Chapter 1

A SURVEY OF MODEL-BASED SENSOR DATA ACQUISITION AND MANAGEMENT

Saket Sathe
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
saket.sathe@epfl.ch

Thanasis G. Papaioannou
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
thanasis.papaioannou@epfl.ch

Hoyoung Jeung
SAP Research
Brisbane, Australia
hoyoung.jeung@sap.com

Karl Aberer
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
karl.aberer@epfl.ch

Abstract In recent years, due to the proliferation of sensor networks, there has been a genuine need of researching techniques for sensor data acquisition and management. To this end, a large number of techniques have emerged that advocate *model-based* sensor data acquisition and management. These techniques use mathematical models for performing various, day-to-day tasks involved in managing sensor data. In this chapter, we survey the state-of-the-art techniques for model-based sensor data acquisition and management. We start by discussing the techniques for acquiring sensor data. We, then, discuss the application of models in sensor data cleaning; followed by a discussion on model-based meth-

ods for querying sensor data. Lastly, we survey model-based methods proposed for data compression and synopsis generation.

Keywords: model-based techniques, data acquisition, query processing, data cleaning, data compression.

1. Introduction

In recent years, there has been tremendous growth in the data generated by sensor networks. Equivalently, there are pertinent techniques proposed in recent literature for efficiently acquiring and managing sensor data. One important category of techniques that have received significant attention are the model-based techniques. These techniques use mathematical models for solving various problems pertaining to sensor data acquisition and management. In this chapter, we survey a large number of state-of-the-art model-based techniques for sensor data acquisition and management. Model-based techniques use various types of models: statistical, signal processing, regression-based, machine learning, probabilistic, or time series. These models serve various purposes in sensor data acquisition and management.

It is well-known that many physical attributes, like, ambient temperature or relative humidity, vary smoothly. As a result of this smoothness, sensor data typically exhibits the following properties: (a) it is continuous (although we only have a finite number of samples), (b) it has finite energy or it is band-limited, (c) it exhibits Markovian behavior or the value at a time instant depends only on the value at a previous time instant. Most model-based techniques exploit these properties for efficiently performing various tasks related to sensor data acquisition and management.

In this chapter, we consider four broad categories of sensor data management tasks: data acquisition, data cleaning, query processing, and data compression. These tasks are pictorially summarized in the toy example shown in Figure 1.1. From Figure 1.1, it is interesting to note how a single type of model (linear) can be used for performing these various tasks. For each task considered in this chapter, we extensively discuss various, well-researched model-based solutions. Following is the detailed discussion on the sensor data management tasks covered in this chapter:

- **Data Acquisition:** Sensor data acquisition is the task responsible for efficiently acquiring samples from the sensors in a sensor network. The primary objective of the sensor data acquisition task is to attain energy efficiency. This objective is driven by the fact that most sensors are battery-powered and are located in inaccessible locations (e.g., environmental monitoring sensors are sometimes located at high altitudes and are surrounded by highly inaccessible terrains). In the literature, there

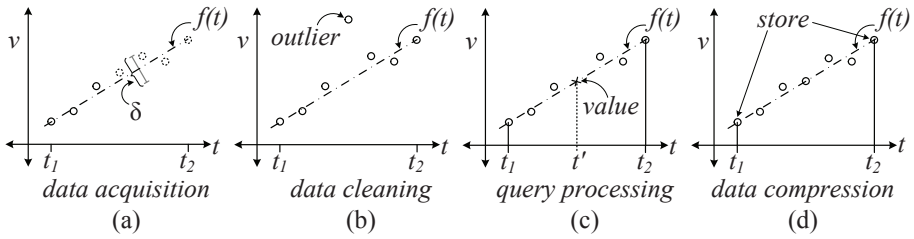


Figure 1.1: Various tasks performed by models-based techniques. (a) to improve acquisitional efficiency, a function is fitted to the first three sensor values, and the remaining values (shown dotted) are not acquired, since they are within a threshold δ, (b) data is cleaned by identifying outliers after fitting a linear model, (c) a query requesting the value at time t' can be answered using interpolation, (d) only the first and the last sensor value can be stored as compressed representation of the sensor values.

are two major types of acquisition approaches: pull-based and push-based. In the pull-based approach, data is only acquired at a user-defined frequency of acquisition. On the other hand, in the push-based approach, the sensors and the base station agree on an expected behavior; sensors only send data to the base station if the sensor values deviate from such expected behavior. In this chapter, we cover a representative collection of model-based sensor data acquisition approaches [2, 12, 17, 16, 18, 27, 28, 41, 66].

- **Data Cleaning:** The data obtained from the sensors is often erroneous. Erroneous sensor values are mainly generated due to the following reasons: (a) intermittent loss of communication with the sensor, (b) sensor's battery is discharged, (c) other types of sensor failures, for example, snow accumulation on the sensor, etc. Model-based approaches for data cleaning often use a model to infer the most probable sensor value. Then the raw sensor value is marked erroneous or outlier if the raw sensor value deviates significantly from the inferred sensor value. Another important approach for data cleaning is known as declarative data cleaning [32, 46, 54]. In this approach, the user registers SQL-like queries that define constraints over the sensor values. Sensor values are marked as outliers when these constraints are violated. In addition to these methods, we also discuss many other data cleaning approaches [31, 73, 23, 21, 52, 65]

- **Query Processing:** Obtaining desired answers, by processing queries is another important aspect in sensor data management. In this chapter,

we discuss the most significant model-based techniques for query processing. One of the objectives of these techniques is to process queries by accessing/generating minimal amount of data [64, 5]. Model-based methods that access/generate minimal data, and also handle missing values in data, use models for creating an abstraction layer over the sensor network [18, 33]. Other approaches model the sensor values by a hidden Markov model (HMM), associating state variables to the sensor values. It, then, becomes efficient to process queries over the state variables, which are less in number as compared to the sensor values [5]. Furthermore, there are approaches that use dynamic probabilistic models (DPMs) for modeling spatio-temporal evolution of the sensor data [33, 29]. In these approaches, the estimated DPMs are used for query processing.

- **Data Compression:** It is well-known that large quantity of sensor data is being generated by every hour. Therefore, eliminating redundancy by compressing sensor data for various purposes (like, storage, query processing, etc.) becomes one of the most challenging tasks. Model-based sensor data compression proposes a large number of techniques, mainly from the signal processing literature, for this task [1, 72, 22, 53, 7]. Many approaches assume that the user provides an accuracy bound, and based on this bound the sensor data is approximated, resulting in compressed representations of the data [24]. A large number of other techniques exploit the fact that sensor data is often correlated; thus, this correlation can be used for approximating one data stream with another [24, 67, 49, 3].

This chapter is organized as follows. In Section 2, we define the preliminaries that are assumed in the rest of the chapter, followed by a discussion of important techniques for sensor data acquisition. In Section 3, we survey model-based sensor data cleaning techniques, both on-line and archival. Model-based query processing techniques are discussed in Section 4. In Section 5, model-based compression techniques are surveyed. At the end, Section 6 contains a summary of the chapter along with conclusions.

2. Model-Based Sensor Data Acquisition

In this section, we discuss various techniques for model-based[1] sensor data acquisition. Particularly, we discuss pull- and push-based sensor data acquisi-

[1]We use *model-based* and *model-driven* interchangeably.

tion methods. In general, model-based sensor data acquisition techniques are designed for tackling the following challenges:

Energy Consumption: Obtaining values from a sensor requires high amount of energy. In contrast, since most sensors are battery-powered, they have limited energy resources. Thus, a challenging task is to minimize the number of samples obtained from the sensors. Here, models are used for selecting sensors, such that user queries can be answered with reasonable accuracy using the data acquired from the selected sensors [2, 17, 16, 27, 28].

Communication Cost: Another energy-intensive task is to communicate the sensed values to the base station. There are, therefore, several model-based techniques proposed in the literature for reducing the communication cost, and maintaining the accuracy of the sensed values [41, 18, 66, 12].

Table 1.1: Summary of notations.

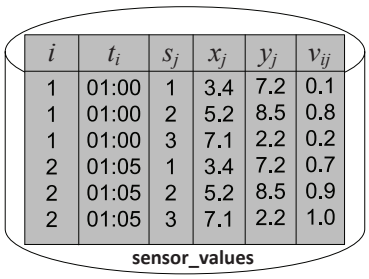
Symbol	Description
$\mathbf{S}$	Sensor network consisting of sensors s_j, where $j = (1, \ldots, m)$.
s_j	Sensor identifier for a sensor in $\mathbf{S}$.
v_{ij}	Sensor value observed by the sensor s_j at time t_i, such that $v_{ij} \in \mathbb{R}$.
v_i	Row vector of all the sensor values observed at time t_i, such that $v_i \in \mathbb{R}^m$.
V_{ij}	Random variable associated with the sensor value v_{ij}.

2.1 Preliminaries

We start by describing our model of a sensor network and establishing the notation that is utilized in the rest of the chapter. The sensor network considered in this chapter consists of a set of stationary sensors $\mathbf{S} = \{s_j | 1 \leq j \leq m\}$. The value sensed by a sensor s_j at time t_i is denoted as v_{ij}, which is a real number. In addition, note that we use s_j, where $j = (1, \ldots, m)$, as sensor identifiers. In certain cases the sampling interval could be uniform, that is, $t_{i+1} - t_i$ is same for all the values of $i \geq 1$. In such cases, the time stamps t_i become irrelevant, and it is sufficient to use only the index i for denoting the time axis.

In this chapter, we assume a scenario where the sensors are used for environmental monitoring. We assume that all the sensors are monitoring/sensing only one environmental attribute, such as, ambient temperature[2]. As discussed in Section 1, we assume that the environmental attribute we monitor is sufficiently smooth and continuous. If necessary for rendering the discussion complete and convenient, we will introduce other attributes being monitored by the sensors. But, in most cases, we restrict ourselves to using only ambient

[2]We use *ambient temperature* and *temperature* interchangeably.



i	t_i	s_j	x_j	y_j	v_{ij}
1	01:00	1	3.4	7.2	0.1
1	01:00	2	5.2	8.5	0.8
1	01:00	3	7.1	2.2	0.2
2	01:05	1	3.4	7.2	0.7
2	01:05	2	5.2	8.5	0.9
2	01:05	3	7.1	2.2	1.0

sensor_values

Figure 1.2: Database table containing the sensor values. The position of the sensor s_j is denoted as (x_j, y_j). Since the sensors are assumed to be stationary, the position can also be stored using a foreign-key relationship between s_j and (x_j, y_j). But, for simplicity, we assume that the `sensor_values` table is in a denormalized form.

temperature. Figure 1.2 shows a conceptual representation of the sensor values in a form of a database table, denoted as `sensor_values`.

