2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

M-DB: A Continuous Data Processing and Monitoring Framework for IoT Applications

Vaibhav Arora Mohammad Javad Amiri Divyakant Agrawal Amr El Abbadi
Department of Computer Science, University of California Santa Barbara
Santa Barbara, California
{vaibhavarora, amiri, agrawal, amr}@cs.ucsb.edu

***Abstract*—IoT devices influence many different spheres of society and are predicted to have a huge impact on our future. Extracting real-time insights from diverse sensor data and dealing with the underlying uncertainty of sensor data are two main challenges of the IoT ecosystem In this paper, we propose a data processing architecture, *M-DB*, to effectively integrate and continuously monitor uncertain and diverse IoT data. M-DB constitutes of three components: (1) *model-based operators* (MBO) as data management abstractions for IoT application developers to integrate data from diverse sensors. Model-based operators can support event-detection and statistical aggregation operators, (2) *M-Stream*, a dataflow pipeline that combines model-based operators to perform computations reflecting the uncertainty of underlying data, and (3) M-Store, a storage layer separating the computation of application logic from physical sensor data management, to effectively deal with missing or delayed sensor data. M-DB is designed and implemented over Apache Storm and Apache Kafka, two open-source distributed event processing systems. Our illustrated application examples throughout the paper and evaluation results illustrate that M-DB provides a real-time data-processing architecture that can cater to the diverse needs of IoT applications.**

***Index Terms*—IoT, Real-time Processing, Abstractions, Prediction**

I. Introduction

Internet connected devices such as smart cars, wearables like health monitors and fitness trackers, and smart home appliances like surveillance cameras, thermostats etc. are slowly becoming ubiquitous and are the future of the Internet. Such devices are collectively referred to as Internet of Things (IoT). Gartner has predicted [5] that there would be 26 billion IoT devices by the end of year 2020. One of the most challenging aspects for the IoT ecosystem is to get insights from data, which is being continuously collected and transmitted from diverse sensors connected to such physical devices [14], [35].

On one hand, integrating data from multiple diverse homogeneous or heterogeneous sensors over time will facilitate the ability to extract deeper insights about the physical environment [9], [28]. On the other hand, the underlying uncertainty of sensor data needs to be taken into account; the sensed data may have errors due to the underlying device errors or a failure in the communication pipeline. Additionally, sensor data might be delayed or missing because of network connectivity issues, failure of the IoT devices, or energy saving mechanisms at IoT devices. To deal with such an uncertainty, an IoT framework must be able to first, *predict* the delayed or missing values and second, output *confidence* guarantees for the computation results. In addition, it is desirable for an IoT framework to provide higher-level *abstractions* for the application developers to express the integration of data from different sensors.

While over the last decade, applications have employed stream processing architectures [11], [26], [36], [37] for the continuous real-time processing of data ingested from multiple sources, these architectures can sometime act as bottlenecks for the IoT data use-case. The push based processing model of stream processing does not fit well with continuous monitoring requirements and the nature of sensor data. As described above, sensor data can be missing or delayed. Push-based architectures might delay or block processing while waiting for delayed or missing sensor data. From the IoT application point of view, processing has to be performed in real-time. Some recent systems [1], [6], [13] can support data delays and pipeline errors by defining triggers, which can be used to produce results based on time-based or data arrival based conditions. However, even these systems do not define how to deal with missing or delayed sensor values nor express the uncertainty of the underlying data in the computation results.

Furthermore, current data management systems do not provide any abstractions to the application developer to express the integration of sensor data. Any logic for integration or expressing computation on sensor data needs to be written by the developer. Developers building applications which interact with IoT devices, should not have to deal with the device API's or individual sensors. Currently, data management systems provide either simple key-value operations which burden the application developer with the complexity of writing entire solutions or transactional abstractions, which are not suited for IoT data. Traditional SQL-based databases [7], [8] provide a transaction abstraction over multiple requests reading and updating the data. Execution of a transaction implies the atomic execution of its composed sub-requests. Many IoT applications on the other hand have to deal with the uncertainty of the sensor data, and cannot be certain of all the encompassing observations. Additionally, the abstractions provided should be able to efficiently support event detection and statistical aggregation operators.

In this paper, we propose a data processing architecture, *M-DB*, to effectively integrate and continuously monitor uncertain and diverse IoT data. M-DB constitutes three components: abstractions to integrate sensor data, a dataflow pipeline to perform computations reflecting the uncertainty of the underlying data, and a storage layer separating the computation of business logic from the physical sensor data management.

First, *Model-based Operators* are proposed as *abstractions* for the IoT application developer, to express the integration of diverse sensor data. Model-based Operators (MBO) can integrate data in spatial, temporal or spatio-temporal domains, and support event-detection and statistical aggregation operators. MBOs are defined by specifying a model of execution using threshold-based, aggregate or user-defined functions to integrate data, and output a degree of *confidence* in the execution to express the underlying uncertainty of sensor data.

Second, we propose a computation framework named *M-Stream*, which represents the processing pipeline of the application by combining model-based operators in the form of a dataflow graph and continuously executing the model-based operators at defined time intervals.

Third, a datastore interface, *M-Store*, is employed to hide the complexity of sensor data aggregation from the real-time data processing in M-Stream. M-Store provides access to sensor and non-sensor data, and gives M-DB the ability to employ *prediction models* to incorporate missing or delayed values.

The paper makes the following contributions:

- Model-based operators are proposed as *abstractions* to IoT applications to integrate data from diverse sensors, and enable temporal, spatial or spatio-temporal integration. MBOs provide support for event-detection or statistical aggregation like operations, and their execution reflects the uncertainty in the underlying data.
- M-DB provides the ability to express the uncertainty of computational results in a data-processing pipeline via *confidence* values provided by MBOs. M-Stream combines MBOs in a dataflow graph, which ensures that the underlying uncertainty in the data can flow along the computation pipeline.
- M-DB's architecture generates real-time results, even in the presence of missing or delayed sensor values, first, by separating the execution in M-Stream from the physical handling of sensor data in M-Store, and second, by utilizing *prediction models* in M-Store to fill in missing or delayed sensor data.

The rest of the paper is organized as follows. Section II gives an overview of M-DB. The details of model-based operators are presented in Section III. Section IV introduces M-DB's architecture. The implementation of M-DB over Apache Storm and Apache Kafka is described in Section V. Experimental results are presented in Section VI and Section VII discusses the related work. The paper concludes in Section VIII.

