Resilient Edge Data Management Framework

Ivan Lujic, Student Member, IEEE, Vincenzo De Maio, Member, IEEE, and Ivona Brandic, Member, IEEE

Abstract—Internet of Things (IoT) based systems produce nowadays an unprecedented amount of data. Managing these systems becomes difficult with emerging low latency requirements of IoT applications and limited underlying network infrastructure. Hence, traditional processing of IoT data in the cloud is often infeasible, imposing essential placement of data analytics at the network edge. Nonetheless, limited storage capability and high failure rate in IoT systems, face us with the challenge of making timely decisions that rely on a limited quantity of data. Furthermore, due to network and monitoring system failures, collected data are often incomplete or incorrect, deteriorating quality of near real-time decisions. Here, we propose a three-layer architecture model for edge data management. Based on this model, we design EDMFrame, a framework that incorporates a novel mechanism for recovery of multiple gaps in incomplete datasets and an adaptive edge storage management mechanism for maintaining limited edge storage, keeping only amount of data necessary for predictive analytics. Finally, to improve recovery of incomplete data, we present a mediator component that recommends a sufficient amount of data for corresponding gap lengths based on projection recovery maps. We evaluate proposed approaches using time series coming from real-world IoT systems such as smart buildings.

Index Terms—Edge computing, Internet of Things, data, storage, forecasting, solution referenced architectures, data flow architecture.

1 INTRODUCTION

The introduction of Internet of Things (IoT) has attracted attention from both academia and industry. Billions of devices will be connected to the Internet by 2020 [1], [2], generating huge amounts of data. Nowadays, sensor-based data are used in many applications, like eHealth systems [3], intelligent traffic management [4], automated smart home and building systems [5], and smart cities [6].

Managing such systems mainly requires three steps: sensors data collection, data processing and taking actions based on results of analysis. In most of IoT systems, data are processed by employing geographically distributed and massive cloud data centers [7], [8]. As the size of such systems is constantly increasing, producing consequently larger amount of data, performing data analytics in the cloud may be infeasible, because current network infrastructures cannot easily scale with the growing amount of data [9]. This raises challenges to meet strict latency and accuracy requirements [10] for decision making processes.

Edge analytics is a promising solution to these challenges. This technique relies on edge nodes such as micro data centers, which are smaller scale cloud data centers, that can be deployed closer to IoT systems [11]. Performing data processing in micro data centers allows near real-time decision making for IoT systems [12]. At the same time, utilizing edge nodes raises several challenges: first, missing values and outliers may appear in data collected by sensors, due to (i) the highly distributed nature of IoT systems; (ii) monitoring system failures; (iii) data packet loss in sensor networks; (iv) aging of the sensor; (v) changes in external conditions [13]. Performing analytics on incomplete/invalid datasets can lead to inaccurate results and imprecise decisions, as shown by [5], [14]. Second, edge nodes have limited storage and are not scalable as cloud data centers. Such limitations can hinder accuracy of edge analytics and consequently decisions for critical applications such as eHealth or traffic monitoring systems [4].

The reconstruction of incomplete datasets is discussed in several papers [15], [16]. However, these approaches are not validated on IoT systems and do not consider improvement of data quality by using different forecasting techniques. Works like [17], [18] discuss IoT data reduction and compression, without considering the edge storage management problem including the quality of near real-time analytics for critical IoT applications.

We bridge this gap by introducing a three-layer architecture model for efficient edge data management, called Edge Data Management Framework (EDMFrame), featuring: (i) a novel semi-automatic mechanism for recovery of incomplete time series, incorporating a recovery cycle that ensures outliers removal, detection and forecasting of each gap; (ii) an edge storage management mechanism that achieves a trade-off between reducing the amount of data stored at the edge and maintaining high accuracy for predictive analytics; (iii) a mediator component that detects an optimal trade-off between the gap size and a necessary range of historical data by managing Projection Recovery Maps (PRM) for each dataset. Preliminary results for (i) and (ii) are shown in [19], [20], respectively. Here, we describe the complete EDMFrame, introducing the novel mediator component and show an entire data path through EDMFrame.

We evaluate EDMFrame experimentally using six representative real-world time series from smart buildings and homes. Results show that EDMFrame is able to efficiently recover multiple gaps of various lengths, achieving a forecasting error below 2.68%, and running time lower than 0.68s; to obtain forecast accuracies above user-defined threshold, while reducing amount of data stored by 39.9% on average per cycle; and to facilitate the choice of recovery methods and ranges of recommended past data for the recovery.

I. Lujic, V. De Maio and I. Brandic are with the Institute of Information Systems Engineering, Vienna University of Technology, A-1040 Vienna, Austria. (e-mail: {ivan, vincenzo, ivona}@ece.tuwien.ac.at)

1. https://github.com/lujic/EDMFrame
We describe EDMFrame architecture in Section 2. Section 3 presents data recovery mechanism, while Section 4 shows the design principles and algorithms for edge storage management. Section 5 contains main principles of mediator component. Experimental evaluation and discussion are given in Section 6. Related work is outlined in Section 7, while Section 8 concludes the paper.

2 ARCHITECTURE MODEL OVERVIEW

2.1 Background and Motivation

IoT systems mostly rely on cloud resources to perform data processing [7]. However, due to increasing data streams from smart sensors and emerging latency requirements from recent IoT systems, transferring all data to the cloud for analysis and decision making becomes infeasible [10]. In addition, underlying network infrastructures meet difficulties for future expansion of IoT data sources [9], including increased energy consumption and high cost of data transmission. At the same time, it is crucial to make fast decisions in IoT services like smart grids and cities requiring distributed ML at the edge [21]. For example, in the case of consistent ML models that must be updated when data streams evolve over time, it poses critical challenges to observe correct data at the right time [22]. Hence, to ensure fast and accurate decision making in IoT systems, performing efficient edge data management is of paramount importance [23].