2.2 The Sensor Data Acquisition Query

Sensor data acquisition can be defined as the processes of creating and continuously maintaining the `sensor_values` table. In existing literature, naturally, many techniques have been proposed for creating and maintaining the `sensor_values` table. We shall discuss these techniques briefly, describing their important characteristics and differences with other techniques. We use the sensor data acquisition query shown in Query 1.1 for discussing how different sensor data acquisition approaches process such a query. Query 1.1 is a query that triggers the acquisition of ten sensor values v_{ij} from the sensors s_j at a sampling interval of one second. Moreover, Query 1.1, is the typical sensor data acquisition query that is used by many methods for collecting sensor data.

```
SELECT s_j, v_ij FROM sensor_values SAMPLE INTERVAL 1s FOR 10s
```

Query 1.1: Sensor data acquisition query.

2.3 Pull-Based Data Acquisition

Broadly, there are two major approaches for data acquisition: pull-based and push-based (refer Figure 1.3). In the pull-based sensor data acquisition approach, the user defines the interval and frequency of data acquisition. Pull-based systems only follow the user's requirements, and pull sensor values as defined by the queries. For example, using the `SAMPLE INTERVAL` clause

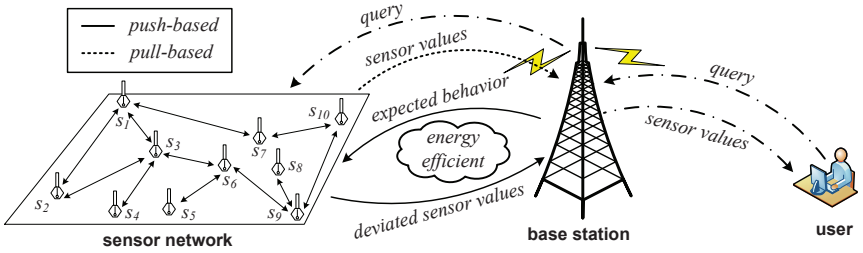


Figure 1.3: Push- and pull-based methods for sensor data acquisition.

of Query 1.1, users can specify the number of samples and the frequency at which the samples should be acquired.

In-Network Data Acquisition. This approach of sensor data acquisition is proposed by TinyDB [45, 44, 43], Cougar [69] and TiNA [58]. These approaches tightly link query processing and sensor data acquisition. Due to the lack of space, we shall only discuss TinyDB in this subsection.

TinyDB refers to its in-network query processing paradigm as *Acquisitional Query Processing* (ACQP). Let us start by discussing how ACQP processes Query 1.1. The result of Query 1.1 is similar to the table shown in Figure 1.2. The only difference, as compared to Figure 1.2, is that the result of Query 1.1 contains $10 \times m$ rows. The naïve method of executing Query 1.1 is to simultaneously poll each sensor for its value at the sampling interval and for the duration specified by the query. This method may not work due to limited range of radio communication between individual sensors and the base station.

Data Acquisition using Semantic Overlays: TinyDB proposes a tree-based overlay that is constructed using the sensors $\mathbf{S}$. This tree-based overlay is used for aggregating the query results from the leaf nodes to the root node. The overlay network is especially built for efficient data acquisition and query processing. TinyDB refers to its tree-based overlay network as *Semantic Routing Trees* (SRTs). A SRT is constructed by flooding the sensor network with the *SRT build request*. This request includes the attribute (ambient temperature), over which the SRT should be constructed. Each sensor s_j, which receives the build request, has several choices for choosing its parent: (a) if s_j has no children, which is equivalent to saying that no other sensor has chosen s_j as its parent, then s_j chooses another sensor as its parent and sends its current value v_{ij} to the chosen parent in a *parent selection message*, or (b) if s_j has children, it sends a parent selection message to its parent indicating the range of ambient temperature values that its children are covering. In addition, it locally stores the ambient temperature values from its children along with their sensor identifiers.

Next, when Query 1.1 is presented to the root node of the SRT, it forwards the query to its children and prepares for receiving the results. At the same time, the root node also starts processing the query locally (refer Figure 1.4). The same procedure is followed by all the intermediate sensors in the SRT. A sensor that does not have any children, processes the query and forwards the value of v_{ij} to its parent. All the collected sensor values v_{ij} are finally forwarded to the root node, and then to the user, as a result of the query. This completes the processing of the sensor data acquisition query (Query 1.1). The SRT, moreover, can also be used for optimally processing aggregation, threshold, and event based queries. We shall return to this point later in Section 4.1.

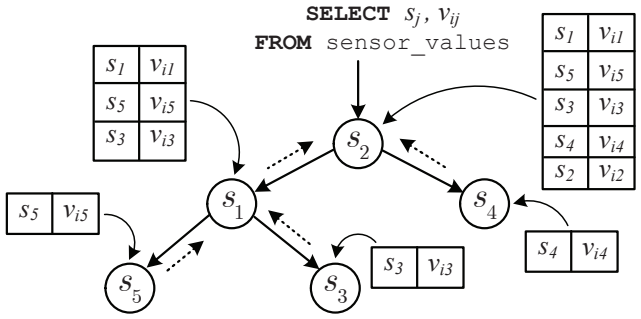


Figure 1.4: Toy example of a Semantic Routing Tree (SRT) and Acquisitional Query Processing (ACQP) over a sensor network with five sensors. Dotted arrows indicate the direction of query response. A given sensor appends its identifier s_j and value v_{ij} to the partial result, which is available from its subtree.

Multi-Dimensional Gaussian Distributions. The Barbie-Q (BBQ) system [17, 16], on the other hand, employs multi-variate Gaussian distributions for sensor data acquisition. BBQ maintains a multi-dimensional Gaussian probability distribution over all the sensors in $\mathbf{S}$. Data is acquired only as much as it is required to maintain such a distribution. Sensor data acquisition queries specify certain confidence that they require in the acquired data. If the confidence requirement cannot be satisfied, then more data is acquired from the sensors, and the Gaussian distribution is updated to satisfy the confidence requirements. The BBQ system models the sensor values using a multi-variate Gaussian probability density function (pdf) denoted as $p(V_{i1}, V_{i2}, \ldots, V_{im})$, where $V_{i1}, V_{i2}, \ldots, V_{im}$ are the random variables associated with the sensor values $v_{i1}, v_{i2}, \ldots, v_{im}$ respectively. This pdf assigns a probability for each possible assignment of the sensor values v_{ij}. Now, let us discuss how the BBQ system processes Query 1.1.

In BBQ, the inferred sensor value of sensor s_j, at each time t_i, is defined as the mean value of V_{ij}, and is denoted as $\bar{v}_{ij}$. For example, at time t_1, the inferred sensor values of the ambient temperature are $\bar{v}_{11}, \bar{v}_{12}, \ldots, \bar{v}_{1m}$. The BBQ system assumes that queries, like Query 1.1, provide two additional constraints: (i) error bound ϵ, for the values $\bar{v}_{ij}$, and (ii) the confidence $1 - \delta$ with which the error bound should be satisfied. Admittedly, these additional constraints are for controlling the quality of the query response.

Suppose, we already have a pdf before the first time instance t_1, then the confidence of the sensor value v_{1j} is defined as the probability of the random variable V_{1j} lying in between $\bar{v}_{1j} - \epsilon$ and $\bar{v}_{1j} + \epsilon$, and is denoted as $P(V_{1j} \in [\bar{v}_{1j} - \epsilon, \bar{v}_{1j} + \epsilon])$. If the confidence is greater than $1 - \delta$, then we can provide a probably approximately correct value for the temperature, without spending energy in obtaining a sample from sensor s_j. On the other hand, if a sensor's confidence is less than $1 - \delta$, then we should obtain one or more samples from the sensor (or other correlated sensors), such that the confidence bound is satisfied. In fact, it is clear that there could be potentially many sensors for which the confidence bound may not hold.

As a solution to this problem, the BBQ system proposes a procedure to chose the sensors for obtaining sensor values, such that the confidence bound specified by the query is satisfied. First, the BBQ system samples from all the sensors $\mathbf{S}$ at time t_1, then it computes the confidence $\mathcal{B}_j(\mathbf{S})$ that it has in a sensor s_j as follows:

$$\mathcal{B}_j(\mathbf{S}) = P(V_{1j} \in [\bar{v}_{1j} - \epsilon, \bar{v}_{1j} + \epsilon] | v_1), \tag{1.1}$$

where $v_1 = (v_{11}, v_{12}, \ldots, v_{1m})$ is the row vector of all the sensor values at time t_1. Second, for choosing sensors to sample, the BBQ system poses an optimization problem of the following form:

$$\min_{\mathbf{S}_o \subseteq \mathbf{S} \text{ and } \mathcal{B}(\mathbf{S}_o) \geq 1-\delta.} \mathcal{C}(\mathbf{S}_o), \tag{1.2}$$

where $\mathbf{S}_o$ is the subset of sensors that will be chosen for sampling, $\mathcal{C}(\mathbf{S}_o)$ and $\mathcal{B}(\mathbf{S}_o) = \frac{1}{|\mathbf{S}_o|} \sum_{j:s_j \in \mathbf{S}_o} \mathcal{B}_j(\mathbf{S})$ are respectively the total cost (or energy required) and average confidence for sampling sensors $\mathbf{S}_o$. Since the problem in Eq. (1.2) is NP-hard, BBQ proposes a greedy solution to solve this problem. Details of this greedy algorithm can be found in [17]. By executing the proposed greedy algorithm, BBQ selects the sensors for sampling, then it updates the Gaussian distribution, and returns the mean values $\bar{v}_{11}, \bar{v}_{12}, \ldots, \bar{v}_{1m}$. These mean values represent the inferred values of the sensors at time t_1. This operation when performed ten times at an interval of one second generates the result of the sensor data acquisition query (Query 1.1).

2.4 Push-Based Data Acquisition

Both, TinyDB and BBQ, are pull-based in nature: in these systems the central server/base station decides when to acquire sensor values from the sensors. On the other hand, in push-based approaches, the sensors autonomously decide when to communicate sensor values to the base station (refer Figure 1.3). Here, the base station and the sensors agree on an expected behavior of the sensor values, which is expressed as a model. If the sensor values deviate from their expected behavior, then the sensors communicate only the deviated values to the base station.

PRESTO. The Predictive Storage (PRESTO) [41] system is an example of the push-based data acquisition approach. One of the main arguments that PRESTO makes against pull-based approaches is that due to the pull strategy, such approaches will be unable to observe any unusual or interesting patterns between any two pull requests. Moreover, increasing the pull frequency for better detection of such patterns, increases the overall energy consumption of the system.

The PRESTO system contains two main components: PRESTO proxies and PRESTO sensors. As compared to the PRESTO sensors, the PRESTO proxies have higher computational capability and storage resources. The task of the proxies is to gather data from the PRESTO sensors and to answer queries posed by the user. The PRESTO sensors are assumed to be battery-powered and remotely located. Their task is to sense the data and transmit it to PRESTO proxies, while archiving some of it locally on flash memory.