II. M-DB Overview

M-DB supports event-detection and statistical aggregation use-cases, by building on model-based operators, and combining and executing them via dataflow pipelines. Figure 1 illustrates M-DB's architecture. M-DB builds on the Model-based operators abstraction, and employs a computational pipeline, M-Stream. A storage abstraction, M-Store, is used for accessing and managing sensor and non-sensor data.

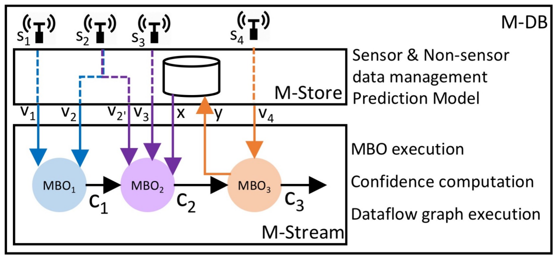

Fig. 1: M-DB Architecture

M-DB employs the model-based operation abstraction to integrate sensor, as well as non-sensor data. Offering a model-based operator abstraction to the application developer at the data management layer provides the advantage of hiding the complexity of integrating data from multiple sensors. Furthermore, it allows for better modularity and code re-use as multiple IoT applications can be built using the same model-based operator abstractions. Model-based operators are of 4 different types, to support temporal, spatial, and two varieties of spatio-temporal integration respectively. Different model-based operators can be defined, based on the needs of the particular application. Following are three applications to show different use-cases for model-based operators.

- Detecting abnormal vibration in a turbine where a pre-defined percentage of sensor values in a given period exceed a pre-defined threshold [21] (*Event-detection via Temporal Integration of Homogeneous sensors*).
- A fitness app that tracks aggregate statistics like distance run, time elapsed etc. over a group of runners (*Statistical Aggregation via Spatio-temporal Integration of Homogeneous sensors*).
- Making decision to provide drug doses to patients based on multiple medical sensors calculating, blood pressure, heart rate etc. (*Event-detection via Spatial Integration of Heterogeneous sensors*).

M-Stream provides the ability to combine the defined model-based operation executions in a processing pipeline in the form of a dataflow graph, to represent the computational needs of IoT applications. In Figure 1, MBO_1 illustrates a model-based operator, which integrates data from two sensors, s_1 and s_2 and outputs the outcome with confidence c_1. Model-based operator MBO_2 integrates values from sensors s_2, s_3, a non-sensor input value, x, and the output from MBO_1's execution, c_1, and outputs c_2. Finally, MBO_3 integrates values from sensor s_4, with the the output of MBO_2. If MBO_3 succeeds, which is determined by the definition of MBO_3 (in case of a thresholding operation), then data item y will be updated in the datastore.

M-Stream's design aids M-DB in dealing with missing or uncertain sensor data. To capture the uncertainty of the underlying data, M-Stream utilizes the confidence in the computation of model-based operations. The confidence function associated with the model-based operator definition is used to reflect the certainty in the computation as a function of the input data. The dataflow pipeline in M-Stream then ensures that the uncertainty in a MBO computation captured by its confidence can flow through the data processing pipeline, as an input to connected model-based operators. This way, the uncertainty can be reflected in downstream computations.

M-Stream employs a continuous processing model to compute the model-based operations periodically after a defined time period. A continuous processing and pull-based model is combined with the traditional push-based stream processing model to perform computations.

To separate sensor data management from the continuous real-time dataflow computations defined on corresponding data, M-DB employs M-Store. M-Store stores, processes and provides API access to sensor and non-sensor data in M-Stream. The separation of concerns also gives M-Store the ability to incorporate prediction models to fill in missing or delayed sensor data values. M-Stream can then use the confidence values to reflect the uncertainty of MBO computations on predicted data.

III. Model-based Operator Abstraction

Model-based operators provide the ability to integrate data from homogeneous or heterogeneous sensors in both spatial and temporal domain. In this section, we introduce four different model-based operators to integrate data over the temporal, spatial, and spatio-temporal domains. We mainly focus on two classes of applications: event detection and statistical aggregation. In event detection applications, a thresholding function is defined to determine whether an event occurs or not, and in statistical aggregation, an aggregation function is defined to integrate data over the temporal or spatial domain. Note that aggregation functions are mainly targeted for homogeneous sensors, where all the sensor values are of the same type. In addition to event detection and statistical aggregation application classes, we allow developers to define their own (user-defined) functions to integrate data.

We first define two functions ρ and ψ to integrate data over the temporal and spatial domains respectively and then define model-based operators.

Function ρ is defined on the values received from a sensor over a time period. The definition of ρ depends on the application class. For event detection applications, ρ is a function with an input consisting of four elements: a set of sensor values, a threshold τ, an inequality operator σ (σ is $<$, $>$, $\leq$, $\geq$, $=$, or $\neq$), and a percentage p. ρ returns *true* if at least p percent of the values received from the sensor satisfy the threshold τ based on the operator σ. In statistical aggregation use-cases, ρ is an aggregation function, e.g., *sum*, *average*, *min*, and *max*, with a set of sensor values as input and an aggregated value as output. In general-purpose applications, ρ is a user-defined function with a set of sensor values as input and a user-specified output.

Function ψ is defined in a similar way on values received from a set of sensors at one time-point (one value per sensor). Since values are from different sensors, for event detection applications, we need to define a threshold τ and an inequality operator σ for each sensor (value). Same as before, function ψ returns *true*, if at least p percent of the values received from the sensors satisfy their own threshold τ based on their own operator σ. In statistical aggregation applications, ψ is an aggregation function. Finally, in general-purpose applications, ψ has sensor values as input and a user-specified output. We now proceed to define different model-based operators.

A. *Temporal Model-based Operator*

We first provide an abstraction to integrate the data from a single sensor in the temporal domain. The values are considered as real numbers which are received on a regular basis. The time domain is modeled as an ordered, infinite set of discrete time points where each time point is basically a sequence number. A temporal model-based operator $T\!-\!MBO$ integrates received values at each time period. To specify time periods we use time-based sliding windows [32]. Note that sliding windows are generic enough to express windows with arbitrary progression steps or even tumbling windows (fixed-sized, non-overlapping time intervals). When a time window is finished, the operator integrates received values in that time window using function ρ. In the case of event detection applications, the model-based operator is said to be executed if function ρ returns *true* (at least p percent of the values received from the sensor in the time window satisfy threshold τ). The confidence c of execution is determined by a user-defined function φ. O is also a finite set of outputs of the model-based operator if the operator is considered to be executed successfully. The outputs may either be written to the datastore or might be inputs to other model-based operators. Note that model-based operators can fuse non-sensor inputs with sensor inputs as well. A non-sensor input such as a data-item with key k and value val can be represented as input from sensor k with value val.