2.2 Proposed Architecture

Figure 1 represents our EDMFrame architecture. The aim is to devise a process to automatically manage accurate near real-time decisions while coping with (i) incomplete data, (ii) big volume of data and (iii) limited storage resources at the network edge, without intervention of third party or providers. Each edge monitoring process includes three software layers, namely: gathering layer, edge layer and cloud layer. Even though the main focus of this paper is the edge layer, we describe all of them for completeness.

Gathering layer transmits IoT measurements to the edge layer to reduce communication costs, save bandwidth and meet latency requirements in distributed sensor networks. Gateways perform aggregation of sensor data sending them in an appropriate format and size to the monitoring component. In step (1) (see Figure 1), data are collected from smart buildings and then in step (2) transferred to the edge layer.

Edge layer manages data through different stages of the EDMFrame, to perform accurate and timely analytics. It is composed of edge nodes, such as edge gateways and micro data centers [11], aiming to perform data processing closer to data sources. It consists of the following components:

Monitoring component. This component receives and analyses data to detect outliers and missing values, notifies mediator component about incomplete data, prepares data for the data recovery mechanism, and triggers control commands to IoT actuators based on local edge analytics. It can extrapolate data characteristics useful for further analytics.

Specification list. Once certain amount of data is transmitted to the edge layer, user specifications are checked in step (3). Specification list contains application dependent and user-defined information, useful for both data recovery and edge data management process, for example, the forecast horizon, monitoring frequency, preferred forecast method, accuracy threshold, conditions and other rules;

Data recovery mechanism. Adaptive recovery process is performed in step (4). It receives data from the monitoring component and performs semi-automatic recovery of multiple gaps incorporating recovery cycles. The output is dataset without gaps and cleaned from outliers.

Storage. Edge storage carries limited capacities. It stores data coming from the data recovery mechanism, and communicates with the edge storage management, mediator component and local edge analytics processes;

Edge storage management. In step (5), edge storage management mechanism maintains limited storage keeping only data relevant for near real-time decisions. It checks available data, validates the specification list and implements the edge storage management phases. The available data are used in step (6) for local analytics processes whose output is forwarded either to the storage or to the monitoring component sending commands to actuators in step (7).

Mediator component. The mediator manages PRM to support data recovery mechanism. In step (8), mediator component communicates with the cloud data repository. It transfers the necessary range of data from/to the cloud. It can perform data filtration and data transformation to improve data transfer between the edge and cloud layer.

Cloud layer contains the data repository storing historical data collected from IoT systems. It performs compute intensive big data batch analytics and delivers information and results based on entire datasets. In Sections 3, 4 and 5, each component of the edge layer is described in detail.

3 DATA RECOVERY MECHANISM

We present an adaptive mechanism for the recovery of acquired time series. First, we define gaps that are present within incomplete data. A gap is a sequence of one or more missing or invalid consecutive values, irregularly distributed in time series. By missing values we mean data that are missing due to sensors/monitoring failures, and by invalid data we mean outliers due to errors in the measurement. Formally, Definition 1. A gap $G_n$ represents the $n$-th gap in an incomplete dataset with $k$ missing/invalid values.

For example, $G_{17}^{2}$ refers to the second gap containing 17 missing values. In Figure 2, we provide a flowchart of the...
The goal of this component is to prepare an incomplete dataset for the recovery process. To this end, we apply a set of operations that detect each gap in the dataset. Data preparation process is described in Algorithm 1. First, line 1 creates an empty vector for indexes of missing values in the dataset. Outliers are identified according to minimum and maximum values, for particular sensors, that can be either application-dependent or predefined by user. If a data value is out of bounds, it is replaced by a missing value indicator such as N/A (Not Available) (line 2), so that the correct value can be efficiently estimated in the recovering cycle. Missing values can occur for different reasons, like a monitoring system or sensor failures. Once the system is recovered, the next received data point is stored immediately after the last generated time stamp, thus making it difficult to identify a gap in the time series without checking timestamps. We propose a solution where the monitoring component receives data and stores either corresponding data value or a missing value indicator, for each time stamp generated in advance (lines 4-13). Counters i and j (line 3) count data from input and framed data, respectively. If time stamps from prepared framed data and input match (line 5), the measurement is moved to the framed data beside the corresponding time stamp (line 6). Otherwise, a missing value indicator N/A is stored to the framed data beside the current time stamp (line 9), and the index of missing data point is stored in the created vector indexmiss (line 10). Once the while loop terminates, the vector indexmiss contains all indexes of missing data, amount of which is calculated and placed in the variable nomiss (line 14).

### 3.2 Gap Identification

The gap identification phase (described in Algorithm 2) is responsible for detecting multiple gaps in a given dataset. It identifies the beginning and the end of each gap, using this information for the separate recovering process. Each gap is then processed separately, to enable selection of an appropriate forecasting technique according to characteristics of previous data or user predefined specifications. The counter i (line 1) is used to iterate over the vector indexmiss, while data index of the first missing value is copied to the beginning of the vector gapcurr and stored also in the temporary variable temp (lines 2-3). As long as there are missing values in nomiss (line 4), the counter i is looking for the next index of missing value, that is, the counter i is updated with the next index of missing value, while the index, stored in the variable temp, is incremented by 1 (line 6). This allows to check whether a gap of consecutive missing values exists (line 6). If indexes of missing values are consecutive, the corresponding index is copied to the vector gapcurr (line 7). Otherwise, if there are no more consecutive indexes (line 8), all missing values from the current gap are detected and the vector indexmiss is updated in line 9.