Now, let us discuss how PRESTO processes the sensor data acquisition query (Query 1.1). For answering such a query, the PRESTO proxies always maintain a time-series prediction model. Specifically, PRESTO maintains a seasonal ARIMA (SARIMA) model [60] of the following form for each sensor:

$$v_{ij} = v_{(i-1)j} + v_{(i-L)j} - v_{(i-L-1)j} + \theta e_{i-1} - \Theta e_{i-L} + \theta\Theta e_{i-L-1}, \quad (1.3)$$

where θ and Θ are parameters of the SARIMA model, e_i are the prediction errors and L is known as the seasonal period. For example, while monitoring temperature, L could be set to one day, indicating that the current temperature (v_{ij}) is related to the temperature yesterday at the same time ($v_{(i-L)j}$) and a previous time instant ($v_{(i-L-1)j}$). In short, the seasonal period L allows us to model the periodicity that is inherent in certain types of data.

In the PRESTO system the proxies estimate the parameters of the model given in Eq. (1.3), and then transmit these parameters to individual PRESTO sensors. The PRESTO sensors use these models to predict the sensor value $\hat{v}_{ij}$, and only transmit the raw sensor value v_{ij} to the proxies when the absolute difference between the predicted sensor value and the raw sensor value is greater

than a user-defined threshold δ. This task can be summarized as follows:

$$|v_{ij} - \hat{v}_{ij}| > \delta, \quad \text{transmit } v_{ij} \text{ to proxy.} \tag{1.4}$$

The PRESTO proxy also provides a confidence interval for each predicted value it computes using the SARIMA model. Like BBQ (refer Section 2.3.0), this confidence interval can also be used for query processing, since it represents an error bound on the predicted sensor value. Similar to BBQ, the PRESTO proxy queries the PRESTO sensors only when the desired confidence interval, specified by the query, could not be satisfied with the values stored at the PRESTO proxy. In most cases, the values stored at the proxy can be used for query processing, without acquiring any further values from the PRESTO sensors. The only difference between PRESTO and BBQ is that, PRESTO uses a different measure of confidence as compared to BBQ. Further details of this confidence interval can be found in [41].

Ken. For reducing the communication cost, the Ken [12] framework employs a similar strategy as PRESTO. Although there is a key difference between Ken and PRESTO. PRESTO uses a SARIMA model; this model only takes into account temporal correlations. On the other hand, Ken uses a dynamic probabilistic model that takes into account spatial and temporal correlations in the data. Since a large quantity of sensor data is correlated spatially, and not only temporally, Ken derives advantage from such spatio-temporal correlation.

The Ken framework has two types of entities, *sink* and *source*. Their functionalities and capabilities are similar to the PRESTO proxy and the PRESTO sensor respectively. The only difference is that the PRESTO sensor only represents a single sensor, but a source could include more than one sensor or a sensor network. The sink is the base station to which the sensor values v_{ij} are communicated by the source (refer Figure 1.3).

The fundamental idea behind Ken is that both, source and sink, maintain the same dynamic probabilistic model of data evolution. The source only communicates with the sink when the raw sensor values deviate beyond a certain bound, as compared to the predictions from the dynamic probabilistic model. In the meantime, the sink uses the sensor values predicted by the model.

As discussed before, Ken uses a dynamic probabilistic model that considers spatio-temporal correlations. Particularly, its dynamic probabilistic model computes the following pdf at the source:

$$p(V_{(i+1)1}, \ldots, V_{(i+1)m} | v_1, \ldots, v_i) = \int p(V_{(i+1)1}, \ldots, V_{(i+1)m} | V_{i1}, \ldots, V_{im}) \\ p(V_{i1}, \ldots, V_{im} | v_1, \ldots, v_i) dV_{i1} \ldots dV_{im}. \tag{1.5}$$

This pdf is computed using the observations that have been communicated to the sink; the values that are not communicated to the sink are ignored by the source, since they do not affect the model at the sink. Next, each sensor contained in the source computes the expected sensor value using Eq. (1.5) as follows:

$$\bar{v}_{(i+1)j} = \int V_{(i+1)j} p(V_{(i+1)1}, \ldots, V_{(i+1)m}) dV_{(i+1)1} \ldots dV_{(i+1)m}. \quad (1.6)$$

The source does not communicate with the sink if $|\bar{v}_{(i+1)j} - v_{(i+1)j}| < \delta$, where δ is a user-defined threshold. If this condition is not satisfied, the source communicates to the sink the smallest number of sensor values, such that the δ threshold would be satisfied. Similarly, if the sink does not receive any sensor values from the source, it computes the expected sensor values $\bar{v}_{(i+1)j}$ and uses them as an approximation to the raw sensor values. If the sink receives a few sensor values form the source, then, before computing the expected values, the sink updates its dynamic probabilistic model.

A Generic Push-Based Approach. The last push-based approach that we will survey is a generalized version of other push-based approaches [38]. This approach is proposed by Silberstein *et al.* [61]. Like other push-based approaches, the base station and the sensor network agree on an expected behavior, and, as usual, the sensor network reports values only when there is a substantial deviation from the agreed behavior. But, unlike other approaches, the definition of expected behavior proposed in [61] is more generic, and is not limited to a threshold δ.

In this approach a sensor can either be an updater (one who acquires or forwards sensor values) or an observer (one who receives sensor values). A sensor node can be both, updater and observer, depending on whether it is on the boundary of the sensor network or an intermediate node. The updaters and the observers maintain a model encoding function f_{enc} and a decoding function f_{dec}. These model encoding/decoding functions define the agreed behavior of the sensor values. The updater uses the encoding function to encode the sensor value v_{ij} into a transmission message g_{ij}, and transmits it to the observer.

The observer, then, uses the decoding function f_{dec} to decode the message g_{ij} and construct $\hat{v}_{ij}$. If the observer finds that v_{ij} has not changed significantly, as defined by the encoding function, then the observer transmits a `null` symbol. A `null` symbol indicates that the sensor value is *suppressed* by the observer. Following is an example of the encoding and decoding functions [61]:

$$f_{enc}(v_{ij}, v_{i'j}) = \begin{cases} g_{ij} = v_{ij} - v_{i'j}, & \text{if } |v_{ij} - v_{i'j}| > \delta; \\ g_{ij} = \texttt{null}, & \text{otherwise.} \end{cases} \quad (1.7)$$

$$f_{dec}(g_{ij}, \hat{v}_{(i-1)j}) = \begin{cases} \hat{v}_{(i-1)j} + g_{ij}, & \text{if } g_{ij} \neq \texttt{null}; \\ \hat{v}_{(i-1)j}, & \text{if } g_{ij} = \texttt{null}. \end{cases} \quad (1.8)$$

In the above example, the encoding function f_{enc} computes the difference between the model predicted sensor value $v_{i'j}$ and the raw sensor value v_{ij}. Then, this difference is transmitted to the observer only if it is greater than δ, otherwise the `null` symbol is transmitted. The decoding function f_{dec} decodes the sensor value $\hat{v}_{(i-1)j}$ using the message g_{ij}.

The encoding and decoding functions in the above example are purposefully chosen to demonstrate how the δ threshold approach can be replicated by these functions. More elaborate definitions of these functions, which are used for encoding complicated behavior, can be found in [61].

3. Model-Based Sensor Data Cleaning

A well-known characteristic of sensor data is that it is uncertain and erroneous. This is due to the fact that sensors often operate with discharged batteries, network failures, and imprecision. Other factors, such as low-cost sensors, freezing or heating of the casing or measurement device, accumulation of dirt, mechanical failure or vandalism (from humans or animals) heavily affect the quality of the sensor data [31, 73, 23]. This may cause a significant problem with respect to data utilization, since applications using erroneous data may yield unsound results. For example, scientific applications that perform prediction tasks using observation data obtained from cheap and less-reliable sensors may produce inaccurate prediction results.

To address this problem, it is essential to detect and correct erroneous values in sensor data by employing *data cleaning*. The data cleaning task typically involves complex processing of data [71, 30]. In particular, it becomes more difficult for sensor data, since true sensor values corresponding to erroneous data values are generally unobservable. This has led to a new approach – *model-based data cleaning*. In this approach, the most probable sensor values are inferred using well-established models, and then anomalies are detected by comparing raw sensor values with the corresponding inferred sensor values. In the literature there are a variety of suggestions for model-based approaches for sensor data cleaning. This section describes the key mechanisms proposed by these approaches, particularly focusing on the models used in the data cleaning process.

3.1 Overview of Sensor Data Cleaning System

A system for cleaning sensor data generally consists of four major components: *user interface, stream processing engine, anomaly detector*, and *data storage* (refer Figure 1.5). In the following, we describe each component.

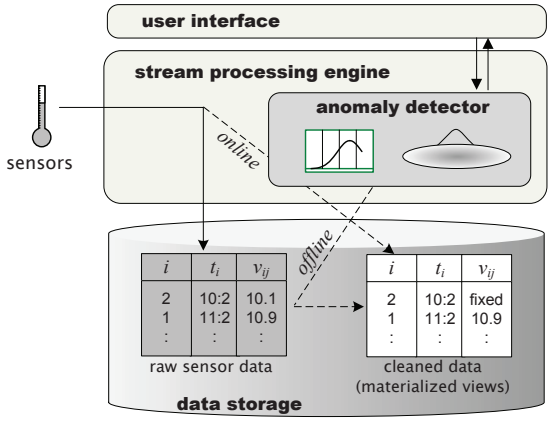


Figure 1.5: Architecture of sensor data cleaning system.

User Interface: The user interface plays two roles in the data cleaning process. First, it takes all necessary inputs from users to perform data cleaning, e.g., name of sensor data and parameter settings for models. Second, the results of data cleaning, such as 'dirty' sensor values captured by the anomaly detector, are presented using graphs and tables, so that users can confirm whether each candidate of such dirty values is an actual error. The confirmed results are then stored to (or removed from) the underlying data storage or materialized views.

Anomaly Detector: The anomaly detector is a core component in sensor data cleaning. It uses models for detecting abnormal data values. The anomaly detector works in online as well as offline mode. In the online mode, whenever a new sensor value is delivered to the stream processing engine, the dirtiness of this value is investigated and the errors are filtered out instantly. In the offline mode, the data is cleaned periodically, for instance, once per day. In the following subsections, we will review popular models used for online anomaly detection.

Stream Processing Engine: The stream processing engine maintains streaming sensor data, while serving as a main platform where the other system components can cooperatively perform data cleaning. The anomaly detector is typically embedded into the stream processing engine, it may also be implemented as a built-in function on database systems.

Data Storage: The data storage maintains not only sensor values, but also the corresponding cleaned data, typically in materialized views. This is because applications on sensor networks often need to repeatedly perform data cleaning over the same data using different parameter settings for the models, especially when the previous parameter settings turn out to be inappropriate later. There-

fore, it is important for the system to store cleaned data in database views without changing the original data, so that data cleaning can be performed again at any point of time (or time interval) as necessary.

3.2 Models for Sensor Data Cleaning

This subsection reviews popular models that are widely used in the sensor data cleaning process.

Regression Models. As sensor values are a representation of physical processes, it is naturally possible to uncover the following properties: continuity of the sampling processes and correlations between different sampling processes. In principle, regression-based models exploit either or both of these properties. Specifically, they first compute the dependency from one variable (e.g., time) to another (e.g., sensor value), and then consider the regression curves as standards over which the inferred sensor values reside. The two most popular regression-based approaches use polynomial and Chebyshev regression for cleaning sensor values.

