOCALIZATION QUERIES

In this subsection, we present our selection of data localization queries used for the static and dynamic fleet scenarios. Please note that these queries are tailored to the two datasets/fleets employed, each of which has a known (geographic) focus. In the general case, basic a-priori knowledge, e.g., vehicle type or region (which can be assumed to be known to the vehicle manufacturer), can be used to select a subset of a fleet before deploying the actual query over the now filtered fleet.

Queries for the static fleet

We introduce here a set of 15 queries, representative of possible vehicular analysis tasks. The queries match typical interesting events occurring in Vehicular Networks [24] (driving close to POIs such as parking spaces, detecting traffic jams, etc.), thus giving meaningful insights into the fleet's behavior. They were chosen to represent different requirements (on time interval, queried sensors, geographic constraints, sampling constraints, etc.). They furthermore have distinct positive answer rates ranging from about 60% to about 1%. Table 1 presents (cf. § II for notations) the query q's 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 QR (rounded to closest percentage) for *Geolife* and *Volvo* datasets. The parameters of the first 10 queries have been slightly tuned between the two datasets (in Table 1 the additional column for $q.P$'s description indicate differing parameters in the query's condition in *Volvo*) so that each query in both datasets has a similar fraction of positive answers. Recall that size(q) corresponds to the size of the program plus the extra data required to check $q.P$, cf. § IV-B. Of the queries, 10 are run over both datasets whereas 5 additional queries focus on signals only contained within the *Volvo* dataset. 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, if not stated otherwise, all queries will require $n = 50$ answers to get resolved. Setting an adequate value for the parameter n is a non-trivial task that is both query- and data-dependent and is linked to the post-treatment of the vehicle selection process and the end-application. For the case of statistical estimation of the true answer rate QR, The impact of the choice of n with the presented algorithms is explored thoroughly in § V-D, where the value $n = 50$ is shown to provide a good trade-off between estimation accuracy and excessive vehicle involvement over the queries analysed in this work.

Queries for the dynamic fleet

As mentioned in § IV-A, short queries (in terms of resolution time) entail a similar behavior in a dynamic fleet as the fleet remains stable during the time used to resolve the query. All queries defined so far fall in this category, as most of the time, they get resolved in less than one second – whereas the churn for 30s is below 5% of vehicles (cf. § IV-A). To be able to ob-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

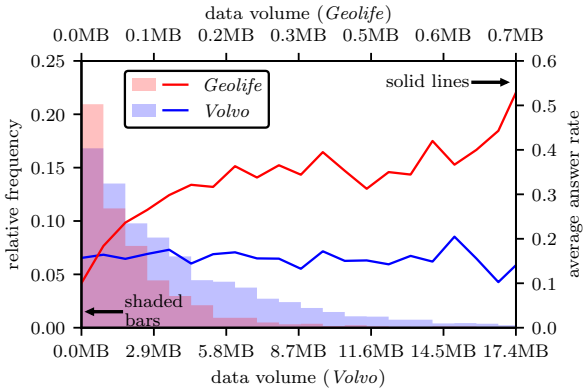

FIGURE 3: Left axis: Histogram of data volumes per car over each respective dataset. Right: Average answer rate for a vehicle with a certain amount of data. Results shown for the *Geolife* (red) and *Volvo* (blue) datasets.

serve differences in the algorithms' behaviors, we introduce "long" versions of the ad-hoc queries previously introduced and described in detail in Table 1. The long versions are obtained by multiplying both the transfer time of the requests and the time needed to process them by a constant of 1000, for a realistic distribution of answering times representative of a fleet with a higher amount of local data or heavier computational tasks used for queries' conditions.

D. DISTRIBUTION OF DATA AND QUERY ANSWERS RATES

The average answer rates over all queries as well as the distribution of the data volumes are presented in Figure 3 for the *Volvo* and *Geolife* dataset; the x-axes range over data volumes in MB within *Volvo* (lower axis) and *Geolife* (upper axis) datasets. For *Geolife*, the average query answer rate (red line) appears to be positively linked to the data volume (shaded red bars); thus, vehicles with larger amounts of data will have a higher chance to answer requests. For *Volvo* (blue line), the average query answer rate is almost flat, which indicates that vehicles with a large amount of data (shaded blue bars) are roughly as likely to answer "yes" to a request as vehicles with only little data. Concerning the distribution of data volumes among the fleet (shaded bars), the *Volvo* dataset presents a significantly longer tail, indicating that inter-vehicular differences in data volume are greater.

V. EVALUATION RESULTS

We show in this section the experiments' results. To compare the performance of the different algorithms, we will use the evaluation metrics defined in § II-D, namely the query resolution time and the fleet workload. To show the results, we will frequently use violin plots, which indicate the median of a distribution with a horizontal bar, while the distribution itself is shown vertically in shaded color.

A. PARAMETRIZATION OF THE ALGORITHMS

To choose well-fitting parameters for our evaluation, we explore the parameter space for BALANCEREQUESTS in the *Geolife*