Definition: A *Temporal Model-based Operator* is a tuple $T\!-\!MBO = (s, \mathcal{V}, w, \mathcal{T}, \gamma, \rho, \varphi, c, O)$ where

- s is a single sensor,
- $\mathcal{V} \subset \mathbb{R}$ is a set of (sensor) values,
- w is the length of each time window,
- $\mathcal{T}$ is a set of time-points,
- $\gamma : \mathcal{V} \to \mathcal{T}$ is a mapping that assigns time points to values,
- ρ is the temporal integration function that integrates data over each time window,
- φ is a (user-defined) mapping that returns the execution confidence $c \in [0, 1]$ for each time window based on the mappings γ and ρ, and
- O is a set of outputs.

For example, in the case of the vibration detection use-case described earlier, the application developer wants the detection to be made based on values received from a single sensor over a time period. Consider a MBO with a time window of length 6. where $\mathcal{V} = \{5, 7, 10, 9, ..\}$ is the set of received values and $\gamma(5) = 1$, $\gamma(7) = 3$, $\gamma(10) = 5$, and $\gamma(9) = 7$ are the time-points within the two first time windows (data is received from the sensor every 2 time units). Since vibration detection is an event detection application, ρ is defined as a thresholding function. Let $\tau = 8$, $p = 0.6$ and the inequality operator be "$<$" (a sensor value satisfies the threshold if it is less than τ). Here, in the first time window, since 0.66 of the time-points (time-points 1 and 3) have values less than 8, the threshold condition is satisfied, function ρ returns *true* and the model-based operator executes successfully, but in the second time window, only time-point 3 (with value 7) satisfies the threshold condition, so function ρ returns *false* and the operator fails. The user-defined function φ can be specified as the ratio of values that satisfy the threshold τ to the total number of values (time points) in a time window. Using this definition, φ returns 0.66 as the execution confidence for the first time window in the above example. The definition of φ given here is one way of computing the confidence. Applications might use a different definition based on the context.

B. *Spatial Model-based Operator*

The next model based operator is defined to integrate data over multiple sensors. Note that we use the term spatial referring to different physical sensors being usually present at different locations. Values from different sensors can represent values from different related locations, like soil moisture readings from different locations on a farm. Alternatively, they might provide information about different physical attributes of a single location, e.g., integrating data from multiple sensors that send information about a patient's blood pressure, heart rate, insulin level etc. to perform continuous health monitoring and administer medicines.

The Spatial Model-based Operator is composed of n sensors, s_1, s_2, ..., s_n. Each sensor s_i has a value associated with it, v_i, which is emitted periodically. The spatial model-based operator, $S{-}MBO$, receives values from all sensors at each time point $t_i \in \mathcal{T}$ and immediately integrates them using function ψ (unlike the temporal model-based operator where we wait till the end of a time window). To handle *non-synchronized* sensor values, a time interval δ is also defined. Value v is considered to be received at time point t if v arrives in $[t{-}\delta, t{+}\delta]$. In the case of event detection applications, a spatial model-based operator executes successfully if function ψ returns *true* (at least p percent of the n sensors values satisfy their defined thresholds). Similar to the temporal operator, the confidence c is determined by a user-defined function φ.

Definition: A *Spatial Model-based Operator* is a tuple $S{-}MBO = (S, \mathcal{V}, \mathcal{T}, \delta, \gamma, \psi, \varphi, c, O)$ where

- S is a finite set of sensors,
- $\mathcal{V} \subset \mathbb{R}$ is a set of (sensors) values,
- $\mathcal{T}$ is a set of time-points,
- δ is a time interval that is used to synchronize values,
- $\gamma : S \times \mathcal{T} \to \mathcal{V}$ is a partial mapping that assigns a value to each sensor at each time point,
- ψ is the spatial integration function,
- φ is a (user-defined) mapping that returns the execution confidence $c \in [0, 1]$ based on mappings γ and ψ, and
- O is a set of outputs.

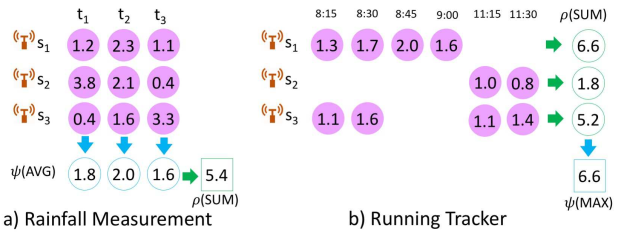

Fig. 2: Two Applications of Spatio-temporal Operators

C. *Spatio-Temporal Model-based Operators*

The last two model based operators are defined to integrate data over both spatial and temporal domains where the system is composed of n sensors, s_1, s_2, ..., s_n and each sensor s_i sends a set of values $v_{i1}, v_{i2}, ..., v_{im}$ over m time-points within a time window. We define two different operators $ST{-}MBO$ and $TS{-}MBO$. In $ST{-}MBO$, the operator first integrates data over the spatial domain (using function ψ) and then over the temporal domain (using function ρ). Whereas, in $TS{-}MBO$, the operator first integrates data over the temporal domain and then over the spatial domain. The confidence c is also returned by a user-defined function φ using the values returned by ψ and ρ.

In $ST{-}MBO$, a function ψ integrates data at each time-point t_i and returns some value v_i. Theses values are then integrated over the temporal domain using a function ρ. Then, a user-defined function φ is used to determine confidence c. Similar to Spatial operator, a time interval δ is also defined to handle non-synchronized sensor values (value v is considered to be received at time-point t if v arrives in $[t{-}\delta, t{+}\delta]$).

Definition: A *Spatio-Temporal Model-based Operator* is a tuple $ST{-}MBO = (S, \mathcal{V}, w, \mathcal{T}, \gamma, \delta, \psi, \rho, \varphi, c, O)$ where

- S is a finite set of sensors,
- $\mathcal{V} \subset \mathbb{R}$ is a set of (sensor) values,
- w the length of each time window,
- $\mathcal{T}$ is a set of time-points,
- $\gamma : S \times \mathcal{T} \to \mathcal{V}$ is a partial mapping that assigns a value to each sensor at each time-point,
- δ is a time interval that is used to synchronize values,
- ψ is a (spatial) function that integrates values at each time point and returns a single value,
- ρ is a (temporal) function that integrates values resulted from ψ and returns the final value,
- φ is a (user-defined) function that returns the execution confidence $c \in [0, 1]$ using the values from functions ψ and ρ, and
- O is a set of outputs.

