On Crowdsensed Data Acquisition using Multi-Dimensional Point Processes

Saket Sathe[†], Timos Sellis[‡], Karl Aberer[⋆]

[†]*IBM Research – Australia.*
ssathe@au.ibm.com

[‡]*RMIT University, Australia.*
timos.sellis@rmit.edu.au

[⋆]*EPFL, Switzerland.*
karl.aberer@epfl.ch

***Abstract*—Crowdsensing applications are increasing at a tremendous rate. In crowdsensing, mobile sensors (humans, vehicle-mounted sensors, etc.) generate streams of information that is used for inferring high-level phenomena of interest (e.g., traffic jams, air pollution). Unlike traditional sensor network data, crowdsensed data has a highly skewed spatio-temporal distribution caused largely due to the mobility of sensors [1]. Thus, designing systems that can mitigate this effect by acquiring crowdsensed at a fixed spatio-temporal rate are needed. In this paper we propose using multi-dimensional point processes (MDPPs), a mathematical modeling tool that can be effectively used for performing this data acquisition task.**

I. Introduction

Recent times have seen a dramatic rise of crowdsensing [2]–[4]. In this paradigm *mobile sensors*, like smartphones[1], sensors mounted on vehicles (cars or buses), or humans themselves are used for gathering information and inferring various aspects regarding a phenomena of interest. Here the sensors (or humans) are mobile and therefore the data produced by them is in the form of spatio-temporal data streams. Humans are involved for acquiring information that is hard to sense using sensors (e.g., traffic jams). The data acquired using crowdsensing principles is typically used for performing high-level inference [2] or phenomena detection [1].

One would rightly ask, how is crowdsensed data acquisition different from traditional wireless sensor network (WSN) data acquisition? The principal differences between them are as follows: (1) in crowdsensing, sensors are mobile and not stationary, (2) the number of mobile sensors in a particular region and time is unpredictable and is spatio-temporally skewed, (3) the rate at which these sensors generate data cannot be easily controlled, this is especially true for humans, who may decide to perform or not perform certain tasks depending on the incentive offered.

Thus, we need a method for acquiring mobile crowdsensed data streams (MCDS) that addresses the aforementioned differences yet is simple, elegant, and enables declarative specification of data acquisition queries. At its core, we believe that the most simplest acquisitional queries for crowdsensed data acquisition will have to provide at least the following information: (1) the attribute they are interested in acquiring, (2) the region or sub-region from which the attribute should be acquired, (3) the rate (per unit area and time) at which this attribute should be acquired.

Given our straightforward world-view of crowdsensed data acquisition, it is apparent that it drastically mismatches the characteristics exhibited by MCDS. On the one hand, we have the most simplest description of acquiring crowdsensed data streams, and on the other hand, we have mobile sensors having uncontrollable mobility and varying data generation rates. The techniques proposed in this paper bridges this gap by accepting user queries for acquiring MCDS and ensures (at least in a probabilistic sense) that these queries are answered satisfactorily. To achieve this goal, we propose a novel set of probabilistic streaming operators that are derived using MDPPs and can be implemented using only a few lines of code. Similar to existing stream processing operators [5]–[7], these operators can be connected to form an execution topology for simultaneously processing a large number of acquisitional queries.

Related Work: Acquisitional queries for fixed sensors are extensively surveyed in [8]. These works either use an overlay network (tree-based or ad-hoc routing protocols) or probabilistic approaches for acquiring and querying sensor data [9]. Unfortunately, these principles cannot be extended for acquiring MCDS, as MCDS acquisition is a fundamentally different problem as compared to WSN data acquisition. Stream processing architectures [5]–[7] propose declarative syntax for defining an execution topology over a set of stream processing operators. This paper significantly enhances such systems by providing stream processing operators for continuously acquiring MCDS at probabilistically guaranteed user-specified rates.