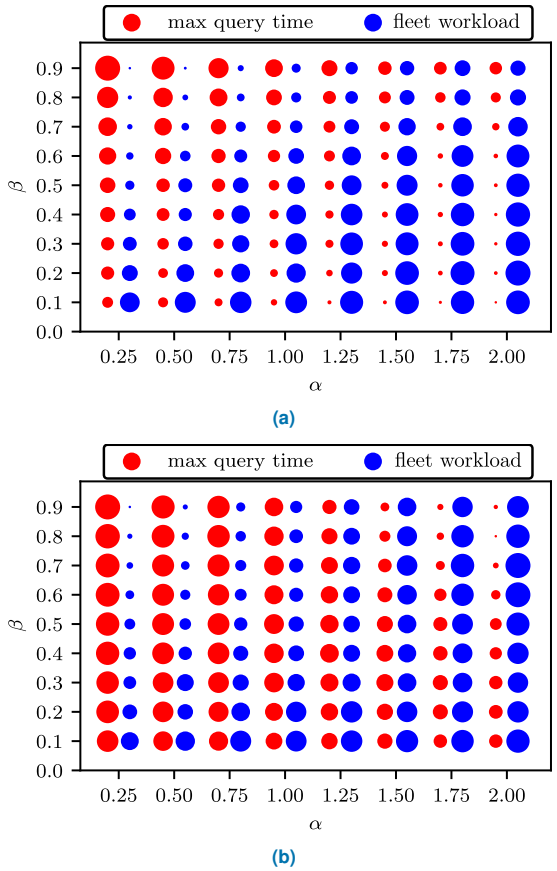

FIGURE 4: Maximum query resolution time and fleet workload *(static model)* needed to resolve all queries over the *Geolife* and *Volvo* datasets for BALANCEREQUESTS for different α, β. Circle size scales with maximum query resolution time (red) and fleet workload (blue), respectively.

and *Volvo* dataset. We run 10000 times the query sets with different values for the parameter α (proportion of vehicles to ask; higher value translates to asking more vehicles) and β (fraction of vehicles to wait before asking next batch; higher fraction translates to longer waiting time between two request batches). For each run, we measure the time needed to resolve all queries (i.e., the maximum query resolution time among the query set) and the fleet workload and present them on a 2D plot in Figures 4 (note that absolute values are given in Figure 5).

Based on the fleet workload (blue) displayed in Figure 4, in both datasets, for lower values for β and higher values for α, more vehicles than necessary tend to be requested while not waiting for everyone's answer before requesting a new batch of vehicles. The consequence in this setting is on the one hand a high analysis cost, as more vehicles participate in the queries resolving task, but on the other hand, the resolution time is relatively lower than other configurations of (α, β). Focusing on the maximum query resolution time (red), the situation is different between the *Geolife* or *Volvo* dataset: When vehicles tend to answer "no" because of lack of data (as for the *Geolife* dataset, cf. § IV-D), hence responding much quicker than positive vehicles, and β is rather small, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

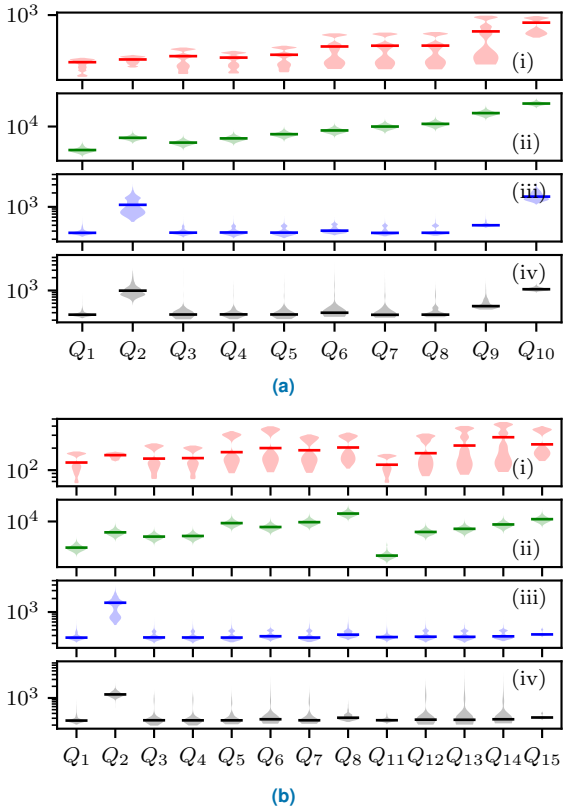

FIGURE 5: Query resolution time (in ms) *(static model)* for all valid queries executed over the (a) *Geolife* and (b) *Volvo* dataset for (i) BASEEAGER, (ii) BASELAZY, (iii) BALANCEREQUESTS, (iv) BALANCELOAD.

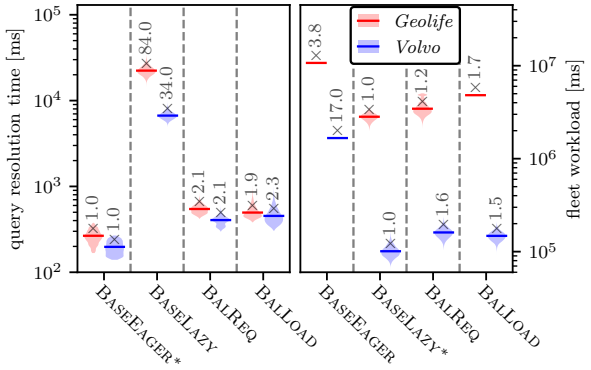

FIGURE 6: Query resolution time (left) and fleet workload (right) *(static model)* of the four algorithms for the *Geolife* (red) and *Volvo* dataset (blue). The starred algorithm is the respective baseline, the y-axis is logarithmic to suit the different scales.

estimation p (cf. Algorithm 3) of positive answers will be too low; then (as β is small) many vehicles are requested rapidly in the first few rounds. The consequence is a shorter resolution time but higher fleet workload, as seen in the *Geolife* experiments. On the contrary, if the data is distributed over the fleet more fairly (as for the *Volvo* dataset, cf. § IV-D), and when β is low, there will be a bias towards positive answers with queries that require a full data scan before they can be answered negatively, whereas vehicles answering positively need only find the first matching record(s). The consequence is that p becomes an overestimation of the real fraction of yes-answers and one observes a succession of small batches of vehicles being requested, as observed in the *Volvo* experiments. When β approaches 1, the estimation p becomes unbiased and better trade-offs are obtained; however, note that a high β is impractical for longer queries, as is shown in § V-E. For the remainder of this section, we set $\alpha = 1.25$ and $\beta = 0.7$ as these values present a suitably balanced trade-off between the two measured performance metrics over both datasets; other nearby values for (α, β) produce similar results that only slightly advantage one metric over the other, as explained above.