Polynomial Regression: Polynomial regression finds the best-fitting curve that minimizes the total difference between the curve and each raw sensor value v_{ij} at time t_i. Given a degree d, polynomial regression is formally defined as:

$$v_{ij} = c + \alpha_1 \cdot t_i + \cdots + \alpha_d \cdot t_i^d, \tag{1.9}$$

where c is a constant and $\alpha_1, \ldots, \alpha_d$ are regression coefficients.

Polynomial regression with high degrees approximate given time series with more sophisticated curves, resulting in theoretically more accurate description of the raw sensor values. Practically, however, low-degree polynomials, such as constant ($d = 0$) and linear ($d = 1$), also perform satisfactorily. In addition, low-degree polynomials can be more efficiently constructed as compared to high-degree polynomials. A (weighted) moving average model [73] is also regarded as a polynomial regression.

Chebyshev Regression: Chebyshev regression is another popular model class for fitting sensor values, since they can quickly compute near-optimal approximations for given time series. Suppose that time values t_i vary within a range $[\min(t_i), \max(t_i)]$. We, then, obtain normalized time values t_i' within a range $[-1, 1]$, by using the following transformation function $f(t_i)$ and its inverse transformation function $f^{-1}(t_i')$ as follows:

$$f(t_i) = \left(t_i - \frac{\max(t_i) + \min(t_i)}{2} \right) \cdot \frac{2}{\max(t_i) - \min(t_i)}, \tag{1.10}$$

$$f^{-1}(t_i') = \left(t_i' \cdot \frac{\max(t_i) - \min(t_i)}{2} \right) + \frac{\max(t_i) + \min(t_i)}{2}. \tag{1.11}$$

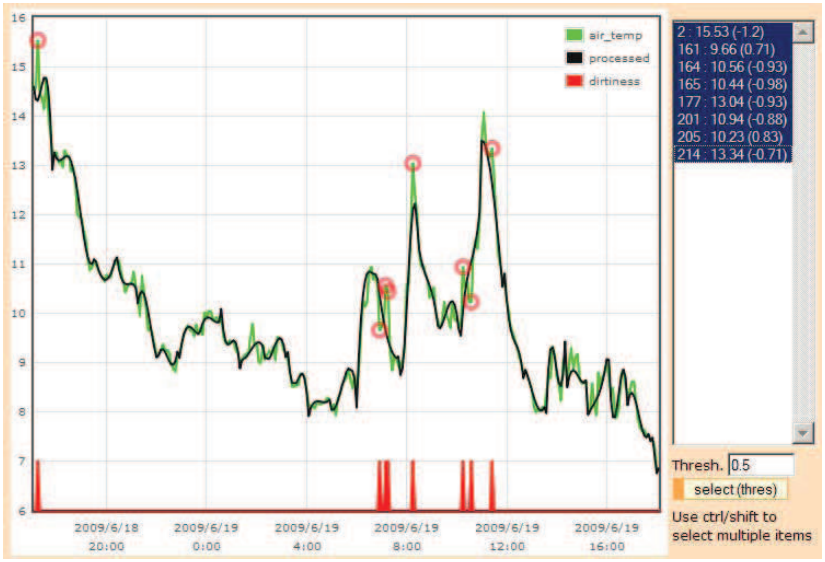


Figure 1.6: Detected anomalies based on 2-degree Chebyshev regression.

Next, given a degree d, Chebyshev polynomial is defined as:

$$v_{ij} = f^{-1}(\cos(d \cdot \cos^{-1}(f(t_i)))).$$

Figure 1.6 illustrates a data cleaning process using degree-2 Chebyshev polynomials. Here, the raw sensor values are plotted as green curves, while the inferred values, obtained by fitting a Chebyshev polynomials, are overlaid by black curves. The anomaly points are then indicated by the underlying red histograms as well as red circles.

Probabilistic Models. In sensor data cleaning, inferring sensor values is perhaps the most important task, since systems can then detect and clean dirty sensor values by comparing raw sensor values with the corresponding inferred sensor values. Figure 1.7 shows an example of the data cleaning process using probabilistic models. At time $t_i = 6$, the probabilistic model infers a probability distribution using the previous values $v_{2j}, \ldots, v_{5j}$ in the sliding window. The expected value $\bar{v}_{6j}$ (e.g., the mean of the Gaussian distribution in the future) is then considered as the inferred sensor value for sensor s_j.

Next, the anomaly detector checks whether the raw sensor value v_{6j} resides within a reasonably accurate area. This is done in order to check whether the value is *normal*. For instance, the 3σ range can cover 99.7 % of the density in the figure, where v_{6j} is supposed to appear. Thus, the data cleaning process can consider that v_{6j} is not an error. At $t_i = 7$, the window slides and now

contains raw sensor values $v_{3j}, \ldots, v_{6j}$. By repeating the same process, the anomaly detector finds v_{7j} resides out of the error bound (3σ range) in the inferred probability distribution, and is identified as an anomaly [57].

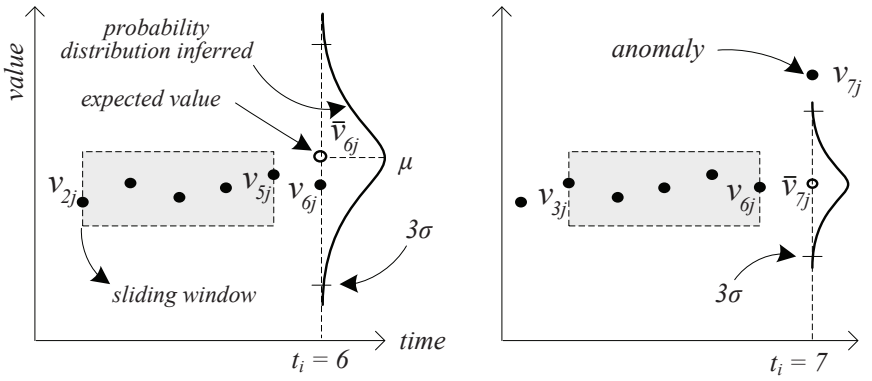


Figure 1.7: An example of data cleaning based on a probabilistic model.

A vast body of research work has utilized probabilistic models for computing inferred values. The ***Kalman filter*** is perhaps one of the most common probabilistic models to compute inferred values corresponding to raw sensor values. The Kalman filter is a stochastic and recursive data filtering algorithm that models the raw sensor value v_{ij} as a function of its previous value (or state) $v_{(i-1)j}$ as follows:

$$v_{ij} = Av_{(i-1)j} + Bu_i + w_i,$$

where A and B are matrices defining the state transition from time t_{i-1} to time t_i, u_i is the time-varying input at time t_i, and w_i is the process noise drawn from a zero mean multi-variate Gaussian distribution. In [63], the Kalman filter is used for detecting erroneous values, as well as inter/extrapolating missing sensor values. Jain *et al.* [29] also use the Kalman filter for filtering possible dirty values.

Similarly, Elnahrawy and Nath [21] proposed to use Bayes' theorem to estimate a probability distribution P_{ij} at time t_i from raw sensor values v_{ij}, and associate them with an error model, typically a normal distribution. Built on the same principle, a neuro-fuzzy regression model [52] and a belief propagation model based on Markov chains [13] were used to identify anomalies. Tran *et al.* [65] propose a method to infer missing or erroneous values in RFID data. All the techniques for inferring sensor values also enable quality-aware processing of sensor data streams [36, 37], since inferred sensor values can serve as the bases for indicating the quality or precision of the raw sensor values.

Outlier Detection Models. An *outlier* is a sensor value that largely deviates from the other sensor values. Obviously, outlier detection is closely related to the process of sensor data cleaning. The outlier-detection techniques are well-categorized in the survey studies of [51, 8].

In particular, some of the outlier detection methods focus on sensor data [59, 71, 15]. Zhang *et al.* [71] offer an overview of such outlier detection techniques for sensor network applications. Deligiannakis *et al.* [15] consider correlation, extended Jaccard coefficients, and regression-based approximation for model-based data cleaning. Shen *et al.* [59] propose to use a histogram-based method to capture outliers. Subramaniam *et al.* [62] introduce distance- and density-based metrics that can identify outliers. In addition, the ORDEN system [23] detects polygonal outliers using the triangulated wireframe surface model.

3.3 Declarative Data Cleaning Approaches

From the perspective of using a data cleaning system, supporting a declarative interface is important since it allows users to easily control the system. This idea is reflected in a wide range of prior work that proposes SQL-like interfaces for data cleaning [32, 46, 54]. These proposals hide complicated mechanisms of data processing or model utilization from the users, and facilitate data cleaning in sensor network applications.

More specifically, Jeffery *et al.* [31, 32] divide the data cleaning process into five tasks: *Point, Smooth, Merge, Arbitrate*, and *Virtualize*. These tasks are then supported within a database system. For example, the SQL statement in Query 1.2 performs anomaly detection within a spatial granule by determining the average of the sensor values from different sensors in the same proximity group. Then, individual sensor values are rejected if they are outside of one standard deviation from the mean.

As another approach, Rao *et al.* [54] focus on a systemic solution, based on rewriting queries using a set of cleansing rules. Specifically, the system offers the rule grammar shown in Figure 1.8 to define and execute various data cleaning tasks. Unlike the prior relational database approaches, Mayfield *et al.*

```
DEFINE       [rule name]
ON           [table name]
FROM         [table name]
CLUSTER BY   [cluster key]
SEQUENCE BY  [sequence key]
AS           [pattern]
WHERE        [condition]
ACTION       [DELETE | MODIFY | KEEP]
```

Figure 1.8: An example of anomaly detection using a SQL statement.

```
SELECT spatial_granule, AVG(temp)
FROM data s [Range By 5 min]
     (SELECT spatial_granule, avg(temp) as avg,
     stdev(temp) as stdev
     FROM data [Range By 5 min]) as a
WHERE a.spatial_granule = s.spatial_granule
     AND a.avg + (2*a.stdev) < s.temp
     AND a.avg - (2*a.stdev) > s.temp
```

Query 1.2: An example of anomaly detection using a SQL statement.

[46] model data as a graph consisting of nodes and links. They, then, provide an SQL-based, declarative framework that enables data owners to specify or discover groups of attributes that are correlated, and apply statistical methods that validate and clean the sensor values using such dependencies.

4. Model-Based Query Processing

In this section we elaborate another important task in sensor data management – query processing. We primarily focus on in-network and centralized query processing approaches. We consider different queries assuming the sensor network described in Section 2.1, and then discuss how each approach processes these queries. In Section 2, however, we followed an approach where we chose a singe query (i.e., Query 1.1) and demonstrated how different techniques processed this query. On the contrary, in this section, we chose different queries for all the approaches, and then discuss these approaches along with the queries. We follow this procedure since, unlike Section 2, the assumptions made by each query processing technique are different. Thus, for highlighting the impact of these assumptions and simplifying the discussion, we select different queries for each approach.