Here since we use both ρ and ψ functions, two classes of applications can be combined. However, the combination of different application classes might not be always meaningful, e.g., if ψ is a thresholding function (returns *true/false*), then ρ cannot be a thresholding or an aggregation function.

Figure 2(a) shows a rainfall measurement application where a set of sensors are distributed in different regions. For simplicity, we consider a set of 3 sensors ($S = \{s_1, s_2, s_3\}$) and one time window consisting of time-points t_1, t_2, and t_3. Let ψ be a function that returns the average rainfall at each time-point (since the sensors are homogeneous, we can have such a function), so $\psi(t_1) = 1.8(mm)$, $\psi(t_2) = 2.0$, and $\psi(t_3) = 1.6$, and ρ be a sum function over the values resulted by ψ, so ρ returns 5.4 that means, on average, we had 5.4 mm rainfall per region.

$ST{-}MBO$ is useful for applications where all sensors send values in all the defined time-points. In such situation, the ψ function can be computed for each time point as soon as its values are received and does not need to wait till the end of the time window. However, if each sensor sends data in some (not all) of the defined time-points, integrating data at each time point might not be possible. To capture those situations, $TS{-}MBO$ is introduced. $TS{-}MBO$ is defined similar to $ST{-}MBO$, except it changes the order of execution of ψ and ρ. First, ρ integrates data received from each sensor over the temporal domain and then ψ plays its role to integrate the resulted values of ρ and return the final value.

Figure 2(b) represents a running tracker application that keeps track of running distances per day and returns the most active runner among a group of users. This value could be used later for some health care analysis. $TS{-}MBO$ can be used in this scenario. For simplicity we consider a group of 3 users (s_1, s_2, and s_3) and sensors send data every 15 minutes. Here each user's device has a sensor and the time window duration is a day. The figure only shows the time points that data is received from any of the users. As can be seen user 1 runs from 8 to 9 at morning (the values are sent at time-points 8:15, 8:30, 8:45 and 9:00), user 2 runs from 11 to 11:30, and user 3 runs from 8 to 8:30 at morning and then 11 to 11:30. Let ρ be a sum function, so $\rho(s_1)$= $6.6(miles)$, $\rho(s_2)$= 1.8, and $\rho(s_3)$= 5.2, and ψ be a "Max" function over the values resulted by ρ, so ψ returns 6.6.

D. *Delayed or Missing Sensor data*

As described earlier, one of the major challenges for sensor data integration is that some of the values needed for integration of sensor values can be delayed or missing. As M-DB employs model-based operators to encode the computations needed by the applications, it is well suited to handle delayed or missing sensor data. To deal with delayed or missing data we can either ignore the missing values or use statistical Prediction Models.

In the presence of a thresholding function, the model-based operator can ignore the missing data and continue processing assuming that the missing data does not satisfy the threshold. The confidence of the model-based operator execution is then used to reflect the uncertainty of the underlying data and decisions. We can ignore the missing values in statistical aggregation operators as well. For example, if the aggregation function is average, the operator returns the average of the received values (and ignores the missing ones). Similar as before, the confidence of the model-based operator execution can be used to reflect the uncertainty of the underlying data.

A more appropriate approach to deal with delayed or missing data is to use statistical prediction models. A prediction model can be used to predict any missing input to the node at a particular time stamp with a probability. If some of the values are missing or delayed, and do not arrive up-to a defined grace period (δ) after the periodic computation interval, the operator is executed using the predicted values.

For event detection applications, the prediction model returns the estimation of the predicted value satisfying the threshold, i.e., for each missing value, the prediction model returns a probability. In that way function ρ returns *true* if the number of values that satisfy threshold τ based on the operator σ plus sum of the probabilities that are returned by the prediction model for missing values over the total number of values (received and missing) is greater than or equal to percentage p. Similarly, we can define ψ, with the above technique used for computing each of the thresholds in the set of thresholds (one per sensor).

For statistical aggregation as well as general-purpose applications, the prediction model estimates the missing values where for each missing value, the prediction model returns a set of intervals $[\alpha_i, \beta_i]$ with the probability of each interval. These interval values and the concrete (received) values are then integrated using the function ρ or ψ.

IV. MDB

M-DB is a data processing architecture, combining model-based operators to continuously integrate and execute real-time computations on diverse sensor data. In this section M-DB's major components: M-Stream and M-Store, are described.

A. *M-Stream*

M-Stream is a computational pipeline, that provides the ability to write the business logic of IoT applications. The computation pipeline is represented as a directed acyclic graph. Each node in the graph represents a Model-based Operator (MBO) computation. For a particular node in the graph, any of the four model-based operators defined in Section III can be used. Each node has incoming and outgoing edges. The incoming edges are inputs to the model-based operator, which can be inputs from external sensors, from other model-based operators or from non-sensor outputs. The outgoing edges are the outcome of the model-based operations: the outputs of the operators and the confidence in the computation. M-Stream accesses M-Store for processing input and output values.

Each model-based operator can perform either a spatial, temporal or a combined spatio-temporal integration of sensor values. Each model-based operator is computed continuously after a defined time period. After the pre-defined period, each MBO pulls the required inputs from M-Store and is executed. A pull request to M-Store comprises the timestamp of the request, type of the model-based operator along with the sensor-id of the sensor to be read. Once the inputs are read, the operator is computed, and the confidence in the operator execution is produced as output. This can be an input to other model-based operator executions. Additionally, the operator execution can have other outputs, like writing to M-Store. The reading and writing can be executed as a transaction to ensure that other concurrent operations accessing the datastore have not modified the accessed items. Transaction isolation guarantees might be required for applications.

B. *M-Store*

M-Store gives access to input data in M-Stream. M-Store is responsible for managing sensor, as well as non-sensor data, accessed by computations in M-Stream. M-Store provides an interface to access sensor data. M-Stream can access the data from a sensor using M-Store's interface. By employing a separate datastore interface, M-Store can expose the underlying uncertainty of the data, and incorporate prediction models for missing or delayed sensor data. Model-based operators can handle such uncertainty and M-Stream's dataflow then ensures that the uncertainty flows through the computations.

M-Store exposes an interface for reading and writing values. M-Store internally manages data as key and values. Both sensor and non-sensor data items have a key associated with them. Additionally, sensor data items also have a timestamp associated with them. M-Store uses the timestamp to ascertain whether a particular sensor value is delayed or missing. For a non-sensor data item, each key only has one value associated to it. A sensor data item can have multiple values associated to the key, indexed by the timestamp of the value. Values associated to a sensor data item also have a configurable expiry period, after which they can be garbage collected.