II. Background

Consider a geographical area of interest denoted by $\mathcal{R}$. We assume that there are m mobile sensors, denoted as $s_1, s_2, \ldots, s_m$, in the region $\mathcal{R}$. These mobile sensors comprise of sensors (smartphones, wearables, handheld sensors, etc.) carried by humans or vehicle-mounted sensors. In certain instances, humans assist in a sensing task by responding to a question displayed on their smartphones related to them or their surrounding. Each mobile sensor is assumed to have local memory to store sensed information.

[1]A standard smartphone has about 10 sensors. See http://bit.ly/sg3specs.

We assume that the mobile sensors have agreed to share all the information required for processing queries with a central server. Examples of such information could be GPS location, microphone level, phone status, values from an embedded sensor, or a response to a crowdsensed attribute etc.

For simplicity, we assume that there are a fixed set of attributes of interest $A^{\langle 1\rangle}, A^{\langle 2\rangle}, \ldots, A^{\langle k\rangle}$. These attributes are either sensed by humans or by sensors. Human-sensed attributes are the ones that are typically hard to sense with a sensor. For e.g., whether it is currently raining around a mobile sensor could be a human-sensed attribute. Sensor-sensed attributes are observed using a sensor, e.g., the current ambient temperature around a mobile sensor. We assume that each mobile sensor generates a stream of tuples. A tuple of attribute $A^{\langle j\rangle}$ is denoted as $(t_i^{\langle j\rangle}, x_i^{\langle j\rangle}, y_i^{\langle j\rangle}, a_i^{\langle j\rangle})$, where the first three entries are the space-time co-ordinates, $a_i^{\langle j\rangle}$ is the value of attribute $A^{\langle j\rangle}$, and i is a unique tuple identifier across sensors. The position can also include a z-coordinate; for brevity reasons, we only work with 2-D coordinates.

Before we define multi-dimensional point processes (MDPPs) and discuss how they can be used for acquiring crowdsensed streams, we present two running examples that will be used throughout the paper. **Rain monitoring** is an example where we are interested to acquire data about whether it is raining or not in a given sub-region $\mathcal{R}' \subseteq \mathcal{R}$. This requires acquisition of the human-sensed boolean attribute $A^{\langle 1\rangle} = rain$. **Ambient temperature monitoring** is about knowing the ambient temperature sampled from a chosen subset of mobile sensors in region $\mathcal{R}' \subseteq \mathcal{R}$. This requires that we acquire the sensor-sensed real value of the attribute $A^{\langle 2\rangle} = temp$. In both the examples above, we are interested in continuously acquiring crowdsensed data streams at a user defined spatio-temporal acquisition rate.

III. Acquiring Crowdsensed Data Streams

Recall that the simplest queries for acquiring MCDS will have to specify the following parameters: 1) The attribute $A^{\langle j\rangle}$ they want to acquire, 2) The region from which they want to acquire the given attribute, 3) the rate at which they want to acquire the attribute. Our task is to acquire a crowdsensed data stream given the three aforementioned parameters. An example of a acquisitional query on attribute $A^{\langle 1\rangle}$, denoted as $Q^{\langle 1\rangle}$, could be the following:

$Q^{\langle 1\rangle}$: Acquire the attribute $A^{\langle 1\rangle} = rain$ from region $\mathcal{R}' \subset \mathcal{R}$ at the rate of 10 /km^2/min.

The output of this query is a MCDS of tuples $(t, x, y, rain)$ at the rate of 10 /km^2/min.

Undoubtedly, the most challenging component of the above query is to maintain the expected spatio-temporal rate. Once we have accomplished the task of acquiring the desired MCDS at the specified rate, there are several other existing methods, systems, and query languages that can be used for processing such streams [5]–[7]. In pure sensor data acquisition systems [9], maintaining such a rate is not significantly difficult, since the sensors in a WSN are stationary and can be sampled any time as long as they are powered and queries do not depend on human input.

When queries enable human input and sensors have unpredictable mobility, such assumptions on any-time sampling are difficult to justify. The reason is that we have nearly no control on the mobile sensors. For example, if a human user is requested to respond to the question: *Is it raining around you?* His/her reply could be unpredictably delayed for several reasons: he/she is not interested in responding at this moment, he/she thinks that the incentive offered for responding is not enough or he/she has moved to a different location, which now is not of interest to the query.