4.1 In-Network Query Processing

In-network query processing first builds an overlay network (like, the SRT discussed in Section 2.3.0). Then, the overlay network is used for increasing the efficiency of aggregating sensor values and processing queries. For instance, while processing a threshold query, parent nodes send the query to the child nodes only when the query threshold condition overlaps with the range of sensor values contained in the child nodes, which is stored in the parent node's local memory.

Consider the threshold query given in Query 1.3. Query 1.3 requests the sensor identifiers of all the sensors that have sensed a temperature greater than 10°C at the current time instance. Before answering this query, we assume that we have already constructed a SRT as described in Section 2.2 (refer Fig-

ure 1.4). Query 1.3 is sent by the root node of the SRT to its children that are a part of the query response. The child nodes check whether the sensor value they have sensed is greater than 10°C. If the sensor value is greater than 10°C at a child node, then that child node appends its sensor identifier to the query response. The child node, then, forwards the query to its children and waits for their response. Once all the children of a particular node have responded, then that node forwards the response of its entire sub-tree to its parent. In the end, the root node receives all the sensor identifiers s_j that have recorded temperature greater than 10°C.

SELECT s_j **FROM** sensor_values **WHERE** $v_{ij} > 10^{\circ}\text{C}$ **AND** t_i == NOW()

Query 1.3: Return the sensor identifiers s_j where $v_{ij} > 10^{\circ}$C.

4.2 Model-Based Views

The MauveDB [18] approach proposes standard database views [19] as an abstraction layer for processing queries. These views are maintained in a form of a regression model; thus they are called *model-based* views. The main advantage of this approach is that the model-based view can be incrementally updated as fresh sensor values are obtained from the sensors. Furthermore, incremental updates is an attractive feature, since such updates are computationally efficient.

Before processing any queries in MauveDB, we have to first create a model-based view. The query for creating a model-based view is shown in Query 1.4. The model-based view created by this query is called `RegModel`. `RegModel` is a regression model in which the temperature is the dependent variable and the sensor position (x_j, y_j) is an independent variable (refer Figure 1.9). Note that `RegModel` is incrementally updated by MauveDB. At time t_1 values from sensors s_1, s_3 and at time t_2 the value from sensor s_2 are respectively used to update the view. The view update mechanism exploits the fact that regression functions can be updated. Further details regarding the update mechanism can be found in [18].

CREATE VIEW RegModel **AS FIT** v **OVER** x^2, xy, y^2, x, y
TRAINING_DATA SELECT x_j, y_j, v_{ij} **FROM** sensor_values
WHERE $t_i > t_{start}$ **AND** $t_i < t_{end}$

Query 1.4: Model-based view creation query.

Once this step is performed many types of queries can be evaluated using the `RegModel` view. For instance, consider Query 1.5. MauveDB evaluates this query by interpolating the value of temperature at fixed intervals on the

x- and y-axis; this is similar to database view materialization [19]. Then the positions (x, y) where the interpolated temperature value is greater than $10°\text{C}$ are returned.

Admittedly, although updating the model-based view is efficient, but for processing queries the model-based view should be materialized at a certain fixed set of points. This procedure produces a large amount of overhead when the number of independent variables is large, since it dramatically increases the number of points where the view should be materialized.

SELECT x, y **FROM** RegModel **WHERE** $v > 10°\text{C}$

Query 1.5: Querying model-based views.

4.3 Symbolic Query Evaluation

This approach is proposed by the FunctionDB [64] system. FunctionDB, like MauveDB, also interpolates the values of the dependent variable, and then uses the interpolated values for query processing.

As discussed before, the main problem with value interpolation is that the number of points, where the sensor values should be interpolated, increase dramatically as a function of the number of independent variables. As a solution to this problem, FunctionDB symbolically executes the filter (for example, the `WHERE` clause in Query 1.5) and obtains feasible regions of the independent variables. These feasible regions are the regions that include the exact response to the query, at the same time contain a significantly low number of values to interpolate. FunctionDB evaluates the query by interpolating values only in the feasible regions, followed by a straightforward evaluation of the query.

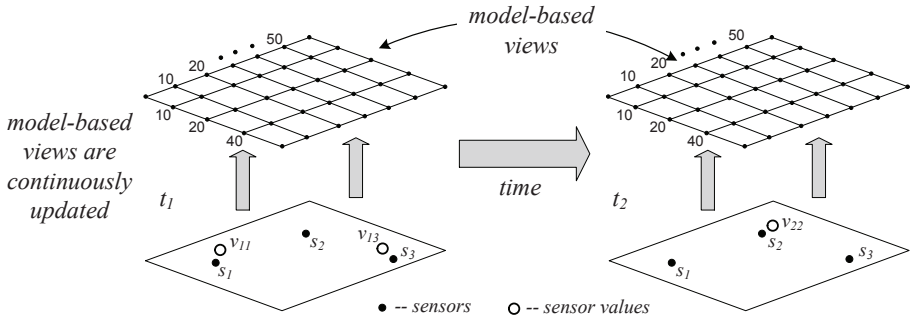


Figure 1.9: Example of the `RegModel` view with three sensors. `RegModel` is incrementally updated as new sensor values are acquired.

Moreover, FunctionDB treats the temperature of the sensor s_j as a continuous function of time $f_j(t)$, instead of treating it as discrete values sampled at time stamps t_i. An example of a query in the FunctionDB framework is given in Query 1.6. This query returns the time values t between t_{start} and t_{end} where the temperature of the sensor s_1 is greater than 10°C. Note that the time values t are not necessarily the time stamps t_i where a particular sensor value was recorded.

SELECT t **WHERE** $f_1(t) > 10^{\circ}\text{C}$ **AND** $t > t_{start}$ **AND** $t < t_{end}$ **GRID** `t 1s`

Query 1.6: Continuous threshold query.

For defining the values of the time axis t (or any continuous variable), FunctionDB proposes the `GRID` operator. The `GRID` operator specifies the interval at which the function $f_1(t)$ should be interpolated between time t_{start} and t_{end}. For instance, `GRID t 1s` indicates that the time axis should be interpolated at one second intervals between time t_{start} and t_{end}. To process Query 1.6, FunctionDB first symbolically executes the `WHERE` clause and obtains the feasible regions of the time axis (independent variable). Then, using the `GRID` operator, it generates time stamps T_I in the feasible regions. The sensor value is interpolated at the time stamps T_I using regression functions. Lastly, the query is processed on these interpolated values, and time stamps $T'_I \subseteq T_I$ where the temperature is greater than 10°C are returned.

4.4 Processing Queries over Uncertain Data

In this form of query processing the assumption is that sensor data is inherently uncertain. This uncertainty can arise due to various factors: loss of calibration over time, faulty sensors, unsuitable environmental conditions, low sensor accuracy, etc. Thus, the approaches that treat sensor data as uncertain, assume that each sensor value is associated with a random variable, and is drawn from a distribution. In this subsection, we discuss two such methods that model uncertain data by either a dynamic probabilistic model or a static probability distribution.

Dynamic Probabilistic Models. Dynamic probabilistic models (DPMs) are proposed for query processing in [33, 29]. These models continuously estimate a probability distribution. The estimated probability distribution is used for query processing. Mainly, there are two types of models that are frequently used for estimating dynamic probability distributions: particle filters and Kalman filters. Particle filters are generalized form of Kalman filters. Since we have already discussed Kalman filters in Section 3.2, here we will focus on particle filtering.

Consider a single sensor, say s_1, the particle filtering approach [4], at each time instant t_i, estimates and stores p weighted tuples $\{(w_{i1}^1, v_{i1}^1), \ldots, (w_{i1}^p, v_{i1}^p)\}$, where the weight w_{i1}^1 denotes the probability of v_{i1}^1 being the sensor value of the sensor s_1 at time t_i, and so on. An example of particle filtering is shown in the `pf_sensor_values` table in Figure 1.10.

Now, consider Query 1.7 that requests the average temperature $\mathsf{AVG}(v_{ij})$ between time t_{start} and t_{end}. To evaluate this query, we assume that we already have executed the particle filtering algorithm at each time instance t_i and have created the `pf_sensor_values` table. We, then, perform the following two operations:

1. For each time t_i between t_{start} and t_{end}, we compute the expected temperature $\bar{v}_{i1} = \sum_{l=1}^{p} w_{i1}^l \cdot v_{i1}^l$. The formal SQL syntax for computing the expected values using the `pf_sensor_values` table is as follows:

   SELECT $t_i, \sum_{l=1}^{p} w_{i1}^l \cdot v_{i1}^l$ **FROM** `pf_sensor_values` **WHERE** $t_i > t_{start}$ **AND** $t_i < t_{end}$ **GROUP BY** t_i

2. The final result is the average of all the $\bar{v}_{i1}$ that we computed in Step 1.

Essentially, the tuples $\{(w_{i1}^1, v_{i1}^1), \ldots, (w_{i1}^p, v_{i1}^p)\}$ represent a discretized pdf for the random variable V_{i1}. Moreover, the most challenging tasks in particle filtering are to continuously infer weights $w_{i1}^1, \ldots, w_{i1}^p$ and to select the optimal number of particles p, keeping in mind a particular scenario and type of data [4].

SELECT $\mathsf{AVG}(v_{i1})$ **FROM** `pf_sensor_values` **WHERE** $t > t_{start}$ **AND** $t < t_{end}$

Query 1.7: Compute the average temperature between time t_{start} and t_{end}.

i	t_i	s_j	x_j	y_j	p	v_{ij}^p	w^p
1	01:00	1	3.4	7.2	1	1.1	0.1
1	01:00	1	3.4	7.2	2	3.0	0.6
1	01:00	1	3.4	7.2	3	5.2	0.3
2	01:05	2	5.2	8.5	1	3.1	0.4
2	01:05	2	5.2	8.5	2	7.9	0.3
2	01:05	2	5.2	8.5	3	6.4	0.3

pf_sensor_values

Figure 1.10: Particle filtering stores p weighted sensor values for each time instance t_i.

Static Probabilistic Models. Cheng *et al.* [9–11] model the sensor value as obtained from an user-defined uncertainty range. For example, if the value of a temperature sensor is $15°$C, then the actual value could vary between $13°$C and $17°$C. Furthermore, the assumption is that the sensor value is drawn from a static probability distribution that has support over the uncertainty range.

Thus, for each sensor s_j we associate an uncertainty range between l_{ij} and u_{ij}, in which the actual sensor values can be found. In addition, the pdf of the sensor values of sensor s_j is denoted as $p_{ij}(v)$. Note that the pdf has non-zero support only between l_{ij} and u_{ij}. Consider a query that requests the average temperature of the sensors s_1 and s_2 at time t_i. Since the values of the sensors s_1 and s_2 are uncertain in nature, the response to this query is a pdf, denoted as $p_{avg}(v)$. This pdf gives us the probability of the sensor value v being the average. $p_{avg}(v)$ is computed using the following formula:

$$p_{avg}(v) = \int_{max(l_{i1,v-u_{i2}})}^{min(u_{i1,v-l_{i2}})} p_{i1}(y)p_{i2}(v-x)dx. \tag{1.12}$$

Naturally, Eq. (1.12) becomes more complicated when there are many (and not only two) sensors involved in the query. Additional details about handling such scenarios can be found in [9].

