EnviroMeter: A Platform for Querying Community-Sensed Data

Saket Sathe
EPFL, Switzerland.
saket.sathe@epfl.ch

Arthur Oviedo
EPFL, Switzerland.
arthur.oviedo@epfl.ch

Dipanjan Chakraborty
IBM Research India.
cdipanjan@in.ibm.com

Karl Aberer
EPFL, Switzerland.
karl.aberer@epfl.ch

ABSTRACT

Efficiently querying data collected from Large-area Community driven Sensor Networks (LCSNs) is a new and challenging problem. In our previous works, we proposed adaptive techniques for learning models (e.g., statistical, non-parametric, etc.) from such data, considering the fact that LCSN data is typically geo-temporally skewed. In this paper, we present a demonstration of EnviroMeter. EnviroMeter uses our adaptive model creation techniques for processing continuous queries on community-sensed environmental pollution data. Subsequently, it efficiently pushes current pollution updates to GPS-enabled smartphones (through its Android application) or displays it via a web-interface. We experimentally demonstrate that our model-based query processing approach is orders of magnitude efficient than processing the queries over indexed raw data.

1. INTRODUCTION

Community-driven sensing relies on on-board or smartphone-embedded sensors carried by the community (buses, cars, people) to sense an environmental phenomenon of interest (e.g., pollution). The main focus of research until now has been on design and implementation of novel deployments to collect and process community-sensed data. Large-scale community-driven sensor networks are fundamentally different from traditional sensor networks, due to their autonomous and unstructured sensing behavior [5].

An example of such a community-sensed deployment is the OpenSense project [5]. The primary objective of the OpenSense project is to efficiently and effectively monitor environmental pollution using wireless and mobile sensors. The project adopts complex utility driven approaches towards sensing and data management. The geographical granularity for monitoring environmental pollution is on the level of a city or state. Pollution data is collected using sensors installed on public transport buses.

Unfortunately, LCSNs cannot be tightly controlled especially when deployments cover large areas, which makes it difficult to produce a homogeneous view of the phenomenon. Thus, the data collected by sensors is geo-temporally skewed. Data skewness drastically affects the efficiency and accuracy of query processing in the following ways: (a) due to semi-controlled or uncontrolled mobility of sensors, sensor values may not be always available at a particular position and time, (b) the accuracy of an estimated pollution value is not high if a distant sensor value is used to approximate the pollution value at the current position. Queries that are typically processed on such data are of two types: point queries and continuous queries. Point queries return the pollution value at a given position, while continuous queries are registered by a mobile object, which is interested in continuously knowing the pollution around it.

In our prior research [6], we demonstrated that multi-model approaches are suitable for modeling geo-temporally skewed community-sensed data. Previous literature exists on scalable high-speed data processing on the server that addresses efficient storing and querying of models from raw data [9, 10]. In the past, research has focused on the problem of inherent unreliability of the sensors due to their autonomous human-influenced nature or surrounding weather conditions, due to which sensors become error-prone or run out of battery [7, 8].

Although the previous works investigate various ways for the community to perform sensing, they are not concerned with using the collected data and knowledge in providing feedback and information about the phenomenon, back to the community. This work, is focused in closing the gap in the loop. In this demonstration of EnviroMeter, we use our proposed techniques from [6] for succinctly representing community-sensed data in the form of models.

We demonstrate how to use the learned models for efficiently processing user queries. Our techniques adapt to the changing nature of the sensed phenomenon by adjusting the geographical granularity of the models, to capture the phenomena with high fidelity. EnviroMeter is a complete framework that senses data in a community-driven sensor network. It processes the results efficiently, and uses lazy update policies to present them to users on their mobile devices, while significantly reducing network bandwidth and processing delay.

2. SYSTEM DESIGN

The system designed for supporting EnviroMeter consists of three main components. First component uses the adaptive techniques for learning single or multiple models over

Proceedings of the VLDB Endowment, Vol. 6, No. 12

the concerned geographical area. Second component consists of the different query processing methods that use the learned models for answering continuous queries. Third component optimizes the bandwidth required for communicating the requests/responses during query processing. In the following sections we discuss each of these components.