In addition to this challenge, our design goal is to process a large number of queries simultaneously. The naïve strategy of processing each query from scratch (i.e., individually), is not cost effective especially for the human-sensed attributes. This is because the data acquired for a particular attribute will not be re-used across queries. Instead, multiple query optimization principles need to be employed in this setting [10].

To address these challenges, we propose two novel ideas: (1) we characterize the spatio-temporal arrival rate of crowdsensed data at the server using MDPPs [11], (2) in Section IV, we propose algebraic operators that can be used for performing a wide variety of operations (rate modifications, partitioning, etc.) on MDPPs. These operators can be implemented in a few lines of code and can re-use data to simultaneously fabricate crowdsensed data streams for multiple queries.

A. Multi-Dimensional Point Processes (MDPPs)

A MDPP is a n-dimensional generalization of the Poisson process [11], which operates only in the time dimension. A Poisson process has a parameter known as the *rate*, denoted as λ. It indicates on an average how many samples are expected in a particular time window. For example, if a Poisson process models the length of a queue at a shopping counter, then the parameter λ indicates the number of customers arriving in a given time interval.

Similarly, a MDPP's rate parameter indicates how many samples are expected in a given n-dimensional window. We model the spatio-temporal arrival of data from each attribute as a 3-D MDPP, where the three dimensions refer to space and time. The intuition behind using MDPPs is primarily: (a) it has been shown that a large variety of spatio-temporal arrival patterns can be modeled as MDPPs [11], [12], (b) as discussed later, MDPPs have several elegant properties that can be exploited for managing crowdsensed data streams.

We use two types of MDPPs: homogeneous and inhomogeneous. A *homogeneous* MDPP[2] for attribute $A^{\langle j\rangle}$ has a constant positive rate $\lambda^{\langle j\rangle}$ over space and time and is denoted as $P^{\langle j\rangle}(\lambda, \mathcal{R})$, where $\mathcal{R}$ indicates its spatial extent. To make the notation concise, we drop the superscript $\langle j\rangle$ of λ in the notation $P^{\langle j\rangle}(\lambda, \mathcal{R})$, as it is clear that we are referring to attribute $A^{\langle j\rangle}$. The *inhomogeneous* MDPP has a positive rate $\tilde{\lambda}^{\langle j\rangle}(t, x, y)$ that is a function of both space and time and is denoted as $\tilde{P}^{\langle j\rangle}(\tilde{\lambda}, \mathcal{R})$. Often the rate of an inhomogeneous MDPP is modeled using a parametrized form as,

$$\tilde{\lambda}^{\langle j\rangle}(t, x, y; \boldsymbol{\theta}) = \theta_0 + \theta_1 t + \theta_2 x + \theta_3 y, \qquad (1)$$

[2]All MDPPs in this paper are homogeneous, unless explicitly specified.

where $\boldsymbol{\theta} = \{\theta_0, \ldots, \theta_3\}$. $\tilde{\lambda}^{\langle j \rangle}(t, x, y; \boldsymbol{\theta})$ is also known as the *conditional rate*; it establishes the rate of an inhomogeneous MDPP at a point (t, x, y), given the parameters $\boldsymbol{\theta}$. Such a parametrization has two advantages: (1) given a set of acquired tuples for an attribute $A^{\langle j \rangle}$, we can estimate the rate of an inhomogeneous MDPP using techniques like maximum-likelihood estimation [12], (2) it gives us a continuous function with parameters (t, x, y) for querying the rate at a given location. We use the conditional rate model given in Eq. (1) to model the spatio-temporal variation of the rate of an inhomogeneous MDPP.