### 3.3 Data Processor

This component performs extrapolation of data characteristics and parameters needed for application of particular forecasting technique. Necessary characteristics are obtained during analysis of available data up to the first

---

**Algorithm 1: Data Preparation**

**Input:** inputdata[timestamp, value], framedata[timestamp, value]

**Output:** vector indexmiss

1. Create vector indexmiss
2. Replace all outliers by a missing value indicator N/A, based on defined thresholds for monitored sensor.
3. i ← 1; j ← 1
4. while i < length(inputdata) do
5. if framedata[j, 1] := inputdata[i, 1] then
6. framedata[j, 2] ← inputdata[i, 2]
7. i ← i + 1; j ← j + 1
8. else
9. framedata[j, 2] ← N/A
10. Add index j in vector indexmiss
11. j ← j + 1
12. end if
13. end while
14. nomiss ← length(indexmiss)

---

**Table 1: Main Notations of Data Recovery Mechanism**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputdata</td>
<td>2D array that represents incomplete dataset.</td>
</tr>
<tr>
<td>framedata</td>
<td>2D array that represents data to be stored.</td>
</tr>
<tr>
<td>indexmiss</td>
<td>Vector that contains all indexes of missing values.</td>
</tr>
<tr>
<td>nomiss</td>
<td>Variable that counts number of missing values based on indexmiss.</td>
</tr>
<tr>
<td>gapcurr</td>
<td>Vector that stores missing indexes of the current gap.</td>
</tr>
</tbody>
</table>
missing index of a potential gap. Once the current gap is identified in the last component, in order to efficiently forecast missing values, predecessor data are analyzed to derive parameters that are necessary for the forecasting process. Parameter selection depends on the forecasting technique. Semi-automatic mechanism allows two possible scenarios: (i) single-technique recovery (STR) and (ii) multiple-technique recovery (MTR). In the first scenario, a single technique, that can be specified by users, is used for recovering all gaps. In the latter scenario, a technique selection is performed for different gaps. Currently, we assume that techniques are predefined in the algorithm repository.

3.4 Forecasting Process

In this component, a forecasting technique is selected from the repository and applied on the current gap. Figure 3 shows the adaptive recovery process including results of aforementioned components. After corresponding missing indexes are stored by the preparation component and the first gap identified by the gap identification component, data processor component analyzes predecessor data before the gap. Selected forecasting technique is then applied for the recovering process. The figure shows our approach, where forecasting process component applies different techniques (t1, t2 and t3) for different gaps. The choice of suitable techniques depends on data characteristics and forecast objectives as described in [24]. In this work we select two techniques, according to different dataset characteristics: (i) the Autoregressive Integrated Moving Average (ARIMA) method can be used if data contain stationary characteristics, such as trend stationarity, that can be explored by methods proposed in [25]; (ii) the Exponential Smoothing method (ETS), although overlapping in same cases with ARIMA models, can be used for short-term seasonal series or with multiple complex seasonality [26]. If seasonality occurs in time series, by checking periodicity, the data processor component can forward that information to the next component. Users can also specify additional information about the data, such as a monitoring frequency for seasonality: for example, if temperature data are collected every five minutes, then the seasonal parameter value 288, representing the expected periodicity, used to set up a forecast method [28], or seasonality over a certain period, used to set up a forecast method [28].

3.5 Algorithm Complexity

By analyzing Algorithm 1, looking at the while loop in line 4, the algorithm iterates over available dataset making it \( O(n) \), where \( n \) represents number of data points in the acquired dataset. Further, entering in the while loop of Algorithm 2 also in line 4, it iterates the vector of indexes of missing values that are always less than the number of data in the available dataset. For all other lines, complexity is \( O(1) \). In case forecasting process uses the ARIMA technique, the time complexity to forecast certain amount of data is \( O(n) \), where \( n \) is the number of data points in the training data used for forecasting, and thereby it results in the overall complexity \( O(n) \). Running time is affected by different factors, such as the size of the gap that has to be recovered, the amount of available historical data (finding an optimal trade-off between the gap size and necessary amount of historical data, is given by the mediator component in Section 5) or seasonal complexity of time series. Since the proposed mechanism targets resource-limited edge nodes and analysis for near real-time decisions, we expect that the input size and dimensionality of incompleteness will not cause violation of latency requirements.

4 Edge Storage Management

Effectiveness of a limited edge storage node depends on the accuracy of determining amount of necessary data to perform accurate near real-time decisions. Hence, an edge node should keep only relevant data for local data analytics, discarding the rest if they are irrelevant. Phases of the edge storage management algorithm are shown in Figure 4, based on the architectural model in Figure 1. Main notations are shown in Table 2. The learning phase is executed only once as it provides information used by all the other phases. Here we describe each phase:

Learning phase. The aim of this phase is to derive information about data, such as time series pattern recognition, that is used to determine the most appropriate method for that specific pattern [27], or seasonality over a certain period, used to set up a forecast method [28].

Validation of the specification list. In this phase the specification list defined by a user is checked. During the execution of the proposed algorithm, users can make some
changes in the specification list anytime, such as setting forecast accuracy threshold, a new forecast horizon or different forecast methods. Any changes made by the user can affect the overall edge storage management. Thus, there is the need of checking this list each time a cycle starts.

**Multiple forecast iteration on available dataset.** This phase takes one of the forecasting methods (in our case ETS or ARIMA) with the accuracy threshold and the forecast horizon from the specification list. The available dataset is divided into training and test data. Test data are equal to the number of data points specified by the user in the specification list (the forecast horizon). The amount of training data is reduced in each iteration by a certain amount of data to find parts of dataset resulting in required forecast accuracy. At the end of each iteration, forecast accuracy measures are added in the vector $\gamma$ to be used in the next phase.

**Detection of stable accuracy clusters.** Here, stable clusters of accuracy values have to be found in the vector $\gamma$.

**Definition 2.** We define a stable cluster $C_{st}$ as a set of subsequent data points in the vector $\gamma$ whose standard deviation for contained values is less than a given percentage $C_{l pct}$ of the standard deviation of entire vector $\gamma$, that is,

$$C_{st} \subset \gamma \text { AND } sd(C_{st}) < C_{l pct} \times (sd(\gamma))$$  \hspace{1cm} (1)

We define that a cluster contains at least three members. Accordingly, to provide reliable information regarding future system behaviors, our predictions must be stable. When the multiple forecast iteration process is finished, cluster detection is applied on the vector that consists in measuring forecast accuracy from each of forecast iterations. The method finds stable clusters of forecast accuracies that are close to the threshold level defined in the specification list. More details about this step in Section 4.1.

