A Real-time and Energy-aware Framework for Data Stream Processing in the Internet of Things

Egberto A. R. de Oliveira[1], Flavia C. Delicato[2], Atslands R. da Rocha[3] and Marta Mattoso[1]
[1]*PESC/COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil*
[2]*Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brazil*
[3]*Universidade Federal do Ceará, Fortaleza, CE, Brazil*

Keywords: IoT, Internet of Things, Data Streams, Data Stream Processing, Edge Computing, Adaptive Sampling.

Abstract: The Internet of things (IoT) has transformed the internet, enabling the communication between every kind of objects (things). The growing number of sensors and smart devices increased the possibilities of data generation and collection. This led to an explosion of data streams being produced which are challenging to be processed in real-time. Regarding the nature of the data, the huge volume, heterogeneity, continuity, disordering, noise and unpredictable rate are some challenging aspects to tackle. Regarding the data processing, the core activities from the data acquisition to the production of high-level knowledge also pose challenges related to limited computational and energy resources and high network latency. In this context, we propose a framework to support activities of a data stream processing workflow for IoT. It aims allowing real-time data processing with low power consumption. Edge computing is used to bring the data processing closer to the data sources and allow actions to be triggered quickly. An adaptive sampling strategy combined with a data prediction model are adopted to reduce the network traffic, thus decreasing the power consumption of the network devices. Experiments show that the proposed framework is able to achieve up to 60.58% average energy consumption savings to sensor nodes and still meet a strict execution time threshold of 1s without compromising the accuracy of the output data on different scales of input streams.

1 INTRODUCTION

The Internet of things (IoT) is transforming the internet, enabling the communication between every kind of object (things) and creating a vision of "anytime, anywhere, any media, anything" communications (Atzori et al., 2010). Initially, IoT was mainly obtained by the use of RFIDs, nowadays such a concept has evolved to a broader view that refers to the interconnection of sensors, actuators, smart objects, and wireless sensor networks (WSN)(Akyildiz et al., 2002). The growing number of sensors and smart devices led to an explosion of volume, variety and velocity of generated data, empowering a new way of value creation to people and corporations (Dias de Assunção et al., 2018). The processing of these "firehoses" of data from existing and emerging applications poses several challenges and brings novel research opportunities.

The challenges involved in IoT data stream processing may be analyzed at least from two dimensions: (i) the data itself (generated by heterogeneous, distributed and often constrained devices), and (ii) the data processing, i.e. the core activities from the data acquisition to the production of high level knowledge.

Regarding the nature of the data, IoT devices/sensors generate, possibly in a continuous way, a huge amount of data, typically consisting of time-series values, which are sampled over a specific time period, thus characterizing a data stream (Karkouch et al., 2016). Often, there is no control over the order or frequency of streamed data, which is transient or non-persisted. The input rate of a data stream is unpredictable and bursty in nature, ranging from a few bytes to several gigabits per second. In addition, the data is highly heterogeneous, as it is generated by multiple types of devices, in different formats and to feed a wide range of applications, also heterogeneous. Besides the potentially massive volume of data, an IoT environment is also characterized by high dynamism and volatility. In many IoT applications, such as traffic accident monitoring or river flooding prediction, the potential value of data depends on its timely processing, under strict time

R. de Oliveira, E., Delicato, F., R. da Rocha, A. and Mattoso, M.
A Real-time and Energy-aware Framework for Data Stream Processing in the Internet of Things.
DOI: 10.5220/0010370100170028
In *Proceedings of the 6th International Conference on Internet of Things, Big Data and Security (IoTBDS 2021)*, pages 17-28
ISBN: 978-989-758-504-3

requirements. Otherwise, the processing results and actions become less valuable or even worthless. Finally, quality-related data features also need to be considered. According to Qin et al. (Qin et al., 2016), data quality in IoT considers the following features: (i) uncertainty, (ii) ambiguity and inconsistency, (iii) incompleteness, and most of them are a direct consequence of the data being produced by sensors. Sensors are fail-prone devices. Information and decisions derived from raw data generated by sensors will also be subject to failure (Klein and Lehner, 2010). Therefore, identifying errors/inconsistencies on a sensor generated data stream is crucial to improve the accuracy of the data being processed. These errors/inconsistencies are called outliers, which are readings considered outside the regular state of the data being collected. Data points that differ significantly from others in a data set can represent either errors or events of importance to the application (Karkouch et al., 2016).

Regarding the data processing, the demand for computational resources capable of processing large volumes of data has historically been an obstacle for creating high volume and/or high speed data processing solutions (Dautov et al., 2018). Because of the huge availability of resources offered by cloud computing platforms, cloud-based approaches are widely adopted in IoT systems. The data is pushed to the cloud to be processed and the outcome is delivered back to the local system. However, the internet backbone is not always able to meet the real-time requirements to transport a huge amount of data coming at a high speed. This creates a communication bottleneck that leads to proposing non-cloud based alternatives, to handle and process IoT generated data streams(Janjua et al., 2019).

A promising approach recently emerged is Edge Computing (Dautov et al., 2018). It consists of bringing the data processing activities physically closer to the data sources. Edge computing is potentially useful and has been adopted in several domains such as smart buildings, healthcare, autonomous vehicles, and environmental monitoring. In these applications, data is processed by an edge device such as a smart gateway, to extract meaningful information from it and take necessary actions immediately (Janjua et al., 2019), thus preserving the usability of time-sensitive data. Therefore, with the support of edge computing, some of the activities of a data stream processing workflow can be performed by devices at the edge of the network. Other activities, more demanding in terms of processing, may continue to be carried out in the cloud. Still others can be performed on the sensor device itself, that is, on the data sources since, although restricted in terms of CPU and memory, such sensors are capable of performing less complex processing. In this context, another challenge arises related to data stream processing in IoT, which consists of using the available resources in a rational way. In addition to the restricted processing and communication capabilities, several sensor devices are powered by non-rechargeable batteries. Keeping the sensors working as long as possible is a major challenge in all sensing-based systems, and it has been extensively investigated in the WSN community.