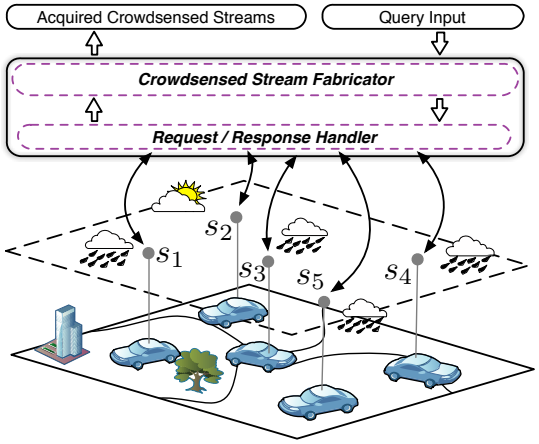

Fig. 1: Architecture of CrAQR.

IV. System Architecture

The system architecture used for query processing is shown in Figure 1. The region $\mathcal{R}$ is partitioned into a $\sqrt{h} \times \sqrt{h}$ sized grid. h is a user-defined parameter and controls the granularity at which queries can be processed. As we shall discuss in Section V, this partitioning is entirely logical, in reality only the grid cells that are useful for query processing are materialized. A grid cell with xy-coordinates (q, r) is denoted as $\mathcal{R}^{(q,r)}$ and

$$area(\mathcal{R}) = \sum_{\forall (q,r)} area(\mathcal{R}^{(q,r)}), \tag{2}$$

where $area(\cdot)$ is a function that computes the area of its argument. A single-attribute query should be on a region with area at least $area(\mathcal{R}^{(q,r)})$.

A. Request/Response Handler

The request/response handler has the task of sending data acquisition requests to mobile sensors and collecting their responses. It uses a parameter known as the *budget*. Budget is defined as the number of acquisitional requests per attribute and per grid cell that can be sent in a given duration of time. For example, the $rain$ attribute could have a budget of 100 /hr on the region $\mathcal{R}^{(1,2)}$. The budget specification does not need a spatial component, as all the grid cells are of equal size. The budget for an attribute $A^{\langle j \rangle}$ on $\mathcal{R}^{(q,r)}$ is denoted as $\beta^{\langle j \rangle}_{(q,r)}$. Data acquisition requests are sent to a randomly selected set of mobile sensors. Mobile sensors are sampled with or without replacement, depending on the number of mobile sensors available.

B. Crowdsensed Stream Fabricator

This is the most important component responsible for performing the operations required for answering acquisitional queries. It uses a novel set of proposed operators called point process transformation (PMAT) operators. PMAT are algebraic operators that are used for manipulating point processes. They can be used for performing tasks such as, converting an inhomogenous MDPP into a homogeneous MDPP or converting one MDPP into another with lower rate, etc.

Here, we will discuss in detail the functionality provided by some of these operators. In the following section, we will show how they can be interconnected together for constructing an execution topology similar to traditional stream data management systems. Such a topology can be used for simultaneously fabricating the required MCDS for multiple queries.

1) Point Process Transformation Operators: One of the most elegant properties of MDPPs is that their rate can be modified by randomly dropping a few tuples [11]. PMAT operators use such properties to define a set of algebraic operators on point processes. All PMAT operators are probabilistic and approximate with provable expected behaviour [11]; thus dramatically simplifying their implementation. We have researched many more operators than presented below, but due to space constraints and in order to convey the key ideas, we only discuss four most important operators.

Flatten: This operator converts a single-attribute inhomogeneous MDPP $\tilde{P}^{\langle j \rangle}(\tilde{\lambda}, \mathcal{R}^\star)$ to an *approximately* homogeneous point process $P^{\langle j \rangle}(\bar{\lambda}, \mathcal{R}^\star)$ with rate $\bar{\lambda}^{\langle j \rangle}$, where $\mathcal{R}^\star \subset \mathcal{R}$. The intuition behind the flatten operator is that a point process can be made homogeneous by retaining a random subset of tuples, such that more tuples are retained in areas of low rate and less tuples are retained in areas of high rate [12].