**Detection of an appropriate cluster.** Since previous step might return more than one stable cluster, we define how to select the most appropriate one. Stable clusters can differ in mean value and the amount of used data. Therefore, it is needed to set selection priorities. We propose the twofold priority for cluster selection and a corresponding algorithm. First priority is to satisfy threshold specified by user.

**Definition 3.** A stable cluster is appropriate if its mean value is the closest to the threshold value in the specification list, as defined in Equation 2,

$$C_{ap} = \arg \min_{C[i]} \left( |C[i]^{mean\_value} - acc_{th}| \right)$$ \hspace{1cm} (2)

where $C[i]^{mean\_value}$ is the mean value of forecast accuracies included in cluster $C[i]$, and $acc_{th}$ denotes forecast accuracy threshold specified by client in the specification list.

The appropriate cluster $C_{ap}$ becomes the one with the minimum absolute difference between $acc_{th}$ and $C[i]^{mean\_value}$. Second priority is to find a cluster that has higher accuracy but using less data. If such stable cluster exists, it becomes the appropriate cluster (see Section 4.2).

**Data management action.** This step releases irrelevant data from the storage. According to the appropriate cluster, we define that central data index of this cluster indicates a border between relevant and irrelevant data. There are three possible cases: (i) We can reach an appropriate cluster among stable clusters respecting desired accuracy threshold with fewer amount of data. In this case, all data not needed to obtain the observed accuracy cluster are released; (ii) None of resulting stable clusters meets the specified accuracy threshold by the client. In this case data management action will select the one with less data points; (iii) Forecast accuracy of stable clusters is higher with increased amount of training data, for example, forecast based on all available data from the storage. In that case, the mediator component can retrieve more data from the cloud repository.

**Validation of available dataset.** The adaptive algorithm is continuously repeated and in each cycle checks storage for newly collected data. Depending on application and data generation rate, the next cycle of edge storage management can act as a triggered event. In the next cycle, both new data received from the recovery mechanism and potentially from the mediator component, are included. The relevance of old information can be lower, unless prediction accuracy level shows that some stable clusters occur (close to specified threshold) based on older data. In that case, if algorithm feedback shows that accuracy increases with historical data, and amount of current stored data exceeds edge storage limitations, then this data analytics can be performed in the cloud environment. Hence, the approach requires to monitor variations of the accuracy rate for performed forecasts.

### 4.1 Detection of Stable Clusters

Detection of smooth behaviors for consecutive forecast accuracies, previously calculated due the forecast iteration phase, represents the cornerstone of our algorithm. There

---

**Fig. 4. Edge storage management design principles.**

**Table 2 Main Notations of Edge Storage Management**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>Vector containing forecast accuracies from the forecast iteration phase.</td>
</tr>
<tr>
<td>$S_{factor}$</td>
<td>Scaling factor dividing standard deviation of $\gamma$ used to determine threshold for finding clusters.</td>
</tr>
<tr>
<td>$v_{\gamma}$</td>
<td>Standard deviation (volatility) of $\gamma$.</td>
</tr>
<tr>
<td>$CL_{th}$</td>
<td>Threshold in identifying stable accuracy clusters.</td>
</tr>
<tr>
<td>$temp_{ac}$</td>
<td>Temporal vector that contains standard deviations calculated from the sampled vector $\gamma$.</td>
</tr>
<tr>
<td>$C$</td>
<td>Matrix containing recognized stable accuracy clusters with corresponding data.</td>
</tr>
<tr>
<td>$acc_{th}$</td>
<td>Forecast accuracy threshold.</td>
</tr>
<tr>
<td>$CL_{ap}$</td>
<td>Appropriate cluster with stable forecast accuracy.</td>
</tr>
<tr>
<td>$f_{th}$</td>
<td>Forecast horizon - the amount of data into the future for which the forecasts are to be calculated.</td>
</tr>
<tr>
<td>$S_{data}$</td>
<td>Array representing available dataset in storage.</td>
</tr>
<tr>
<td>$d_{factor}$</td>
<td>Decrement factor that decreases available dataset</td>
</tr>
<tr>
<td>$S_{data}$</td>
<td>Array during the forecast iteration phase.</td>
</tr>
</tbody>
</table>
are many clustering techniques [29] such as partitioning, hierarchical or density-based, but they are not suitable for our case, because often they require specification of a certain number of clusters beforehand and additionally they separate the entire dataset based on similarity. Our case requires a dynamic approach in which we discover as few clusters as possible based on Definition 2, and considering only corresponding parts of the entire dataset. The process consists of three steps: first, we calculate overall standard deviation for all forecast accuracies and mark it as a baseline. Second, forecast accuracies are grouped into clusters of fixed length and standard deviation is calculated per cluster. Third, obtained deviations are compared to the baseline considering the previously calculated threshold. Consequently, stable clusters show where the forecast accuracies are stable. Pseudo-code for detecting stable accuracy clusters is presented in Algorithm 3. The method requires vector $\gamma$ consisting of forecast accuracy measures (MAPA) from the forecast iteration process and scaling factor $S_{factor}$ that is used for the threshold calculation. The threshold $CL_{th}$ differs between different datasets, because each measurement has its own scale of values with unpredicted volatility. Based on experiments, by default the scaling factor is always equal to 5 in the first attempt of stable clusters detection. This means that only stable clusters with $CL_{pct}$ equal to 20% (see Definition 2) of the baseline standard deviation will be selected. However, even with the fixed threshold it is possible to have no clusters. In case that is impossible to meet any stable clusters for the specified threshold, that is, since forecast accuracies show greater dispersion, threshold is increased and the process is repeated. For example, decreasing the scaling factor from 5 to 4, the $CL_{pct}$ becomes 25% of the baseline for detecting stable clusters, by setting in Algorithm 5. In line 1, the algorithm calculates standard deviation of the entire vector $\gamma$ and divides the result, in line 2, by scaling factor $S_{factor}$ for the purpose of setting a threshold $CL_{th}$ for finding clusters. In line 3, standard deviations will be calculated for each of grouped iteration results in sliding window in vector $\gamma$ and then stored in temporal vector $temp_c$. Before searching for stable clusters, algorithm initializes two counters and creates one empty matrix in lines 4-5, that is, counter $i$ will count clusters in $temp_c$, and counter $j$ will denote discovered stable clusters in matrix $C$ including appropriate attributes such mean value of forecast accuracies in cluster, and corresponding range of cluster indexes. To detect stable clusters, the algorithm starts from the beginning of $temp_c$ (line 6) and checks if standard deviation for the first cluster is below threshold $CL_{th}$ (line 7). If cluster is recognized as a stable, corresponding data will be added in a new row of matrix $C$ (lines 8-11). In some cases, stable clusters can be wider, so it is necessary to check the neighbor cluster (line 12), and if the new one is recognized as stable (line 13) then it will continue to check other neighbors (line 14). For each new cluster in a row recognized as stable, algorithm extends existing cluster updating its corresponding mean value and end index (lines 15-16) and check the next cluster (line 17). When there are no more stable clusters in a row, a place for new stable cluster is prepared increasing counter $j$ in line 19. In case the next cluster is not recognized as stable (line 20), algorithm will simply close the existing cluster and check the next cluster (line 21). For each non-stable cluster (line 22), the algorithm will increment counter $i$ (line 24) and loop back to line 6. Finally, matrix $C$, whose rows represent stable clusters with the attributes, is returned.