Each read request to M-Store comprises keys to be read, and the timestamp of the request. Read requests from M-Stream will query with the timestamp corresponding to when a particular model-based operator is invoked. For non-sensor values, the timestamp is ignored and the data associated to the key is returned. For reading a sensor-value, M-Store checks whether there is a value which is within the defined δ period of the timestamp requested. If so, the corresponding value is returned. Otherwise, the value is considered as missing.

Missing values are treated either by ignoring these values, or by using a prediction model to fill them in, as specified in Section III-D. If a prediction model is employed by M-Store, then the processing of missing values also depends on the type of model-based operation. The type of the operation, i.e. whether it is a thresholding or an aggregate operation, determines the value to the returned. In case of a thresholding operation, the model returns the probability with which the predicted value satisfies the threshold. For an aggregation operator, the model returns a set of confidence intervals and the probabilities associated with the interval.

M-Store specifies the interfaces of the prediction model, to support the two different operators described above. Application developer can implement and integrate any prediction model to handle missing values. For example, to return the probability of a missing value satisfying a threshold, a developer might implement a weighted-moving average over a defined period to predict the missing values corresponding to

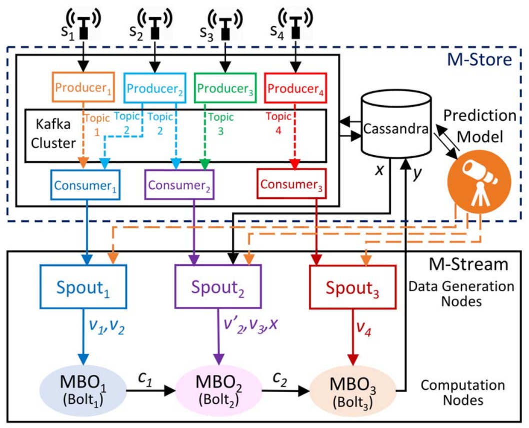

Fig. 3: M-DB Implementation over Apache Storm and Kafka

a particular sensor. As M-Store internally stores timestamp-indexed values, the implementation can use the sensor values from the past to predict the current value. If no prediction model is specified, the values are ignored, and the model-based operator is executed as described in Section III-D.

Each write request to M-Store comprises the key and the corresponding value to be written, and the timestamp of the request. Each write to a sensor key creates a new version indexed by the timestamp. Whereas, for a non-sensor key, the timestamp is ignored and only a single version of the value is maintained. The writes to a non-sensor key from a particular MBO execution in M-Stream, can be transactional. A transactional write request also specifies the keys and corresponding values read as input to the MBO execution. The write in this case would ensure that the inputs read have not been modified. If the items read have been modified, the transaction is considered to be failed, and is aborted.

V. M-DB's Design and Implementation

M-DB is architected and implemented over Apache Storm [4], [37], Kafka [3], [25] and Cassandra [2], three open-source systems. Developing the computation framework of M-DB over a mature distributed stream processing system like Storm enables use of basic stream processing constructs, and provides applications the ability to define topologies using Storm's APIs. It also enables the use of Storm for scheduling tasks and jobs in a distributed setting. Kafka and Cassandra are used for processing and storing incoming sensor data.

Figure 3 shows the implementation of the M-DB architecture in Figure 1 where the components of M-Stream and M-Store are implemented using Storm, Kafka and Cassandra. In this section, M-DB's implementation including M-Stream, M-Store, and Model-based operators, are described in detail.

A. *M-Stream*

M-Stream builds over the stream processing constructs of Storm. It combines model-based operators to define data processing. These model-based operators are implemented using stream transformation interfaces in Storm (*bolts*). The stream generator elements of Storm, *spouts*, are used to process sensor data from M-Store and emit them to the defined MBOs. Like in

Storm, a computation topology is used to specify the data processing pipeline. The topology comprises connections among the model-based operators and data generating elements.

M-Stream employs Storm for scheduling the execution of defined topologies. The topology of execution in M-Stream is a directed acyclic graph. The graph comprises nodes and the connections among the nodes. Each node defines a computation. The M-Stream framework consists of two different types of nodes: *Data Generation Nodes* and *Computation Nodes*.

1) Computation Nodes

Each computation node is defined by a model-based operator computation. M-Stream implements computation nodes using *bolts* in Storm. Each computation node receives stream of tuples as input, which are MBO inputs. Figure 3 shows three computation nodes mbo_1, mbo_2, and mbo_3.

Each computation node receives the tuples from other data generation and computation nodes and continuously executes the defined operation after every *computation interval.*

The *outgoing* edges of the computation node constitute a stream combining confidence of the execution, and other continuous values (streaming outputs). Apart from the streaming outputs, the operation execution can also have outputs which are to be written to M-Store.

2) Data Generation Nodes

Data generation nodes are the nodes responsible to process and read sensor data from M-Store. The data generation nodes are implemented using the spouts interface in Storm. The application developer defines a data generation node for each computation node by specifying the sensors values to be read, and a periodic time period after which the data generation node will send a read request to M-Store's interface. The data generation node execution periodically reads the specified sensors' values, and the corresponding timestamps from M-Store and emits them to the corresponding computation node. Figure 3 shows a M-Stream comprising three data generation nodes, i.e, $Spout_1$ to $Spout_3$, reading and emitting data periodically. For example, $Spout_1$ reads values v_1 and v_2 coming from the first two sensors and sends them to MBO_1.

B. M-Store

M-Store manages sensor and non-sensor data. M-Store's implementation has three main components: incoming sensor data management, persistent datastore and a prediction model.

M-Store exposes an interface where the incoming sensor data from the devices, or continuous inputs produced from other systems, can be written and read. Internally, M-Store employs Kafka for ingesting and accessing the incoming data from sensors. Data is ingested through Kafka producers. Kafka producers write sensor data to different Kafka topics, with each topic corresponding to a particular sensor. The read requests for sensor values from the data generation nodes are served by M-Store through the Kafka consumers. Consumers read sensor data from the appropriate topic for each sensor.

As described in Section IV, M-Store stores data as key and values. Additionally, the sensor values are also indexed with timestamps. For persisting such key and values, M-Store employs Cassandra [2], an open-source persistent NOSQL key-value store. The Kafka producer ingests sensor data into Cassandra, and the consumer reads the data.