In general, a basic strategy for preserving energy in WSNs consists of: (i) keeping the nodes in hibernation as long as possible and (ii) reducing the data traffic in the network as much as possible. One strategy called Adaptive Sampling (Anastasi et al., 2009) simultaneously tries to do both by varying the interval between samplings according to the behaviour of the sensed data. Adaptive sampling methods might not be suitable to sample physical phenomena with sudden variations, which is a common characteristic of data streams. However, this problem can be minimized when Adaptive Sampling is combined with a data prediction model to compute estimated future sensor readings (Monteiro et al., 2017).

In this work, we propose a framework which supports activities of a data stream processing workflow. The framework aims at addressing the challenges of real-time, power consumption and data accuracy. We adopt the edge computing paradigm to deal with the network bandwidth vs. data production bottleneck, allowing for applications with real-time requirements. We use adaptive sampling to reduce the network traffic, and, as a consequence, the power consumption of the sensor nodes. A data prediction model identifies and removes outliers producing an accurate aggregate output. The main contributions of the proposed framework are:

- To provide an energy-aware data gathering component with adaptive sampling to reduce the network traffic and, as a consequence, the power consumption of the sensor nodes;
- To develop a data prediction model which takes readings from multiple sensor nodes over a short predefined window as inputs, applies a density-based clustering algorithm to identify and remove outliers and produces an accurate aggregated output.

The major benefit expected by adopting the proposed framework is being able to deploy long running real-time processing systems on remote outdoor environments such as forests, open fields and watercrafts. In such environments, there is no access to continuous

sources of electricity thus requiring the use of batteries, solar panels or other types of limited power sources.

The rest of this paper is organized as follows. Section 2 presents relevant works which individually tackle the issues we aim to solve together. Section 3 describes the proposed framework in details. Experiments to evaluate the proposal is presented and discussed in Section 4. Finally, Section 5 concludes the paper and provides additional information about the ongoing and future work on this research.

2 RELATED WORK

Real-time/low latency response is an important and very frequent requirement in IoT and Data Stream Processing (DSP) applications (Li et al., 2015). A recent survey on solutions for real-time processing in big data streams (Mehmood and Anees, 2020) points out there has been a growing number of publications on this topic during the last 10 years. The study identifies in-memory computing, support to non-structured or semi-structured data, low latency and the usage of machine learning algorithms, among others, as key challenges on this field. The authors conclude that there is a lack of flexibility in the available solutions, since they are too specific for the use cases they were designed to tackle. In addition, the survey states that cloud-based approaches still represent the majority of current solutions analyzed. It is important to mention that the quality of the output data is always a major concern while energy consumption do not appear among the challenges or objectives of the evaluated solutions for real-time data stream processing.

Another recent survey on IoT architecture challenges (Samizadeh Nikoui et al., 2021) highlights energy efficiency as one of the major concerns when designing IoT systems. Time efficiency (real-time or near real-time responses) is also mentioned as an important and frequent requirement. Quality awareness can also be inferred as a key aspect for IoT architectures since data integrity and accuracy concerns are present in most of the described approaches. The study, in turn, does not explicitly correlate energy and time efficiency on the analyzed solutions in any way. In order to find this correlation, we cross checked the lists of publications addressing energy and time efficiency. From the 29 works listed on this survey, only 4 aim at tackling both energy and time efficiency together: the solutions proposed in (Xu and Helal, 2016) and (Catarinucci et al., 2015) are cloud-based data stream processing solutions that rely on fully offloading the collected data to the cloud. As already mentioned in section 1, this kind of approach leads to a communication bottleneck which makes it impossible to meet strict low latency requirements. The architecture described in (Loria et al., 2017) is an effective real-time stream processing solution which relies on a robust infrastructure of servers on the edge. It is not designed for constrained gateway devices and, in fact, it does not address energy efficiency of gateways or sensor nodes. OSCAR (Vučinić et al., 2014) is not related to data stream processing. It is an energy efficient architecture for real-time communication (machine-to-machine and multicast) focusing on security.

Despite relevant proposals addressing separately the issues of real-time responses, data accuracy or power consumption can be found in the field of data stream processing for IoT, to the best of our knowledge, no solution tackling these three concerns together has been found so far. This makes it difficult to deploy solutions that can efficiently respond to real-time events in power-constrained environments, such as a forest fire suppression system, a malfunction detection system on small ships, etc.

IRESE (Janjua et al., 2019) presents an outlier (so called "rare-event") detection system that applies unsupervised machine learning techniques at the edge to quickly identify events on audio data streams. Despite the significant results achieved in terms of data accuracy and real-time response, no concern regarding energy consumption is mentioned. Therefore, this solution might not be feasible on environments with limited power sources, which is a major concern in our work. In addition, IRESE was designed to handle a very specific type of data: audio streams. Our work is more agnostic and not restricted to a single data type or use case.

Dual prediction techniques are presented in (Monteiro et al., 2017), (Al-Hoqani and Yang, 2015) and (Gupta et al., 2011). They are combinations of adaptive sampling with data prediction models based on exponential time series. The core idea is based on a set of lightweight calculations performed at the WSN nodes. These computations allow the sensor nodes, instead of delivering only a single sensor reading to the sink, delivering a function that allows predicting sensor readings in the time interval between the current and the next reading. Data prediction models are inserted in this context in order to avoid jeopardizing the quality of the data being generated, by the decrease in the sampling frequency. The main goal of such a combined approach is finding a good balance between the energy consumption and the quality of the data being produced by a WSN. A common fea-

ture observed on these works that employ dual prediction schemes is the fact that their prediction models are applied individually by each sensor. The framework proposed in this paper differs from these works since it considers the readings of a group of sensors. It not only uses an aggregate function to predict data, but it also identifies and eliminates incorrect readings from its computations.

The works presented in this section propose effective solutions to address requirements of real-time responses, data accuracy or power consumption, but none of them tackle all these three concerns together. Combining these three requirements in the same solution is complex because the approach used to solve one problem can negatively impact the solution of another. For example, statistical methods based on intensive computing can efficiently solve the problem of lack of data accuracy but they demand a high energy consumption from the devices. The contribution of our work consists of combining approaches such as those described, promoting the necessary adaptations so that the three requirements are jointly met.

