DrOPS: Model-Driven Optimization for Public Sensing Systems

Damian Philipp, Jarosław Stachowiak, Patrick Alt, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
Email: firstname.lastname@ipvs.uni-stuttgart.de

***Abstract*—The proliferation of modern smartphones has given rise to Public Sensing, a new paradigm for data acquisition systems utilizing smartphones of mobile participants. In this paper, we present DrOPS, a system for improving the efficiency of data acquisition in Public Sensing systems. DrOPS utilizes a model-driven approach, where the number of required readings from mobile smartphones is reduced by inferring readings from the model. Furthermore, the model can be used to infer readings for positions where no sensor is available. The model is directly constructed from the observed phenomenon in an online fashion. Using such models together with a client-specified quality bound, we can significantly reduce the effort for data acquisition while still reporting data of required quality to the client. To this effect, we develop a set of online learning and control algorithms to create and validate the model of the observed phenomenon and present a sensing task execution system utilizing our algorithms in this paper. Our evaluations show that we obtain models in a matter of just hours or even minutes. Using the model-driven approach for optimizing the data acquisition, we can save up to 80 % of energy for communication and provide inferred temperature readings for uncovered positions matching an error-bound of $1°C$ up to 100 % of the time.**

I. Introduction

The proliferation of modern smartphones has given rise to Public Sensing (PS), a new paradigm for sensor data acquisition utilizing smartphones of mobile participants [1]. These devices are equipped with various sensors such as accelerometers, cameras, light sensor, and positioning sensors like GPS. Together with their capability of processing and communicating captured sensor data, smartphones have become powerful networked sensor platforms that can be used to obtain sensor data without the cost of managing a dedicated fixed sensor network.

However, due to the mobility of participants, capturing sensor data at certain points of interests (POI) is more challenging than for fixed sensor networks where a sensor could simply be installed at each POI. In [2], we introduced the concept of *virtual sensors* (v-sensors for short) as a mobility-transparent abstraction of the PS system. Similar to the sensors of a fixed sensor network, v-sensors can be placed at POIs and report readings with a certain client-defined sampling rate. The PS system is responsible for selecting suitable mobile devices in the vicinity of the v-sensor to capture the data associated with this v-sensor.

While being a powerful abstraction, the PS system faces the problem of a v-sensor being temporarily unavailable if no mobile device is in its vicinity. Clearly, the probability for a v-sensor being unavailable increases with an increasing v-sensor sampling rate and a decreasing sensing device density. We alleviate this problem by a model-driven data acquisition approach, which uses a model to infer readings of unavailable v-sensors. Moreover, we can save energy for sensing and communicating data on mobile devices by inferring readings of v-sensors even if they are available when inferred readings are sufficiently accurate. Keeping the energy consumption for PS to a minimum is important, as otherwise participants might be unwilling to support PS.

Model-driven sensing has already proven to increase the performance of fixed sensor networks [3], [4]. These approaches target long-running queries and use either expert knowledge or a pilot deployment, typically lasting at least several days, to learn the model a priori. However, to provide significant energy savings in PS, any optimization technique may take at most a fraction of the recharge cycle duration, which is typically one or two days, to set up. Furthermore, PS systems are built for heterogeneous types of queries with a-priori unknown duration, spontaneously issued by clients. For example, people commuting to their workplace may be interested in the microclimate along potential routes to avoid bad weather conditions. Such queries, each defining an individual set of v-sensors, would be posted just before the start of a commute and canceled shortly thereafter. Clearly, maintaining models proactively for all possible types of queries is counter-productive if the query is never issued. Therefore, we propose model-*dr*iven *O*ptimizations for *PS* (*DrOPS*), a PS system that obtains a model *online* on-demand within minutes from the start of a query.

In detail, the contributions of this paper are: (1) An execution model for optimized PS systems using model-driven data acquisition. (2) An online learning algorithm (OLA) for multivariate Gaussean models of spatially distributed phenomenons that can obtain training data from running queries and creates a model suitable for model-driven sensing in a few minutes. To minimize energy as well as the time to learn a model, we re-use historic readings and only update portions of the model if possible. (3) An online model validity check algorithm (MOCHA)

Published in Proceedings of the 11th IEEE International Conference on Pervasive Computing and Communications (PerCom 2013), pages 185–192, San Diego, CA, USA, March 2013.
©IEEE 2013
http://dx.doi.org/10.1109/PerCom.2013.6526731

that verifies whether a model still accurately reflects the underlying phenomenon. To minimize the energy spent for checking the validity, MOCHA uses only a small fraction of additional control readings which are continuously compared to inferred readings. Moreover, MOCHA applies smoothing techniques to avoid abandoning models too quickly.

Our evaluations show that we obtain models in a matter of minutes on average. Using our model-driven approach, we can save up to 80 % of energy for communication and provide inferred readings for unavailable positions matching an error-bound of $1°C$ up to 100 % of the time.

The remainder of this paper is structured as follows. We present the system model and problem statement in Section II before discussing the basic operation of DrOPS in Section III. Section IV details the optimized operation and gives a brief overview of to the spatial models used in DrOPS, while Section V presents our online learning and validation algorithms. We evaluated DrOPS in a real-world testbed and a simulated environment, whose setup and results are discussed in Section VI. Related work is discussed in Section VII, before we conclude this paper with a summary and outlook onto future work in Section VIII.

