A Case for User-Defined Governance of Pure Edge Data-Driven Applications

João Mafra, Francisco Brasileiro, and Raquel Lopes

Federal University of Campina Grande, Paraíba, Brazil
jvmafra@lsd.ufcg.edu.br, {fubica,raquel}@computacao.ufcg.edu.br

Abstract. The increasing popularity of smartphones, associated with their capability to sense the environment, has allowed the creation of an increasing range of data-driven applications. In general, this type of application collects data from the environment using edge devices and sends them to a remote cloud to be processed. In this setting, the governance of the application and its data is, usually, unilaterally defined by the cloud-based application provider. We propose an architectural model which allows this kind of application to be governed solely by the community of users, instead. We consider members of a community who have some common problem to solve, and eliminate the dependence on an external cloud-based application provider by leveraging the capabilities of the devices sitting on the edge of the network. We combine the concepts of Participatory Sensing, Mobile Social Networks and Edge Computing, which allows data processing to be done closer to data sources. We define our model and then present a case study that aims to evaluate the feasibility of our proposal, and how its performance compares to that of other existing solutions (e.g. cloud-based architecture). The case study uses simulation experiments fed with real data from the public transport system of Curitiba city, in Brazil. The results show that the proposed approach is feasible, and can aggregate as much data as current approaches that use remote dedicated servers. Differently from the all-or-nothing sharing policy of current approaches, the approach proposed allows users to autonomously configure the trade-off between the sharing of private data, and the performance that the application can achieve.

Keywords: Edge-computing · Community · Analytics · Privacy

1 Introduction

Smartphones are now spread all over the world, being used by the most diverse people, following different cultures and lifestyles. According to statista, we

This work was supported by the Innovation Center, Ericsson Telecomunicacoes S.A., Brazil and by EMBRAPPI-CEEI.

1 https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.
reached in 2020 the mark of 3.5 billion smartphone users. Each smartphone comes with processing and networking capabilities, as well as a set of sensors (e.g. GPS - Global Positioning System, ambient light sensors and microphones). With these small computers around and online all the time, there is a great variety of applications already in place, leveraging their capabilities.

Many of these applications aim at getting sensoring information from the personal smartphones and other types of sensors in place and merging the collected data to extract new information on a larger scale. This technique known as participatory sensing has emerged in 2006 [5] and took a new dimension more recently when associated with Mobile Social Networks (MSN) [7]. Mobile Social Networks are virtual communities of individuals that have some common interests and keep in touch using their mobile devices. By considering the sensing capabilities of the users’ devices, the users can share (their local) data and access the merged data to extract rich information to measure, map, analyze or estimate processes of common interest. For example, GreenGPS, relies on participatory sensing data to map fuel consumption on city streets, allowing drivers to find the most fuel efficient routes for their vehicles between arbitrary end-points [6]. Many applications like that came to life [13,18,19], all of them exploiting the participatory and opportunistic sensing capabilities of mobile devices.

At the heart of these applications are the users with a common goal. These users usually trade personal local data for global information that can be achieved just by collaboration. Only with the combined data collaboratively shared the users satisfy their common goal. It is common to see applications that are built, advertised and, after that, adopted by interested users, gathering data in a collaborative way and, eventually, using the applications to fulfill their own needs. These applications are commonly hosted in cloud infrastructures, requiring some sort of sponsorship, management and technical support to operate them. Many of these applications use machine learning and other artificial intelligence (AI) techniques to extract useful predictions/answers from raw data. This computational model usually centralizes the (collaboratively shared) data and processing to a central server (usually in the cloud), where the machine learning models are trained. Thus, the adoption of such a model boosts the need for external sponsorship and technical support.

Obviously, when the data shared by the users is sensitive there are privacy issues that must be considered. Federated Learning [3] has been proposed as a new architecture to build global machine learning models without data sharing. This architecture considers that AI models are built independently in the users’ devices and then merged with the global model that is located in the cloud. Although data is not shared, the users themselves have no governance over the global model achieved, which is typically hosted in cloud providers.

In summary, participatory sensing, mobile social networks and federated learning are useful frameworks that allow shared knowledge to be extracted from raw data that is collaboratively gathered. However, they require a centralized hosting service where the back-end application runs, usually in the cloud. Keeping application governance in external hands may be inconvenient for a number
of reasons. Just to start, we can mention that users are subject to unilateral changes on usage policies, decided by the provider. Yet more severe, providers can run out of business, or simply decide to stop supporting the applications, usually without any liabilities with respect to the users.

The fact that all data shared by the users is held by a single entity that governs the service gives opportunities to data misuse that can lead to privacy issues. In order to use the application, interested users (that have their local data to share) have to agree with a policy describing how sensitive data will be used. Despite leaving users aware of the use of their information, this data governance model might be risky. First, application providers can omit important details about how the data is used. An example of such a situation occurred recently, when Facebook was subject to a huge penalty from the government of the United States of America for misusing user data. Second, the use of the application is often conditional on acceptance of the terms of use which are established by the external sponsor, giving no chance for negotiation. Therefore, if the user does not agree with the policy, he/she may not be able to utilize the service. In other words, this is usually an all-or-nothing decision. Hereafter, we refer to all these issues discussed so far as *the problem of external governance*.

Central to this problem is the fact that external governance takes from the community the right of deciding how their data is shared, where it is stored, when and by whom it is used. Besides, there is a cost for maintaining such service in the cloud. In this chapter we present an approach that exploits the edge computing resources to provide a *community-governed service* as an alternative way to deliver analytics services to a community of users sharing a common goal.

Community-governed services involve data analysis, typically to help users to find better alternative answers to their common problem. Driven by their shared interest, they exchange data among themselves so they can make better decisions based on more information. For example, neighbors interested in finding the best spots to catch up outdoors can share air pollution related data they gather from the neighborhood to build a comprehensive report about the air quality in the neighborhood. Users indoors, such as a shopping mall, may be interested in the quietest places, and can resort to data collectively gathered to spot those places [21]. Members of a community may be interested in traffic conditions. Google maps does this by aggregating in a centralized environment data collected by users’ smartphones to generate reports about traffic conditions.