2.1 Adaptive Models

The architecture of the EnviroMeter framework is shown in Figure 1. It assumes a geographical region $\mathcal{R}$, over which environmental pollution is sensed using community-driven approaches. The sensed data is stored in a database in the form of *raw tuples*. The adaptive modeling approach that we propose creates a multi-model abstraction or a *model cover* over the raw tuples dumped in the region $\mathcal{R}$. A model cover is defined as a set of models $\mathcal{M} = \{M_1, \ldots, M_O\}$ that are respectively responsible for modeling the sub-regions $R_1, R_2, \ldots, R_O$ of $\mathcal{R}$. The sub-regions taken together cover the entire region $\mathcal{R}$.

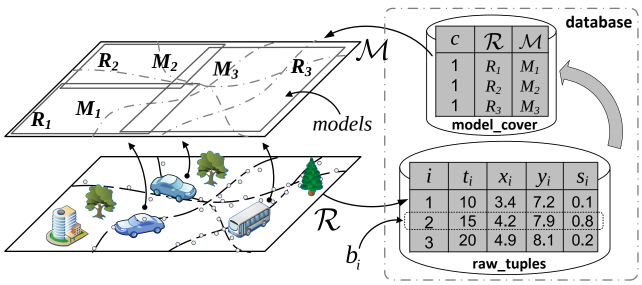


Figure 1: Architecture of the framework.

We denote the raw tuple as $b_i = (t_i, x_i, y_i, s_i)$, where s_i is the raw sensor value, and t_i and (x_i, y_i) are the time and the position corresponding to the sensor value s_i. We assume that the model cover is computed using a window of raw tuples $\mathcal{W}_c = \langle b_i | cH \leq t_i \leq (c+1)H \rangle$, where c is a positive integer and H is the window length.

We briefly present the adaptive method, called *adaptive k-means* or Ad-KMN, that gave us the best results among many candidates we designed [6]. This method partitions the region $\mathcal{R}$ adaptively (i.e., only when and where it is necessary) and estimates the models $M_1, M_2, \ldots, M_O$. The standard k-means algorithm uses the Euclidean distance for creating the clusters. Instead, in the Ad-KMN method, we use the model approximation error as an additional clustering criteria. An example of the Ad-KMN method on toy data is shown in Figure 2.

Assume that before executing the Ad-KMN method, we compute two centroids μ_1 and μ_2 by executing the standard k-means algorithm using the positions (x_i, y_i) from $\mathcal{W}_c$ (refer Figure $2(a)$). Then, (a) we partition the sensor values in $\mathcal{W}_c$, such that R_1 and R_2 contain sensor values that are nearest to μ_1 and μ_2 respectively, and (b) for the sensor values in R_1 and R_2 we estimate linear regression models M_1 and M_2 and compute the approximation error [1].

Next, we check whether the approximation error is within a user-defined threshold τ_n. In the regions (R_1 or R_2) where the approximation error is greater than τ_n, we introduce an additional cluster centroid (equivalent to splitting the region) and re-estimate all the centroids. This procedure is

[1] *approximation error* is the average percentage error compared to the normal range of s_i in the environment (pollutant specific).

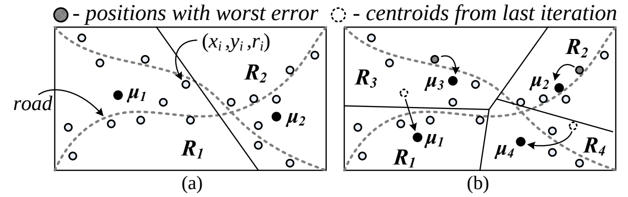


Figure 2: Example on toy data: (a) initial regions, and (b) two new regions R_3 and R_4 added after an Ad-KMN iteration.

continued until all the regions meet the approximation error threshold τ_n. We denote the cluster centroids $(\mu_1, \ldots, \mu_O)$ as $\boldsymbol{\mu}$.