Concretely, it works as follows. Given a batch of size n of the spatio-temporal co-ordinates $\{t_i^{\langle j \rangle}, x_i^{\langle j \rangle}, y_i^{\langle j \rangle}\}_{i=1}^n$ of the crowdsensed tuples of attribute $A^{\langle j \rangle}$ obtained from the region $\mathcal{R}^\star$ and the desired rate $\bar{\lambda}^{\langle j \rangle}$, it computes the probability for each tuple to be retained in the batch. This probability is called the *retaining probability* and is defined as

$$p_i^{\langle j \rangle} = \frac{\bar{\lambda}^{\langle j \rangle}}{\tilde{\lambda}^{\langle j \rangle}(t_i^{\langle j \rangle}, x_i^{\langle j \rangle}, y_i^{\langle j \rangle}; \boldsymbol{\theta}) \times \lambda_c^{\langle j \rangle}}, \tag{3}$$

where $\lambda_c^{\langle j \rangle} = \sum_{i=i}^{n} \tilde{\lambda}^{\langle j \rangle}(t_i^{\langle j \rangle}, x_i^{\langle j \rangle}, y_i^{\langle j \rangle}; \boldsymbol{\theta})^{-1}$ and is a constant term over the batch. The retaining probability $p_i^{\langle j \rangle}$ of the tuple i is high, if it is from an area of low rate, as compared to $\bar{\lambda}^{\langle j \rangle}$.

It could happen that $p_i^{\langle j \rangle} > 1$. This is when the denominator of Eq. (3) is less than $\bar{\lambda}^{\langle j \rangle}$. The operator labels such instances as rate violations and reports *percent rate violation* in a batch. Percent rate violation is denoted as $N_v^{\langle j \rangle}$, and the retaining probabilities of such violations are rounded to 1. If $N_v^{\langle j \rangle}$ increases, it means that sufficient tuples are not present in the batch to create a point process with rate $\bar{\lambda}^{\langle j \rangle}$. As we will discuss in Section V, $N_v^{\langle j \rangle}$ is used for tuning the rate at which the request/response handler sends acquisition requests.

In the next step, flatten generates a Bernoulli random variable b with probability $p_i^{\langle j \rangle}$; a tuple $(t_i^{\langle j \rangle}, x_i^{\langle j \rangle}, y_i^{\langle j \rangle})$ is forwarded to the next operator if $b = 1$ and is discarded otherwise.

If necessary, the discarded tuples can be stored separately. As shown in [12], this procedure produces an approximately homogeneous point process. The flattening operation can also be performed over sliding windows, as opposed to batches. This can be done using online parameter estimation algorithms like stochastic gradient descent to estimate parameters $\boldsymbol{\theta}$ in [13] and maintaining $p_j^{(i)}$ and $N_v^{\langle j \rangle}$ over a sliding window. Lastly, the flatten operator is depicted as follows:

$$\tilde{P}^{\langle j \rangle}(\tilde{\lambda}, \mathcal{R}^\star) \longrightarrow \boxed{\mathcal{F}} \longrightarrow P^{\langle j \rangle}(\bar{\lambda}, \mathcal{R}^\star), \quad \boxed{\mathcal{F}} \dashrightarrow N_v^{\langle j \rangle}$$

Thin: Converts a MDPP $P^{\langle j \rangle}(\lambda_1, \mathcal{R}^\star)$ into another MDPP $P^{\langle j \rangle}(\lambda_2, \mathcal{R}^\star)$ whose rate is strictly less than the original MDPP, i.e. $\lambda_2^{\langle j \rangle} < \lambda_1^{\langle j \rangle}$ and $\mathcal{R}^\star \subset \mathcal{R}$. It is drawn as the following block:

$$P^{\langle j \rangle}(\lambda_1, \mathcal{R}^\star) \longrightarrow \boxed{\mathcal{T}} \longrightarrow P^{\langle j \rangle}(\lambda_2, \mathcal{R}^\star)$$

Thinning works as follows: (1) compute a probability $p = \frac{\lambda_2^{\langle j \rangle}}{\lambda_1^{\langle j \rangle}}$, (2) draw a random sample b from a Bernoulli distribution with probability p, which is equivalent to a biased coin toss with bias p, (3) if $b = 1$ then the tuple is forwarded to the next operator, otherwise it is dropped. It can be shown that this simple procedure produces a point process with the desired rate $\lambda_2^{\langle j \rangle}$.