### 4.2 Detection of the Appropriate Cluster

The cluster selection process is described in Algorithm 4. It starts by checking the amount of stable clusters. The else branch in line 11 is executed only if one stable cluster is recognized and it will become the appropriate cluster (line 12), otherwise, algorithm will find appropriate cluster (lines 2-10). Considering the first priority, appropriate cluster becomes the one that is closest to the specified accuracy threshold (line 2). Further, all stable clusters (line 3) that have better accuracy than selected $CL_{ap}$, that is, higher mean value, and whose start index begins after end index of selected $CL_{ap}$ (line 4), become potential appropriate clusters (line 5). If there are such clusters (lines 8-10), the one including less data, that is, which has the lowest start index (line 9), is selected as a new appropriate cluster $CL_{ap}$. Finally, appropriate cluster $CL_{ap}$ is returned in line 14.

### 4.3 Adaptive Algorithm

The adaptive algorithm includes all design principles shown in Figure 4, and includes calls on described Algorithms 3 and 4. Algorithm 5 requires a forecast horizon $f_h$ and accuracy threshold $acc_h$ that are specified in the specification list, and array $Sdata$ that denotes data available in storage. As shown in Figure 4, the learning phase is inevitable. One of the possibilities can be finding periodicity as a necessity for determining the seasonality and thereby to make better forecast, as described in [30]. Next, at the beginning of the algorithm, the decrement factor $d_{factor}$ is calculated utilizing Equations 4 and 5. The calculated $d_{factor}$ will be decreasing storage data $Sdata$ in the multiple forecast

---

**Algorithm 3 DetectionOfStableClusters**

**Input:** Vector of iteration results $\gamma$, scaling factor $S_{factor}$

**Output:** Matrix $C$

1: $v_\gamma \leftarrow sd(\gamma)$  \> Calculate standard deviation (volatility) of entire vector $\gamma$
2: $CL_{th} \leftarrow \frac{v_\gamma}{S_{factor}}$  \> Calculate threshold $CL_{th}$
3: Create and fill vector $temp_c$, applying standard deviation on sliding window of length 3 on vector $\gamma$
4: $i \leftarrow 1, j \leftarrow 1$  \> Initialize counters $i$ and $j$
5: Create empty matrix $C$ with 3 columns representing mean value, start and end index of detected stable clusters
6: while $i < length(temp_c)$ do  \> Satisfying conditions in Equation 1
7: if $temp_c[i] < CL_{th}$ then  \> Satisfying conditions in Equation 1
8: Add cluster in $C$ such that $C[j, 1] \leftarrow mean$ value of corresponding range in $\gamma$
9: $C[j, 2] \leftarrow start$ data index of corresponding range
10: $C[j, 3] \leftarrow end$ data index of corresponding range
11: $i \leftarrow i + 1$  \> Incrementing $i$ to check next potential cluster member
12: if $temp_c[i] < CL_{th}$ then  \> Satisfying conditions in Equation 1
13: while $temp_c[i] < CL_{th}$ do  \> Calculate standard deviation on sliding window $v_\gamma$
14: Update mean value $C[j, 1]$
15: Update end index $C[j, 3]$
16: $i \leftarrow i + 1$
17: end while
18: $j \leftarrow j + 1$
19: else
20: $i \leftarrow i + 1, j \leftarrow j + 1$
21: end if
22: if $temp_c[i] < CL_{th}$ then
23: else
24: $i \leftarrow i + 1$
25: end if
26: end while
27: Return $C$
Algorithm 4 DetectionOfTheAppropriateCluster
Input: Matrix $C$, $acc_{m}$
Output: Appropriate cluster $CL_{ap}$
1: if $C$ has more than 1 cluster then
2: Compute $CL_{ap}$ using Equation 2
3: for each cluster $i \in C$ do
4: if $C[i]\text{mean-value} > CL_{ap}\text{mean-value}$ AND $C[i]\text{start-index} > CL_{ap}\text{end-index}$ then
5: Add $C[i]$ to temporary matrix $A$
6: end if
7: end for
8: if $A$ is not empty then
9: $CL_{ap} \leftarrow A$, with minimum starting index
10: end if
11: else
12: $CL_{ap} \leftarrow C[0]$
13: end if
14: Return $CL_{ap}$