II. System Model and Goals

Before introducing the system model, we first define some terminology. Moreover, we formulate the problem to be solved by our PS system.

A. Definitions

We refer to applications requesting data from the PS system as clients. Smartphones carried by participants sharing resources for the PS system are referred to as *mobile nodes*. Clients can define virtual sensors (v-sensors for short). Each v-sensor *v* is associated with a type of reading *v.type*, e.g., temperature or light intensity, a point in space *v.loc* indicating its location where data is supposed to be sensed, and a coverage area *v.area* defined relative to its location. *v.area* can be defined arbitrarily, as long as the coverage areas of all v-sensors are pairwise disjoint to ensure a unique mapping of mobile nodes to v-sensors. If there is at least one mobile node located in *v.area* to capture sensor readings for *v*, we say that *v* is *available*. Otherwise, *v* is *unavailable*.

Virtual sensors provide measurements called virtual readings, which can either be *effective readings* or *inferred readings*. An effective reading *r* is taken by a mobile node at position $pos(r) \in v.area$, whereas an inferred reading is computed using a model of the observed phenomenon without interaction with any mobile node.

B. System Model and Architecture

Next, we describe the components and interfaces of DrOPS. Note that this system model follows the general design of PS systems, e.g., [5], [6]. Therefore, other systems could be extended with the DrOPS query mechanisms.

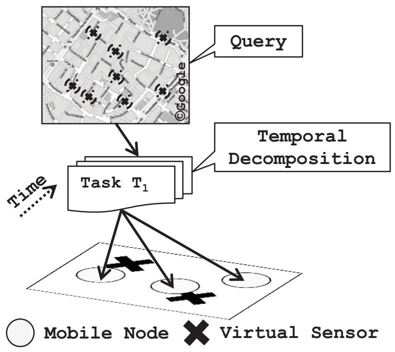

Figure 1. Overview of Sensing Task Execution

The DrOPS system consists of a gateway server and a set of mobile nodes. The gateway serves as an interface to submit queries to the system. Note that to ensure scalability of the system, the gateway server may be implemented as a distributed service using a partitioned service area. However, such an extension is beyond the scope of this work.

Mobile nodes can determine their position using a positioning system, e.g., GPS, and use a set of environmental sensors (light, sound, temperature, air pollution, etc.) to observe their environment. DrOPS does not make any assumption about the mobility of nodes. We assume that all nodes are equipped with the appropriate sensor hardware to provide readings for all data requests posted to the system. Furthermore, nodes are equipped with a mobile Internet connection allowing for communication with the gateway.

DrOPS accepts queries *Q = (V,p,QoS)* issued by clients, which are executed until canceled by the client. In a query, *V* represents the set of v-sensors. Note that the v-sensors are created by each query independently. Parameter *p* denotes the sampling period, while *QoS* is a set of quality parameters required by the client. The content of *QoS* depends on the selected algorithms and, therefore, will be explained in detail in the corresponding sections. At the end of every sampling period, a result set R_Q is sent to the client. R_Q contains all effective readings as well as inferred readings computed by the gateway for virtual sensors where no effective reading was taken.

C. Problem Statement

Our goal is to create a system for efficiently obtaining data on spatially distributed environmental phenomena according to a client-defined quality bound $Q.QoS$. We maximize the number $|V'|$ of virtual sensors for which the quality constraint is fulfilled while minimizing the number of effective readings taken.

III. Basic Sensing Algorithm

Next, we explain the basic execution of queries, before we describe the optimized model-driven operation and the process of model maintenance in subsequent sections. The basic sensing algorithm is used to illustrate the operation of DrOPS and will later serve as a baseline for comparing the performance of our optimized approaches.

To issue a query *Q = (V,p,QoS)*, the client submits it to the gateway. The gateway then performs a *temporal decomposition* of the query by generating a new task $T = (V, QoS)$ every p seconds at the beginning of each sampling period. Since the basic algorithm is not model-driven, all sensors should report effective readings. As the gateway does not track the position of mobile nodes, it does not know which nodes are close to a v-sensor. Therefore, each T is broadcast to all mobile nodes. This process is illustrated in Fig. 1.

On receiving a task, each node determines whether it is located within the coverage area of any virtual sensor $v \in V$. In this case, it takes a reading *r* and returns $(v, r, pos(r))$ to the gateway. Otherwise, it discards the task. In case there are multiple readings taken for *v*, the gateway selects the reading $\overline{r}$ with minimum Euclidean distance $\delta(pos(\overline{r}), v.loc)$. The gateway stores all selected readings together with the starting time of the corresponding sampling period. At the end of each sampling period, the collected effective readings are returned to the client as result R_Q. Note that with this basic algorithm readings are returned only for available v-sensors.

In the following sections, we extend this basic sensing algorithm with our model-driven approach to compensate for unavailable v-sensors and to minimize the number of necessary effective readings.