B. COMPARISON OF THE ALGORITHMS IN THE STATIC MODEL

To give a general idea of the resolution time for the different queries, we present quantitative results in Figure 5 (an intra-algorithms comparison is done in Figure 6): the query resolution time is measured over 10000 experiment repetitions consisting in resolving all 10 (*Geolife*, a) / 15 (*Volvo*, b) queries arriving in random order; for the *Volvo* dataset, Q_9 and Q_{10} have been removed here and for all following experiments as all vehicles end up being contacted (there are fewer than 50 positive answers in this case, violating the assumption $k_q > n$ from § II-A). The main findings to note are: BASEEAGER's and BASELAZY's resolution time varies clearly depending on the queries' answer rate (lower answer rate QR is associated with larger resolution times, see Table 1 for QR per query), while BASELAZY is one to two orders of magnitude slower; and BALANCEREQUESTS and BALANCELOAD present similar query resolution times that do not vary significantly with the queries' answer rate (except for Q_2 and Q_{10} [*Geolife*]). Also, note these computationally heavier queries Q_2, Q_{10} (requiring to check spatial proximity to multiple points of interest) get resolved significantly slower than lightweight queries. BASEEAGER shows large variations in resolution time for the same algorithm and query because, contrary to all other algorithms, the algorithm itself is purely deterministic and highly dependent on the order in which queries arrive: indeed, if a "heavy" query is sent first to every vehicle, all the nodes will need to process it before moving on to the next query (cf. the FIFO task queue as described in § II-A), potentially slowing down subsequent lighter queries (this also explains why the balanced-algorithms may outperform BASEEAGER by contacting smaller subsets of vehicles, see Figure 10).

As a summary, Figure 6 presents the query resolution time (left side) and fleet workload (right side) over all queries for the four algorithms relative to the average resolution time of BASEEAGER and the average fleet workload of BASELAZY, respectively (marked by stars). The query resolution time is almost two orders of magnitude higher for BASELAZY (in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

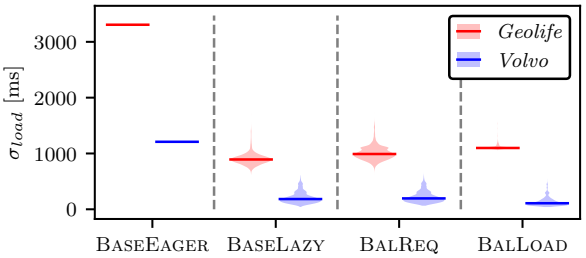

FIGURE 7: Standard deviation σ_{load} of local workloads *(static model)* between vehicles over both datasets and 10000 experiments.

Geolife dataset) than for BaseEager, whereas the Balance-algorithms are almost as fast as BaseEager, which shows the best resolution times in both datasets. For the fleet workload, BaseLazy outperforms BaseEager by a factor of up to 17. The Balance* algorithms perform again well in this metric on both datasets, having an average cost close to the baseline.

Finally, BalanceRequests's fleet workload shows the dependence on the distribution of data in the fleet: with skewed data (as in the *Geolife* dataset), it outperforms BalanceLoad by a margin of 40%, whereas in a uniformly spread dataset (e.g., *Volvo*) it performs marginally worse.

C. FAIRNESS OF THE ALGORITHMS

The presented algorithms distribute clearly differently the workload over the vehicles. We measured the standard deviation σ_{load} of the local workloads *between* the vehicles with non-zero workloads, for executing all queries (cf. § IV-C) over 10000 experiments and for both datasets, presented by Figure 7. Low values of σ_{load} indicate that all vehicles have a similar workload, and vice versa for high σ_{load}. In BaseEager, the workload is distributed deterministically as every vehicle checks every query (even though the execution order may vary in different runs), and thus the value for BaseEager results only from vehicles needing different times to execute all queries. The inter-dataset differences may be explained by the fact that *Geolife* exhibits larger variance in the average answer rate between vehicles, cf. Figure 3. All other data localization algorithms show a smaller spread of the workload, hence a fairer distribution, as more vehicles have similar workloads. BaseLazy provides the fairest outcome in this sense in the *Geolife* dataset, closely matched by BalanceRequests and BalanceLoad, while the latter provides small improvements over the three in the *Volvo* dataset.

D. ESTIMATION OF THE FRACTION OF YES-ANSWERS

In all previous experiments, the number n of required answers was set to 50. This section now investigates how this parameter influences the outcome of the different presented data localization algorithms. Recall that the number of required answers allows one to select a fixed number of vehicles from the fleet satisfying the query's condition for further analysis. One may estimate in this fashion the true fraction QR of vehicles satisfying the query in the full fleet, with a higher number of required answers providing intuitively a