The idea of community-governed services is to exploit the participatory sensing concept, but limiting the data exchange to the trusted partners in the social network, and eliminating the need of third party services such as cloud-based application providers. In order to do that, community-governed services follow a peer-to-peer (P2P) architecture: data is generated and processed at the edge of the network without the need to be transferred to the cloud to be processed. The community-governed services use the processing power of the edge devices themselves to gather, process and store the shared data. Of course, these devices are limited in terms of processing, storage and energy consumption, and some applications cannot be built that way, as we explore in Sect. 2.3.
Edge computing [23] is the core of the community-governed services, since the same (edge) devices that collect the data also store and process it. As expected, edge computing avoids data flooding in the cloud, saves bandwidth and reduces applications’ latency, providing better user experience. But these are not the only benefits. By adopting community-governed services built on top of pure edge computing technology (not including fog servers, mobile edge computing servers [15] nor VM-based cloudlets [22]), we empower the data owners and service users to have their services the way the community desires. It is a pure P2P application built to unite the sensing and computing power of edge computing devices, driven and guided by those who feed and use the service.

Our previous work [14] shows a simplified evaluation of this service model, assuming that all members of the community trust each other when sharing data. Since this is impractical, in this chapter, in addition to revisiting the idea of community-governed service, we further analyze how trust relationships among community members impact the quality of the proposed solution.

The rest of the chapter discusses the architectural model of pure edge data-driven applications that allows for community-governed services. We discuss its components, application requirements, and limitations. We also present a use case that analyzes the trade-offs involved in using the proposed model from a service performance point of view.

2 Community-Governed Services Model

2.1 Definition

One of the main advantages of the Community-Governed Services is that it empowers users to jointly define the governance of the data that is manipulated by the service. Also, since the service is cooperatively implemented by the software running at the user’s devices, failures or departure of users are likely to lead to a gentle degradation of the service, and not a complete disruption of the service. In this way, individuals have greater control over the use of their collected data, storing it locally and sharing only with those they trust. In addition, data processing can be done at the edge of the network, using the same devices that collect the information from the environment, avoiding the need for sponsorship to keep the application running in the cloud. To guarantee the governance to users, the proposed service has some fundamental principles:

- **Participatory Sensor**: Environment data collection using sensors installed on smartphones, smartwatches, bracelets, among other devices that are on the edge of the network. To guarantee the engagement of individuals, it is important that they have some common goal to be satisfied by the service as a form of motivation.
- **Data Sharing**: The users can communicate with each other using an MSN and share the collected data individually with their trusted peers. Taking advantage of the mobility pattern of individuals, this can be done at some point in which they are in a zone of local proximity.
– **Edge Processing:** Using Edge Computing paradigm, the collected data is processed on the users’ own mobile devices.

Thus, a community of users has a common goal and harness the power of its own edge devices to collect and process the data collected. Between data collection and processing, users can share their data with trusted users in the same community. Since everything is done at the edge of the network, the need for a third party (logically) centralized server running at a cloud provider is obviated. It increases the robustness of the service by removing the dependency on a central entity represented by the server running in the cloud, eliminates the bottleneck in the communication with the centralized server, and most importantly, allows the community of users to jointly define and manage the governance of the service, which among other things mitigates privacy issues.

### 2.2 System Model

The system is composed by a number of personal devices running the community-governed service. The users utilize the service agent running in their devices to both collect and share data in a participatory sensing way, and query the service. The service agent that runs at each personal device is illustrated in Fig. 1. It consists of six modules: participatory sensor, community sensor, community data collector, community data filter, model builder and query dispatcher.

The *participatory sensor* component is responsible for collecting data in the vicinity of the device. The *community sensor* component takes care of discovering other members of the community. The *community data collector* contacts other members of the community in order to increase the amount of data that is available locally. The *community data filter* component regulates which data should be shared with other members of the community, in both directions, i.e. to whom local collected data can be shared, and from whom data should be requested. The *model builder* component is in charge of creating the service model from all the data collected. Finally, the *query dispatcher* provides the interface to the service.

![Fig. 1. Components of agent [14].](image)
When a new request is received by the query dispatcher component, it uses the model generated by the model builder component in order to answer the request. Whenever a new data item is made available (either by the participatory sensor or the community data collector), the model builder assesses if a new model needs to be created. If this is the case, it uses all the data available to train the new model.

Periodically, the community sensor tries to identify members that are online. This information is passed to the community data filter that, in turn, decides to which members local data could be shared (upon request), and from which other members data could be requested. The community data collector contacts other community data collectors obeying the community data filter decision. Periodically, or upon the detection of an event of interest, the community data collector tries to collect data from the accepted members that are online. When contacted by an external member, the community data collector decides whether it should provide the data locally stored to the contacting member.

2.3 Application Requirements

First, the personal devices of the users must be able to collect data in a passive or active way. Second, the users share common service interests and form a community. So, at some point in time, these devices connect to each other through the formation of mobile social networks, a common local area network, or even a regular Internet connection. The creation of the community allows the users to share the collected data among the trusted peers (participatory sensing). Third, these services involve data analysis through analytical and/or machine learning models. We expect the quality of answers provided by the models to be proportional to the quantity and diversity of the data gathered. Thus, the more data is available to users, the more benefits they can get from the service.

The use of the edge devices to run the community-governed service, as well as to execute the sensing to collect data, limits these activities. In other words, the execution of the service, as well as the sensing activity must be lightweight, and ideally the battery consumption due to these activities should be acceptable to the user. Moreover, data storage consumption should also be low. There are a number of ways to mitigate the impact of these limitations. For instance, training machine-learning models tend to be a compute-intensive procedure. This could be executed only when the device is fully charged, and connected to the charger. Alternatively, simpler statistical models can be used to reduce the computation demand (in our case study we describe one example). Regarding data storage, in many cases a sliding-window approach can be used to release data that is old enough, and as a result less important.