2.2 Continuous Query Processing

The query processing framework is depicted in Figure 3. It consists of a mobile object v_q that transmits the query tuple $q_l = (t_l, x_l, y_l)$ at time t_l from position (x_l, y_l) to the server using mobile data services (GPRS or 3G). Here, we assume a single mobile object (individual or vehicle) continuously querying for pollution around it. The query that we consider is formally defined as follows:

Query 1. ***Continuous Value Query.*** *Given a mobile object v_q that continuously transmits the query tuple $q_l = (t_l, x_l, y_l)$ at time t_l, interpolate the sensor value $\hat{s}_l$ at position (x_l, y_l) and transmit it to v_q.*

Here, the sensor value could be any of the pollutants that are typically monitored: carbon dioxide (CO_2), carbon monoxide (CO), suspended particulate matter, etc. We assume that the mobile object transmits a query tuple with uniform interval, i.e., $|t_{l+1} - t_l|$ is always the same. We propose the following three methods for processing Query 1.

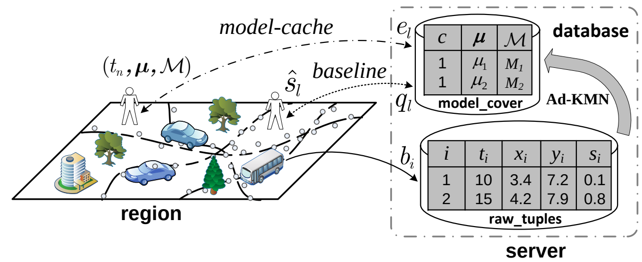


Figure 3: Continuous query processing framework.

Naïve: In this method, the server does an exhaustive search in the window $\mathcal{W}_c$ to find all the raw tuples that are in a radius r centered at (x_l, y_l). Then the interpolated value $\hat{s}_l$ is computed as the average value of the sensor values s_i found in the radius r. This interpolated value $\hat{s}_l$ is then returned to the object v_q.

Metric Space Indexing: This method is similar to the naïve method, but it uses a metric space index (e.g., R-tree or VP-tree) to enhance the performance of finding the raw tuples in window $\mathcal{W}_c$ that are within radius r of (x_i, y_i).

Model Cover: This method uses the model cover $\mathcal{M}$ and the cluster centroids $\boldsymbol{\mu}$ for query processing. In this method, we first find the cluster centroid μ_* in $\boldsymbol{\mu}$ that is nearest to (x_l, y_l). Then the model $M_* \in (M_1, \ldots, M_O)$ corresponding to μ_* is used for interpolating the sensor value $\hat{s}_l$.

2.3 Bandwidth Optimization Techniques

It is a well-known fact that smartphones spend significant amount of battery power and bandwidth in transmitting data via GPRS or 3G data services. In order to optimize the bandwidth- and power-usage, we propose a caching technique referred to as *model-cache*. Model-cache stores the model cover on the smartphone and only queries the server when the cached model cover becomes invalid.

Model-Cache: As a system initialization step v_q sends a *model request*, denoted as e_l, to the server (refer Figure 3). In response to e_l the server sends the following items: (i) the coefficients of all the models in $\mathcal{M}$, (ii) the cluster centroids $\boldsymbol{\mu}$, and (iii) the time t_n until which the current model cover is valid. v_q stores $(t_n, \boldsymbol{\mu}, \mathcal{M})$ in its local memory.

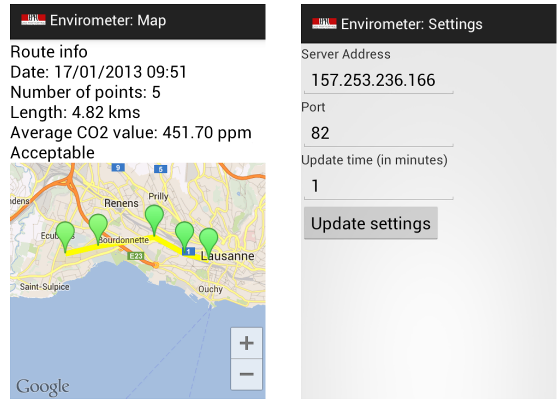
Now, when the user, who has EnviroMeter running on his/her smartphone, needs a pollution update, a query tuple q_l is generated. Then, EnviroMeter checks whether $t_l \leq t_n$. If $t_l \leq t_n$, then it finds the nearest cluster centroid μ_* to (x_l, y_l). It uses the model M_* corresponding to μ_* for computing the value $\hat{s}_l$, without contacting the server. If $t_l > t_n$, then the current model cover is invalid, and a new model request e_l is sent to the server for updating $(t_n, \boldsymbol{\mu}, \mathcal{M})$. Since in practice it often happens that mobile objects have limited mobility or predefined trajectories, we save a considerable amount of bandwidth by caching $(t_n, \boldsymbol{\mu}, \mathcal{M})$.