Algorithm 5 AdaptiveAlgorithm
Input: forecast horizon $f_{h}$, $acc_{m}$, array $Sdata$ representing storage data
1: Calculate $dfactor$ using Equations 4 and 5
2: while length($Sdata$) $> 2 \times f_{h}$ do
3: Perform method $(Sdata, f_{h})$
4: Calculate MAPA (See Equation 7)
5: Add MAPA to vector $\gamma$
6: $Sdata \leftarrow Sdata$ decreased by $dfactor$
7: end while
8: $Sfactor \leftarrow 5$ \hspace{1cm} \triangleright Set threshold on 20\% of overall standard dev., that is, represented as $\psi$
9: $C \leftarrow \text{FindStableClusters}(\gamma, Sfactor)$
10: while $C$ is empty do
11: $Sfactor \leftarrow Sfactor - 1$ \hspace{1cm} \triangleright Decrease $Sfactor$
12: $C \leftarrow \text{FindStableClusters}(\gamma, Sfactor)$
13: end while
14: $CL_{ap} \leftarrow \text{FindAppropriateCluster}(C)$ \hspace{1cm} \triangleright Create matrix $CL_{ap}$
15: Release data from $Sdata$ based on range between the oldest and the central index of the appropriate cluster $CL_{ap}$, retaining only relevant data in the current cycle.

Iteration Phase. Forecast iterations (lines 2-7) will continue until amount of data in $Sdata$ becomes less than the two lengths of a forecast horizon (more details in Section 4.4). Appropriate forecast method uses storage data $Sdata$ and other attributes, such as periodicity, to make prediction for defined forecast horizon $f_{h}$ and calculates mean absolute percentage accuracy MAPA (see Equation 7) in lines 3-4. At the end of each iteration, the MAPA is stored in vector $\gamma$ (line 5) and a certain amount of old data is removed (line 6) based on decrement factor $dfactor$. Next, for the purpose of the detection of stable accuracy cluster phase (lines 9-13), scaling factor $Sfactor$ is set to number 5 representing the impact of 20\% in determining the threshold for finding stable clusters in algorithm 4. If any stable cluster is recognized, the matrix $C$ gets corresponding data (line 9) as: mean value, start and end index of the cluster. Otherwise, if stable clusters cannot be found (line 10), the algorithm will decrease the $Sfactor$ and keep looking for the clusters (lines 11-12). Line 14 finds the appropriate cluster $CL_{ap}$. Finally, data in array $Sdata$ are released in range between the oldest index and the central index of the appropriate cluster $CL_{ap}$ (line 15). Adaptive algorithm repeats itself based on demands in the specification list.

4.4 Optimal Parameter Settings
Our goal is to set up necessary parameters enabling continuous operation of the edge storage management mechanism. To allow a proper performance of proposed algorithms, storage must always contain enough data for training plus additional test data (amount of which is equal to defined forecast horizon), and there must be enough number of forecast iterations in each cycle in order to find stable clusters. This is ensured by keeping training data as

$$T = 2 \cdot f_{h} + k, \hspace{1cm} k \geq 3$$

where $T$ denotes amount of training data points, $f_{h}$ denotes a forecast horizon that is application dependent and given in the specification list (see Figure 1), and $k$ is a natural number. With this insurance, there will be always minimum 4 forecast iterations resulting in 4 MAPA measures as a basis to find at least one stable cluster (see definition in Section 4).

Further, running time heavily depends on number of multiple forecast iterations while this number is affected by the decrement factor. The decrement factor is calculated based on Equations 4 and 5, that is,

$$\psi = \text{round}(df_{pet} \cdot (T - 2 \cdot f_{h}))$$

where $\text{round()}$ represents a function that rounds the result half away from zero to integer and $df_{pet}$ denotes decrement factor percentage. In order to set optimal value for $df_{pet}$, we performed experiments using all possible ranges of $df_{pet}$ on different datasets. The evaluation is done on 144 data points, representing data collected every 5 min over 12 h with 1 h forecast horizon, satisfying Equation 3 to have enough iterations to find stable clusters. Thus, maximum $df_{pet}$ of 30\%, that is a representative from range of 26\%-33\%, gives necessary 4 forecast iterations. Results showed that as $df_{pet}$ becomes very low (1\%-2\%) and very high (8\%-30\%), the algorithm cannot always find stable clusters in the first run of the calculation (Algorithm 5, line 8) while at the same time having number of clustered MAPA measures near 100\%. Among rest $df_{pet}$ values, in order to satisfy the goal of finding a setting that releases more data without significantly decreasing the accuracy of the appropriate cluster, we set $df_{pet}$ at 3\% resulting in 34 iterations on average. Further, we assume that prediction of data, that can potentially contain seasonality, requires at least twice as much data points comparing to the forecast horizon. This assumption is derived from the constraint that prediction of one period of seasonal time series requires at least two periods of prior data. Accordingly, to calculate the decrement factor, training dataset is reduced by two lengths of the $f_{h}$ (Equation 4).

4.5 Complexity Analysis
Considering Algorithm 5, its complexity is $O(n^{2})$, where $n$ represents the amount of data in the dataset. ARIMA method (line 3) has $O(n)$ complexity and the outer while loop (lines 2-7) iterates whole dataset until $n$ is equal to the two lengths of specified forecast horizon. In the worst case, it is decreased by 1 at each iteration, leading to a $O(n)$ complexity. Further, both Algorithms 3 and 4 have complexity of $O(n)$. The while loop (line 6) from the Algorithm 3 iterates over the size of the temporal vector $temp_{m}$. The inner while loop (line 14) uses the same counter as the outer loop resulting in the same $O(n)$ and decreasing the space complexity by not creating new objects in line 8. Algorithm 4 instead iterates over each cluster in the for loop in line 3,
whose number is always less than $n$. All other operations have either a $O(1)$ or a $O(n)$, resulting into an overall $O(n^2)$ time complexity. Such complexity may be reduced by using less accurate forecasting methods. Even though a $O(n^2)$ is not suitable for big datasets, it provides acceptable response time in this context, since we target small datasets due to the storage limitations at the edge.