3 Case Study

In order to shed some light on the feasibility of the community-governed service model we conducted a simulation-based case study in the area of public trans-
portation. The choice of the application was based on the fact that it fits the features discussed above, and we have real data to feed our simulation model.

In many cities, urban bus schedules are made public and followed in a very strict way. In cities that use technology in an intensive way, buses can be equipped with sensors and tracked in real-time using a cloud-based service, or even using 5G-based solutions [11], so that unanticipated delays can be spotted, and alternative bus routes can be chosen. Nevertheless, in many places, especially in large cities of developing countries, knowing the actual time that a bus will leave a bus stop can be difficult. In these places, the timetables provided by the bus companies are rarely followed, due to many reasons, including traffic jams, unanticipated maintenance, etc. Not knowing the actual bus departure time can increase wait time at the bus stop, leading to wasted time and, in more serious situations, can make passengers more susceptible to urban violence. Our case study is focused on the latter scenario, and on a particular community of users: students in a big university campus.

A large number of college students use public transportation every week day, to get from home to the university and back. These students share a common interest regarding the bus transportation schedule. By forming a community, they can take advantage of their collective mobility pattern, which can be exploited by a community-governed service. Whenever they leave the university in a bus, they can collect information about which bus line was used, and what time the bus departed from the university. When students get back to the university, all the information they have previously collected is available in their devices. Thus, in this scenario, students that are online at the campus at the same time are able to share their collected data following the model described in Sect. 2.2. This data collected and shared is then processed and analyzed to satisfy common demands of this community.

With that in mind, the users’ common goal in our case study is to estimate the departure time of buses in a bus line at a university campus, using only past travel data collected by the university community that uses this means of transportation, and evaluate how our proposed community-governed service behaves compared to other scenarios, like a typical cloud-based service.

4 Materials and Methods

4.1 Data Source

We have used public transportation data from Curitiba, a city in the South of Brazil with a population of around 2 million inhabitants. A brief characterization of the data and how it was collected can be seen in the work by Braz [17]. He used bus schedule data, raw GPS and smart card records to reconstruct trips at the passenger-level from the original data provided by Curitiba’s Public Transport Department. In the trace rebuilt by Braz, each registry represents a bus travel made by a user. From it, we can get the bus stop that the bus departed, the time of departure, and the bus line associated to the bus. An example of an event in
the trace might be a user whose id is 123456, who left bus stop 151 at 8:00 a.m on May 12 using bus line 500\(^2\).

From the city map, we have selected the bus stops that are in the vicinity of our target university campus, in our case the Universidade Técnica Federal do Paraná (UTFPR), and considered only the trips that left or arrived from these bus stops. These trips, representing a set of arrival and departure events, were then arranged in chronological order.

Some adaptations to the trace and assumptions had to be made, so that we could somehow established the periods of time when a user was at the campus. These are signaled by both arrival and departure events in the trace. For each day and each user in the trace, there may be zero or more of such events. When an arrival event is followed by a departure event at the same day, then we assume that the user stayed at the campus for the period comprised between the arrival time and the departure time. However, if only a departure event is present, without an earlier arrival event at the same day, then we arbitrated that the user had arrived one hour before the time of the departure event. Similarly, if an arrival event is present with no later departure event, then we arbitrate that the user stayed at the campus for one hour since its arrival.

Our adapted trace has information about 74,907 trips leaving (45,346) or arriving (29,561) at bus stops near the university campus, between May 2017 and July 2017, from 18,662 different users.

4.2 Simulation Model

Every departure from the campus that appears in the trace considered generates a request to the service. Let \( t_d \) be the time of a departure logged in the trace. We assume that at some time \( t_r \), prior to \( t_d \), the user wants to know the estimated time he/she should leave the campus, if he/she prefers to take a particular bus line \( b \). In other words, before leaving the campus at \( t_r \), the user asks the service: “at what time should I go to the bus stop to get the next line \( b \) bus leaving the campus, so that I wait as little as possible at the bus stop?” We arbitrate the time \( t_r \) when the request is issued to the service as a time that is draw from a time interval that starts at most one hour before the actual time of the departure \( t_d \), following a uniform distribution. This interval can be smaller than one hour if the last arrival event of the same user, say \( t_a \), happened less than one hour before the departure time:

\[
t_r = U(\max(t_a, t_d - 1h), t_d).
\]

A request that is made at time \( t_r \) asking the estimated time \( t_e \) to go to the bus stop in the vicinity of the campus in order to get the next bus of line \( b \) is denoted by \( R_{t_r}^b \).

\(^2\) There are more attributes in the original data, but we just cite the data we use in our model.
Since $t_d$ is not known to the service, given a request $R_{b, t_r}$, we use a prediction algorithm that is fed with the past data available to estimate the most appropriate time for the user to go to the bus stop to take a bus from line $b$. The objective of the prediction algorithm is to minimize the wait time at the bus stop. Since the goal of this work is not to provide the best solution for this problem, but rather to understand how the amount of available information impact the performance of a particular solution, we have chosen a quite simple algorithm, which is in line with the small footprint required for the service, as discussed in Sect. 2.3. We consider an algorithm that simply recommends the smallest departure time contained in the past data available (for any previous day) between time $t_r$, when the request was issued, and the next hour. So, if a user makes a request at 8:05 a.m, the algorithm will get from available historical data all trips between 8:05 a.m and 9:05 a.m, and recommend the earliest departure time as the most appropriate one. (To simplify, we do not take into account the time that the user needs to walk to the bus stop.)