IV. Model-Driven Approach

DrOPS learns and maintains a model of the phenomenon observed in a query Q to optimize the data acquisition process. Initially, DrOPS uses the basic sensing algorithm to execute the query and learns the model in parallel. Once a model is created, DrOPS switches to an optimized operation phase and uses the model-driven sensing algorithm. A validity check algorithm continuously monitors model accuracy and causes an update of the model if necessary. Next, we present the model-driven optimizations before introducing the learning and validation algorithms in Section V.

Since our focus is on obtaining data for spatially distributed environmental phenomena, we use multivariate Gaussian distributions (MGD), a popular modeling technique for such phenomena (e.g., [4], [7]).

In an MGD, a set of v-sensors is modeled as a set of correlated one-dimensional Gaussean distributions, stored as a mean vector and a covariance matrix. Capturing the correlation of values between all v-sensors of Q without depending on indirect criteria such as proximity of v-sensors is the core feature of an MGD. Consider a placement of light sensors at buildings on a university campus. Sensors placed on walls facing the sun will report values similar to each other and sensors placed on walls facing away from the sun will report similar values, i.e., sensors in each group show a high correlation. Note that capturing other phenomena, e.g., discrete environmental phenomena such as individual events (e.g., lightning strikes), may require a different model.

We exploit this knowledge about the correlation of values. First, we can infer readings for unavailable v-sensors using the model. The details of this process are omitted due to space constraints but can be found in [7]. We will refer to this inference process as function INFER(MGD, *P*), where *P* denotes available effective readings. Second, we can identify strongly correlated v-sensors which can be used to infer other sensors' values. We use this to minimize the number of effective readings, thus, reducing the effort required for task execution.

We will briefly show how to apply MGDs to optimize readings in principle, before we present an extended version of DrOPS based on these principles. For more in-depth information about the use of MGDs, we refer to [4], [7].

A. Near-Optimal Sensor Selection

As stated before, we strive to minimize the number of effective readings to achieve the best optimization. The rationale behind this is that for sets of strongly correlated v-sensors, effective readings for a small subset are sufficient to yield accurate inferred readings (i.e., readings with a variance below a given threshold σ^2_{max}) for all v-sensors.

Guestrin et al. have shown that this problem is NP hard [4] and have developed a set of near-optimal heuristic algorithms to address this problem. Given a task *T = (V,QoS)* and an MDG model, their GREEDY algorithm partitions V into two subsets V_{eff} and V_{inf}, where V_{eff} contains all v-sensors for which effective readings should be taken and V_{inf} denotes all v-sensors for which readings are computed using INFER. The v-sensors in V_{eff} are selected according to a mutual information criterion which minimizes the variance of inferred readings. Note that the selection only depends on the properties of the model, not on current observations.

Whereas the original algorithm in [4] limits the set of v-sensors to a fixed size, we modify the algorithm to use a target maximum variance $QoS.\sigma^2_{max}$ provided as a quality parameter. The modified sensor selection algorithm will then add as many sensors as necessary to V_{eff} to ensure that the variance of any v-sensor is less or equal to $QoS.\sigma^2_{max}$.

As discussed previously, the gateway does not track positions of mobile nodes and thus is not aware of v-sensor availability. Therefore, selecting V_{eff} is done in an optimistic fashion, assuming that all $v \in V$ will be available. This is not a problem for a dense coverage. For instance, as our experiments show, selecting unavailable v-sensors reduces the quality of inferred readings by at most 12 percentage points. Compensating for unavailable v-sensors in sparse environments is subject to future work.

B. Model-Driven Sensing Algorithm

Based on INFER and GREEDY, we now present the model-driven sensing algorithm. Given an MGD and a sensing task

```
Require: $Q = (V, p, QoS)$, $MGD_V$
  for all $task \in$ temporalDecomposition($p$) do
2:   $V_{\text{eff}} \leftarrow$ GREEDY($V, MGD_V, QoS.\sigma^2_{max}$)
     $task.V \leftarrow V_{\text{eff}}$; $readings \leftarrow \emptyset$
4:   $R_Q \leftarrow$ execute($task$)
     $R_Q \leftarrow R_Q \cup$ INFER($MGD_V, R_Q$)
6:   return Final Result $R_Q$
  end for
```

Figure 2. Model-driven sensing task execution

T = (V,QoS), we modify the operations of DrOPS to use the model as shown in Fig. 2. At the beginning of each sampling period, we use the modified GREEDY-Algorithm to select a set $V_{\text{eff}} \subseteq V$ of virtual sensors (l. 2). As the selection of V_{eff} only depends on the model, it is not strictly necessary to recompute V_{eff} for every task. However, this execution model allows us to easily integrate our validity check algorithm later on. Note that the selection of v-sensors does not change for the same model. However, due to node mobility, it is unlikely that a node has to provide multiple consecutive readings unless it remains stationary for a longer period of time. Therefore, we did not include explicit mechanisms for fair load balancing.