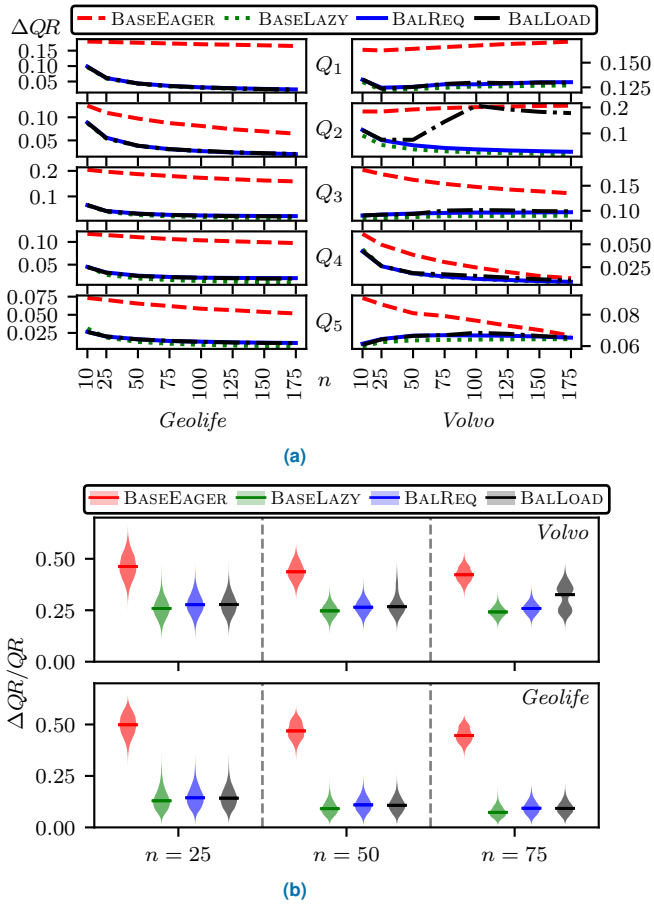

FIGURE 8: (a) Average absolute error ΔQR and (b) relative error $\Delta QR/QR$ of the estimation of yes-answers *(static model)* for queries $Q_1 - Q_5$ with different data localziation algorithms over both datasets.

better estimation of that fraction. The estimation is given by n/m, where n yes-answers have been collected over m total received answers. Here, the validity of the aforementioned intuition will be investigated. Figure 8a presents the average absolute error $\Delta QR = |QR - n/m|$ on the estimation of yes-answers among the fleet for the first five defined queries with $QR = 56/60\%, 42/43\%, 29/28\%, 18\%$ and 12%, respectively (*Geolife/Volvo*, cf. Table 1 for details about the queries), and n ranging from 10 to 175 required answers. For each n and each algorithm, 10000 experiments were conducted where all 5 queries are being resolved in parallel. Then, for each experiment and each query, the share n/m of yes-answers provided by the algorithm at the moment that the particular query is resolved is recorded. Since BaseEager asks every vehicle in the fleet, the estimation is provided based only upon the fastest vehicles to answer and ends up providing the least precise estimation of all tested algorithms. On the contrary, BaseLazy, by asking one vehicle at a time chosen randomly upon receiving negative answers, bases its estimation on a purely random pool of vehicles, hence providing the best estimation unbiased by the time vehicles require to answer the query. In between, BalanceRequests and BalanceLoad provide reasonable trade-offs; as both base

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

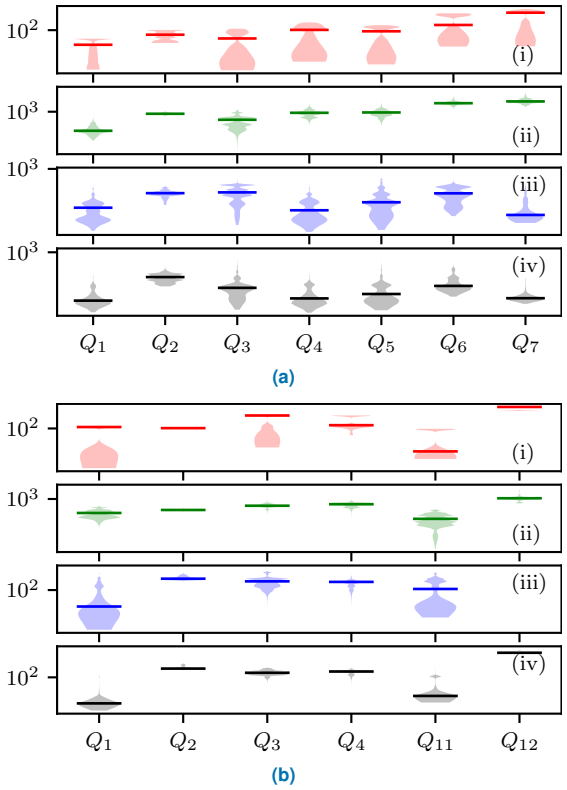

FIGURE 9: Query resolution time (in s) *(dynamic model)* and all valid queries executed over the (a) *Geolife* and (b) *Volvo* dataset for (i) BASEEAGER, (ii) BASELAZY, (iii) BALANCEREQUESTS, (iv) BALANCELOAD.

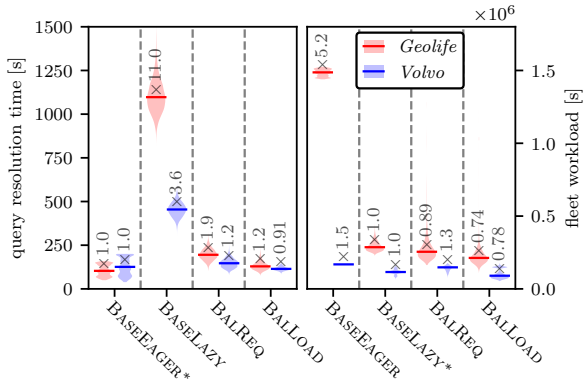

FIGURE 10: Query resolution time (left) and fleet workload (right) *(dynamic model)* of the four algorithms for the *Geolife* (red) and *Volvo* dataset (blue). The starred algorithm is the respective baseline.