The amount of historical data available to the prediction algorithm depends on how users are assumed to behave. We consider different configurations for that. In particular, we consider cases where users do not share data, neither among themselves, nor with centralized servers, cases where data is made available to centralized servers at different points in time, and cases where data is exchanged among users that are in the campus at the same time. As discussed before, data stored in the user device, directly sensed by the user him/herself or received from other users, is kept for some time. Our trace has a three-month duration, but the individual data units are quite small, thus, in our simulations we considered that all data gathered in the devices were kept until the end of the simulation. We present below the data sharing configurations evaluated in this proof of concept study:

- **Baseline.** The baseline configuration is as naive as possible. It does not use any historical data to estimate the time to go to the bus stop; it simply suggests the request time ($t_r$) as such time.
- **Offline.** In this configuration, the model is built using only trips collected by the user making the request; it represents the situation in which users never share their data with other members of the community.
- **Cloud.** Here all the collected data is made available at the very time the data is collected, since in this case the data is sent to a central cloud; all data available in the server can be considered by the prediction algorithm used to answer users’ requests.
- **Cloudlet.** In this configuration we consider the existence of a local server in the university campus that is accessible only when the user is at the campus; data is made available to this server whenever a user arrives at the campus (and not at the time the data was collected, as in the previous configuration).
Community. In this case, a user \( u \) that is at the campus at time \( t \) will share its data with a user \( u' \), provided that \( u' \) is also at the campus at time \( t \), \( u \) is willing to share data with \( u' \), and \( u' \) trusts \( u \) as a data provider; in this case, all data that \( u \) has collected until that point in time (directly or indirectly) will be made available to \( u' \); all data available at a user’s device can be considered by the prediction algorithm used to answer queries.

Clearly, the amount of data used when processing a particular request in the offline configuration, except from unusual corner cases, is less than that used in the community configuration, which is, in turn, less than the amount used in the cloudlet configuration, which is less than what is used in the cloud configuration. The focus is to evaluate how feasible is to use just the data available for the community configuration, and also to compare the accuracy of the estimations done using these different levels of information available. Moreover, we consider different trust settings for the community configuration, which leads to different amounts of data available for performing predictions, as we describe in the following.

4.3 Defining Trust Relationship

We first consider that all users trust each other, so if two members of the community are at the campus at the same time, then they can exchange their data. This allows us to evaluate the best performance that the community approach can deliver. Then, we consider the case that all users in the trace have the same number \( m \) of other community members willing to share data with them, and evaluate the performance of the algorithm as the value of \( m \) varies from 10 to 80 (we refer to these configurations as community-\( m \), \( m \in \{10, 20, 30, 40, 50, 60, 70, 80\} \)). The \( m \) data providers of a user \( u \) are randomly chosen from all members of the community. Clearly, this assumption is not realistic, but it helps to understand how the performance of the algorithm degrades, as the amount of information available diminishes. Finally, we consider the trust relationship in real social networks to arbitrate which members of the community trust each other, and evaluate the performance of the community-governed service in a more realistic setting.

For the last case described above, we took the following approach. We considered 100 social networks extracted from Facebook that connect students enrolled in universities in the United States of America [24]. We grouped these social networks considering several features, and selected one representative of each group. Then, for each social network graph considered, we randomly chose one member of the graph and started a Breadth First Search (BFS) from that node, until the number of nodes traversed in the graph was equal to the number of users in our trace. At this point, we built a new social network graph that contained only the nodes that have been traversed, and the connections that these nodes had with other nodes in the new graph. Finally, we randomly mapped each user in the trace to a different node in the graph. Users that were connected in the graph, trusted each other.
We used k-means [9] to group the 100 social networks in \( k \) groups, and both the silhouette [20] and the Within-cluster Sum of Squares (WSS) [8] methods to define an appropriate value for \( k \) (see Fig. 2 and Fig. 3). Based on the results achieved, we have chosen \( k = 7 \).

After grouping the 100 social networks in 7 groups, we chose the network which was closest to the centroid of each group as the social network that represented the group (we refer to this configurations as community-real-\( i \), \( i \in \{1, 2, 3, 4, 5, 6, 7\} \)).

### 4.4 Experimental Design

The different users in our trace have quite different profiles. In particular, most users have just one departure event, and no arrival events, i.e. they never come back to share with others their unique trip collected. Thus, they can only share this information when the cloud configuration is used. Also, the distribution of “return-trips” performed by different users is quite skewed, as can be seen in Fig. 4. A lot of users have only one or two of such trips, but there are users with as many as 30.

Because of the skewness in the data, in addition to the whole trace, we considered two subsets of the trace, based on the users associated with the trips present in the trace: i) one that filtered out users that had no “return-trips”; and, ii) one that considered only the most active users of the community. For the latter, we computed the 90-percentile of the total trips that each user collected over time. This led us to consider only users who made available at least 9 trips over time. The summary of traces considered is presented in Table 1.

| Name        | Description                          | Number of users |
|-------------|--------------------------------------|-----------------|
| All         | Considers all users of the original trace | 18,662          |
| Contributors| Considers only users who made at least one “return-trip” | 8,918 |
| Most actives| Considers only users who made at least 9 “return-trips” | 845 |

We have executed three simulation experiments, which differ on the values that we used for two factors: the trace, and the data sharing configuration. Only the first experiment considers the three traces defined, since the results of this experiment show that this factor has minimal impact on the results. In this experiment, it is assumed that all users trust each other (i.e. there are no restrictions on data sharing among them). This represents the best possible case of the community configuration in terms of the amount of information that can be shared and aggregated. This was the only experiment reported in our previous work [14].
Fig. 2. Silhouette.

Fig. 3. WSS.

Fig. 4. Distribution of number of “return-trips”.
In the other two experiments we have only used the trace with the most active users, since these are the users that would benefit most from the service. They consider the existence of social networks (synthetic or real), which limit the data sharing among users, as discussed in Sect. 4.3. The values considered for the factors in each experiment are summarized in Table 2.

| Experiment | Trace          | Data sharing configuration                      |
|------------|----------------|-----------------------------------------------|
| I          | All            | Baseline                                      |
|            | Contributors   | Offline                                       |
|            | Most actives   | Cloudlet                                      |
|            |                | Cloud                                         |
|            |                | Community (all users trust each other)         |
| II         | Most actives   | Community-\(m\) (every user trusts \(m\) other users) |
| III        | Most actives   | Community-real-\(i\) (trust based on friendship relations of 7 real social networks) |

The community configurations in both Experiment II and Experiment III, have a stochastic behavior. In Experiment II, the \(m\) users that provide data to a particular user \(u\) are randomly drawn from the trace, while in Experiment III, the starting node for the BFS algorithm is randomly chosen from the social network, and after the new graph is generated, there is a random mapping of users in the trace to nodes in the new graph (see Sect. 4.3). Thus, for these settings, we replicated the execution of the experiments for a number of times that were enough to produce results with the required accuracy—50 for Experiment II, and 500 for Experiment III.