5 Mediator Component

Storage space limitations prevent us from storing all the measurements collected by edge systems. Many IoT systems require both local edge and global batch data analytics [12], [22]. Consequently, an edge node can keep only relevant data, and send all acquired data to be integrated into the cloud data repository. Once all data are available in the cloud, batch analytics can be applied. The mediator component can then be used: (i) when local analytics require more data to reach certain analytic tasks; (ii) when the mediator requires more data according to Projection Recovery Maps (PRM) that recommend ranges of data for recovering current gaps detected by the monitoring component (see Figure 1). In both cases the mediator can retrieve necessary data from the cloud data repository and can temporarily store them locally. Here, we describe the second case. Analytics can be executed in the cloud using historical data and then resulted PRM are sent to the mediator component. Finding recommended ranges of data for certain lengths of detected gaps can be done by slightly modifying concepts from the edge storage management mechanism. Selecting an appropriate cluster $CL_{ap\_med}$ means selecting a cluster with the highest accuracy, that is,

$$CL_{ap\_med} = \arg \max_{C[i]} (C[i]^{mean\_value})$$

With this rule, we omit the previously described twofold priority that considers accuracy threshold and usage of less data points in the appropriate cluster. Once the $CL_{ap\_med}$ is detected, upper and lower bound for number of used data points and corresponding MAPA measure are stored. This process is repeated for selected ranges of missing data points, and then merged in PRM for each monitored process or a dataset. To check accuracy of gap recovery, we test it on historical data, assuming it will work for the upcoming data. In cases where big data are expected, compute-intensive process can perform projection recovery maps using non-consecutive number of missing values and later, if required, interpolate ranges that are not calculated.

6 Performance Evaluation

We implement simulations of EDMFrame using R language. Simulations are run on a 64-bit Windows 10 PC, with a 2.70-2.90 GHz Intel i7-7500U CPU and 16 GB memory.

6.1 Data and Measures

We evaluate our approaches on data from real-world IoT systems. We target sensor-based time series data, typical in IoT data sources for applications like smart buildings and homes [31]. The main characteristics of datasets are presented in Table 3, namely: (i) Smart homes. Datasets from this source (h_1-3) contain traces from the Smart project [32] for optimization of energy consumption smart homes design, obtained by UMass Trace Repository [33]. It represents variety of environmental and electrical type of data, such as temperature, relative humidity (RH), wind information and heat index; (ii) Smart buildings. Datasets from this source (b_1-3) come from the monitoring system of Austria’s largest Plus-Energy Office High-Rise Building. Sensor-based data collection contains various measurements such as temperature, indoor air quality (IAQ), electricity consumption and production, that are used for operative decisions such as automatic heating and cooling, ventilation, efficient energy consumption. Regarding the forecast accuracy evaluation, we use Mean Absolute Percent Error (MAPE), that allows us to compare accuracy of the recovery among different types of datasets due to its scale independence. We also define Mean Absolute Percent Accuracy (MAPA) as in Equation 7:

$$MAPE(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

$$MAPE(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

where $MAPE(Y, \hat{Y})$ is equal to $\frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$, $Y$ and $\hat{Y}$ are respectively the set of the original values and the set of forecasts, $n$ is the number of data points, $y_i - \hat{y}_i$ represents the forecast error, and $y_i$ and $\hat{y}_i$ represent respectively the i-th original value of a data point and its prediction.

6.2 Data Recovery Process

We evaluate the applicability of the proposed approach (see Section 3) by recovering multiple gaps on different datasets. We see an example in Figure 5. The upper graph shows an incomplete subset of the dataset $h_1$ before the recovery, while the lower graph shows a complete dataset after recovery. Gray shaded areas indicates four gaps. In all datasets by source 2, we observe several gaps in collected data affecting data analytics. Therefore, after datasets analysis, we identify these gaps and artificially make several gaps with same sizes in the dataset $h_1$ (precisely, gaps with $1 (G_1^1), 2 (G_2^1), 17 (G_3^1)$ and 30 ($G_4^1$) consecutive missing/invalid data values). Then, these gaps are recovered using the proposed mechanism, and forecast accuracy is evaluated by comparing predicted data with actual data from each gap. The black solid line represents the actual state of a dataset at the edge. The black dashed line shows actual data for corresponding gaps, while the red solid line represents predicted values of missing/invalid data as an output of applied forecasting techniques. In this case, multiple gaps are automatically recovered using ARIMA, representing the STR scenario. Sensor-based time series, as shown in the first

<table>
<thead>
<tr>
<th>Source</th>
<th>Dataset</th>
<th>Sensor type</th>
<th>Range of values</th>
<th>Volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b_1</td>
<td>heat index [F]</td>
<td>52.11-88.79</td>
<td>8.74</td>
</tr>
<tr>
<td></td>
<td>b_2</td>
<td>room temp. [F]</td>
<td>68.90-79.70</td>
<td>2.51</td>
</tr>
<tr>
<td></td>
<td>b_3</td>
<td>RH [%]</td>
<td>30.2-92.7</td>
<td>18.52</td>
</tr>
<tr>
<td></td>
<td>b_1</td>
<td>el. meter [kWh]</td>
<td>19.86-19.97</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>b_2</td>
<td>flow temp. [C]</td>
<td>41.3-48.1</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>b_3</td>
<td>IAQ [ppm]</td>
<td>458.11-707.88</td>
<td>62.53</td>
</tr>
</tbody>
</table>
sub-figure, can contain different behaviors from stationary to trend and volatile patterns. For this reason, the MAPE of reconstructed gaps $G_1^1$, $G_2^2$ and $G_1^3$ are significantly low, 0.1843%, 0.1317% and 0.3366% respectively, while the MAPE for the $G_4^3$ is 2.6797%. We also see that, as the gap increases, the forecasting error increases too. Finally, the result confirms that the proposed mechanism is able to efficiently recover all gaps with running time of 0.68s. On the other hand, based on the proposed mechanism, multiple techniques can be involved in recovery process of each gap separately, that is, the MTR scenario. The MTR will be employed by the mediator component in Section 6.4. Full-automatic selection at runtime is left for future research.