For the prediction model, M-Store provides an interface to the different accesses that needs to be supported by the prediction model (for event detection and statistical aggregation operations). By default, M-Store's implementation predicts the values using a weighted moving-average scheme.

C. Execution of Model-based Operators

Each computation node comprises a model-based operator. The model-based operator is executed periodically every computation interval, which is defined by the application developer when instantiating the computation node. The model-based operator might be integrating values from multiple sensors (spatial), a single sensor value over time (temporal), or multiple sensors over a time period (spatio-temporal).

The computation node continuously gets the data from other computation nodes, and from data generator nodes, which pull the data from M-Store. Depending on the type of the model-based operator, computation nodes integrate the data. Note that for T, TS and ST MBOs, values corresponding to the time period of the integration are stored in-memory at the computation node. Once the inputs have been processed, the model-based operation is executed and generates its outputs.

Before emitting the streaming outputs, a defined set of the outputs are written to the datastore in M-Store. If the processing of the non-sensor outputs is specified as transactional, the outputs are written only if the transaction isolation guarantees are satisfied. The transactional guarantees are provided using the compare-and-set guarantees in Cassandra. If the transactional writes fail, then the model-based operator execution is considered unsuccessful. After the model-based operator execution, the computed confidence of execution and the streaming outputs are sent to the next computation nodes

VI. Evaluation

Depending on the context, IoT applications differ in their settings in terms of the number of sensors, the required amount of computations, and the frequency of sensor data arrival. Extensive experimental evaluation is carried out to study M-DB's performance in various configurations.

M-DB is implemented over Apache Storm, Kafka and Cassandra, and deployed over a cluster of machines. Kafka topics are also distributed across different machines. Each machine in the cluster runs its own Cassandra instance. A data generator utility is implemented to simulate sensors. The data generator generates data representing a set of sensors continuously sending data at a defined time-interval. It sends the input data to M-Store. Internally, these requests are written to Kafka producers, as described in Section V-B.

M-DB is evaluated with varied MBO topologies and different input sensor configurations. Four different sets of experiments are performed. In the first scenario, we employ a topology where each MBO has multiple input sensors, and then vary the frequency of sensor data arrival. In the second scenario, a topology with a relatively smaller number of

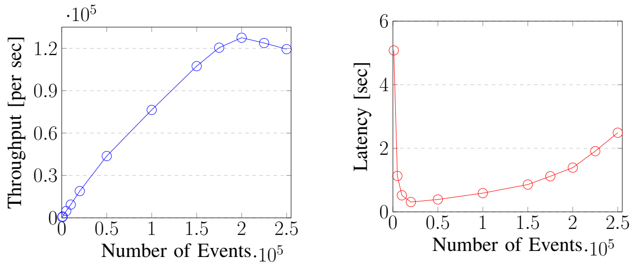

Fig. 4: Measuring (a) Throughput and (b) Latency with varying the frequency of events

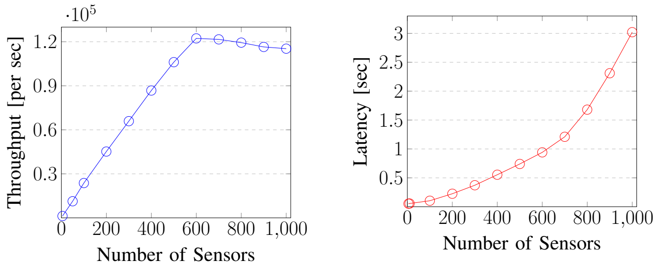

Fig. 5: Measuring (a) Throughput and (b) Latency with varying the number of sensors

model-based operators is employed, and the number of sensors are varied. The third scenario evaluates the scalability of M-DB by increasing the number of machines while the number of sensors, the frequency of sensor data arrival, and the topology are fixed. Finally, the performance of M-DB is measured in the presence of missing values. To this end, we employ the simple prediction model implemented in M-Store (Section V-B). To simulate missing data, we modify the data generator utility to skip sending a defined percentage of sensor values.

The experiments measure both the throughput, in terms of the number of sensor events processed, and the latency of M-Stream and M-Store. The experiments are conducted on a cluster of machines where each machine has 8-core Intel Xeon E31235 processor clocked at 3.20 GHz and 16 GB of RAM.

Unless otherwise mentioned, we employ a topology with 90 sensors and 10 MBOs, one spatial, eight spatio-temporal (TS and ST), and one temporal, which are connected to each other serially. The first node is a spatial MBO that receives data from 10 sensors and also reads a non-sensor tuple from Cassandra. The next eight MBOs are spatio-temporal ones where each MBO receives data from 10 homogeneous sensors and the previous MBO. The last operator is a temporal one that gets input values from the ninth operator. Each sensor value is input to only one model-based operator. The first spatial operator uses the average function for integration. The eight spatio-temporal operators employ a combination of aggregation and thresholding functions, and the temporal MBO uses a thresholding function. The framework is deployed on a cluster of five machines and the model-based operators are equally distributed on the cluster.

A. Varying the Frequency of Data Arrival

In this experiment, the interval of sensor data arrival is varied from 1000 ms to 2 ms to increase the number of events. The duration of each experiment is 60 seconds. The computation interval of the model-based operators are the same as the time interval of the sensors and the time window for all spatio-temporal and temporal operators is 6 times the time interval of sensors. As a result, the number of events increases from around 500 to $250,000$.

Figure 4(a) shows the throughput of M-DB when the total number of events is increasing. As can be seen, the system can process upto $127K$ events per second.

Figure 4(b) represents the average pipeline latency to process a tuple. Note that as the computation interval of the model-based operators is the same as the time interval of the sensors, by increasing sensor data arrival frequency, the latency should decrease. For example, if sensors send data every 1 second, ideally, it takes 10 seconds for a tuple to go through all 10 operators. However, if the sensors interval becomes 10 ms, the ideal computation will take 100 ms. Therefore, the latency decreases till a point and then increases due to a higher number of unprocessed events. The results illustrate that M-DB achieves sub-second latency even for $160K$ events, efficiently supporting a high frequency of sensor data arrival.

B. Varying the Number of Input Sensors

In this experiment, we use a simpler topology with five spatio-temporal model-based operators and vary the number of sensors from 5 to 1000, distributed equally between the operators. The sensor data arrival frequency is set to 8 ms and the time window of the model-based operators is 16 ms (at each time window two values are received from each sensor). Thus, the number of events increases from 1250 to $250,000$.