4.5 Query Processing over Semantic States

The MIST framework [5] proposes to use Hidden Markov Models (HMMs) for deriving semantic meaning from the sensor values. HMMs allow us to capture the hidden states, which are sometimes of more interest than the actual sensor values. Consider, as an example, a scenario where the sensors $\mathbf{S}$ are used to monitor the temperature in all the rooms of a building. Generally, we are only interested to know which rooms are hot or cold, rather than the actual temperature in those rooms. We, then, can use a two-state HMM with states *Hot* (denoted as H) and *Cold* (denoted as C) to continuously infer the semantic states of the temperature in all the rooms.

Furthermore, MIST proposes an in-network index structure for indexing the HMMs. This index can be used for improving the performance of query processing. For instance, if we are interested in finding the rooms that are *Hot* with probability greater than 0.9, then the in-network model index can efficiently prune the rooms that are surely not a part of the query response. Due to the lack of space, we shall not cover the details of index construction and pruning. We encourage the interested reader to read the following paper [5].

4.6 Processing Event Queries

Event queries are another important class of queries that are proposed in the literature. These queries continuously monitor for a particular event that could probably occur in sensor data. Consider a setup consisting of RFID sensors in a building. An event query could monitor an event of a person entering a room or taking coffee, etc. Moreover, event queries can also be registered, not only to monitor a single event, but a sequence of events that are important to the user. Again, due to space constraints, we shall not cover any of the event query processing approaches in detail. The interested reader is referred to the prior works on this subject [55, 65, 68, 45].

5. Model-Based Sensor Data Compression

Recent advances in sensor technology has resulted in the availability of a multitude of (often privately-held) sensors. Embedded sensing functionality (e.g., sound, accelerometer, temperature, GPS, RFID, etc.) is now included in mobile devices, like, phones, cars, or buses. The large number of these devices and the huge volume of raw monitored data pose new challenges for sustainable storage and efficient retrieval of the sensor data streams. To this end, a multitude of model-based regression, transformation and filtering techniques have been proposed for approximation of sensor data streams. This section categorizes and reviews the most important model-based approaches towards compression of sensor data. These models often exploit spatio-temporal correlations within data streams to compress the data within a certain error norm; this is also known as *lossy compression*. Moreover, several standard orthogonal transformation methods (like, Fourier or wavelet transform) reduce the amount of storage space required by reducing the dimensionality of data.

Unlike the assumptions of Section 2, where we assumed a sensor network consisting of several sensors, here we assume that we only have a single sensor. We have dropped the several sensors assumption to simplify the notation and discussion in this section. Furthermore, we assume that the sensor values from the single sensor are in a form of a *data stream*. Let us denote such a data stream as a sequence of data tuples (t_i, v_i), where v_i is the sensor value at time t_i.

5.1 Overview of Sensor Data Compression System

The goal of the sensor data compression system is to approximate a sensor data stream by a set of functions. Data compression methods that we are going to study in this section permit the occurrence of approximation errors. These errors are characterized by a specific error norm. Furthermore, a stan-

dard approach to sensor data compression is to segment the data stream into *data segments*, and then approximate each data segment, so that a specific error norm is satisfied. For example, if we are considering the L_∞ norm, then each sensor value of the data stream is approximated within an error bound ϵ.

Let us assume that we have K segments of a data stream. We denote these segments as $g^1, g^2, \ldots, g^K$, where g^1 approximates the data tuples $((t_1, v_1), \ldots, (t_{i_1}, v_{i_1}))$, while g^k, where $k = 2, \ldots, K$, approximates the data items $((t_{i_{k-1}+1}, v_{i_{k-1}+1}), (t_{i_{k-1}+2}, v_{i_{k-1}+1}), \ldots, (t_{i_k}, v_{i_k}))$. Similar to [20], we distinguish between two classes of the segments used for approximation, namely *connected segments* and *disconnected segments*. In connected segments, the ending point of the previous segment is the starting point of the new segment. On the contrary, in disconnected segments, the approximation of the new segment starts from the subsequent data item in the stream. Disconnected segments offer more approximation flexibility and may lead to fewer segments; however, for linear approximation [35], they necessitate the storage of two data tuples (i.e., start tuple and end tuple) per data segment, as opposed to connected segments.

Since functions are employed for approximating data segments, only the approximated data segments are stored in the database, instead of the raw sensor values of the data stream [64, 50]. A schema for linear segments is presented in [64], consisting of a table, referred to as `FunctionTable`, where each row represents a linear model with attributes `start_time`, `end_time`, `slope` and `intercept` (i.e., base) of the segment. In case of connected segments [20], the `end_time` attribute can be omitted.

A more generic schema for storing data streams, approximated by multiple models was proposed in [50] that consists of one table (`SegmentTable`) for storing the data segments, and a second table (`ModelTable`) for storing the model functions, as depicted in Figure 1.11. A tuple of the `SegmentTable` contains the approximation data for a segment in the time interval `[start_time, end_time]`. The attribute `id` stands for identification of the model that is used in the segment. The primary key in the `SegmentTable` is the `start_time`, while in the `ModelTable` it is `id`. When, both, linear and non-linear models are employed for approximation, `left_value` is the lowest raw sensor value encountered in the segment, and `right_value` is the highest raw sensor value encountered in the segment. In this case, `start_time`, `end_time`, `left_value` and `right_value` define a rectangular bucket that contains the values of the segment.

The attribute `model_params` stores the parameters of the model associated with the model identifier `id`. For example, regression coefficients are stored for the regression model. The attribute `model_params` has variable length (e.g., `VARCHAR` or `VARBINARY` data types in SQL) and it stores the concatenation of the parameters or their compressed representation, by means

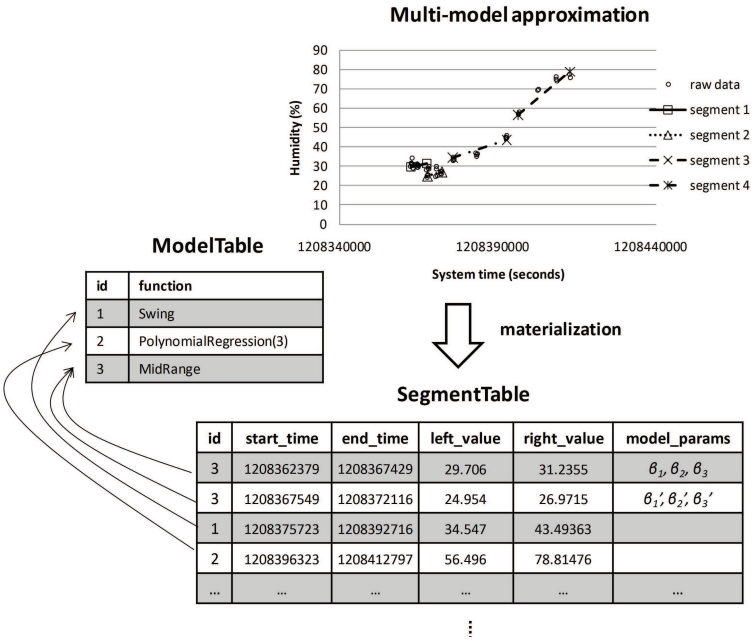


Figure 1.11: The database schema for multi-model materialization.

of standard lossless compression techniques (refer Section 5.7) or by a bitmap coding of approximate values, as proposed in [3]. Each tuple in the ModelTable corresponds to a model with a particular `id` and `function`. The attribute `function` represents the name of the model and it maps to the names of two user defined functions (UDFs) stored in the database. The first function implements the mathematical formula of the model, and the second function implements the inverse mathematical formula of the model, if any. Both the UDFs are employed for answering value-based queries. While the first function is used for value regeneration over fixed time steps (also referred to as *gridding*), the second function is used for solving equations.

5.2 Methods for Data Segmentation

In [34], the piecewise linear approximation algorithms are categorized in three groups: sliding window, top-down and bottom-up. The sliding window approach expands the data segment as long as the data tuples fit. The bottom-up approach first applies basic data segmentation employing the sliding window approach. Then, for two consecutive segments, it calculates merging cost in terms of an approximation error. Subsequently, it merges the segments with the minimum cost within the maximum allowed approximation error, and up-

dates the merging costs of the updated segments. The process ends when no further merging can be done without violating the maximum approximation error. The top-down approach recursively splits the stream into two segments, so as to obtain longest segments with the lowest error until all segments are approximated within the maximum allowed error.

Among these three groups, only the sliding window approach can be used online, but it employs look-ahead. The other two approaches perform better than the sliding window approach, but they need to scan all data, hence they cannot be used for approximating streaming data. Based on this observation, Keogh *et al.* [34] propose a new algorithm that combines the online processing property of the sliding window approach and the performance of the bottom-up approach. This approach needs a predefined buffer length. If the buffer is small, then it may produce many small data segments; if the buffer is large, then there is a delay in returning the approximated data segment. The maximum look-ahead size is constrained by the maximum allowed delay between data production and data reporting or data archiving.

5.3 Piecewise Approximation

Among several different data stream approximation techniques, piecewise linear approximation has been the most widely used [34, 39]. Piecewise linear approximation models the data stream with a separate linear function per data segment. Piecewise constant approximation (PCA) approximates a data segment with a constant value, which can be the first value of the segment (referred to as the cache filter) [47], the mean value or the median value (referred to as poor man's compression - midrange (PMC-MR) [39]).

In the cache filter, for all the sensor values in a segment g^k, the following condition should be satisfied:

$$\left|v_{i_{k-1}+p} - v_{i_{k-1}+1}\right| < \epsilon \qquad \text{for } p = 1, \ldots, i_k, \tag{1.13}$$

where ϵ is the maximum allowed approximation error according to the L_∞ norm. Also, for PMC-Mean and PMC-MR the sensor values in a segment g^k should satisfy the following condition:

$$\max_{1 \le p \le i_k} v_{i_{k-1}+p} - \min_{1 \le p \le i_k} v_{i_{k-1}+p} \le 2\epsilon\,. \tag{1.14}$$

Furthermore, for PMC-Mean, the approximation value for the segment g^k is given by the mean value of the sensor values in segment g^k. But, for PMC-MR it is given as follows:

$$\frac{\max_{1 \le p \le i_k} v_{i_{k-1}+p} - \min_{1 \le p \le i_k} v_{i_{k-1}+p}}{2}\,.$$

The data segmentation approach for PMC-MR is illustrated in Figure 1.12.

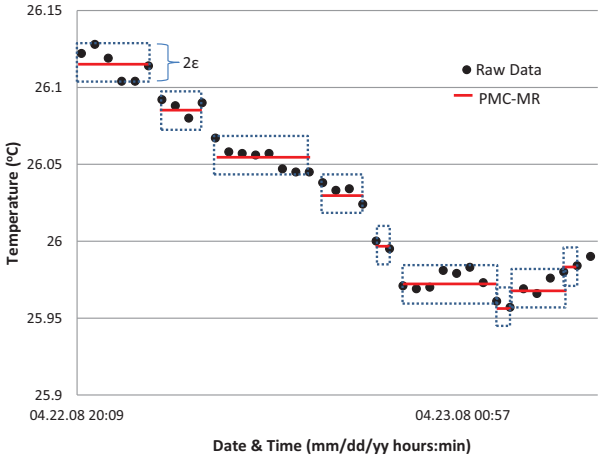


Figure 1.12: Poor Man's Compression - MidRange (PMC-MR).