their estimation on the first 70% of vehicles to answer a particular query (as $\beta = 0.7$), they both feature bias as BASEEAGER, but to a lesser degree. The main difference between the two algorithms is that BALANCELOAD introduces another bias on top of using the 70% fastest vehicles, which is selecting vehicles with a current lower load rather than random ones as in BALANCEREQUESTS; this additional bias seems strongest in Q_2 in the *Volvo* dataset. On most queries, the balanced algorithms perform nearly as well as BASELAZY. We note that they present almost identical estimations except for query Q_2, where BALANCELOAD, prioritizing spreading the queries fairly among the fleet, under-performs in the *Volvo* dataset. Figure 8b summarizes the estimation performance of all four algorithms on both datasets by presenting the average relative error $\Delta QR/QR$ of the estimation of yes-answers for the 10000 experiments. The advantage of the introduced algorithms is clear: they provide mostly good estimates independently of the required number of answers, especially considering the low values of n compared to the size of the fleets (*Geolife*: 10528; *Volvo*: 3462), while the disadvantage of the secondary bias of BALANCELOAD becomes apparent again in the *Volvo* dataset.

E. COMPARISON OF THE ALGORITHMS IN THE DYNAMIC MODEL

Recall that in the dynamic model, vehicles may join and leave the fleet at any time during a query's resolution. For the churn values to be in line with those observed in real setups (cf. § IV-C), we have modeled the arrival and departure of vehicles during a time interval by using real-world traces: in the following experiments, the set V_t representing the fleet at time t consists of all vehicles that have records within 7.5s of t. Furthermore, here long queries defined in § IV-C have been used where the multiplicative factor has been set to 1000. This shifts the fastest vehicles from answering within milliseconds to seconds and the slowest from a few hundreds of milliseconds to minutes (on selected queries); similarly, the transfer time of 1-20ms becomes 1-20s (the equivalent of 1-20 MB of data having to be transferred per query). The timeout (introduced in § III-C) is set to 100s and the number of answers required per query is set to $n = 50$ as in previous experiments.

Figure 9 presents the query resolution time in the dynamic model following the same conventions as Figure 5. We use in the experiments a query batch of 7 queries (*Geolife* dataset) and 6 queries (*Volvo*) with a starting time of 18:00; the remaining other queries were discarded as not solvable considering only vehicles active past that point in time. The main outcomes in regards to the adaptation of the algorithms to dynamicity are as follows: (i) BASEEAGER's and BASELAZY's resolution time is less dependent on the queries' positive answer rate; (ii) BALANCELOAD performs similarly to BALANCEREQUESTS with a slight improvement thanks to spreading the requests over more vehicles, which then decreases the chance that a vehicle leaves the network before having emptied its local request queue; and (iii) queries that require new arrivals to get resolved, such as Q_{12}, display high resolution times regardless of the spreading algorithm used (however, we note that in these situations, BALANCEREQUESTS is more likely to fail to collect enough answers, as it does for Q_{12} in Figure 9 b).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

Figure 10 shows, as a summary, the average query resolution time and fleet workload over all selected queries[3] of both datasets for each algorithm relative to BASEEAGER and BASELAZY, respectively (in a similar fashion as Figure 6). Contrary to the static setting, where the best-performing of the Balance- algorithms varies depending on the distribution of data over the fleet (cf. § V-B), in a dynamic environment, BALANCELOAD clearly performs best. BALANCEREQUESTS displays overall favorable trade-offs, performing close to each baseline, but slightly slower and with a higher cost than BALANCELOAD. The latter performs close to or better than BASEEAGER in terms of time, and better than BASELAZY in terms of load. This is the consequence of the way the vehicles are selected in the algorithm; those with the lowest current local load are requested first (hence favoring freshly arrived vehicles), which in our datasets biases the selection process towards vehicles with higher chances of answering positively or vehicles answering faster (see discussion in § V-A). A clustering of relevant data in those vehicles may further exhibit the causes of BALANCELOAD's higher performance.

F. SUMMARY OF THE RESULTS

Table 2 summarizes the average resolution time and relative fleet workload (compared to BASEEAGER) over all queries for all algorithms and datasets in the static and dynamic fleet model. On average, the proposed algorithms resolve queries up to 40 times faster than BASELAZY while consuming only $1/3$rd of the resources of BASEEAGER (BALANCEREQUESTS, static model, *Geolife*).

The presented solutions (a well-tuned BALANCEREQUESTS and BALANCELOAD) provide substantially improved trade-offs of query resolution time versus on-board workload compared to baseline solutions, and allow tuning between the tradeoffs by varying the estimation of required vehicles to ask in the next iteration (via the parameter α) and the waiting times for slow-processing vehicles (via β). Furthermore, a query's resolution time in the proposed algorithms is shown to not be negatively impacted by a low positive answer rate among the fleet.

BALANCELOAD, presenting shorter resolution time and slightly larger fleet workload, produces a workload more fairly spread over the vehicles; however, it may provide a less accurate estimation of the fraction of positively answering vehicles once queries are resolved. BALANCEREQUESTS provides the most balanced trade-offs overall, performing almost as good as each baseline solution both when considering a uniform distribution of positive answers (*Volvo* dataset) or a skewed distribution (*Geolife* dataset). Finally, BALANCELOAD is overall more suited to "dynamic" scenarios, i.e.,when the queries require long enough processing times for the fleet churn to become noticeable.

[3]All queries described in Figure 9, except for *Volvo* where Q_{12} has been excluded from the resolution time plot, as it did not always terminate.

TABLE 2: Summary of the results (average over all queries) for both datasets, all algorithms, and static & dynamic models.