### 4.5 Evaluation Metrics

The focus of the evaluation of the proposed service model is on the quantity of data that users can aggregate and that will be used to make the predictions in each configuration. To do this, we have computed the amount of historical data used to answer each request, in addition to the proportion of requests that can be answered using some data from the past.

Despite the focus on data aggregation, we have also illustrated an example of a simple prediction algorithm (described in Sect. 4.2), in which greater data aggregation leads to the delivery of a better Quality of Service (QoS). To evaluate the QoS of the service, we have collected the wait time at the bus stop and the percentage of requests for which users could not catch a bus.

The relationship between the amount of data obtained and the quality of the predictions can vary depending on the niche of the application, the characteristics of the collected data and the prediction algorithm used. As already stated, our
goal is not to find the best solution to the problem of estimating bus departure times, but to show that the proposed model is feasible, and to assess the trade-off between the amount of data that users share, and the amount of data that they can aggregate. The metrics introduced are further detailed below:

- **Proportion of Requests $R^b_{t_e}$ that Can Be Predicted using Past Data (PP).** As described earlier, the prediction algorithm uses data from past trips to infer when the next bus of line $b$ will leave the bus stop. However, there are cases in which no data is available for the time interval associated with the request $([t_r - 1h, t_r])$—in this case, the baseline strategy is used, instead. This metric aims at measuring the proportion of requests whose predictions are done based on past data, and not on the baseline strategy.

- **Data Amount used to Perform Prediction (DA).** This metric indicates how many past trips were used to answer a given request. It is measured in number of trips.

- **Wait Time at the Bus Stop (WT).** This metric measures the amount of time that the user waits in the bus stop, until the next bus arrives. We note that this bus does not need to be the same whose departure (at $t_a$), registered in the trace, triggered the request in the first place. This is because the time the user gets to the bus stop ($t_e$, estimated by the prediction algorithm) can be both earlier than $t_a$—in which case the user might get another bus from the same line $b$ that left the bus stop after $t_e$ and before $t_a$—, or later than $t_a$—in which case it is not even guaranteed that there will be a bus from line $b$ departing at a time later than $t_a$. To avoid having the user waiting indefinitely, we assume that if the bus does not arrive in as much as one hour after $t_e$, then the user gives up waiting, and we register the wait time as 1 h.

- **Missing Rate (MR).** This metric indicates the percentage of requests for which users could not catch a bus. In other words, the percentage of requests $R^b_{t_e}$, such that there is no bus of line $b$ departing at a time $t_d, t_e \leq t_d \leq t_e + 1h$.

The $PP$ and the $MR$ metrics are computed only for Experiment I, and reported as the computed value for the single simulation run executed for each scenario. The other two metrics are computed in all three experiments. The $DA$ metric is reported as the mean value for all the requests processed in the single run simulations. For the replicated experiments, we compute the mean value of each execution, and report the mean of these means, together with its associated 95% confidence interval. The $WT$ metric is reported similarly, but using the median, instead of the mean. We use the median as the statistic to assess $WT$ because its distribution is not symmetric, and the mean may be affected by extreme values.

5 Results and Discussion

5.1 Best Case Analysis: No Restrictions on Data Sharing

Table 3 shows results for DA, PP, MR and WT for each configuration. In general, the community configuration is able to aggregate as much data as in the clouddlet
Table 3. Summary of all metrics [14].

| Trace    | Configuration | DA  | PP      | MR      | WT     |
|----------|---------------|-----|---------|---------|--------|
| All      | Baseline      | 0.0 | 0.0%    | 0.008%  | 5.48 min |
|          | Offline       | 1.3 | 39.8%   | 0.366%  | 5.13 min |
|          | Community     | 51.9| 92.6%   | 0.602%  | 3.65 min |
|          | Cloudlet      | 53.7| 96.0%   | 0.965%  | 3.35 min |
|          | Cloud         | 57.1| 97.7%   | 0.882%  | 3.30 min |
| Contributors | Baseline | 0.0 | 0.0%    | 0.005%  | 5.40 min |
|          | Offline       | 1.6 | 50.7%   | 0.460%  | 4.81 min |
|          | Community     | 54.0| 94.5%   | 0.547%  | 3.33 min |
|          | Cloudlet      | 55.6| 96.7%   | 0.744%  | 3.21 min |
|          | Cloud         | 57.1| 97.7%   | 0.747%  | 3.33 min |
| Most actives | Baseline | 0.0 | 0.0%    | 0.008%  | 5.36 min |
|          | Offline       | 3.6 | 75.8%   | 0.239%  | 4.46 min |
|          | Community     | 53.4| 96.0%   | 0.168%  | 3.16 min |
|          | Cloudlet      | 53.9| 96.5%   | 0.186%  | 3.16 min |
|          | Cloud         | 54.4| 97.0%   | 0.195%  | 3.13 min |

Fig. 5. 95% confidence intervals for mean of amount of historical data used in each scenario.
and cloud configurations, in addition to being able to aggregate much more data than a situation in which there is no data exchange between community members (offline configuration). Moreover, in general, the more data is available, the better the prediction algorithm performs.

**Data Aggregation.** From the amount of information used in each of the predictions (accumulated locally on the user’s device or aggregated in a centralized server, depending on the simulated configuration), we have calculated the 95% confidence interval for mean. As we can see in Fig. 5, the community configuration showed results very close to the cloudlet and cloud configurations, in which there is a central server that aggregates the data. On average, these configurations use between 52 and 57 historical data to generate the prediction model that serves the requests, considering all traces. Considering the “Most actives” trace, the difference from the community configuration to the other two is even smaller, and the confidence intervals of the three configurations overlap. In addition, in the community configuration it is possible to aggregate much more data than in the offline configuration, where there is no data sharing among users. In the “Most actives” trace, for example, on average, only 3 data items from the past are used to generate the prediction model.