Figure 5(a) shows the system throughput when the number of sensors (and as a result the total number of events) increases. As the number of input sensors processed increases, the throughput increases upto 600 sensors, to a value of $122K$ events per second. Further increasing the number of sensors does not increase throughput due to the increased load.

Figure 5(b) illustrates the event processing latency. Unlike the previous experiments, since the computation interval of the model-based operators does not change, increasing the number of sensors increases the latency. However, upto 620 sensors the framework has sub-second latency which illustrates that M-DB is able to efficiently support a large number of sensors.

C. Varying the Number of Machines

The number of machines are varied from 1 to 10 to evaluate the performance of M-DB while scaling-out. The topology comprises of 10 MBOs as before, distributed equally among the machines. The interval of sensor data arrival is $2ms$ and the time window of the model-based operators is set to $8ms$ (in total $200,000$ events are generated each second).

Figure 6(a) illustrates that by scaling-out, the throughput of M-DB increases upto 181 thousand events per second.

Figure 6(b) shows the event processing latency. Since the sensors value arrive at a very high frequency, a small number of machines leads to overloading and the latency increases due to the unprocessed events. Increasing the number of machines distributes the load between them and decreases the latency.

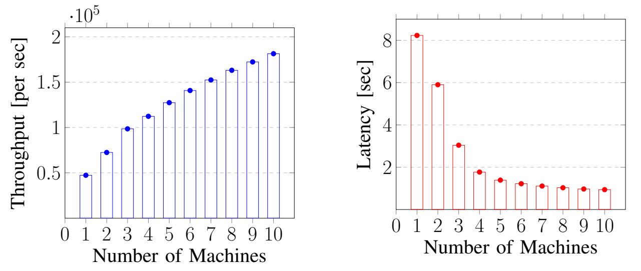

Fig. 6: Measuring (a) Throughput and (b) Latency with varying the number of machines

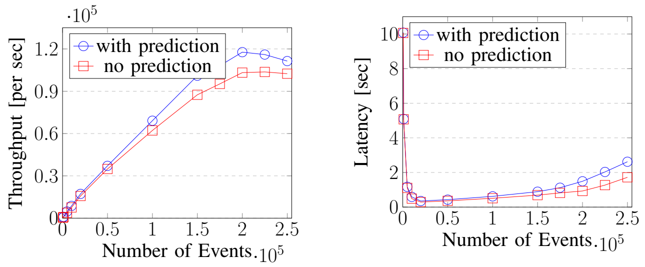

Fig. 7: Measuring (a) Throughput and (b) Latency in the presence of a prediction model

D. Employing a Prediction Model

M-DB's performance is now studied in the presence of missing values. The prediction model predicts a missing value using a weighted moving average over the past values. We repeat the experiments for varying the data arrival frequency, with the data generator utility skipping 20% of the values.

Figure 7(a) presents the throughput of the system. The prediction model helps the framework to process more events by filling-in the missing values. Each predicted value goes through a number of MBOs, thus the number of events and consequently, the throughput is increased. Note that the number of events here is the expected number of events, in the absence of any missing values. Figure 7(b) shows the average pipeline latency to process a tuple in the topology. While with small number of events the behaviour of both approaches is similar, by increasing the number of events, in the presence of a prediction model, the latency increases more. This is expected because the model predicts the missing values and as a result the framework has to process more events (as shown in 7(a)), resulting in increased latency. In addition, the output of the model using the prediction model is comparable to the output of the same model with no missing sensor values.

VII. Related Work

M-DB is motivated by and related to a wide range of research in the areas of database procedures, stream processing, dataflow models and wireless sensor networks.

Traditional database systems expose transactions [19] as abstractions to the application developer. The developer can express that an event would be considered executed only if all the sub-events comprising the transaction take place atomically. Stored procedures [10] are used by developers to express a set of functions to be executed on the data. They allow the expression of the business requirements and lead to consolidation of logic. Triggers [33] allow a set of functions to be executed, in response to a certain event, like data insertion. Model-based operators, like stored procedures and triggers, are executed when a set of defined conditions are met, and allow for code-reuse, ease in programmability, and pushing the computation to the data management layer. However, each model-based operator execution also outputs a confidence value to reflect the uncertainty of the underlying data and is designed to deal with uncertain sensor data.

Stream processing architectures [11], [26], [37] have been employed to perform continuous real-time data processing. Systems like Storm [26] use a dataflow architecture and push-based processing, to perform computation and push the results on data arrival. However, Storm can lead to blocking processing when data is delayed or missing, and can not reflect the underlying sensor data uncertainty in computed results. M-DB builds on the stream-processing model and defines each node in the dataflow graph to be executed at defined time periods, so as to not block for missing or delayed sensor values. M-DB is implemented on top of an open-source distributed stream-processing architecture like Storm, so as to reuse the basic dataflow constructs and communication between components. Architectures to combine traditional transaction processing and stream processing have been proposed as well [16]. Like the traditional stream processing architectures, they are not suitable for handling the underlying uncertainty of sensor data.

Akidau et al. [13] propose the dataflow model, which deals with unbounded data by providing trade-offs between correctness, latency, and cost. Dataflow model handles delayed or missing data, by introducing triggers and separating it from defined windows. Using triggers, applications can define when to emit results (based on estimates of data processed, or at particular time periods etc). The model provides the ability to retract or discard old results when the entire data comes in. This model is the basis of Apache Beam [1] and Google Cloud Dataflow [6]. M-DB also aims to produce real-time results, while dealing with uncertain data, by providing developers with model-based operators abstraction. These operators can reflect the uncertainty in the underlying data used for computation, which can then flow through the data-processing pipeline. Furthermore, by having a datastore interface, M-DB allows different prediction models to be employed for filling-in delayed or missing sensor data. Some tailored solutions have also been proposed for dealing with uncertain data streams for applications like clustering [23] and skyline queries [41].

Macrobase [15] is an analytical engine which combines streaming and learning capabilities to detect anomalies in data streams. Macrobase exposes transformation, classification and explanation operators, and focuses on learning techniques to automatically analyze outliers. Analogous to model-based operators, these analytic operators are aimed at providing abstractions over high frequency data streams.