Partition: Partitions a point process $P^{\langle j \rangle}(\lambda, \mathcal{R}^\star)$ into two point processes of the same rate $\lambda^{\langle j \rangle}$ but on different regions $\mathcal{R}_1^\star \subset \mathcal{R}^\star$ and $\mathcal{R}_2^\star \subset \mathcal{R}^\star$, such that $\mathcal{R}_1^\star \cap \mathcal{R}_2^\star = \emptyset$. It is denoted as follows:

$$P^{\langle j \rangle}(\lambda, \mathcal{R}^\star) \longrightarrow \boxed{\mathcal{P}} \longrightarrow P^{\langle j \rangle}(\lambda, \mathcal{R}_1^\star), \quad \boxed{\mathcal{P}} \longrightarrow P^{\langle j \rangle}(\lambda, \mathcal{R}_2^\star)$$

This operator is implemented by checking to which region the incoming tuple belongs, and then transmitting it to the appropriate output branch. This operator can be easily extended to partition processes into multiple regions.

Union: It unions two MDPPs $P^{\langle j \rangle}(\lambda, \mathcal{R}_1^\star)$ and $P^{\langle j \rangle}(\lambda, \mathcal{R}_2^\star)$ to form a process $P^{\langle j \rangle}(\lambda, \mathcal{R}_3^\star)$ where $\mathcal{R}_3^\star = \mathcal{R}_1^\star \cup \mathcal{R}_2^\star$. It is denoted as follows:

$$P^{\langle j \rangle}(\lambda, \mathcal{R}_1^\star) \longrightarrow \boxed{\mathcal{U}}, \quad P^{\langle j \rangle}(\lambda, \mathcal{R}_2^\star) \longrightarrow \boxed{\mathcal{U}} \longrightarrow P^{\langle j \rangle}(\lambda, \mathcal{R}_3^\star)$$

Notice that for computing $\mathcal{R}_1^\star \cup \mathcal{R}_2^\star$ the rectangles should be adjacent and with a common side of equal length. This operator can be easily extended to union multiple MDPPs at once.

V. Query Processing

In this section we will discuss how the PMAT operators are used for simultaneously processing multiple acquisitional queries, and ensuring that the desired rates for each attribute of the query are satisfied over the query region.

We explain query processing in detail using our toy examples from Section II. Recall, $A^{\langle 1 \rangle} = rain$ and $A^{\langle 2 \rangle} = temp$ and suppose we have queries $Q_1^{\langle 1 \rangle}$, $Q_2^{\langle 2 \rangle}$, and $Q_3^{\langle 2 \rangle}$ over the regions $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ respectively, and they have requested rates $\lambda_1^{\langle 1 \rangle}$, $\lambda_2^{\langle 2 \rangle}$, and $\lambda_3^{\langle 2 \rangle}$. In addition, we assume that $\lambda_1^{\langle 1 \rangle} > \lambda_2^{\langle 2 \rangle} > \lambda_3^{\langle 2 \rangle}$. Then, query processing consists of the following two phases:

A. Topology Construction

In this phase the data structures required for query processing are initialized. First, a hashmap is constructed where the keys of the hashmap are the xy-cordinates of grid cells $\mathcal{R}^{(q,r)}$. The value of a key (q, r) is the execution topology that is responsible for processing all the tuples that are crowdsensed in $\mathcal{R}^{(q,r)}$. This execution topology is formed by interconnecting a set of PMAT operators as explained below.

Query Insertions: For a given query region, we compute the amount of overlap that it has with each grid cell $\mathcal{R}^{(q,r)}$. In our example, say we are inserting $Q^{\langle 1 \rangle}$, then we compute the overlap between all grid cells and $\mathcal{R}_1$. Next, for each grid cell that has a non-zero overlap with the query region, we check whether the key for that grid cell is present in the hashmap.