3 A FRAMEWORK FOR DATA STREAM PROCESSING IN IoT

There are many tools and platforms for ingesting, processing, storing, and managing data streams, making it a difficult task for professionals to select the right combination to perform their analysis. The authors in (Isah et al., 2019) identified the main components of a modern data stream processing system (DSPS), which can be integrated into a framework. A data stream processing framework can be considered as a cornerstone for guiding the building of DSPS, addressing all the activities involved in the stream processing workflow (Isah et al., 2019). In our work, we consider the general model for a DSPS framework proposed by (Isah et al., 2019). It includes (i) a data stream ingestion layer, responsible for accepting data into the DSPS; (ii) a data stream processing layer, which preprocesses and analyzes data in one or more steps; (iii) a persistence layer that stores, indexes and manages the data and the generated knowledge; (iv) a resource management layer, which coordinates the functions of provisioning resources for data processing and its communication with external applications; and (v) an output layer that directs the output data to services and applications.

3.1 Framework Components

In this work, we consider that IoT systems are organized in a three-tier architecture as described in (Li et al., 2017). In such architecture, the bottom is the Things tier and it comprises physical sensors, WSNs and embedded devices, responsible for collecting data from the monitored environment and feeding them to the IoT system. Next comes the edge/fog tier composed of devices located physically close to the things and responsible for less compute intensive tasks such as preprocessing the incoming data. The upper tier is the cloud, encompassing robust devices (data centers) capable of handling more compute intensive processing tasks and/or permanently storing relevant data (archiving).

Considering such an organization for IoT systems and the model for DSPS described in (Isah et al., 2019), in this work we propose a DSP framework (depicted in Fig. 1) to be deployed at the things and edge tiers. Its goal is to provide real-time and energy-aware data processing for IoT streamed data. In the figure, each physical entity (PE) denotes a physical quantity monitored by a group of physical sensing units (sensors - S) at the things tier. The Orchestrator is a software component placed at the edge tier. It in-

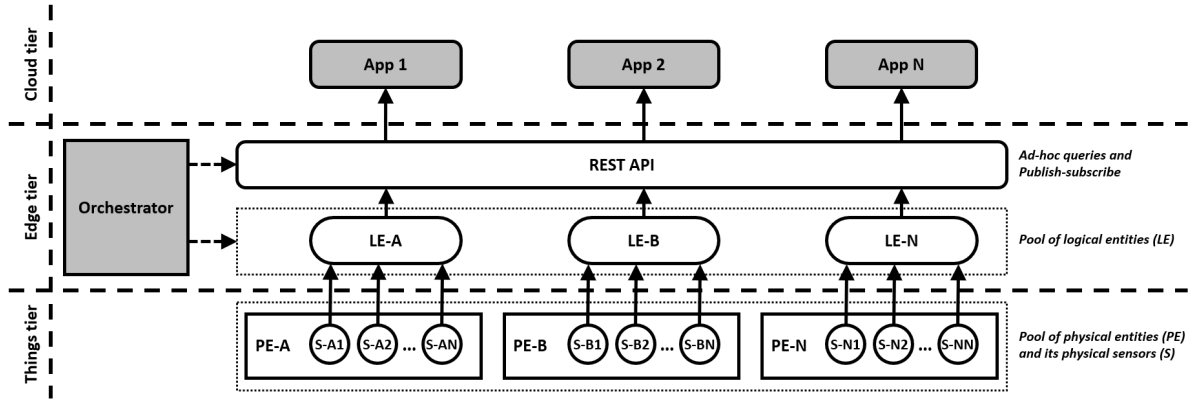

Figure 1: Schematic view of the proposed framework for real-time and energy-aware data stream processing.

stantiates logical representations of each physical entity, called logical entities (LE), and also provides a Representational State Transfer Application Program Interface (REST API) (Richards, 2006) to allow communication between LEs and applications. The sensor nodes at the Things tier push readings to LEs via Constrained Application Protocol (CoAP) (Shelby et al., 2014). Applications running at the cloud tier, or even at the edge tier, consume data from LEs via Hypertext Transfer Protocol (HTTP) (Fielding and Reschke, 2014). The proposed framework supports both synchronous queries and asynchronous patterns such as publish / subscribe (Eugster et al., 2003).

Figures 2 and 3 illustrate a use case of a single entity monitored by three physical sensors and serving three different applications. We use a simplified scenario to allow the graphic representation of the components and their interactions. The adoption of decoupled components and the possibility of hosting them using lightweight virtualization techniques (such as containers) make the proposed framework potentially scalable in terms of physical entities, sensors and applications. The software components that encompass the proposed framework are described as follows, along with their correspondence to the layers for data stream processing systems proposed by (Isah et al., 2019):

- *Sampler*: it is a CoAP client that runs on the sensor nodes and it represents the data ingestion layer. Its function is to sample the physical entity, send the data to the correspondent LE and receive a time interval as a response from the LE. This time interval is used to put the sensor node in a sleep state. When the sensor node goes back to the active state, this process is repeated.
- *Orchestrator*: it is a service that runs on the edge node to coordinate instantiating of LEs and the communication between LEs, sensors and applications. It represents the resource management layer.
- *Logical Entity (LE)*: it consists of a set of decoupled modules (described below) that run on the edge node and work together to provide an abstract view of a monitored entity or phenomena:
  - *Gatherer*: a CoAP server that listens to requests on a specific UDP port. Its function is to receive data sampled from sensor nodes and respond back with a time interval until the next sampling. Section 3.2 describes how these time intervals are calculated.
  - *Buffer*: to meet strict real-time requirements for processing requests within milliseconds, an in-memory data store that keeps the data in the random access memory (RAM) is necessary (Zhang et al., 2015). This is used to persist sensor readings for a short predefined time. It represents the persistence layer.
  - *Predictor*: this is a component responsible to retrieve data from the buffer, identify and discharge incorrect readings and output a calculated value based on a predefined aggregation function. It represents the DSP layer. Section 3.3 describes how these operations are carried out.
- *REST API*: it is an HTTP endpoint (Richards, 2006) provided by the Orchestrator to standardize the communication between LEs and applications. It represents the output layer.