The task is then executed in lines 3 to 4 as in the basic algorithm. At the end of the sampling period, after collecting all effective readings at the gateway, the gateway performs the inference step. Using INFER, readings for v-sensors in $V^+_{\text{inf}} = V_{\text{inf}} \cup$ *unavailable v-sensors in* V_{eff} are inferred locally from the effective readings reported by mobile nodes in line 5. Thus, virtual readings for all v-sensors are provided by the system. For mimicking a classic sensor network, we output the inferred mean values for each $v \in V^+_{\text{inf}}$ as an inferred reading. If a client demands additional information, we also include the variances in the result R_Q.

V. Model Management

Next, we present our algorithms for creating and maintaining the model required for model-driven sensing. Existing approaches for optimizing data acquisition, e.g., [3], [4] aim to create a model that is accurate at all times. For instance, consider modelling temperatures which rise in the morning and drop in the afternoon. Training data from several days is required for an accurate model. However, as motivated in Section I, optimizations in a PS system must use training periods significantly less than a single day.

Therefore, the basic idea of our approach is to derive a model of the observed phenomenon on demand that is sufficiently accurate for the near future using an *online learning algorithm (OLA)*. The runtime of a query is divided into basic operation phases and optimized operation phases. During a basic operation phase, queries are executed using the basic sensing algorithm, i.e., $V_{\text{eff}} = V$. In parallel, OLA works to create a new MGD. As soon as a new model is output by OLA, the basic operation phase ends.

```
Require: Final Result $R_Q$, Control Readings $C_{V_{\text{ctl}}}$, Threshold $QoS.T$, Acceptable
    Violations $QoS.violations$
    RMSE $\leftarrow 0$
2:  for all $c \in V_{\text{ctl}}$, $c$ available do
      RMSE $\leftarrow$ RMSE $+ (R_c - C_c)^2$
4:  end for
    RMSE $\leftarrow \sqrt{\frac{\text{RMSE}}{|V_{\text{ctl}}|}}$
6:  if RMSE $\geq QoS.T$ then
      Add "violation" to window
8:  else
      Add "no violation" to window
10: end if
    if Num. of violations in window $> QoS.violations$ then
12:   return "Model Invalid"
    end if
14: return "Model Valid"
```

Figure 3. MOCHA algorithm. C denotes the set of v-sensors used for control readings.

After the basic operation phase, we switch to the optimized operation phase, where we use the MGD as described in Section IV to reduce the number of effective readings taken. Since such a model might not reflect changes happening over a longer time period—such as the temperature profile in the previous example—, we continuously monitor model accuracy using an *online model validity check algorithm (MOCHA)*. When MOCHA considers the MGD to be inaccurate, we switch to the next basic operation phase.

Next, we start by presenting our online model validity check algorithm MOCHA, before we present our online learning algorithm in detail.

A. MOCHA

The goal of MOCHA is to check a model for correctness. Intuitively, a model is correct if the Gaussian distributions of inferred readings fit the real data, i.e., inferred values from v-sensors $v \in V^+_{\text{inf}}$ center around the true mean value and the variance matches the true variance. However, checking this property for every v would require constant sampling of all v-sensors and thus render the optimization useless.

Therefore, we take a different approach for MOCHA (see Fig. 3). At the beginning of each sampling period, we randomly choose a set of control sensors $V_{\text{ctl}} \subseteq V_{\text{inf}}$ of size *QoS.ctrl*. In addition to the v-sensors V_{eff} selected by the GREEDY algorithm, we request effective readings for V_{ctl}. At the end of the sampling period, only effective readings from (available) v-sensors in V_{eff} are used as input for the inference algorithm. We then compute the root mean squared error (RMSE, lines 1 to 5) of mean values of inferred readings and their corresponding effective control readings. Using the RMSE, we avoid the problem of comparing individual samples to inferred distributions since we can compare absolute values directly. Furthermore, by adjusting the size of V_{ctl}, we can trade off the costs for effective sampling and the probability of detecting inaccurate models (Quality of Service).

If RMSE $>$ *QoS.T*, where *QoS.T* is a predefined threshold (part of the quality specification *QoS* of a sensing

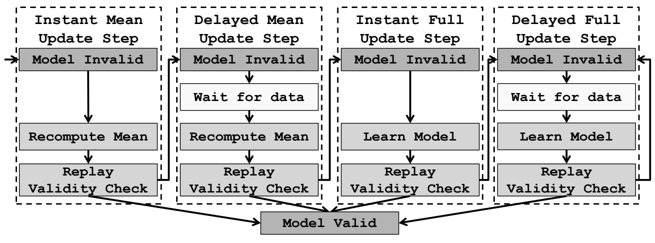

Figure 4. Overview of the basic operation phase

```
Require: $Q = (V, p, QoS)$, $MGD_V$
    $\forall v \in V : M'_V[v] = \perp$
 2: $W \leftarrow \emptyset$
    for all $v \in V$ do
 4:     $data \leftarrow$ getHistoricData($v$, [$QoS.maxAge$, now()]))
        if $|data| > 0$ then
 6:         $M'_V[v] \leftarrow$ mean($data$)
        else
 8:         $W \leftarrow W \cup \{v\}$
        end if
10: end for
    $M'_V[W] \leftarrow$ INFER($MGD_V$, $M'_V$)
12: return $MGD'_V$
```

Figure 5. Instant Mean Update Step

task), we say that the threshold has been violated. To avoid discarding an accurate model in case the observed violation was an outlier, we use a sliding window approach to dampen the reactivity of MOCHA (l. 6–14). We define a model to be inaccurate if there are *QoS.violations* within the last *QoS.win* samples. For example, for *QoS.win* = 2 and *QoS.violations* = 1, we discard the model after two consecutive violations.