	Algorithm	Resolution time (s)		Fleet workload (s)	
		Geolife	*Volvo*	*Geolife*	*Volvo*
static	BASEEAGER	0.27	0.20	10715	1667
	BASELAZY	22.65	6.75	2823	103
	BALANCEREQUESTS	0.55	0.40	3616	162
	BALANCELOAD	0.52	0.48	4816	148
dynamic	BASEEAGER	105	204	1484451	168280
	BASELAZY	1105	556	293245	115858
	BALANCEREQUESTS	197	148	281650	141771
	BALANCELOAD	132	220	227068	101282

VI. RELATED WORK

Having studied in this work the problem of how to localize, efficiently and in a distributed manner, relevant data in a vehicular fleet for analysis applications, in the following paragraphs we discuss work about topics that associate with or have similarities to the problem.

The traditional approach to query a set of vehicles has been through SQL-inspired languages [25], [26], [27] 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 (e.g., [28], [29], [30], [31], [32], [33]) and (ii) the full fleet was queried upon receiving new queries (as in [34]), contrary to our work in which a known and fixed set of *general* queries is deployed from a centralized point to the fleet and only some vehicles in the current fleet may have relevant data to answer the queries. Also, many works in the field are based on an advantageous usage of geographical properties of the distribution of Road Side-Units (among others [31], [32], [33], [35]), whereas our work is only based on the already widespread mobile broadband infrastructure as well as data analysis capabilities already in place at car manufacturers' data centers. 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 [36], [37], [38] or 2-tier architectures [39], [40]) to resolve the query. In this work, we do not presume any connections between vehicles; this positions our work in readily deployable technologies on modern vehicles. A querying approach for vehicle selection was recently studied in [18] in which a request is sent to all available vehicles to detect candidates to participate in Federated Learning. The vehicles send updated responses to the query over time as they are collecting new data, and eventually a subset of the vehicles that answered positively is chosen. In contrast, our algorithms attempt to limit the number of vehicles that are queried for data, thus allowing for more concurrent queries, while novel data discovery is only supported via new queries.

The problem of localizing the relevant data or "data localization" features many similarities with the concept of data aggregation in wireless sensor networks [41], [42], [43], [44], [45]. Usual aspects of data aggregation that differ from data localization include a continuous aspect (rather

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118596, IEEE Access

than an on-the-fly query approach for data localization) and dissemination of information to nearby nodes (rather than to a single sink node for data localization). Since the focus taken here is on the localization of data, approaches for efficient data gathering [14], [20], [7] and aggregation do well complement the initial localization phase. The way our algorithms have been designed also relates to the large field of information gathering (see [46] and references therein). In both concepts, online decisions are iteratively taken to gain knowledge of a hidden state. For instance, in [47], the authors design near-optimal algorithms to pick the right set of tests in order to maximize the *value of information*, with applications to medical diagnosis and troubleshooting. Note such approaches can be used to design the right set of queries (similar to the aforementioned tests) to resolve a particular task whereas our work concentrates on how to efficiently and distributively resolved those queries.

The fundamental opposing metrics studied in our paper that are the time to resolve the queries and the computational overhead induced on the fleet are similarly observed in the field of job scheduling in distributed computing. Parallel and redundant job execution can decrease job execution times in heterogeneous environments (e.g., through speculative scheduling in *MapReduce* [48]), at the cost of increasing contention and overall workload [49]. In contrast, in our work parallel execution is always required, and no node is a priori known to be fundamentally able to fulfill a given task.

In vehicle data analysis, privacy aspects are important when dealing with for example location-based services [50], [51], [52] and privacy-preserving cloud-based query processing [35]. 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.

VII. CONCLUSION

This work proposes two distributed algorithms for data localization in Vehicular Networks. To the best of our knowledge, this paper is the first to propose a data localization mechanism over a Vehicular Network through request spreading, focusing on acquiring only a limited number of answers from the fleet and considering as a performance metric the computing workload of the vehicles. The focus lies on the vehicle selection phase necessarily performed prior to data gathering over large vehicular fleets, typically for selecting vehicles that triggered a particular condition or event [6], [20], [7]. As this work also shows, this vehicle selection mechanism can be used as-is for estimating the occurrence of particular events in the vehicles' recent data while incurring low overhead on the Vehicular Network as a whole. The proposed algorithms balance (i) the overall time needed to identify a subset of vehicles holding relevant data and (ii) the local computational overhead each vehicle pays to check whether a set of properties hold for its data. As shown with analytical argumentation and experimental evaluation, conducted with real-world data traces, the algorithms provide means to tune the trade-off between (i) and (ii) in interesting ways. In particular, it appears that it is possible to significantly reduce the query resolution time, with only a small extra load imposed on the vehicles, compared to the baselines that can optimize only one of these metrics (achieving for example up to 40 times faster resolution while saving more than 65% of the computing resources). These results indicate that the adoption of a data localization phase prior to the execution of additional analysis steps for example in a Federated Learning scenario can occur with little overhead with respect to time and computing resources, thus enabling better learning outcomes for a comparably small price. Furthermore, our results show that the distribution of the work to the vehicles can happen fairly, even for skewed data distributions, to alleviate the risk of overloading individual vehicles. This work sets the basis for several paths to investigate in the future. One avenue is the porting of the proposed algorithms to V2V [53] rather than centralized V2I setups. A second direction is to explore how our algorithms can be integrated within existing simulators (e.g., with a traffic and/or network simulator) to produce richer simulation environments for benchmarking smart analysis in Vehicular Networks. Lastly, investigating the use of correlations between queries could be a promising way of efficiently selecting those vehicles for a query that have answered positively to a similar query in an earlier execution by adaptively changing the parameters of our algorithms during a data localization query's resolution.

ACKNOWLEDGMENT

This work is supported by Volvo Car Corporation; the Swedish Government Agency for Innovation Systems VINNOVA, proj. "Onboard/Offboard Distributed Data Analytics (OODIDA)" (DNR 2016-04260) and "Automotive Stream Processing and Distributed Analytics (AutoSPADA)" (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).