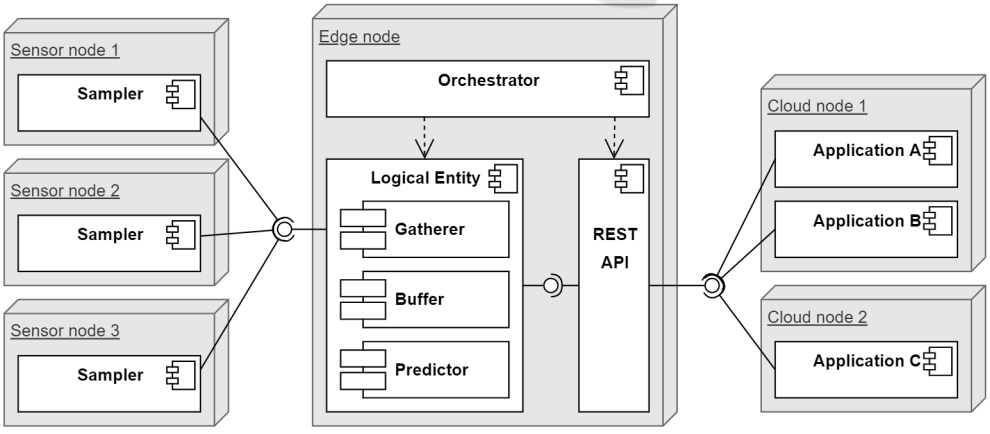

Figure 2: Nodes and components involved on a hypothetical scenario of a single entity monitored with three physical sensors and serving different applications.

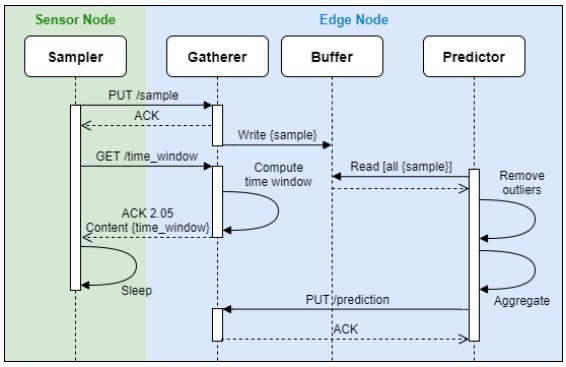

Figure 3: Interactions between software components.

3.2 Adaptive Sampling Strategy

An adaptive sampling strategy consists of dynamically varying the time interval between samplings somehow to follow the variability of the sampled physical entity. Whenever there is little or no variation, the sample interval can be increased. Whenever there is a significant variation, the sample interval must be reduced. The purpose of an adaptive sampling method is to reduce as much as possible the number of samples per unit of time, aiming at reducing the energy consumption of the sensor node. However, this strategy needs to be applied carefully so as not compromising the accuracy of the sampled series (Anastasi et al., 2009).

When combined with a data prediction model, it is possible to calculate the difference between the sampled and predicted values. Thus, a domain specialist can configure a tolerance threshold which shall be used to drive the adaptive sampling strategy. A so-called greedy adaptive sampling strategy increases the sampling interval at each sampling until the prediction exceeds this tolerance threshold. When the threshold is exceeded, the sampling interval is reduced so that the predicted and sampled values are closer again. Also, minimum and maximum thresholds can be added to satisfy application requirements (Monteiro et al., 2017).

The adaptive sampling strategy adopted on the proposed framework is an adaptation made to the one described in DPCAS (Monteiro et al., 2017), also proposed by our research group, for data stream processing in WSNs. DPCAS uses the TCP congestion control algorithm concepts to adjust the sampling interval of each sensor node dynamically. The strategy is based on the TCP CUBIC protocol (Ha et al., 2008), where the size of the windows vary according to a cubic function. The equations of the adaptive sampling method are as follows:

$$W_i = C(t-K)^3 + W_{max} \quad (1)$$

$$K = (\beta W_{max}/C)^{1/3}, 0 < \beta \leq 1 \quad (2)$$

Where W_i represents the sample interval calculated at the i-th sampling, which will be used as the sensor hibernation time until the next sampling, C is a scale factor known as a CUBIC parameter (typically 0.4), t is the elapsed time since the last reduction of the sample interval, W_{max} is the sample interval immediately before the last reduction of the sample interval and β a multiplicative reduction factor (typically 0.2). The factor K, described in Equation 2, is updated only when an event of reduction of the sample interval occurs. An event of reduction of the sampling interval occurs whenever the difference (δ) between the sampled value (Y_i) and its respective prediction (F_i) exceeds the tolerance limit of the application (ε), that is, whenever:

$$|Y_i - F_i| = \delta > \varepsilon \quad (3)$$

In addition, the application can also impose a minimum (S_{min}) and maximum (S_{max}) limit for sampling interval variation, that is:

$$S_{min} \leq W_i \leq S_{max} \quad (4)$$

In our proposal, we assume the presence of multiple sensors sampling the same physical entity, so called multi-sensed entity scenario. In the adaptive sampling model proposed by DPCAS, each sensor node acts in a completely autonomous way. It samples the physical entity and calculates the time window until the next sampling based on its own samples. However, as the cubic function used to compute the time intervals is the same for all sensor nodes, the sensing rate is very close or the same for all nodes. Therefore, there is some synchronization between the sensor node activities since the time windows increase and decrease almost simultaneously. All the sensors would always be sampling at the very same time. Similarly, all the sensors would also be in a sleep state at the same time. This aspect creates a gap that we call a ”blind window”: if a sudden variation occurs when all sensors are sleeping, this variation will only be identified when the sensors wake up.