Moreover, the linear filter [34] is a simple piecewise linear approximation technique in which the sensor values are approximated by a line connecting the first and second point of the segment. When a new data tuple cannot be approximated by this line with the specified error bound, a new segment is started. In [20], two new piecewise linear approximation models were proposed, namely *Swing* and *Slide*, that achieve much higher compression compared to the cache and linear filters. We briefly discuss the swing and slide filters below.

Swing and Slide Filters. The swing filter is capable of approximating multi-dimensional data. But, for simplicity, we describe its algorithm for one-dimensional data. Given the arrival of two data tuples (t_1, v_1) and (t_2, v_2) of the first segment of the data stream, the swing filter maintains a set of lines, bounded by an upper line u^1 and a lower line l^1. u^1 is defined by the pair of points (t_1, v_1) and $(t_2, v_2 + \epsilon)$, while l^1 is defined by the pair of points (t_1, v_1) and $(t_2, v_2 - \epsilon)$, where ϵ is the maximum approximation error bound. Any line segment between u^1 and l^1 can represent the first two data tuples. When (t_3, v_3) arrives, first it is checked whether it falls within the lines l^1, u^1. Then, in order to maintain the invariant that all lines within the set can represent all data tuples so far, l^1 (respectively u^1) may have to be adjusted to the higher-slope (respectively lower-slope) line defined by the pair of data tuples $((t_1, v_1), (t_3, v_3 - \epsilon))$ (respectively $((t_1, v_1), (t_3, v_3 + \epsilon))$). Lines below this new l^1 or above this new u^1 cannot represent the data tuple (t_3, v_3). The segment estimation continues until the new data tuple falls out of the upper and lower lines for a segment. The generated line segment for the completed

filtering interval is chosen so as to minimize the mean square error for the data tuples observed in that interval. As opposed to the slide filter (described below), in the swing filter the new data segment starts from the end point of the previous data segment.

In the slide filter, the operation is similar to the swing filter, but upper and lower lines u and l are defined differently. Specifically, after (t_1, v_1) and (t_2, v_2) arrive, u^1 is defined by the pair of data tuples $(t_1, v_1 - \epsilon)$ and $(t_2, v_2 + \epsilon)$, while l^1 is defined by $(t_1, v_1 + \epsilon)$ and $(t_2, v_2 - \epsilon)$. After the arrival of (t_3, v_3), l^1 (respectively u^1) may need to be adjusted to the higher-slope (respectively lower-slope) line defined by $((t_j, v_j + \epsilon), (t_3, v_3 - \epsilon))$ (respectively $((t_i, v_i - \epsilon), (t_3, v_3 + \epsilon))$), where $i \in [1, 2]$. The slide filter also includes a look-ahead of one segment, in order to produce connected segments instead of disconnected segments, when possible.

Palpanas *et al.* [48] employ *amnesic functions* and propose novel techniques that are applicable to a wide range of user-defined approximating functions. According to amnesic functions, recent data is approximated with higher accuracy, while higher error can be tolerated for older data. Yi and Faloutsos [70] suggested approximating a data stream by dividing it into equal-length segments and recording the mean value of the sensor values that fall within the segment (referred to as segmented means or as piecewise aggregate approximation (PAA)). On the other hand, adaptive piecewise constant approximation (APCA) [6] allows segments to have arbitrary lengths.

Piecewise Linear Approximation. The piecewise linear approximation uses the linear regression model for compressing data streams. The linear regression model of a data segment is given as:

$$v_i = s \cdot t_i + b, \tag{1.15}$$

where b and s are known as the base and the slope respectively. The difference between v_i and t_i is known as the residual for time t_i. For fitting a linear regression model of Eq. (1.15) to the sensor values $v_i : t_i \in [t_b; t_e]$, the ordinary least squares (OLS) estimator is employed. The OLS estimator selects b and s such that they minimize the following sum of squared residuals:

$$RSS(b, s) = \sum_{t_i=t_b}^{t_e} [v_i - (s \cdot t_i + b)]^2.$$

Therefore, b and s are given as:

$$\begin{aligned} b &= \sum_{t_i=t_b}^{t_e} \left(\frac{t_i - \frac{t_b+t_e}{2}}{\sum_{t_i=t_b}^{t_e} (t_i - \frac{t_b+t_e}{2}) t_i} \right) v_i, \\ s &= \frac{\sum_{t_i=t_b}^{t_e} v_i}{t_e - t_b + 1} - b\frac{t_b + t_e}{2}. \end{aligned} \tag{1.16}$$

Here, the storage record of each data segment of the data stream consists of $([t_b; t_e]; b, s)$, where $[t_b; t_e]$ is the segment interval, and s and b are the slope and base of the linear regression, as obtained from Eq. (1.16).

Similarly, instead of the linear regression model, a polynomial regression model (refer Eq. (1.9)) can also be utilized for approximating each segment of the data stream. The storage record of the polynomial regression model is similar to the linear regression model. The only difference is that for the polynomial regression model the storage record contains parameters $\alpha_1, \ldots, \alpha_d$ instead of the parameters b and s.

5.4 Compressing Correlated Data Streams

Several approaches [14, 42, 24] exploit correlations among different data streams for compression. The GAMPS approach [24] dynamically identifies and exploits correlations among different data segments and then jointly compresses them within an error bound employing a polynomial-time approximation algorithm. In the first phase, data segments are individually approximated based on piecewise constant approximation (specifically the PMC-Mean described in Section 5.3). In the second phase, each data segment is approximated by a ratio with respect to a base segment. The segment formed by the ratios is called the ratio segment. GAMPS proposes to store the base segment and the ratio segment, instead of storing the original data segment. The idea here is that, in practice, the ratio segment is flat and therefore can be significantly compressed as compared to the original data segment.

Furthermore, the objective of the GAMPS approach is to identify a set of base segments, and associate every data segment with a base segment, such that the ratio segment can be used for reconstructing the data segment within a L_∞ error bound. The problem of identification of the base segments is posed as a *facility location* problem. Since this problem is NP-hard, a polynomial-time approximation algorithm is used for solving it, and producing the base segments and the assignment between the base segments and data segments.

Prior to GAMPS, Deligiannakis *et al.* [14] proposed the self-based regression (SBR) algorithm that also finds a base-signal for compressing historical sensor data based on spatial correlations among different data streams. The base-signal for each segment captures the prominent features of the other signals, and SBR finds piecewise correlations (based on linear regression) to the

base-signal. Lin *et al.* [42] proposed an algorithm, referred to as adaptive linear vector quantization (ALVQ), which improves SBR in two ways: (i) it increases the precision of compression, and (ii) it reduces the bandwidth consumption by compressing the update of the base signal.

5.5 Multi-Model Data Compression

The potential burstiness of the data streams and the error introduced by the sensors often result in limited effectiveness of a single model for approximating a data stream within the prescribed error bound. Acknowledging this, Lazaridis *et al.* [39] argue that a global approximation model may not be the best approach and mention the potential need for using multiple models. In [40], it is also recognized that different approximation models are more appropriate for data streams of different statistical properties. The approach in [40] aims to find the best model approximating the data stream based on the overall *hit ratio* (i.e., the ratio of the number of data tuples fitting the model to the total number of data tuples).

Papaioannou *et al.* [50] aim to effectively find the best combination of different models for approximating various segments of the stream regardless of the error norm. They argue that the selection of the most efficient model depends on the characteristics of the data stream, namely rate, burstiness, data range, etc., which cannot be always known *a priori* for sensors and they can even be dynamic. Their approach dynamically adapts to the properties of the data stream and approximates each data segment with the most suitable model. They propose a greedy approach in which they employ multiple models for each segment of the data stream and store the model that achieves the highest compression ratio for the segment. They experimentally proved that their multi-model approximation approach always produces fewer or equal data segments than those of the best individual model. Their approach could also be used to exploit spatial correlations among different attributes from the same location, e.g., humidity and temperature from the same stationary sensor.

5.6 Orthogonal Transformations

The main application of the orthogonal transformation approaches has been in dimensionality reduction, since reducing the dimensionality improves performance of indexing techniques for similarity search in large collections of data streams. Typically, sequences of fixed length are mapped to points in an N-dimensional Euclidean space; then, multi-dimensional access methods, such as R-tree family, can be used for fast access of those points. Since, sequences are usually long, a straightforward application of the above approach, which does not use dimensionality reduction, suffers from performance degradation due to the curse of dimensionality [56].

The process of dimensionality reduction can be described as follows. The original data stream or signal is a finite sequence of real values or coefficients, recorded over time. This signal is transformed (using a specific transformation function) into a signal in a transformed space. To achieve dimensionality reduction, a subset of the coefficients of the orthogonal transformation are selected as features. These features form a feature space, which is simply a projection of the transformed space. The basic idea is to approximate the original data stream with a few coefficients of the orthogonal transformation; thus reducing the dimensionality of the data stream.

Discrete Fourier Transform (DFT). The Fourier transform is the most popular orthogonal transformation. It is based on the simple observation that every signal can be represented by a superposition of sine and cosine functions. The discrete Fourier transform (DFT) and discrete cosine transform (DCT) are efficient forms of the Fourier transform often used in applications. The DFT is the most popular orthogonal transformation and was first used in [1, 22]. The Discrete Fourier Transform of a time sequence $x = x_0, \ldots, x_{N-1}$ is a sequence $X = X_0, \ldots, X_{N-1}$ of complex numbers given by:

$$X_k = \sum_{j=0}^{N-1} e^{-i2\pi \frac{k}{N} j}. \tag{1.17}$$

The original signal can be reconstructed by the inverse Fourier transform of X, which is given by:

$$x_j = \sum_{k=0}^{N-1} X_k e^{i2\pi \frac{k}{N} j}. \tag{1.18}$$

In [1], Agrawal *et al.* suggest using the DFT for dimensionality reduction of long observation sequences. They argue that most real signals only require a few DFT coefficients for their approximation. Thus similarity search can be performed only over the first few DFT coefficients, instead of the full observation sequence. This provides an efficient and approximate solution to the problem of similarity search in high-dimensional spaces. They use the Euclidean distance as the dissimilarity measure.