Using MOCHA, we next introduce our online learning algorithm OLA.

B. OLA

During the basic operation phase, OLA aims to create a new MGD of the observed phenomenon. To this end, the basic operation phase is subdivided into four steps (cf. Fig. 4). In each step, a new model is created using all available data obtained in the current basic operation phase and all previous phases (both basic and optimization) for the same query up to a certain age given in *QoS.maxAge*. While reducing *QoS.maxAge* reduces the runtime of the learning algorithm, it should be set to a large enough value to accommodate expected periodic shifts. For example, for temperature shifts in day/night cycles, *QoS.maxAge* should be set to a multiple of the cycle duration. In our experiments, *QoS.maxAge* = 3 days showed best results.

At the end of every step, we replay GREEDY, INFER, and MOCHA on data from the last *QoS.win* sampling periods. If the new model is considered valid by MOCHA, we immediately switch to the next optimized operation phase using this model, thus, ensuring that OLA terminates in the earliest possible step. Next, we explain each step in detail.

In the **instant mean update step** (Fig. 5) we update only the mean vector of the previous MGD from existing data.

```
Require: $Q = (V, p, QoS)$, $MGD_V$
    $waitStart \leftarrow$ now()
 2: repeat
        wait for next sampling period
 4:     $V_{\text{eff}} \leftarrow V$
        request data
 6: until $|\{v \in V \,|\, |$getHistoricData($v$, [$waitStart$, now()])$| \geq QoS.minRdg\}| \geq QoS.minSens$ or $totalWaitTime > QoS.totalWaitTime$
    return INSTANTMEANUPDATE($R$, $MGD_V$)
```

Figure 6. Delayed Mean Update Step

```
Require: $Q = (V, p, QoS)$
    $data \leftarrow$ getHistoricData($V$, [$QoS.maxAge$, now()])
 2: return learnModel($data$)
```

Figure 7. Instant Full Update Step

This is motivated by the observation that when an MGD is considered invalid, the covariance matrix is often still correct while the mean vector failed to account for a global shift in the observed phenomenon. To avoid costly effective readings from all v-sensors, the mean vector is directly computed from data currently available on the gateway (l. 3–10). If no effective readings have been reported in the considered history, a mean value for this v-sensor is inferred using the old model (l. 11). This might yield a greater error in inferred readings, which, however, would be detected by MOCHA.

If the new model is invalid, all v-sensors are queried for effective readings in the **delayed mean update step** (Fig. 6). When *QoS.minRdg* effective readings have been received from at least *QoS.minSens* v-sensors each (l. 6), a new model is constructed as in the previous step (l. 7).

If the model is still considered invalid after the delayed mean update step, we update both the mean vector and covariance matrix in the **instant full update step** (Fig. 7) using an existing offline learning algorithm [8]. If the number of effective readings available for a v-sensor *v* at the end of the basic operation phase is insufficient for the offline learning algorithm, *v* is excluded from the model.

If the new model remains invalid, we continue with the **delayed full update step** (Fig. 8). In this step, we request fresh effective readings from all v-sensors and execute the offline learning algorithm when sufficient data has arrived. We repeat this step until a valid model has been created.

Note that OLA will remain in this step indefinitely if either a sufficiently large fraction of v-sensors remains unavailable or the replay of sensor data causes a model to be falsely considered invalid. Therefore, we introduce a hard runtime limit *QoS.totalWaitTime*, after which a new full model is learned from the available data and considered to be valid without further checking. Should the new model be invalid, this is detected by MOCHA after at most *QoS.window* sampling periods.

VI. EVALUATION

In this section, we evaluate the performance of the DrOPS system. As a proof of concept, we implemented DrOPS in

```
Require: $Q = (V, p, QoS)$
      $waitStart \leftarrow$ now()
  2: repeat
        wait for next sampling period
  4:    $V_{\text{eff}} \leftarrow V$
        request data
  6: until $|\{v \in V || \text{getHistoricData}(v, [waitStart, \text{now}()])| \geq QoS.minRdg\}| \geq QoS.minSens$ or $totalWaitTime > QoS.totalWaitTime$
     $data \leftarrow \text{getHistoricData}(V, [QoS.maxAge, \text{now}()])$
  8: return learnModel($data$)
```

Figure 8. Delayed Full Update Step

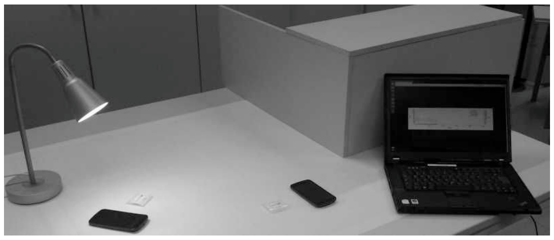

Figure 9. Public Sensing Testbed

a small-scale real-world testbed as discussed in the next section. To get more insight about the behavior of our system for large-scale tasks, we implemented DrOPS in a simulated environment as discussed in later sections.