In such a multi-sensed entity scenario, all the sensors monitoring the same entity must not be in a sleep state simultaneously. This opens up an opportunity that we explore by proposing a collaborative strategy: to distribute different sampling intervals between sensor nodes. It aims at desynchronizing sensor nodes activities to reduce these blind windows and make the adaptive sampling model more responsive to sudden changes in the data stream.

To enable this collaborative and desynchronized approach, the calculations of time intervals between samplings must be carried out at the edge node and not at the sensor nodes. The edge is the only node

that communicates with all sensor nodes and it is also responsible for computing data predictions. Only the edge node has all the information required to calculate sampling intervals. In this way, the gatherer component is responsible for these computations by performing the following steps:

1. It receives a sensor reading from the sampler;
2. It stores the received value in the buffer;
3. It uses the cubic function, the last value computed by the predictor and the last computed sampling window to compute the next sampling window;
4. It sends the sampling window back to the Sampler.

3.3 Data Prediction

A prediction model aims at computing future sensor readings. Simple Exponential Smoothing (SES) (Ha et al., 2008) and Double Exponential Smoothing (DES), also known as Holt Method (Hyndman and Athanasopoulos, 2014) (Wright, 1986) are good examples of data prediction models. They are computationally economical and thus interesting choices for WSNs and IoT (Monteiro et al., 2017). However, both were designed to predict readings of an individual sensor based on its own past readings. As already mentioned in section 3.2, our framework assumes a multi-sensed entity scenario. Thus, our data prediction model must consider readings from different sensors, physically closer to each other, on its computations. From a high level perspective, the data prediction model performed by our Predictor component consists of (i) cleaning noisy/incorrect readings and (ii) computing an output value based on an aggregation function.

Data streams are continuous flows of isolated data points. In an IoT use case with sensor generated data, these data points are represented by sensor readings. Sensors are fail-prone devices. Incorrect sensor readings need to be identified and discarded as best as possible to improve the quality of the information and decisions based on the acquired data. No data point can be considered an outlier on an individual basis analysis. Thus, defining a way to group and analyze these data points together is a major concern when designing an outlier detection task for IoT. Buffering incoming data on a predefined length or time interval to create windows is a common approach to group data points and perform operations on data streams (Tsai et al., 2014).

Clustering is a problem widely studied in data mining and AI literature. However, it is difficult to adapt arbitrary clustering algorithms to the context of data stream processing. The data stream feature of being potentially unbounded in size makes such adaptation especially complex (Aggarwal, 2013). K-means is one of the best-known clustering and also the starting point for a number of variations tailored for stream processing (Tsai et al., 2014). However, since each outlier can represent a different cluster and the number of clusters is an expected input for K-means and its variations, such algorithms are not suitable for outlier detection. Density-based techniques, in turn, are able to determine the number of clusters as an output. Thus, they are more effective and versatile than K-means for the purpose of outlier detection in data streams in the IoT (Aggarwal, 2013). Even not being tied to a specific algorithm, our proposed framework requires a density-based approach to be chosen. A set of good candidates is presented in (Campello et al., 2013). We chose Density-based spatial clustering of applications with noise (DBSCAN) in our implementation of the predictor component.

Choosing an aggregation function can become a complex task depending on the application's use case. In the same way as for the clustering algorithm used in the outlier detection task, this is a feature where the proposed framework gives some autonomy regarding its choice. In the context of real-time data stream processing for IoT, it can be assumed that the input data are sensor readings sampled in a short time window. We must also consider that these data points have been cleaned by an outlier detection task and are trustworthy. Therefore, central tendency statistical measures, such as mean and median, are reasonable choices to represent the value measured in that time interval. For the sake of simplicity, we use the arithmetic average function in our implementation.

The current version of the Predictor component adopts a naive approach which can be summarized as a three-step procedure:

1. It takes all the samples from the buffer as the input;
2. It runs DBSCAN to identify and remove outliers;
3. It computes and outputs the arithmetic average of the remaining samples.

4 EVALUATION

In this section, we describe the experiments performed with the proposed framework in order to evaluate (i) the impact, in terms of accuracy of the output data, of reducing the number of data samples sent by the sensors; (ii) how efficient is the proposed adaptive sampling strategy in terms of energy consump-

tion, and (iii) how fast is the proposed data prediction model to process buffered data.

4.1 Environment

The components of the framework were developed in Python 3 language. The CoAP features were implemented with CoAPthon (Tanganelli et al., 2015). Redis (Sanfilippo and Noordhuis, 2009) was used to implement the Buffer component. The REST API was implemented with Flask (Ronacher, 2020). In terms of infrastructure, not only the development of the software components but also a preliminary proof of concept (PoC) were executed on a Raspberry Pi 3 Model B+ which is a widely used device in many IoT applications. We created an isolated virtual network on a public cloud environment to simulate the experiments on five different input data volumes and allocated computing resources for processing. To optimize the costs incurred by running these experiments on a public cloud with a pay-as-you-go model, each sensor node runs 30 instances of the sampler component to simulate 30 physical sensors running simultaneously. A single edge node gets more allocated resources since the proposed solution is not yet prepared to run in a distributed manner. Table 1 describes the number of sensors ingesting data and also the computing resources of the edge node (vCPUs and RAM) for each round of simulation. All sensor and edge nodes are powered by 64-bit Arm-based processors, with 10 Gbps network bandwidth and Ubuntu Linux 20.04 operating system. This configuration for the virtual machines is the closest setup to a Raspberry Pi 3 B+ we can achieve on a public cloud.

A preliminary data set of real sensor readings was generated from six Adafruit DHT11 sensors (Industries, 2020) continuously collecting temperature and humidity data for one hour. Then, we used this data set to generate synthetic data sets of 480 sensors by adding to these measured values a random value ranging from -1 to 1. Finally, the synthetic data set was used as the input on each round of simulation for two different sampling modes:

1. *Fixed Sampling Mode*: the complete synthetic data set is sent to the edge node according to its timestamp and used by the predictor component to compute the output. This scenario represents a benchmark use case where all the sensors are active all the time.
2. *Adaptive Sampling Mode*: each instance of the sampler component running on a sensor node communicates with the edge node to send data when active and sleeps according to the time windows received. Data points from the synthetic data set in which the timestamp corresponds to the sensor's sleep time are discarded.