APPENDIX.

A preliminary formulation of the problem was presented in [54]. The present article builds on those test results and presents an extensive study that includes a detailed problem formulation, as well as varying system models and parameters along with algorithmic designs for them. For comparison, we list here the main novel contributions of the present work:

1) the system model is made more realistic by introducing a dynamic fleet model where vehicles can leave and join at any time (§ II-B);
2) the data localization algorithms presented here are enhanced to adapt to changes in the set of available

vehicles (§ III-C), in accordance with the new system model;
3) the evaluation has been updated with use-cases that account for the new system model (§ V-E);
4) the evaluation is extended with a thorough comparison of the algorithms' behaviour, using a larger set of performance metrics, as well as enhanced experiment repetitions for higher statistical certainty, and a more extensive analysis of the parametrization of the proposed algorithms (§ V-C, § V-D);
5) the study includes more realistic modeling of the communication delays (§ IV-B); and lastly
6) the evaluation framework and algorithms are openly published[4] to enable replicability of our experiments as well as to spark further research.

REFERENCES

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

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

[3] D. Palyvos-Giannas, B. Havers, M. Papatriantafilou, V. Gulisano, Ananke: a streaming framework for live forward provenance, Proceedings of the VLDB Endowment 14 (3) (2020) 391–403.

[4] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, W. Wang, A survey on mobile edge networks: Convergence of computing, caching and communications, Ieee Access 5 (2017) 6757–6779.

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

[6] Y. Lai, F. Yang, J. Su, Q. Zhou, T. Wang, L. Zhang, Y. Xu, Fog-based two-phase event monitoring and data gathering in vehicular sensor networks, Sensors 18 (1) (2018) 82.

[7] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, M. Papatriantafilou, A. C. Koppisetty, Driven: a framework for efficient data retrieval and clustering in vehicular networks, Future Generation Computer Systems (2020).

[8] E. Schoch, F. Kargl, M. Weber, T. Leinmuller, Communication patterns in vanets, IEEE Communications 46 (11) (2008) 119–125.

[9] M. Gerla, L. Kleinrock, Vehicular networks and the future of the mobile internet, Computer Networks 55 (2) (2011) 457–469.

[10] S. Costache, V. Gulisano, M. Papatriantafilou, Understanding the data-processing challenges in intelligent vehicular systems, in: 2016 IEEE Intelligent Vehicles Symp., IEEE, 2016, pp. 611–618.

[11] D. Palyvos-Giannas, V. Gulisano, M. Papatriantafilou, Haren: A framework for ad-hoc thread scheduling policies for data streaming applications, in: 13th ACM Int'l Conf. on Distributed Event-Based Systems (DEBS), ACM, 2019, pp. 19–30. doi:10.1145/3328905.3329505.

[12] G. Ulm, E. Gustavsson, M. Jirstrand, Oodida: On-board/off-board distributed data analytics for connected vehicles, arXiv preprint arXiv:1902.00319 (2019).

[13] D. Palyvos-Giannas, V. Gulisano, M. Papatriantafilou, Genealog: Fine-grained data streaming provenance at the edge, in: 19th Int'l Middleware Conf., Middleware '18, ACM, New York, NY, USA, 2018, pp. 227–238. doi:10.1145/3274808.3274826.
URL http://doi.acm.org/10.1145/3274808.3274826

[14] R. Duvignau, V. Gulisano, M. Papatriantafilou, V. Savic, Streaming piecewise linear approximation for efficient data management in edge computing, in: 34th ACM/SIGAPP Symposium On Applied Computing SAC'19, 2019, pp. 593–596.

[15] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[16] S. Lu, Y. Yao, W. Shi, Collaborative learning on the edges: A case study on connected vehicles, in: 2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19), 2019.

[17] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-iid data, arXiv preprint arXiv:1806.00582 (2018).

[18] D. Deveaux, T. Higuchi, S. Uçar, C.-H. Wang, J. Härri, O. Altintas, On the orchestration of federated learning through vehicular knowledge networking, in: 2020 IEEE Vehicular Networking Conference (VNC), IEEE, 2020, pp. 1–8.

[19] A. M. Elbir, B. Soner, S. Coleri, Federated learning in vehicular networks, arXiv preprint arXiv:2006.01412 (2020).

[20] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, A. C. Koppisetty, M. Papatriantafilou, Driven: a framework for efficient data retrieval and clustering in vehicular networks, in: 35th IEEE Int'l Conf. on Data Engineering (ICDE 2019), 2019, pp. 1850–1861.

[21] B. Eisenberg, On the expectation of the maximum of iid geometric random variables, Statistics & Probability Letters 78 (2) (2008) 135–143.

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

[23] Measuring mobile broadband performance in the UK - 4G and 3G network performance, http://static.ofcom.org.uk/static/research/mbb.pdf (2014).

[24] S. Ilarri, T. Delot, R. Trillo-Lado, A data management perspective on vehicular networks, IEEE Communications Surveys & Tutorials 17 (4) (2015) 2420–2460.

[25] Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, ACM Sigmod record 31 (3) (2002) 9–18.

[26] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, S. Madden, Cartel: a distributed mobile sensor computing system, in: 4th Int'l Conf. on Embedded networked sensor systems, ACM, 2006, pp. 125–138.

[27] Y. Zhang, B. Hull, H. Balakrishnan, S. Madden, Icedb: Intermittently-connected continuous query processing, in: 23rd IEEE Int'l Conf. on Data Engineering (ICDE 2007), IEEE, 2007, pp. 166–175.