Discrete Wavelet Transform. Wavelets can be thought of as a generalization of the Fourier transform to a much larger family of functions than sine and cosine. Mathematically, a wavelet is a function $\psi_{j,k}$ defined on the real numbers $\mathbb{R}$, which includes an integer translation by k, also called a shift, and a dyadic dilation (a product by the powers of two), known as stretching. The functions $\psi_{j,k}$ play a similar role as the exponential functions in the Fourier transform: $\psi_{j,k}$ form an orthonormal basis for the $L^2(\mathbb{R})$ space. The

$L^2(\mathbb{R})$ space consists of all the functions whose L_2 norm is finite. Particularly, the functions $\psi_{j,k}$, where j and k are integers are given as follows:

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k). \tag{1.19}$$

Similar to the Fourier transform, by using the orthonormal basis functions $\psi_{j,k}$, we can uniquely express a function $f \in L^2(\mathbb{R})$ as a linear combination of the basis functions $\psi_{j,k}$ as follows:

$$f = \sum_{j,k\in\mathbb{Z}} < f, \psi_{j,k} > \psi_{j,k}, \tag{1.20}$$

where $< f, g >:= \int_R f\overline{g}dx$ is the usual inner product of two functions in $L^2(\mathbb{R})$.

The Haar wavelets are the most elementary example of wavelets. The mother wavelet ψ for the Haar wavelets is the following function:

$$\psi_{\text{Haar}}(t) = \begin{cases} 1, & \text{if } 0 < t < 0.5, \\ -1, & \text{if } 0.5 < t < 1, \\ 0, & \text{otherwise.} \end{cases} \tag{1.21}$$

Ganesan *et al.* [26, 25] proposed in-network storage of wavelet-based summaries of sensor data. Recently, discrete wavelet transform (DWT) was also proposed in [53, 7] for sensor data compression. For sustainable storage and querying, they propose progressive aging of summaries and load sharing techniques.

Discussion. The basis functions of some wavelet transforms are nonzero only on a finite interval. Therefore, wavelets may be only able to capture local (time dependent) properties of the data, as opposed to Fourier transforms, which can capture global properties. The computational efficiency of the wavelet transforms is higher than the Fast Fourier transform (FFT). However, while the Fourier transform can accurately approximate arbitrary signals, the Haar wavelet is not likely to approximate a smooth function using few features.

The wavelet transform representation is intrinsically coupled with approximating sequences whose length is a power of two. Using wavelets with sequences that have other lengths require ad-hoc measures that reduce the fidelity of the approximation, and increase the complexity of the implementation. DFT and DCT have been successfully adapted to incremental computation [72]. However, as each DFT/DCT coefficient makes a global contribution to the entire data stream, assigning less significance to the past data is not obvious with these transformations.

5.7 Lossless vs. Lossy Compression

While lossless compression is able to accurately reconstruct the original data, lossy compression techniques approximate data streams within a certain error bound. Most lossless compression schemes perform two steps in sequence: the first step generates a statistical model for the input data, and the second step uses this model to map input data to bit sequences. These bit sequences are mapped in such a way that frequently encountered data will produce shorter output than infrequent data. General-purpose compression schemes include DEFLATE (employed by gzip, ZIP, PNG, etc.), LZW (employed by GIF, compress, etc.), LZMA (employed by 7zip). The primary encoding algorithms used to produce bit sequences are Huffman coding (also used by DEFLATE) and arithmetic coding. Arithmetic coding achieves compression rates close to the best possible, for a particular statistical model, which is given by the information entropy. On the other hand, Huffman compression is simpler and faster but produces poor results.

Lossless compression techniques, however, are not adequate for a number of reasons: (a) as experimentally found in [39], gzip lossless compression achieves poor compression (50%) compared to lossy techniques, (b) lossless compression and decompression are usually more computationally intensive than lossy techniques, and (c) indexing cannot be employed for archived data with lossless compression.

6. Summary

In this chapter, we presented a comprehensive overview of the various aspects of model-based sensor data acquisition and management. Primarily, the objectives of the model-based techniques are efficient data acquisition, handling missing data, outlier detection, data compression, data aggregation and summarization. We started with acquisition techniques like TinyDB [45], Ken [12], PRESTO [41]. In particular, we focused on how acqusitional queries are disseminated in the sensor network using routing trees [44]. Then we surveyed various approaches for sensor data cleaning, including polynomial-based [73], probabilistic [21, 63, 52, 65] and declarative [31, 46].

For processing spatial, temporal and threshold queries, we detailed query processing approaches like MauveDB [18], FunctionDB [64], particle filtering [33], MIST [5], etc. Here, our primary objective was to demonstrate how model-based techniques are used for improving various aspects of query processing over sensor data. Lastly, we discussed data compression techniques, like, linear approximation [34, 39, 48], multi-model approximations [39, 40, 50] and orthogonal transformations [1, 22, 53, 7].

All the methods that we presented in this chapter were model-based. They utilized models – statistical or otherwise – for describing, simplifying or abstracting various components of sensor data acquisition and management.

Acknowledgments

This work was supported by the OpenSense project (Nano-Tera reference number 839_401), NCCR-MICS (http://www.mics.org), and by the OpenIoT project (EU FP7-ICT 287305).

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

[3] A. Arion, H. Jeung, and K. Aberer. Efficiently maintaining distributed model-based views on real-time data streams. In *GLOBECOM*, pages 1–6, 2011.

[4] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[5] A. Bhattacharya, A. Meka, and A. Singh. MIST: Distributed indexing and querying in sensor networks using statistical models. In *VLDB*, pages 854–865, 2007.

[6] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)*, 27(2):188–228, 2002.

[7] K. Chan and W. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999.

[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

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

[10] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Information Systems*, 32(1):104–130, 2007.

[11] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *VLDB*, pages 1271–1274, 2005.

[12] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, pages 48–48, 2006.

[13] F. Chu, Y. Wang, S. Parker, and C. Zaniolo. Data cleaning using belief propagation. In *IQIS*, pages 99–104, 2005.

[14] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *SIGMOD*, pages 527–538, 2004.

[15] A. Deligiannakis, V. Stoumpos, Y. Kotidis, V. Vassalos, and A. Delis. Outlier-aware data aggregation in sensor networks. In *ICDE*, pages 1448–1450, 2008.

[16] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, pages 143–154, 2005.

[17] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[18] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD*, pages 73–84, 2006.

[19] R. Elmasri and S. Navathe. *Fundamentals of database systems*. Addison Wesley, 6th edition, 2010.

[20] H. Elmeleegy, A. Elmagarmid, E. Cecchet, W. Aref, and W. Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. In *VLDB*, pages 145–156, 2009.

[21] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *WSNA*, pages 78–87, 2003.

[22] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[23] C. Franke and M. Gertz. ORDEN: Outlier region detection and exploration in sensor networks. In *SIGMOD*, pages 1075–1077, 2009.

[24] S. Gandhi, S. Nath, S. Suri, and J. Liu. GAMPS: Compressing multi sensor data by grouping and amplitude scaling. In *SIGMOD*, pages 771–784, 2009.

[25] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *SIGCOMM*, pages 143–148, 2003.

[26] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and H. J. An evaluation of multi-resolution storage for sensor networks. In *SenSys*, pages 89–102, 2003.

[27] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: An efficient framework for modeling sensor network data. In *IPSN*, pages 1–10, 2004.

[28] H. Gupta, V. Navda, S. Das, and V. Chowdhary. Efficient gathering of correlated data in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(1):4, 2008.

[29] A. Jain, E. Chang, and Y.-F. Wang. Adaptive stream resource management using Kalman Filters. In *SIGMOD*, pages 11–22, 2004.

[30] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *ICDE*, page 140, 2006.

[31] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In *Pervasive*, pages 83–100, 2006.

[32] S. Jeffery, M. Garofalakis, and M. Franklin. Adaptive cleaning for RFID data streams. In *VLDB*, pages 163–174, 2006.

[33] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, pages 1160–1169, 2008.

[34] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[35] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *SIGKDD*, pages 239–241, 1998.

[36] A. Klein. Incorporating quality aspects in sensor data streams. In *PIKM*, pages 77–84, 2007.

[37] A. Klein and W. Lehner. Representing data quality in sensor data streaming environments. *Journal of Data and Information Quality*, 1(2):1–28, 2009.

[38] Y. Kotidis. Snapshot queries: Towards data-centric sensor networks. In *ICDE*, pages 131–142, 2005.

[39] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *ICDE*, pages 429–440, March 2003.

[40] Y. Le Borgne, S. Santini, and G. Bontempi. Adaptive model selection for time series prediction in wireless sensor networks. *Signal Processing*, 87(12):3010–3020, 2007.

[41] M. Li, D. Ganesan, and P. Shenoy. PRESTO: Feedback-driven data management in sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1256–1269, 2009.

[42] S. Lin, V. Kalogeraki, D. Gunopulos, and S. Lonardi. Online information compression in sensor networks. In *IEEE International Conference on Communications*, 2006.

[43] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[44] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003.

[45] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *TODS*, 30(1):122–173, 2005.

[46] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: A database approach for statistical inference and data cleaning. In *SIGMOD*, pages 75–86, 2010.

[47] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, pages 563–574, 2003.

[48] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, pages 339–349, 2004.

[49] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[50] T. Papaioannou, M. Riahi, and K. Aberer. Towards online multi-model approximation of time series. In *IEEE MDM*, pages 33–38, 2011.

[51] A. Patcha and J.-M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

[52] A. Petrosino and A. Staiano. A neuro-fuzzy approach for sensor network data cleaning. In *KES*, pages 140–147, 2007.

[53] I. Popivanov. Similarity search over time series data using wavelets. In *ICDE*, pages 212–221, 2002.

[54] J. Rao, S. Doraiswamy, H. Thakkar, and L. Colby. A deferred cleansing method for RFID data analytics. In *VLDB*, pages 175–186, 2006.

[55] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, pages 715–728, 2008.

[56] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[57] S. Sathe, H. Jeung, and K. Aberer. Creating probabilistic databases from imprecise time-series data. In *ICDE*, pages 327–338, 2011.

[58] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *MobiDE*, pages 69–76, 2003.

[59] B. Sheng, Q. Li, W. Mao, and W. Jin. Outlier detection in sensor networks. In *MobiHoc*, pages 219–228, 2007.

[60] R. Shumway and D. Stoffer. *Time series analysis and its applications*. Springer-Verlag, New York, 2005.

[61] A. Silberstein, R. Braynard, G. Filpus, G. Puggioni, A. Gelfand, K. Munagala, and J. Yang. Data-driven processing in sensor networks. In *CIDR*, 2007.

[62] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006.

[63] Y. Tan, V. Sehgal, and H. Shahri. SensoClean: Handling noisy and incomplete data in sensor networks using modeling. Technical report, University of Maryland, 2005.

[64] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, pages 791–804, 2008.

[65] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, pages 1096–1107, 2009.

[66] D. Tulone and S. Madden. PAQ: Time series forecasting for approximate query answering in sensor networks. In *EWSN*, pages 21–37, 2006.

[67] L. Wang and A. Deshpande. Predictive modeling-based data collection in wireless sensor networks. In *EWSN*, pages 34–51, 2008.

[68] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, pages 407–418, 2006.

[69] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[70] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[71] Y. Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Survey & Tutorials*, 12(2), 2010.

[72] Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.

[73] Y. Zhuang, L. Chen, X. Wang, and X. Lian. A weighted moving average-based approach for cleaning sensor data. In *ICDCS*, page 38, 2007.