Previous works in the context of wireless sensor networks have proposed techniques to optimize communication with sensor devices for energy efficiency. Techniques have been proposed for probabilistic event detection [12], model-based sensor data acquisition [17], [18], [29], [34] and approximate querying over sensor data [18], [38]. Traub et al. [39] tailor data streams for supporting real-time analytics, by optimizing the data transfer from sensor devices. They propose user-defined sampling functions based on data-demand from queries and multiple queries share sensor reads and data transfers. Trigoni et al. [40] propose a hybrid push and pull approach for data dissemination from sensor nodes. Sensor data is proactively pushed into gateways nodes and then pulled based on queries. Guo et al. [20] propose a interval index to efficiently query over modeled sensor data. Many of these techniques are complementary to M-DB. Data transfer reduction techniques [39], [40] can by used in conjunction with MBO definitions, based on certainty of results desired. As M-DB virtualizes a sensor in M-Store, the proposed probabilistic models [18], [27] could be used to design statistical models to provide probabilistic values of missing sensor data.

Several other optimizations have been proposed to support IoT data management: native integration of signal processing operators [31], faster ingestion of time-series data into analytical systems [30], data cleaning support [24] and fusion of time-series DB writes with relational queries [22].

VIII. Conclusion

In this paper, we propose M-DB, a real-time data-processing and monitoring framework for IoT applications. On one hand, M-DB proposes abstractions for IoT application developers to integrate data from diverse sensors, and on the other hand, it deals with the uncertainty of the underlying data. To provide abstraction, different types of model-based operators are introduced for temporal, spatial, or spatio-temporal integration of sensor values to support event-detection and aggregation operations. M-DB deals with the sensor data uncertainty in two ways. First, a confidence value that expresses the uncertainty of computational results is assigned to the model-based operator computation. These confidence values flow along the computation pipeline (M-Stream). Second, M-Store, a storage layer separating the computation of business logic from the physical sensor data management, is introduced. M-Store employs a prediction model to predict the missing or delayed sensor data.

Acknowledgement

This work is funded by NSF grants CNS-1703560 and CNS-1815733.

References

[1] Apache Beam. https://beam.apache.org/.

[2] Apache Cassandra. http://cassandra.apache.org/.

[3] Apache Kafka. http://kafka.apache.org/.

[4] Apache Storm. http://storm.apache.org/.

[5] Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. http://www.gartner.com/newsroom/id/2636073.

[6] Google Cloud Dataflow. https://cloud.google.com/dataflow/.

[7] MySQL. https://www.mysql.com/.

[8] Oracle 12c. http://www.oracle.com/us/corporate/features/database-12c.

[9] Sensor Fusion. http://www.mouser.com/applications/sensor-fusion-iot/.

[10] Stored Procedures. https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine, 2017.

[11] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, et al. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[12] D. J. Abadi, S. Madden, and W. Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *VLDB*, pages 769–780, 2005.

[13] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *VLDB*, 8(12):1792–1803, 2015.

[14] V. Arora, F. Nawab, D. Agrawal, and A. El Abbadi. Multi-representation based data processing architecture for iot applications. In *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 2234–2239, 2017.

[15] P. Bailis, E. Gan, S. Madden, et al. Macrobase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556. ACM, 2017.

[16] U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, J. Meehan, A. Pavlo, et al. S-store: a streaming newsql system for big velocity applications. *VLDB*, 7(13):1633–1636, 2014.

[17] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*, pages 48–48, 2006.

[18] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, et al. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[19] J. Gray and A. Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

[20] T. Guo, T. G. Papaioannou, and K. Aberer. Model-view sensor data management in the cloud. In *Big Data*, pages 282–290. IEEE, 2013.

[21] I. Horrocks, M. Giese, E. Kharlamov, and A. Waaler. Using semantic technology to tame the data variety challenge. *IEEE Internet Computing*, 20(6):62–66, 2016.

[22] S. Huang, Y. Chen, X. Chen, et al. The next generation operational data historian for iot based on informix. In *Proc. of SIGMOD*, pages 169–176. ACM, 2014.

[23] C. Jin, X. Yu, A. Zhou, and F. Cao. Efficient clustering of uncertain data streams. *Knowledge and Information Systems*, 40(3):509–539, 2014.

[24] A. Karkouch, H. Mousannif, H. Al Moatassime, and T. Noel. Data quality in internet of things: A state-of-the-art survey. *Journal of Network and Computer Applications*, 73:57–81, 2016.

[25] J. Kreps, N. Narkhede, J. Rao, et al. Kafka: A distributed messaging system for log processing. In *NetDB*, 2011.

[26] S. Kulkarni, N. Bhagat, M. Fu, Kedigehalli, et al. Twitter heron: Stream processing at scale. In *ACM SIGMOD*, pages 239–250, 2015.

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

[28] M. Lippi, M. Mamei, S. Mariani, and F. Zambonelli. Coordinating distributed speaking objects. In *ICDCS*, pages 1949–1960, 2017.

[29] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 30(1):122–173, 2005.

[30] J. Meehan, C. Aslantas, S. Zdonik, N. Tatbul, and J. Du. Data ingestion for the connected world. In *CIDR*, 2017.

[31] M. Nikolic, B. Chandramouli, and J. Goldstein. Enabling signal processing over data streams. In *proc. of SIGMOD*, pages 95–108, 2017.

[32] K. Patroumpas and T. Sellis. Window specification over data streams. In *EDBT*, pages 445–464. Springer, 2006.

[33] G. J. Ramakrishnan, Raghu. *Database management systems*. McGraw Hill, 2000.

[34] S. Sathe, T. G. Papaioannou, H. Jeung, and K. Aberer. A survey of model-based sensor data acquisition and management. In *Managing and mining sensor data*, pages 9–50. Springer, 2013.

[35] E. M. Schooler, D. Zage, J. Sedayao, et al. An architectural vision for a data-centric iot: Rethinking things, trust and clouds. In *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 1717–1728, 2017.

[36] Z. Shen, V. Kumaran, M. J. Franklin, et al. Csa: Streaming engine for internet of things. *IEEE Data Eng. Bull.*, 38(4):39–50, 2015.

[37] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, et al. Storm@ twitter. In *ACM SIGMOD*, pages 147–156, 2014.

[38] T. T. Tran, L. Peng, Y. Diao, et al. Claro: modeling and processing uncertain data streams. *VLDB*, 21(5):651–676, 2012.

[39] J. Traub, S. Breß, T. Rabl, A. Katsifodimos, and V. Markl. Optimized on-demand data streaming from sensor nodes. In *SOCC*, pages 586–597. ACM, 2017.

[40] N. Trigoni, Y. Yao, A. J. Demers, J. Gehrke, and R. Rajaraman. Hybrid push-pull query processing for sensor networks. In *GI Jahrestagung (2)*, pages 370–374, 2004.

[41] W. Zhang, A. Li, et al. Probabilistic n-of-n skyline computation over uncertain data streams. *WWW*, 18(5):1331–1350, 2015.