In Section 4, we compare the model-cache technique with a baseline technique, which simply responds to each query tuple with the interpolated sensor value $\hat{s}_l$, without caching the models. We experimentally demonstrate that model-cache is approximately *50 times* bandwidth efficient as compared to the baseline technique.

3. DEMONSTRATION

As a part of the demonstration we present the EnviroMeter Android application. The users are presented with a map of Lausanne, Switzerland. EnviroMeter users can quickly find the CO_2 concentration at their current position. The application has the ability to record routes. After a route has been recorded, the user can view it on a map. In addition, the application presents the average pollution level through the route. An informative text indicating whether this value is acceptable according to the OSHA (Occupational Safety and Health Administration) [1] guidelines is displayed. Moreover, the map shows each of the points in the route with a marker whose color varies from green (safe) to red (hazardous CO_2 levels). Finally, users can set up configuration parameters, like the server address and the interval for the position updates using the settings menu.

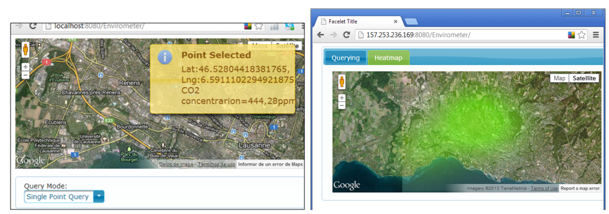
In the second part, we demonstrate a web interface of EnviroMeter. The web interface can be used in three different modes: continuous query (Query 1), point query and heatmap visualization. In the continuous query mode, users select a set of points that constitute the route, and the application computes dynamically and displays the average CO_2 level for each point on the route. Figure 5(a) shows an example of the web interface, along with the CO_2 concentration at a point clicked by the user. In single point query mode, users click on a point in the map, and the application presents the interpolated CO_2 concentration measured in parts per million (ppm) at that point. Finally, the user can visualize a heatmap of the area Figure 5(b). The emitting points are the centroids computed by the Ad-KMN algorithm with its pollution level. The points are colored in a



(a) query mode (b) settings

Figure 4: (a) Map showing the points in a continuous query, and (b) user-defined settings.

scale going from acceptable (green) to dangerous to human health (red).



(a) query mode (b) heatmap visualization

Figure 5: (a) EnviroMeter web interface for single point query, and (b) heatmap of the pollutant concentration.

4. EXPERIMENTAL EVALUATION

We perform the experiments on a real dataset collected in the city of Lausanne, Switzerland for the OpenSense project [5]. The dataset is community-sensed by two public transport buses that are equipped with various environmental pollution sensors. For our experiments, we focus only on CO_2. Our dataset was collected over a period of 1 month and has 176K raw tuples with sampling interval of 60 seconds. We refer to this dataset as *lausanne-data*. In Section 4.1, we compare the various query processing methods for processing Query 1, followed by the experiments on bandwidth optimization methods in Section 4.2.

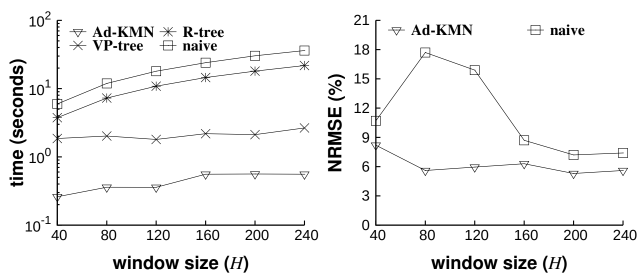
4.1 Query Processing

To evaluate the performance of our query processing methods, we use a varying window size H from 40 to 240 raw tuples (4 hour window), a radius r of 1 km, and error threshold $\tau_n = 2\%$. The naïve and the model cover methods are implemented using Python. For testing the metric space indexing methods, we use Python-based implementations of the R-tree [3] and the VP-tree [4]. We use 5000 point queries for comparing the efficiency, accuracy, and memory consumption of all the query processing methods.