A. Testbed Evaluation

Our testbed, depicted in Fig. 9, consists of a laptop, serving as the gateway and presenting a user interface for submitting queries and browsing obtained data. While it is possible to use our implementation to run a full-scale PS system, an experiment with hundreds of people participating is not feasible. Therefore, we select an evaluation scenario that can be handled by two people. Two smartphones are used for taking light intensity readings. Furthermore, we use a light source with predetermined movement to mimic the changing position of the sun over a day. Two v-sensors are placed directly under the initial position of the light source, two are placed at a distance, and another two are placed behind an obstacle shadowing the v-sensors from the light source. The smartphones then move among these v-sensors.

Figure 10 shows the output of our testbed evaluation. DrOPS runs for roughly three minutes in a basic operation phase to learn a model of the light intensity at the v-sensors. Note that in the basic operation phase in each sampling period only readings for available v-sensors are included in the mean light intensity. Thus, node mobility leads to shifting availability causing readings to appear fluctuating. During the optimized operation phase, readings for all v-sensors are included, yielding a smooth mean. Up to 900 s, we do not change system conditions, thus, the model remains accurate and readings are inferred with low error except for outliers. At 900 s the light source is moved, causing the existing model to become inaccurate. At this point, the error briefly goes up before MOCHA recognizes the model

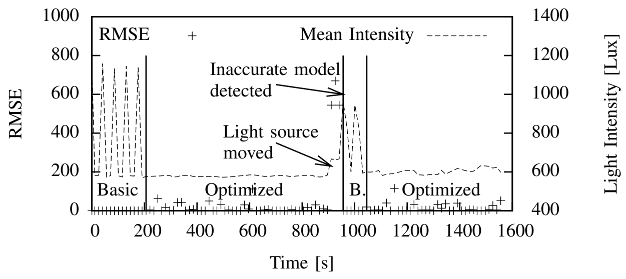

Figure 10. Output of the testbed evaluation. 15 s sampling period.

as inaccurate. After an additional 100 s, a new model is available and readings are again inferred with low error.

B. Simulation Setup

While our real-world testbed allows for insight into the operation of a model-driven PS system, it is far too small to provide information on the overall performance of the DrOPS system. Therefore, we extend our evaluation to a large-scale system using an implementation of DrOPS in a simulated environment implemented in the OMNeT++ network simulator. We first present a brief overview of the simulation setup before presenting simulation results.

To make our evaluation comparable to a real deployment, we use two real-world datasets as input: Ten days from the Intel Lab data set [3] and three non-consecutive weeks of data from the Lausanne Urban Canopy Experiment (LUCE) [9]. Both data sets contain environmental readings, e.g., temperature reported by a large set of fixed sensors.

We generate queries by placing a v-sensor for temperature data at the position of every real sensor in each data set in order to generate a temperature map of the area. The sampling period is adapted to match the interval at which data is provided by the data set. Other quality parameters used in the evaluation are shown in Table I. Simulations run for 6 h each, with a time offset between simulations increasing in steps of 3 h from the start of the data set.

To generate node mobility for the Intel Lab data, we place 200 mobile nodes on an abstract representation of the lab's floor plan. Nodes move around randomly along the available paths. For the LUCE data, we use CanuMobiSim [10] to generate random mobility traces for 400 nodes on a road graph of the deployment area. For reference, we also simulate our algorithms in a static sensor network to analyze how close our approach is to optimum performance.

We use empirical energy models for a 3G radio [11] and built-in sensors [12] to measure energy consumption. Energy for positioning is not taken into account, since our algorithm does not change the number of position fixes taken by each node with respect to the basic sensing algorithm. Exploiting the additional potential for saving energy by reducing the number of position fixes is part of future work.

Parameter		Intel Lab	LUCE
Error Threshold	*Q.T*	$1\,^{\circ}C$	$1\,^{\circ}C$
Max. Variance	$Q.\sigma^2_{max}$	0.1	0.1
Window Size	*Q.win*	10	10
Violations	*Q.violations*	3	4
Control Readings	*Q.ctrl*	3	1
Max. Learning Time	*Q.totalWaitTime*	1 hour	1 hour
Maximum Age	*Q.maxAge*	3 days	3 days
OLA Paramters	*Q.minRdg*	5	5
	Q.minSens	49	98

Table I
QUALITY PARAMETERS USED IN THE SIMULATION

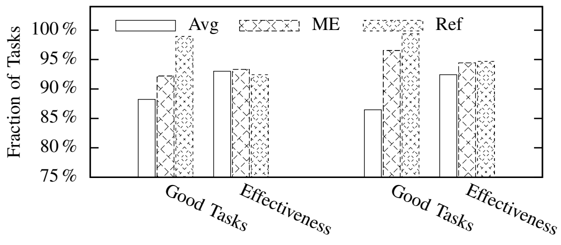

Figure 11. Quality Results. Average over all simulations (Avg), Median over all simulations (ME) and average over all reference simulations (Ref).