Table 1: Infrastructure setup per round of simulation.

Round	Sensors	Edge vCPUs	Edge RAM
1	30	1	4 GiB
2	60	2	8 GiB
3	120	4	16 GiB
4	240	8	32 GiB
5	480	16	64 GiB

In all rounds of simulations, the data prediction task is executed once every five seconds to get the data used to generate the line and scatter plots in this section.

Before evaluating the impacts and benefits of the proposed solution, it is important to clarify the core difference between the two samplings, which will be compared in the following subsections. As it is shown in Fig. 4, while the fixed sampling mode keeps a high number of samples at the buffer all the time, in the adaptive sampling mode, this number varies according to the value of the data being gathered. When the temperature and humidity values are stable, the number of buffered samples is low. When the temperature and humidity values start to change, the number of buffered samples increases quickly. This behavior is observed in all simulation rounds with different scales. As the data volume being gathered increases on each simulation round, the difference in the number of buffered samples over time is even greater. Fig. 4 also presents the difference between rounds 1 and 5.

4.2 Data Accuracy

The reduced number of samples sent to the edge node using the adaptive sampling mode implies that less data is available for the data prediction model.

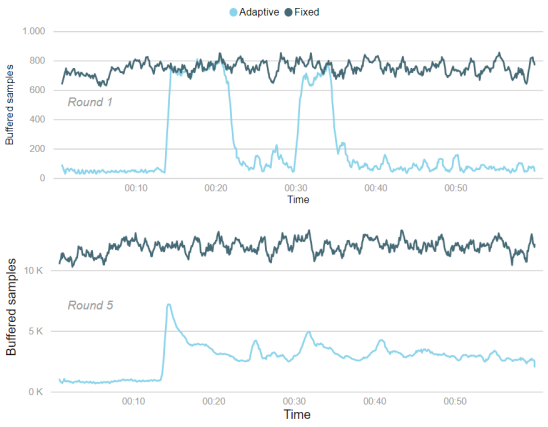

Figure 4: Buffered load over time by sampling strategy.

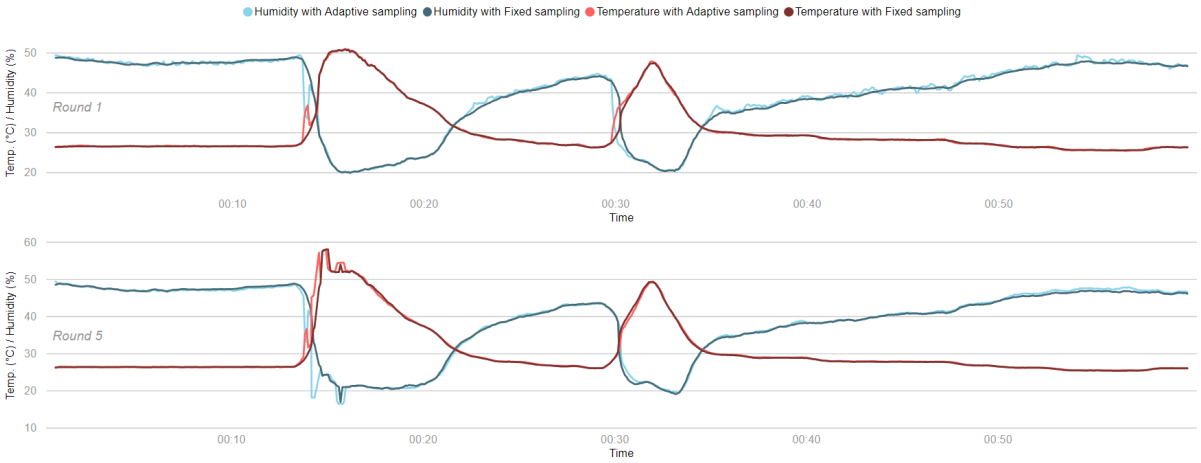

Figure 5: Temperature and humidity readings processed with different sampling strategies.

Thus, there is a concern that data accuracy might be affected. To evaluate how this reduced amount of data available impacts the data prediction output, we use Mean Absolute Error (MAE) metric (Chai and Draxler, 2014). We assume the outputs from the executions using fixed sampling as the real/observed values to calculate the differences from the outputs obtained with adaptive sampling mode on each simulation round. Tables 2 and 3 present the calculated MAE and data range per round of simulation for humidity and temperature. It is possible to say that scaling up sensors and computing resources does not impact MAE.

Despite the considerable difference in the data available at the buffer, both fixed and adaptive sampling modes have a very similar data prediction output. Fig. 5 allows visualizing how close are the outputs for both sampling modes. The red lines represent temperature readings while the blue lines represent humidity readings. The darker lines correspond to the fixed sampling mode while the lighter ones correspond to the adaptive sampling mode. By comparing rounds 1 and 5, it is also possible to note that the greater the amount of data, the closer are the outputs for the different sampling approaches.

Table 2: MAE for humidity per round of simulation.

Round	MAE	Range
1	0.5365	(19.91 - 51.01 %)
2	0.5720	(17.15 - 50.44 %)
3	0.4110	(16.84 - 52.23 %)
4	0.3656	(19.06 - 51.23 %)
5	0.4738	(16.45 - 50.46 %)

4.3 Energy Consumption

Once it is proved that the reduced number of samples does not significantly affect the accuracy of the output data, we need to verify the benefit of the adaptive sampling strategy in terms of energy consumption. To verify how efficient the proposed framework is on reducing the energy consumption of the sensor nodes when compared to a traditional fixed sampling approach, we use PowerPi (Kaup et al., 2014). PowerPi is a power consumption model to calculate the energy consumed by an application running on a Raspberry Pi device (RPi).