Efficiency: Figure 6(a) presents elapsed time for the described scenario. We can observe that the model cover method processes the queries *7.1 times* faster as compared to the VP-tree method for $H = 40$. In addition, it is *39.4 times* faster than the R-tree method for $H = 240$.

Accuracy: For measuring the accuracy of the obtained results, we compare the naïve method and our proposed model cover method. Recall that the naïve method computes the value $\hat{s}_l$ as the average of the sensor values that lie in the radius r of the query tuple. Figure 6(b) shows that our method consistently generates a smaller NRMSE (normalized root-mean-square error) than the naïve method. The R-tree and the VP-tree methods are not considered, since they produce the same result as the naïve method.

Memory Consumption: For demonstrating the saving in memory due to the model cover method, we use a larger window size $H = 5000$. We compared the memory required to store: (a) the complete set of points for the naïve method, (b) the index information for the R-tree and VP-tree methods, and (c) the models generated by the model cover method. The memory required is accurately measured using the Pympler library [2]. Figure 7(a) presents the average memory required by all the methods averaged over 10 independent runs. Observe that the model cover method dramatically reduces memory consumption and requires approximately *7 times, 70 times and 407 times* less memory than the naïve, R-tree and VP-tree methods respectively.



(a) efficiency (b) accuracy

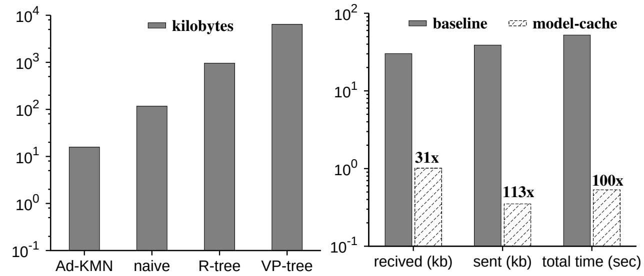
Figure 6: Comparing efficiency and accuracy of query processing. Note the logarithmic scale on the y-axis in (a).

4.2 Bandwidth Optimization

To evaluate the bandwidth savings obtained from using the model-cache technique, we use a continuous query of 100 query tuples. We measured the total number of bytes transmitted and received by the mobile device, and the total time to complete the query. From the results presented in Figure 7(b) we can see that the model-cache technique dramatically reduces the memory consumption and query processing time. Compared to the baseline technique (see Section 2.3), model-cache requires *113 times* less transmitted bytes, *30 times* less received bytes, and approximately *100 times* less time.

5. CONCLUSION

In this demonstration, we introduced EnviroMeter, an efficient and easy-to-use framework for processing queries over community-sensed data. We proposed various design strategies for query processing and optimizing network bandwidth. We presented the main functionalities of the EnviroMeter Android application for smartphones and the web interface. Finally, we evaluated our techniques on a real dataset and clearly demonstrated the orders of magnitude performance enhancements obtained using our methods.



(a) memory (b) bandwidth

Figure 7: (a) Comparing memory requirements of various query processing methods and (b) comparing the bandwidth optimization techniques. Note the logarithmic scale on the y-axis.

6. ACKNOWLEDGMENTS

The work is supported by the OpenSense project (reference number 839 401) supported by the Nano-Tera initiative (http://www.nano-tera.ch).

7. REFERENCES

[1] Occupational Safety and Health Administration:. `http://www.osha.gov/dts/chemicalsampling/data/CH_225400.html`.

[2] Pympler. `http://pythonhosted.org/Pympler`.

[3] Pyrtree. `http://code.google.com/p/pyrtree/`.

[4] Python VP-Tree. `http://www.logarithmic.net/pfh/blog/01164790008`.

[5] K. Aberer, S. Sathe, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Theile. OpenSense: Open community driven sensing of environment. In *IWGS (along with ACM GIS)*, 2010.

[6] S. Cartier, S. Sathe, D. Chakraborty, and K. Aberer. ConDense: Managing data in community-driven mobile geosensor networks. In *IEEE SECON*, 2012.

[7] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[8] A. Krause *et al.*. Towards community sensing. In *IPSN*, 2008.

[9] M. Mokbel, X. Xiong, and W. Aref. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD*, page 634, 2004.

[10] M. Mokbel, X. Xiong, S. Hambrusch, S. Prabhakar, and M. Hammad. PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams. In *VLDB*, 2004.