If the key is absent, it is created and a $\mathcal{F}$-operator is added to it. The first operator is always the $\mathcal{F}$-operator, as all the other operators only work on homogeneous point process, and this is the only operator that has the capability of converting an inhomogeneous MDPP to a homogeneous MDPP. Next, the $\mathcal{T}$-operator is added for each attribute followed by the $\mathcal{P}$-operator to partition-out the overlapping region from the grid cell.

If the key is present, then (1) the $\mathcal{T}$-operators are added such that the rates of all the existing $\mathcal{T}$-operators remain sorted in a descending order and the highest rate $\mathcal{T}$-operator is closest to the $\mathcal{F}$-operator, (2) two $\mathcal{T}$-operators cannot be consecutively placed unless there is a *branching point* between them, otherwise these operators can be combined to form a single $\mathcal{T}$-operator (refer Fig. 2(*b*)), (3) if needed, the output rate of the $\mathcal{F}$-operator is changed to a value greater than the output rate of the first $\mathcal{T}$-operator. Finally, if required the $\mathcal{P}$-operators are added after the $\mathcal{T}$-operators. In our example, $\mathcal{P}$-operators are required only for $Q_3^{\langle 2 \rangle}$, since $Q_1^{\langle 1 \rangle}$ and $Q_2^{\langle 2 \rangle}$ perfectly overlap the grid cells.

Query Deletions: When a query is deleted, then all the crowdsensed data streams that are part of that query are removed. Concretely, referring to Fig. 2(*b*) and Fig. 2(*c*), crowdsensed data streams are deleted from right to left, until we hit a branching point or until all the streams and the key in the hashmap are deleted. For example, if we delete $Q^{\langle 1 \rangle}$, then first the crowdsensed stream $P^{\langle 1 \rangle}(\lambda_1, \mathcal{R}_1)$ is deleted, followed by the $\mathcal{U}$-, $\mathcal{T}$-, and $\mathcal{F}$-operators associated with the regions $\mathcal{R}^{(2,3)}$, $\mathcal{R}^{(3,2)}$ and $\mathcal{R}^{(3,3)}$. Finally, all the entries in the hashmap for these regions are removed. If two consecutive $\mathcal{T}$-operators are created in this process, then they are merged to form a single $\mathcal{T}$-operator.

Budget Tuning: The $\mathcal{F}$-operators report the percent rate violation $N_v^{\langle j \rangle}$ in a batch. We check whether $N_v^{\langle j \rangle}$ is under a user-defined threshold. If $N_v^{\langle j \rangle}$ exceeds the threshold, then

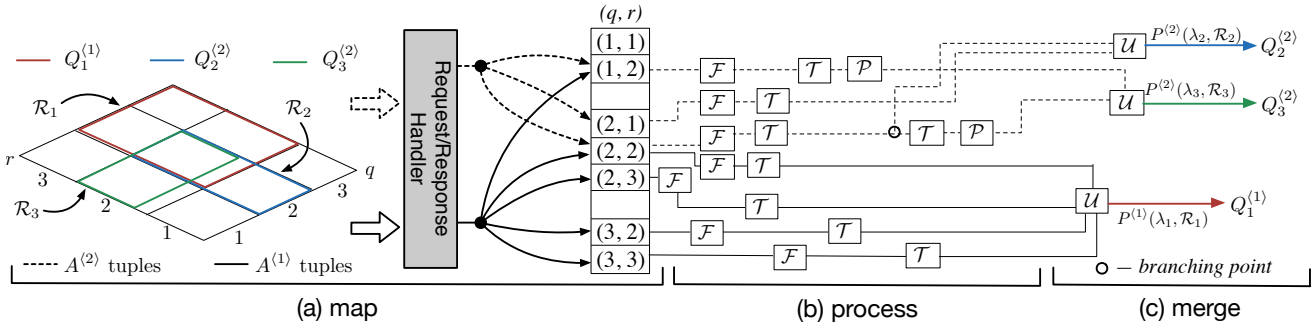

Fig. 2: Query processing: (a) **map** assigns the tuples to their respective key in the hashmap. (b) **process** executes the execution topology for each grid cell and fabricates partial crowdsensed data streams. (c) **merge** aggregates these partial streams appropriately to form the ultimate crowdsensed data stream. The percent rate violation $N_v^{\langle j \rangle}$ (not shown in figure) given by the $\mathcal{F}$-operators are used for tuning request/response handler's budget.

the budget $\beta_{(q,r)}^{\langle j \rangle}$ is increased by $\Delta\beta$, otherwise it is decreased by the same amount. If the budget cannot be increased beyond a limit, then the user is requested to either accept the feasible rate or pay more to obtain the required rate.