To calculate the energy consumption of an application with PowerPi, only the application to be measured must be running on the device, along with essential operating system tasks. Our experiments rely on 30 concurrent processes running the sampler component on a shared virtual sensor node (section 4.1). To allow this calculation, we had to make an assumption: the energy consumption was measured individually on a real Raspberry Pi 3 Model B+ device for each of the four main actions performed by a sensor node (described below). The total energy consumption of a sensor node is computed as a weighted sum of each of these individual measures plus a constant value for idle time (Kaup et al., 2014). The weights are based on the application logs, where every ac-

Table 3: MAE for temperature per round of simulation.

Round	MAE	Range
1	0.1894	(25.44 - 51.01 °C)
2	0.2517	(25.46 - 57.53 °C)
3	0.3369	(25.59 - 50.65 °C)
4	0.2609	(25.31 - 58.59 °C)
5	0.2384	(25.40 - 58.09 °C)

tion of the sensor node is registered. The four actions performed by a sensor node which were individually measured are:

1. *Sampling*: sampling the physical entity to obtain the temperature and humidity values.
2. *Sending*: sending sampled data to the edge node.
3. *Getting Window*: getting the sleep time until the next sampling from the edge node.
4. *Sleeping*: sensor inactive for 1s.

The assumption made can lead to calculated values that might not accurately represent the real energy consumption of the devices. However, for a strictly comparative analysis between the scenarios, the performed calculations are enough. Fig. 6 shows that the adaptive sampling strategy leads to average energy consumption savings ranging from 42.93% to 60.58% on sensor nodes, which is a very significant result. It is also possible to state, from the same Fig. 6, that actions that involve communication between the sensor and edge nodes (specially "Sending") are the most expensive in terms of energy consumption. The adaptive sampling strategy essentially replaces "Sending" activity by less expansive actions: "Sleeping", which is the least expensive activity and "Getting" which is also a communication activity but with much smaller payloads.

The carried out experiments also allowed observing an expected side effect of the proposed adaptive sampling strategy: poor load balancing. Although the total energy consumption of the sensor nodes is significantly reduced in the broad spectrum compared to the fixed sampling strategy, individually, some sensors presented a much higher energy consumption

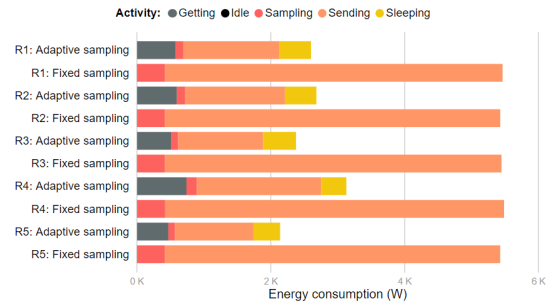

Figure 6: Average energy consumption per sensor node.

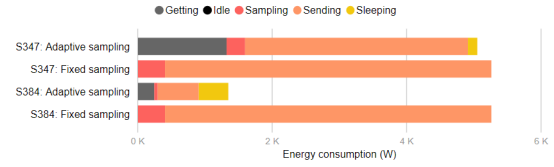

Figure 7: Energy consumption of sensors 347 and 384.

than others in the same group. Load balancing tends to get worse as we increase the number of sensors. Fig 7 shows the high contrast between the energy savings of two sensors on the same group in simulation round 5. For smaller scales (rounds 1 to 3), this anomaly is tiny, almost irrelevant.

4.4 Real-time Data Prediction

Earlier research on computer response times suggests that(Nah, 2004):

- **0.1s:** is the limit for having the users feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result;
- **1.0s:** is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1s but less than 1.0s;
- **2.0s:** is the limit where a response to simple commands becomes unacceptable to users.

According to the definitions above, to meet the real-time requirement, we consider 1.0 s as a threshold for the data prediction task's execution time. To achieve such goal on this multi-sensed environment, it is required to choose fast and non compute intensive algorithms when implementing the Predictor component to run on an edge device. DBSCAN was chosen for data cleaning/pre-processing due to its good capability of finding arbitrarily shaped clusters, what makes it robust to outlier detection (Campello et al., 2013). The performed experiments involve continuous variables, which are real values over a non-empty range. Thus, we believe that a simple average can fairly represent the temperature and humidity values given a set of samples over a given short time. Therefore, the Mean statistical function was chosen as the aggregation function for this implementation.

The time elapsed in each data prediction execution was also registered as an attribute on the output data. This information evaluated how fast the data is being processed, considering the data cleaning and aggregation tasks. Fig 8 shows how the execution time of the data prediction increases as the volume of data processed (number of buffered samples) grows. The graph presents a linear progression, suggesting good scalability of the proposed solution using the chosen algorithms. There were actually a very small number of data points (9) above the threshold (1.0 s), representing less than 0.3% of the total and, therefore, can be considered outliers. These outliers are outside the plotting area to provide a better view of the relevant

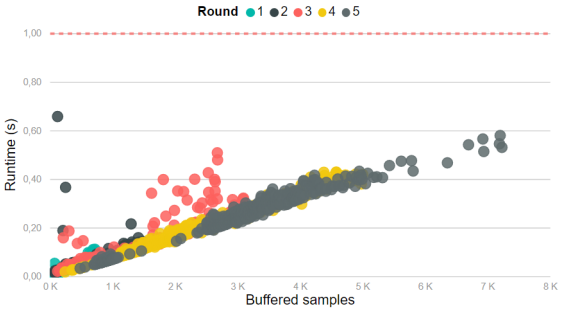

Figure 8: Data prediction execution time per amount of buffered samples.

part of the data. It is important to highlight that even with a number of samples in the order of 7 thousand units, the processing time remains below 0.6 s.

5 FINAL REMARKS

This work presented a real-time and energy-aware data stream processing framework for IoT. Experiments show that the reduced number of samples does not compromise data accuracy due to a combination of an adaptive sampling strategy with a data prediction model. Being energy efficient, the framework has reduced the average energy consumption of sensor nodes up to 60.58%. The results described in 4.4 indicate that the edge data prediction model successfully addresses real-time requirements by meeting the execution time threshold of 1s for the data prediction activity. The main contribution of the proposed framework is its capability of tackling real-time processing, energy consumption and data accuracy requirements all together. Thus, it might be used to enable the development of long running real-time data stream processing IoT systems in remote outdoor environments, where energy sources are scarce and it undesirable or unfeasible replacing batteries frequently.