C. Simulation Results

We analyze the performance of DrOPS with regard to several metrics. *Effectiveness* is the fraction of tasks executed in an optimized operation phase, thus characterizing the time spent where the model-driven sensing algorithm is used. *Good Tasks* is the fraction of tasks where QoS-constraints are met, thus characterizing the data quality a client can expect. In addition, we compute the *average duration of basic operation phases*. Finally, *Relative Energy Consumption* is defined as the total energy consumption of all nodes divided by the total energy consumption of all nodes under the basic sensing algorithm for the same simulation parameters.

Figure 11 depicts the results for *effectiveness* and *good tasks*, shown as the average (Avg) and the median of all simulations (ME), and the average of the reference simulations (Ref). For *effectiveness*, results for both data sets are almost identical to the reference values from the sensor network whereas for *good tasks*, average values are 10 to 13 percentage points below the reference values. This indicates that the overall performance of our approach is basically similar to the reference system, except for a larger number of individual outliers caused by unavailable v-sensors both while learning a model and while optimizing execution. Note that for the LUCE data, in only 5 % of cases *good tasks* is below 50 %. For the Intel Lab data, in less than 5 % of cases the error threshold is violated by more than $0.2\,^{\circ}C$. On average, basic operation phases last for 7.5 minutes for the Intel Lab data and 16 minutes for the LUCE data, showing that DrOPS can learn a model in a matter of minutes.

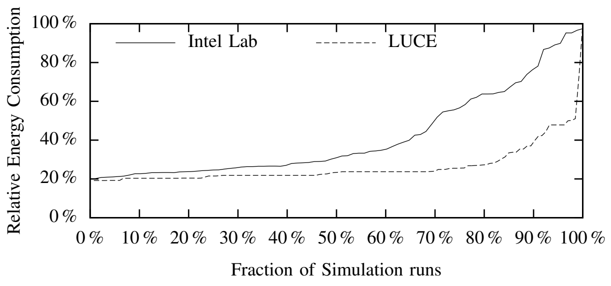

Figure 12. Cumulated relative energy for communication and sensing.

Finally, we analyze the energy consumption of our system. Note that the energy consumption for all nodes in each simulation is nearly uniformly distributed. This indicates that the energy consumption is dominated by the reception of tasks, especially by the size of task messages, which in turn depends on the number of v-sensors selected for effective readings. We can see from Fig. 12 that energy savings up to 80 % compared to the basic sensing algorithm are achieved in 3 % and 6 % of simulations (Intel Lab data and LUCE data, respectively). On average, 59 % of energy is saved for the Intel Lab data, and 74 % for the LUCE data. In a few extreme cases, next to no energy is saved. This happens when effective readings for most or all v-sensors are requested even in an optimized operation phase, i.e., when no correlation could be identified.

VII. Related Work

Research interest in PS has grown over the last few years [1]. Several prototype systems and system architectures have been proposed [5], [6], [13], [14]. However, all of these approaches focus on applications and general system challenges, and neither discusses possible optimizations.

Several prototype systems for monitoring environmental variables have been developed [15], [16]. However, they require energy-intensive constant sampling by all nodes.

Previous work on optimized PS focused on optimizations for individual v-sensors. Lu et al. present a first approach for location-centric sensing task execution [17] at a single v-sensor. In our previous research, we presented optimizations for reading fixed sensors via mobile phones [18] and for executing tasks at multiple v-sensors [2] in parallel. Furthermore, we extended the idea to continuous sampling along road segments [19] and updating of road-maps [20]. However, none of these works deals with optimizing data acquisition across multiple v-sensors.

Reddy et al. present an approach to manually select the best set of mobile nodes for data acquisition in a PS system based on long-term profiles for mobility and participation [21]. In contrast, DrOPS provides automatic operation and short setup times.

Closest to our work, model-driven approaches for fixed sensor networks limit data acquisition to sensors with the best informational value [3], [4]. In actuated sensing, optimal paths for mobile sensors are computed from the model [22], which assumes that node mobility is controlled by the system. None of these systems has to deal with online learning of models.

VIII. Conclusion

In this work, we presented the DrOPS system for monitoring environmental values using Public Sensing. DrOPS uses a model-driven sensing approach based on multivariate Gaussian distributions to infer readings, in order to reduce the set of mobile nodes that are queried for effective readings to reduce the energy consumption. Moreover, we can compensate for missing readings due to unavailable virtual sensors. Furthermore, we introduced OLA, an online learning algorithm to learn multivariate Gaussian distributions over short time periods, and MOCHA, an online model validity check algorithm to determine whether a given multivariate Gaussian distribution fits current sensor readings.

Our evaluations show that we obtain optimization models in a matter of minutes on average. Using the model-driven approach for optimizing the data acquisition, we can save up to 80 % of energy for communication and sensing and provide inferred readings for uncovered positions matching an error-bound of $1°C$ up to 100 % of the time.

In future work, we plan to increase energy savings for communication further by introducing an efficient hybrid 3G/WiFi ad-hoc routing scheme and to reduce the energy cost for positioning by an improved query execution model reducing the number of position fixes. Furthermore, we are going to extend DrOPS to work in sparse environments by detecting and adapting to unavailable v-sensors.