Stream Fabrication: When the request/response handler sends a batch of tuples for attribute $A^{\langle j \rangle}$, it is determined in which grid cells each of the tuple lies, and then it is forwarded to the corresponding execution topology. The tuples pass through various PMAT operators and finally form the per grid-cell MCDS. Lastly, the final MCDS are constructed using the $\mathcal{U}$-operator on the per grid-cell streams, see Fig. 2(c). These streams are returned to the user or can be further processed using well-known stream processing frameworks.

VI. Extensions and Optimizations

This is a work-in-progress, there are several interesting and challenging topics that we plan to explore in the future versions. Below we present a glimpse of such topics:

- **Including incentives:** Currently, if there are significant rate violations then the request/response handler, in the hope of reducing violations, increases its rate of sending acquisition requests. Another alternative is to offer more incentive to the mobile sensors to respond. Therefore, we will include mechanisms to define and optimally distribute such incentives [14].
- **Alternative topologies:** The execution topology presented in Section V is one of the many ways in which queries can be processed. For example, a tree-like topology can be formed. We have already started working on the necessary operators to perform this task.
- **Query optimization:** We should define the cost of processing a single query, and prepare an execution topology that minimizes this cost. Response time, power consumption, communication cost due to operator placement are some of the aspects that we plan to consider during optimization.
- **Handling errors:** Errors can be introduced by sampling constraints, GPS errors, sensors inaccuracies, or errors in human judgment. In the future, we will explore methods for mitigating the effect of such errors on query accuracy.

VII. Conclusion

Crowdsensing applications are growing rapidly and this poses many exciting data management challenges. In this paper we proposed techniques for elegantly acquiring crowdsensed data streams using MDPPs. We proposed a novel set of streaming operators (PMAT) for simultaneously acquiring multiple MCDS. Lastly, we discussed extensions and optimizations that we will explore in the future. We believe this will open exciting research areas for the data management community.

References

[1] K. Aberer, S. Sathe, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Theile, "OpenSense: Open community driven sensing of environment," in *IWGS*, 2010.

[2] "Waze," https://www.waze.com/. [Online]. Available: https://www.waze.com/

[3] "Premise," http://www.premise.com/.

[4] "MobileWorks," https://www.mobileworks.com/.

[5] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora : A new model and architecture for data stream management," *VLDB Journal*, pp. 120–139, 2003.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ : Continuous dataflow processing for an uncertain world," in *CIDR*, 2003.

[7] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language : semantic foundations," *VLDB Journal*, vol. 15, pp. 121–142, 2006.

[8] S. Sathe, T. Papaioannou, H. Jeung, and K. Aberer, "A survey of model-based sensor data acquisition and management," in *Managing and mining sensor data*. Springer, 2013, pp. 9–50.

[9] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *TODS*, vol. 30, no. 1, pp. 122–173, Mar. 2005. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1061318.1061322

[10] T. Sellis, "Multiple-query optimization," *TODS*, vol. 13, no. 1, pp. 23–52, 1988.

[11] D. J. Daley and D. Vere-Jones, *An introduction to the theory of point processes*. Springer, 1988, vol. 1-2.

[12] R. D. Peng, F. P. Schoenberg, and J. A. Woods, "A space–time conditional intensity model for evaluating a wildfire hazard index," *Journal of the American Statistical Association*, vol. 100, no. 469, 2005.

[13] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[14] "Amazon Mechanical Turk," https://www.mturk.com/.