ACKNOWLEDGEMENTS

This work has been partially funded by Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP (grant 2015/24144-7), Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro - FAPERJ (grant 2017/233868) and Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (grant 434874/2018-3). Marta Mattoso and Flavia Delicato are CNPq Fellows.

REFERENCES

Aggarwal, C. C. (2013). Mining Sensor Data Streams. In *Managing and Mining Sensor Data*, pages 143–171. Springer US, Boston, MA.

Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38(4).

Al-Hoqani, N. and Yang, S.-H. (2015). Adaptive sampling for wireless household water consumption monitoring. *Procedia Engineering*, 119:1356 – 1365. Computing and Control for the Water Industry (CCWI2015) Sharing the best practice in water management.

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

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805.

Campello, R. J. G. B., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg. Springer Berlin Heidelberg.

Catarinucci, L., de Donno, D., Mainetti, L., Palano, L., Patrono, L., Stefanizzi, M. L., and Tarricone, L. (2015). An iot-aware architecture for smart healthcare systems. *IEEE Internet of Things Journal*, 2(6):515–526.

Chai, T. and Draxler, R. R. (2014). Root mean square error (rmse) or mean absolute error (mae)? *Geoscientific Model Development Discussions*, 7(1):1525–1534.

Dautov, R., Distefano, S., Bruneo, D., Longo, F., Merlino, G., and Puliafito, A. (2018). Pushing intelligence to the edge with a stream processing architecture. In *Proceedings - 2017 IEEE International Conference on Internet of Things, IEEE Green Computing and Communications, IEEE Cyber, Physical and Social Computing, IEEE Smart Data, iThings-GreenCom-CPSCom-SmartData 2017*.

Dias de Assunção, M., da Silva Veith, A., and Buyya, R. (2018). Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*.

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131.

Fielding, R. T. and Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231.

Gupta, M., Shum, L. V., Bodanese, E., and Hailes, S. (2011). Design and evaluation of an adaptive sampling strategy for a wireless air pollution sensor network. In *2011 IEEE 36th Conference on Local Computer Networks*, pages 1003–1010.

Ha, S., Rhee, I., and Xu, L. (2008). Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74.

Hyndman, R. and Athanasopoulos, G. (2014). *Forecasting: principles and practice*. OTexts.

Industries, A. (2020). *DHT11 basic temperature-humidity sensor*.

Isah, H., Abughofa, T., Mahfuz, S., Ajerla, D., Zulkernine, F., and Khan, S. (2019). A survey of distributed data stream processing frameworks. *IEEE Access*, 7:154300–154316.

Janjua, Z. H., Vecchio, M., Antonini, M., and Antonelli, F. (2019). IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge. *Engineering Applications of Artificial Intelligence*, 84:41–50.

Karkouch, A., Mousannif, H., Al Moatassime, H., and Noel, T. (2016). Data quality in internet of things: A state-of-the-art survey.

Kaup, F., Gottschling, P., and Hausheer, D. (2014). Powerpi: Measuring and modeling the power consumption of the raspberry pi. In *39th Annual IEEE Conference on Local Computer Networks*, pages 236–243.

Klein, A. and Lehner, W. (2010). Quality and Performance Optimization of Sensor Data Stream Processing. *International Journal on Advances in Networks and Services*.

Li, S., Xu, L. D., and Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259.

Li, W., Santos, I., Delicato, F. C., Pires, P. F., Pirmez, L., Wei, W., Song, H., Zomaya, A., and Khan, S. (2017). System modelling and performance evaluation of a three-tier Cloud of Things. *Future Generation Computer Systems*.

Loria, M. P., Toja, M., Carchiolo, V., and Malgeri, M. (2017). An efficient real-time architecture for collecting iot data. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1157–1166.

Mehmood, E. and Anees, T. (2020). Challenges and solutions for processing real-time big data stream: A systematic literature review. *IEEE Access*, 8:119123–119143.

Monteiro, L. C., Delicato, F. C., Pirmez, L., Pires, P. F., and Miceli, C. (2017). Dpcas: Data prediction with cubic adaptive sampling for wireless sensor networks. In Au, M. H. A., Castiglione, A., Choo, K.-K. R., Palmieri, F., and Li, K.-C., editors, *Green, Pervasive, and Cloud Computing*, pages 353–368, Cham. Springer International Publishing.

Nah, F. F.-H. (2004). A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163.

Qin, Y., Sheng, Q. Z., Falkner, N. J., Dustdar, S., Wang, H., and Vasilakos, A. V. (2016). When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications*, 64.

Richards, R. (2006). *Representational State Transfer (REST)*, pages 633–672. Apress, Berkeley, CA.

Ronacher, A. (2010 (accessed August 5, 2020)). *Flask web development, one drop at a time*.

Samizadeh Nikoui, T., Rahmani, A. M., Balador, A., and Haj Seyyed Javadi, H. (2021). Internet of things architecture challenges: A systematic review. *International Journal of Communication Systems*, 34(4):e4678. e4678 IJCS-19-1067.R1.

Sanfilippo, S. and Noordhuis, P. (2009). Redis.

Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). RFC 7252.

Tanganelli, G., Vallati, C., and Mingozzi, E. (2015). Coapthon: Easy development of coap-based iot applications with python. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 63–68.

Tsai, C.-W., Lai, C.-F., Chiang, M.-C., and Yang, L. T. (2014). Data Mining for Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16(1):77–97.

Vučinić, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., and Guizzetti, R. (2014). Oscar: Object security architecture for the internet of things. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–10.

Wright, D. J. (1986). Forecasting data published at irregular time intervals using an extension of holt's method. *Management Science*, 32(4):499–510.

Xu, Y. and Helal, A. (2016). Scalable cloud–sensor architecture for the internet of things. *IEEE Internet of Things Journal*, 3(3):285–298.

Zhang, H., Chen, G., Ooi, B. C., Tan, K., and Zhang, M. (2015). In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948.