The only configuration that is substantially affected by the different traces used is the offline one, specifically in the PP metric. This is expected, since the average amount of information that each user has is the main difference between the three traces. When the offline configuration and the “All” trace are used, only 40% of requests are answered based on the past data collected by the user.
The other 60% of requests resort to the baseline strategy due to lack of data. This is because in this trace, half of users make only one request, and there is no past data collected by them. The value of \( PP \) increases to 75.8% when the offline configuration is used with the “Most actives” trace. Since in this trace all users traveled and collected data at least 9 times, as previously mentioned, when users make a service request, there is a high probability that they have already collected some information in the past that is stored on their smartphone locally and that can be used by the prediction model.

Still considering the PP metric, we also note that the difference between the community configuration and the cloud configuration, which is the best possible configuration regarding the amount of data available, is no more than 5% on the “All” trace. In the “Most actives” trace, this difference is even lower (1%).

Quality of Predictions. As discussed before, each request has a wait time \( WT \) associated and we have calculated the median for each scenario. Figure 6 shows the confidence intervals of the median in each scenario, with a confidence level of 95%. As we can see, in scenarios where more data is available, the wait time is shorter overall. The biggest evolution occurs when we move from the offline configuration to the community configuration. In cloudlet and cloud configurations there is a decrease in wait times, but it is not substantial. The Wilcoxon-rank-sum test confirms that, on average, there is only significant statistical differences between the configurations whose confidence intervals do not overlap. Considering the “All” trace, the wait times measured for the community configuration is less than the offline configuration and greater than the wait times of the cloudlet and cloud configurations. For the other two traces, there is no statistical difference between the wait times of the community configuration and the cloudlet and cloud ones. These results indicate that the community

| Trace         | Configuration | Better | Worse | Equals |
|---------------|---------------|--------|-------|--------|
| All           | Offline       | 23.4%  | 16.4% | 60.2%  |
|               | Community     | 77.7%  | 13.7% | 8.6%   |
|               | Cloudlet      | 81.0%  | 13.8% | 5.2%   |
|               | Cloud         | 83.8%  | 12.6% | 3.6%   |
| Contributors  | Offline       | 29.9%  | 20.8% | 49.3%  |
|               | Community     | 79.8%  | 13.6% | 6.6%   |
|               | Cloudlet      | 82.1%  | 13.3% | 4.6%   |
|               | Cloud         | 83.2%  | 13.2% | 3.6%   |
| Most actives  | Offline       | 46.0%  | 29.8% | 24.2%  |
|               | Community     | 82.0%  | 12.8% | 5.2%   |
|               | Cloudlet      | 82.5%  | 12.8% | 4.7%   |
|               | Cloud         | 82.9%  | 12.9% | 4.2%   |

Table 4. Comparison of wait times between baseline and the other configurations [14].
network that is formed by the online users is, in general, as good as the case in which there is a central server to aggregate all the collected data.

Also, the missing rate for all scenarios simulated is very small, peaking at 0.96%, and smaller than 0.24% for all scenarios that considered the “Most actives” trace (see Table 3).

We also measured the difference between the baseline and all the other configurations. To measure this difference we pair the same requests in each configuration. Table 4 shows the proportion of requests in which the wait time was better (i.e. shorter), worse (i.e. greater) and equal to the baseline configuration.

In configurations with more data available for prediction, the proportion of requests in which the result was better than the baseline increases. Again, the main difference between traces is in the offline configuration. As said before, the baseline strategy is used in 60% of the requests of the offline configuration for the “All” trace. This means that for these cases, the offline configuration results are the same as the baseline. In only 23% of cases we see better results for the offline configuration. In the “Most actives” trace, this proportion increases to 46%. The community configuration had shorter wait times for more than 77% of the requests. Cloudlet and cloud configurations are better than the baseline in more than 80% of the cases.

In Experiment I, we did not take privacy issues into account, and considered that all users trust each other. In the next two experiments, we assess how the performance of the community-governed service is impacted by trust relationships among users.
5.2 Analysis of Data Sharing Using Synthetic Social Network Restrictions

In this experiment we consider the scenario where each user in the trace can receive information from exactly $m$ other users. Figure 7 and Fig. 8 show, respectively, the 95% confidence interval for the mean of $DA$ and the median for $WT$, for all configurations simulated in Experiment I (red), plus a number of configurations for community-$m$, simulated in Experiment II (blue).

As expected, as $m$ increases, so does the mean amount of information used per prediction. As a result, the median of $WT$ decreases as $m$ increases. Moreover, for $m$ as low as 20, the data amount used to make the predictions is already, on average, much better than that obtained in the offline configuration ($26.2 \times 3.6$).

The analysis applies to the $WT$ metric. The $WT$ achieved is much better than that attained when the offline configuration is used—while the community-20 configuration increases $WT$ by 24.3% when compared to the cloud configuration, for the offline configuration the increase on $WT$ is of 42.5% (see Table 5). Allowing the user to receive information from other 80 users (community-80), the result is already very close to the community configuration, where all users trust each other, both for the data amount and the wait time at the bus stop.

![Graph showing confidence intervals for wait times](Color figure online)
5.3 Analysis of Data Sharing Using Real Social Network Restrictions

We now consider the more realistic scenario where real trust relationships are used to inform the simulation model about which data can be shared (see Sect. 4.4). For comparison reasons, we use some results from Experiment I and Experiment II. Table 5 shows results for a single experiment execution (Baseline, Offline, Community, Cloudlet and Cloud), results for replicated experiment executions (community-m and community-real-i), and a result that aggregates all the replicated simulations involving real networks (community-real-all). The columns DA and WT represent, respectively, the mean of the amount of data available and the median of the wait time for the cases where a single experiment is executed, and the mean of the means, and mean of the medians for the cases where replicated experiments are executed. For the latter the table also shows the 95% confidence intervals (columns DA(C.I.) and WT(C.I.)).