[28] U. Lee, E. Magistretti, M. Gerla, P. Bellavista, A. Corradi, Dissemination and harvesting of urban data using vehicular sensing platforms, IEEE transactions on vehicular technology 58 (2) (2008) 882–901.

[29] U. Lee, J. Lee, J.-S. Park, M. Gerla, Fleanet: A virtual market place on vehicular networks, IEEE Trans. on Vehicular Technology 59 (1) (2010) 344–355.

[30] Y. Zhang, J. Zhao, G. Cao, Roadcast: a popularity aware content sharing scheme in vanets, ACM SIGMOBILE Mobile Computing and Communications Review 13 (4) (2010) 1–14.

[31] T. Delot, N. Mitton, S. Ilarri, T. Hien, Decentralized pull-based information gathering in vehicular networks using geovanet, in: 12th IEEE Int'l Conf. on Mobile Data Management, Vol. 1, IEEE, 2011, pp. 174–183.

[32] G. M. N. Ali, E. Chan, W. Li, Supporting real-time multiple data items query in multi-rsu vehicular ad hoc networks (vanets), Journal of Systems and Software 86 (8) (2013) 2127–2142.

[33] Y. Lai, L. Zhang, F. Yang, L. Zheng, T. Wang, K.-C. Li, Casq: Adaptive and cloud-assisted query processing in vehicular sensor networks, Future Generation Computer Systems 94 (2019) 237–249.

[34] Y. Lai, L. Zheng, T. Wang, F. Yang, Q. Zhou, Cloud-assisted data storage and query processing at vehicular ad-hoc sensor networks, in: Int'l Conf. on Security, Privacy and Anonymity in Computation, Communication and Storage, Springer, 2017, pp. 692–702.

[35] Y. Lai, Y. Xu, F. Yang, W. Lu, Q. Yu, Privacy-aware query processing in vehicular ad-hoc networks, Ad Hoc Networks 91 (2019) 101876.

[36] C.-L. Liu, C.-Y. Wang, H.-Y. Wei, Cross-layer mobile chord p2p protocol design for vanet, Int'l Journal of Ad Hoc and Ubiquitous Computing 6 (3) (2010) 150–163.

[37] T. Wang, L. Song, Z. Han, Collaborative data dissemination in cognitive vanets with sensing-throughput tradeoff, in: 1st IEEE Int'l Conf. on Communications in China (ICCC), IEEE, 2012, pp. 41–45.

[38] N. Kumar, J.-H. Lee, Peer-to-peer cooperative caching for data dissemination in urban vehicular communications, IEEE Systems Journal 8 (4) (2014) 1136–1144.

[39] S.-L. Tsao, C.-M. Cheng, Design and evaluation of a two-tier peer-to-peer traffic information system, IEEE Communications 49 (5) (2011) 165–172.

[40] C.-M. Cheng, S.-L. Tsao, Adaptive lookup protocol for two-tier vanet/p2p information retrieval services, IEEE Trans. on Vehicular Technology 64 (3) (2015) 1051–1064.

[4] All code and detailed useage instructions are available at https://github.com/dcs-chalmers/dataloc_vn.

[41] L. Krishnamachari, D. Estrin, S. Wicker, The impact of data aggregation in wireless sensor networks, in: Proceedings 22nd international conference on distributed computing systems workshops, IEEE, 2002, pp. 575–578.
[42] R. Rajagopalan, P. K. Varshney, Data-aggregation techniques in sensor networks: A survey, IEEE Communications Surveys & Tutorials 8 (4) (2006) 48–63.
[43] S. Ozdemir, Y. Xiao, Secure data aggregation in wireless sensor networks: A comprehensive overview, Computer Networks 53 (12) (2009) 2022–2037.
[44] J. He, S. Ji, Y. Pan, Y. Li, Constructing load-balanced data aggregation trees in probabilistic wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems 25 (7) (2013) 1681–1690.
[45] R. Kumar, M. Dave, A framework for handling local broadcast storm using probabilistic data aggregation in vanet, Wireless personal communications 72 (1) (2013) 315–341.
[46] Y. Chen, Near-optimal adaptive information acquisition: Theory and applications, Ph.D. thesis, ETH Zurich (2017).
[47] Y. Chen, J.-M. Renders, M. H. Chehreghani, A. Krause, Efficient online learning for optimizing value of information: Theory and application to interactive troubleshooting, arXiv preprint arXiv:1703.05452 (2017).
[48] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, I. Stoica, Improving mapreduce performance in heterogeneous environments., in: Osdi, Vol. 8, 2008, p. 7.
[49] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, Why let resources idle? aggressive cloning of jobs with dolly, in: USENIX HotCloud, 2012.
[50] J. Xu, Z. Jin, M. Xu, N. Zheng, Mobile-aware anonymous peer selecting algorithm for enhancing privacy and connectivity in location-based service, in: 7th Int'l Conf. on E-Business Engineering, IEEE, 2010, pp. 172–177.
[51] X. Huang, R. Yu, J. Kang, Y. Zhang, Distributed reputation management for secure and efficient vehicular edge computing and networks, IEEE Access 5 (2017) 25408–25420.
[52] D. Christin, A. Reinhardt, S. S. Kanhere, M. Hollick, A survey on privacy in mobile participatory sensing applications, Journal of systems and software 84 (11) (2011) 1928–1946.
[53] A. Gidenstam, B. Koldehofe, M. Papatriantafilou, P. Tsigas, Scalable group communication supporting configurable levels of consistency, Concurrency and Computation: Practice and Experience 25 (5) (2013) 649–671.
[54] R. Duvignau, B. Havers, V. Gulisano, M. Papatriantafilou, Querying large vehicular networks: How to balance on-board workload and queries response time?, in: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), IEEE, 2019, pp. 2604–2611.