Acknowledgments

This work was partially funded by the SpoVNet project of Baden-Württemberg Stiftung gGmbH and the Com'N'Sense project of the German Research Foundation (DFG).

References

[1] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn, "The Rise of People-Centric Sensing," *IEEE Internet Computing*, vol. 12, no. 4, pp. 12–21, 2008.

[2] D. Philipp, F. Dürr, and K. Rothermel, "A sensor network abstraction for flexible public sensing systems," in *Proc. of the 8th IEEE Int. Conf. on Mobile Adhoc and Sensor Syst.*, Oct. 2011, pp. 460–469.

[3] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-Driven Data Acquisition in Sensor Networks," in *Proc. of the 30th Int. Conf. on Very large data bases*. VLDB Endowment, 2004, pp. 588–599.

[4] C. Guestrin, A. Krause, and A. P. Singh, "Near-optimal sensor placements in gaussian processes," in *Proc. of the 22nd Int. Conf. on Machine learning*. ACM, 2005, pp. 265–272.

[5] S. B. Eisenman, "People-centric mobile sensing networks," Ph.D. dissertation, Columbia University, New York, NY, USA, 2008.

[6] E. Kanjo, S. Benford, M. Paxton, A. Chamberlain, D. S. Fraser, D. Woodgate, D. Crellin, and A. Woolard, "MobGeoSen: Facilitating Personal Geosensor Data Collection and Visualization Using Mobile Phones," *Personal and Ubiquitous Computing*, vol. 12, pp. 599–607, 2008.

[7] N. A. C. Cressie, *Statistics for Spatial Data*. Wiley-Interscience, 1993.

[8] A. Schwaighofer, V. Tresp, and K. Yu, "Learning gaussian process kernels via hierarchical bayes," in *Adv. in Neural Information Processing Syst.*, no. 17, 2005, pp. 1209–1216.

[9] D. Nadeau, W. Brutsaert, M. Parlange, E. Bou-Zeid, G. Barrenetxea, O. Couach, M.-O. Boldi, J. Selker, and M. Vetterli, "Estimation of urban sensible heat flux using a dense wireless network of observations," *Environmental Fluid Mechanics*, vol. 9, pp. 635–653, 2009.

[10] I. Stepanov. CANU Mobility Simulation Environment. [Online]. Available: http://canu.informatik.uni-stuttgart.de/

[11] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proc. of the 9th ACM SIGCOMM Conf. on Internet Measurement*, 2009, pp. 280–293.

[12] B. Priyantha, D. Lymberopoulos, and J. Liu, "Eers: Energy efficient responsive sleeping on mobile phones," in *Proc. of the Int. Workshop on Sensing for App Phones*, Nov. 2010.

[13] E. Miluzzo, N. D. Lane, S. B. Eisenman, and A. T. Campbell, "CenceMe – Injecting Sensing Presence into Social Networking Applications," in *Smart Sensing and Context*. Springer Berlin / Heidelberg, 2007, pp. 1–28.

[14] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Cartel: a distributed mobile sensor computing system," in *Proc. of the 4th Int. Conf. on Embedded Networked Sensor Syst.*, 2006, pp. 125–138.

[15] A. Steed and R. Milton, "Using Tracked Mobile Sensors to Make Maps of Environmental Effects," *Personal and Ubiquitous Computing*, vol. 12, no. 4, pp. 331–342, 2008.

[16] N. Maisonneuve, M. Stevens, M. E. Niessen, P. Hanappe, and L. Steels, "Citizen Noise Pollution Monitoring," in *Proc. of the 10th Annu. Int. Conf. on Digital Government Research*, 2009, pp. 96–103.

[17] H. Lu, N. D. Lane, S. B. Eisenman, and A. T. Campbell, "Bubble-sensing: Binding Sensing Tasks to the Physical World," *Pervasive and Mobile Computing*, vol. 6, no. 1, pp. 58 – 71, 2009.

[18] H. Weinschrott, F. Dürr, and K. Rothermel, "Efficient Capturing of Environmental Data with Mobile RFID Readers," in *Proc. of the 10th Int. Conf. on Mobile Data Manage.*, 2009, pp. 41–51.

[19] ——, "StreamShaper: Coordination Algorithms for Participatory Mobile Urban Sensing," in *Proc. of the 7th IEEE Int. Conf. on Mobile Ad-hoc and Sensor Syst.*, Nov. 2010, pp. 1–10.

[20] P. Baier, H. Weinschrott, F. Dürr, and K. Rothermel, "MapCorrect: automatic correction and validation of road maps using public sensing," in *Proc. of the 36th IEEE Conf. on Local Comput. Networks*, Bonn, Germany, Oct. 2011.

[21] S. Reddy, D. Estrin, and M. B. Srivastava, "Recruitment Framework for Participatory Sensing Data Collections," in *Proc. of the Int. Conf. on Pervasive Computing*, May 2010, p. 18.

[22] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings, "Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm," in *Proc. of the 21st Int. joint Conf. on Artifical Intell.*, 2009, pp. 299–304.