| Configuration       | DA    | DA (C.I.) | WT    | WT (C.I.) |
|---------------------|-------|-----------|-------|-----------|
| Baseline            | 0.0   | –         | 5.36  | –         |
| Offline             | 3.6   | –         | 4.46  | –         |
| Community-20        | 26.2  | [26.0, 26.3] | 3.89  | [3.83, 3.95] |
| Community-real-1    | 27.9  | [27.4, 28.4] | 3.91  | [3.88, 3.93] |
| Community-real-2    | 30.4  | [30.2, 30.6] | 3.80  | [3.78, 3.81] |
| Community-real-3    | 33.6  | [33.1, 34.0] | 3.71  | [3.69, 3.73] |
| Community-real-4    | 34.3  | [34.1, 34.5] | 3.67  | [3.66, 3.69] |
| Community-real-5    | 35.3  | [35.2, 35.3] | 3.63  | [3.62, 3.65] |
| Community-real-6    | 38.0  | [37.7, 38.3] | 3.54  | [3.52, 3.55] |
| Community-real-7    | 38.4  | [38.3, 38.6] | 3.53  | [3.51, 3.54] |
| Community-real-all  | 34.0  | [33.8, 34.1] | 3.68  | [3.67, 3.69] |
| Community-40        | 38.1  | [38.0, 38.2] | 3.50  | [3.46, 3.53] |
| Community           | 53.4  | –         | 3.16  | –         |
| Cloudlet            | 53.9  | –         | 3.16  | –         |
| Cloud               | 54.4  | –         | 3.13  | –         |

From the simulation results we can conclude that the trust relationship existent in the real social networks makes the system perform at a level that is between the configurations community-20 and community-40. As mentioned before, results for the configuration community-20 are already much better than the offline configuration, both for the aggregated amount of data and for the quality of predictions made. On the other hand, when compared to the performance of the cloudlet and cloud configurations, the average WT of the community-governed service (community-real-all) is, respectively, 16.5% and 17.6% larger.
In our study, this is the average price that needs to be paid to limit data exposure. We note that if privacy is not a concern, then the community-governed service is able to aggregate more data and provide a service that performs as well as the cloudlet and cloud configurations, but without the external governance limitations that come with the need for a centralized provider.

6 Related Work

Community-governed services take advantage of mobile phone sensor capabilities to collect data from the environment and use it for some purpose in the future. This feature is known as participatory sensing, and its seminal idea is well described by Burke et al. [5]. Mobile crowd sensing is an extension of participatory sensing, which in addition to using data collected from users’ devices, also uses data made available by other users from MSN services [7].

A wide variety of applications can be built by taking advantage of the features described above [6,13,18,19,26]. All of these applications provide a service for the good of a community of users who share a common goal, which is another important pillar of our idea. However, in currently available applications, all data gathered by users and used to provide the service for the community of users is sent to a remote cloud infrastructure to be processed. Our solution aims at giving the complete power to the users who collect, process and, most importantly, govern their data and applications without the need for relying on a single service provider entity, typically hosting the service in a cloud provider.

The works by Bonomi et al. [4] and Shi et al. [23] address a new paradigm named Edge Computing. It extends the cloud paradigm by considering resources that reside between end users and the central cloud, and that provide compute and network services close to users. VM-based cloudlets [22], smart gateways [1] and servers installed in shopping centers and bus stations [12] are some examples of technologies, cited in the literature, deployed closer to data sources to perform computational tasks. Our proposed service uses a completely distributed version of the edge computing paradigm, in which processing is done on the devices themselves, which also act as data sources, with no centralized component.

Community-governed services assume users can meet and share data with whom they trust, thus forming an MSN [16], coupled with a P2P architecture that allows users’ devices to be both data consumers (clients) and data providers (servers) [25].

The idea proposed by Bellavista et al. [2] combines some of the characteristics mentioned above, like crowd sensing and edge computing paradigm, but it focuses on forming an ad hoc network with the devices of users of a community in an opportunistic way. The application monitors regions and detects points whose concentration of people is sufficient to form a network. However, it does not investigate the feasibility of building the type of services that we propose in this work.

Kuendig et al. [10] suggest a community-driven architecture that gets together devices within a zone of local proximity to form a collaborative edge
computing environment in a dynamic mesh topology. Our proposal, in addition to using community users’ devices to process tasks, also addresses the collection and sharing of data among these users, who have some common goal to be achieved.

In order to check the feasibility and efficacy of the community-governed services on the edge, we carried out a simulation-based case study fed with real data. This application aims at estimating the actual departure times of urban buses using past data collected by users. A similar application can be seen in the work by Zhou et al. [26], where users of a community have a common goal of anticipating the bus arrival time. For this purpose they use their mobile phones to collect information while on the move, and thus help in performing predictions. In addition to past information, they also use real-time information. However, the application defined in that work uses a remote cloud as the back-end, while our proposal is based purely upon edge computing principles.

7 Conclusions

In this work, we have proposed an architecture in which individuals can define the governance of a service they are interested in by using principles of Participatory sensing, Mobile Social Networks and Edge Computing. The idea is that members of the community will use resources at the edge of the network, i.e. the sensing and processing capabilities of their mobile devices, to gather data, share it with other members and then process it without having to send it to a remote cloud. This obviates the need for external governance, i.e. a cloud application provider that manages the life of the application and the data used. In this way, the proposed architecture provides more control to users over who has direct access to the data collected.

To evaluate the feasibility of the proposed model, we elaborated a case study in which university students want to know the departure time of the first bus of a particular bus line in the vicinity of the campus. We performed simulations, fed with real data from the Curitiba city public transportation system, to compare the community-governed service approach to other data sharing approaches, such as the state-of-the-practice approach where a server hosted on a cloud provider aggregates all data. The results show that it is possible to aggregate enough data from the community members to make good predictions. Moreover, the amount of data aggregated is far more than what a single user could collect. When privacy is not a concern, the aggregated amount of data is close to the approaches where a central server is needed, without facing the risks associated with the need for external governance. When users limit the exposure of their data, sharing only with whom they trust, the aggregated amount of data and the quality of predictions are impacted, but yet proving reasonable results. Thus, the model allows for a more flexible way to establish a trade-off between increased performance, and reduced data exposure.
References

1. Aazam, M., Huh, E.: Fog computing and smart gateway based communication for cloud of things. In: 2014 International Conference on Future Internet of Things and Cloud, pp. 464–470, August 2014. https://doi.org/10.1109/FiCloud.2014.83

2. Bellavista, P., Chessa, S., Foschini, L., Gioia, L., Girolami, M.: Human-enabled edge computing: exploiting the crowd as a dynamic extension of mobile edge computing. IEEE Commun. Mag. 56(1), 145–155 (2018). https://doi.org/10.1109/MCOM.2017.1700385

3. Bonawitz, K., et al.: Towards federated learning at scale: system design. CoRR abs/1902.01046 (2019). http://arxiv.org/abs/1902.01046

4. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16. ACM, New York (2012). https://doi.org/10.1145/2342509.2342513, https://doi.org/10.1145/2342509.2342513

5. Burke, J., et al.: Participatory sensing. In: Workshop on World-Sensor-Web (WSW 2006): Mobile Device Centric Sensor Networks and Applications, pp. 117–134 (2006)

6. Ganti, R.K., Pham, N., Ahmadi, H., Nangia, S., Abdelzaher, T.F.: GreenGPS: a participatory sensing fuel-efficient maps application. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys 2010, pp. 151–164. ACM, New York (2010). https://doi.org/10.1145/1814433.1814450, https://doi.org/10.1145/1814433.1814450

7. Guo, B., Yu, Z., Zhou, X., Zhang, D.: From participatory sensing to mobile crowd sensing. In: 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), pp. 593–598, March 2014. https://doi.org/10.1109/PerComW.2014.6815273

8. Hartigan, J.A., Wong, M.A.: Algorithm as 136: a k-means clustering algorithm. J. Roy. Stat. Soc. Ser. C (Appl. Stat.) 28(1), 100–108 (1979). http://www.jstor.org/stable/2346830

9. Hartigan, J.A.: Clustering Algorithms, 99th edn. Wiley, New York (1975)

10. Kuendig, S.J., Rolim, J., Angelopoulos, K.M., Hosseini, M.: Crowdsourced edge: a novel networking paradigm for the collaborative community. Technical report (2019). https://archive-ouverte.unige.ch/unige:114607. ID: unige:114607; Paper submitted for publication at the Global IoT Summit 2019

11. Lohmar, T., Zaidi, A., Olofsson, H., Boberg, C.: Driving transformation in the automotive and road transport ecosystem with 5G. Ericsson Technology Review (2019)

12. Luan, T.H., Gao, L., Li, Z., Xiang, Y., Sun, L.: Fog computing: focusing on mobile users at the edge. CoRR abs/1502.01815 (2015), http://arxiv.org/abs/1502.01815

13. Ludwig, T., Reuter, C., Siebigerothero, T., Pipek, V.: CrowdMonitor: mobile crowd sensing for assessing physical and digital activities of citizens during emergencies. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, pp. 4083–4092. ACM, New York (2015). https://doi.org/10.1145/2702123.2702265, https://doi.org/10.1145/2702123.2702265

14. Mafra, J., Brasilheiro, F.V., Lopes, R.V.: Community-governed services on the edge. In: Ferguson, D., Helfert, M., Pahl, C. (eds.) Proceedings of the 10th International Conference on Cloud Computing and Services Science, CLOSER 2020, Prague, Czech Republic, 7–9 May 2020, pp. 498–505. SCITEPRESS (2020). https://doi.org/10.5220/0009765804980505
15. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutorials 19(4), 2322–2358 (2017). https://doi.org/10.1109/COMST.2017.2745201. Fourthquarter 
16. Miluzzo, E., et al.: Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys 2008, pp. 337–350. ACM, New York (2008). https://doi.org/10.1145/1460412.1460445, https://doi.org/10.1145/1460412.1460445 
17. de Oliveira Filho, T.B.: Inferring passenger-level bus trip traces from schedule, positioning and ticketing data: methods and applications. Master dissertation, Universidade Federal de Campina Grande, Paraíba, Brasil (2019) 
18. Predić, B., Yan, Z., Eberle, J., Stojanovic, D., Aberer, K.: ExposureSense: integrating daily activities with air quality using mobile participatory sensing. In: 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 303–305, March 2013. https://doi.org/10.1109/PerComW.2013.6529500 
19. Reddy, S., Shilton, K., Denisov, G., Cenizal, C., Estrin, D., Srivastava, M.: Biketastic: sensing and mapping for better biking. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2010, pp. 1817–1820. ACM, New York (2010). https://doi.org/10.1145/1753326.1753598, https://doi.org/10.1145/1753326.1753598 
20. Rousseuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20, 53–65 (1987). https://doi.org/10.1016/0377-0427(87)90125-7, http://www.sciencedirect.com/science/article/pii/0377042787901257 
21. Ruge, L., Altakrouri, B., Schrader, A.: SoundOfTheCity - continuous noise monitoring for a healthy city. In: 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 670–675, March 2013. https://doi.org/10.1109/PerComW.2013.6529577 
22. Satyanarayanan, M., Bahl, V., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. 8, 14–23 (2009) 
23. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. 3(5), 637–646 (2016). https://doi.org/10.1109/JIOT.2016.2579198 
24. Traud, A.L., Mucha, P.J., Porter, M.A.: Social structure of Facebook networks. Phys. A 391(16), 4165–4180 (2012) 
25. Tsai, F.S., Han, W., Xu, J., Chua, H.C.: Design and development of a mobile peer-to-peer social networking application. Expert Syst. Appl. 36(8), 11077–11087 (2009). https://doi.org/10.1016/j.eswa.2009.02.093, http://www.sciencedirect.com/science/article/pii/S0957417409002498 
26. Zhou, P., Zheng, Y., Li, M.: How long to wait? Predicting bus arrival time with mobile phone based participatory sensing. IEEE Trans. Mob. Comput. 13(6), 1228–1241 (2014). https://doi.org/10.1109/TMC.2013.136