6.3 Storage Management Process
We simulate edge storage management under fixed amount of upcoming data. We use the adaptive algorithm (Algorithm 4) for selection of the appropriate cluster. The evaluated dataset $h_{11}$ is shown in Figure 6 and represents 3 cycles of edge storage management. In the first cycle, the same data from the previously described data recovery process are used. Further, in second and third cycle, we set, in addition to retained data from the previous cycle, the upcoming amount of data on 144 data points per turn, representing collected data over 12h (every 5min). For the experiments, we manually define some rules to simulate the specification list. Forecast horizon $f_h$ is set to 12 data points, representing 1 hour (12-5min) for which the forecasts will be calculated. $f_h$ stays fixed in the current cycle, while the user can change the desired $f_h$ in the specification list. Then, we consider the threshold $CL_{ap}$ for identifying stable accuracy clusters of 90%, since the proposed framework targets near real-time decision making in IoT applications and we expect accurate MAPA measures resulted due to the short $f_h$ in comparison with the available dataset size. Regarding the Figure 6, the upper graph represents the original dataset, where denoted vertical blue dotted line represents the data management decision after the procedures in lower graphs.

The lower graphs represent the result after applying forecasting method and validating the principle for multiple forecast iterations on the available dataset. Among MAPA measurements, orange shaded areas represent stable clusters, while green shaded areas represent selected appropriate clusters. The last 12 data points become test data and the rest will be used in different variations to predict the $f_h$. Considering the first cycle, the algorithm has found appropriate cluster in range between 68 and 108 available data points, corresponding to the cluster between data indexes 168 and 208 in our original dataset (upper graph). Central index of that cluster indicates that data management action will release data points in range 1-188, respectively, indexes in range 189-288 will retain in the edge storage. The process repeats for each next cycle. Figure 7 summarizes all 5 cycles. Sub-figure 7a shows both, released and retained data per each cycle, while Sub-figure 7b shows accuracy of the selected appropriate cluster $CL_{ap}$ and the percentage of clustered MAPA values (lower graphs in Figure 6) per cycle. In Table 4 all 5 cycles are averaged and compared among 6 different datasets (see Table 3). The results have shown that it is possible to retain 39.9% of data points on average, while keeping the high accuracy of the appropriate cluster $CL_{ap}$, depending on parameter settings of proposed algorithms. Generally, we are able to find appropriate clusters while having clustered MAPA measurements always above 50% and based on approximately 34 multiple forecast iterations. Results show that using edge storage management mechanism we can reduce the amount of data stored in edge nodes, respecting user-set prediction accuracy.

6.4 Mediator Process
In Section 4.3 we define two scenarios for recovering gaps in our proposed adaptive algorithm: Section 6.2 describes the recovering mechanism where multiple gaps are recovered using a single method (in our case, ARIMA). In this case the algorithm uses only available edge stored data and all data preceding a gap. In the second scenario we introduce PRM, used by recovery mechanism to select recommended range of required data points that corresponds to the number of missing values. In Figure 8, for each number of missing values from 1 to 144, the algorithm finds recommended range of required data points by finding stable clusters (Algorithm 3) and then calculating the most accurate cluster. Here, the original algorithm is modified by considering stable clusters with highest forecast accuracy (see Definition 6). The blue line represents upper border of the cluster, while the green line depicts the lower border. ETS and ARIMA methods are applied, and the method with the highest
Fig. 6. Evaluation of edge storage management on h_1 dataset - cycles 1-3 showing available dataset (upper graphs) and stable clusters of forecast accuracies (lower graphs). According to calculated appropriate clusters, only relevant data points are retained in the edge storage (blue dotted line).

Fig. 7. Results of the adaptive algorithm after 5 cycles, showing amounts of released/retained data, accuracy of the appropriate cluster and percentage of clustered forecast accuracies in the process.

7 RELATED WORK

Authors in [34] focus on data management solutions, proposing a comprehensive description of components of IoT data management framework. Furthermore, in [35], univariate imputation is used for air pollution data, considering fixed size gaps. The challenge of recovering missing data has been investigated by many researchers, providing methods relying on cubic interpolation [36], Singular Spectrum Analysis [37] or Lomb-Scargle method [38]. However, these works do not target IoT scenarios. Concerning time-series forecasting, authors in [24] describe time series forecast methods, including a self-adaptive approach for method selection based on users’ forecasting objectives. The use of different methods is motivated by the fact that forecast accuracy depends on characteristics of data before each gap. ARIMA and ETS methods are described in [26], [39]. These methods do not require constant user interactions, which makes them suitable for IoT applications. Micro data centers are described in [40]. The problem of reducing data transmission at the edge has been discussed by several works. In [41], a solution for network-edge data reduction targeting IoT devices is presented, without considering latency requirements of IoT applications and improvement of data quality by using different forecasting techniques. Paper [42] proposes a dynamic compression-based technique for sensor data. Works like [43], focus on data storage structure, memory allocation strategy and data compression to efficiently use storage capacity.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we introduce EDMFrame, a framework for data management at the edge featuring a mechanism for
recovery of multiple gaps in time series coming from IoT sources, then a novel approach to support reliable decisions on the space-limited storage at runtime using an adaptive algorithm, and projection recovery maps (PRM) to support efficient data recovery. Results show that EDMFrame can: (i) reduce the amount of data stored on edge nodes, (ii) efficiently recover multiple gaps within incomplete data and (iii) improve the predictive analytics for IoT applications such as smart buildings. In future work, we will extend it by considering density of data generation affecting the PRM calculation for improved data recovery, in particular a dynamic decrement factor percentage, and using EDMFrame towards reliable distributed ML applications at the edge.

ACKNOWLEDGMENTS

The work described in this article has been funded through Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015 and supported with Ivan Lujic’s netidee scholarship by the Internet Foundation Austria. The authors would like to thank staff at TU Wien’s Plus-Energy Office High-Rise Building, especially to Thomas Bednar and Alexander David on valuable discussions and the Federal Real Estate Company (BIG) for providing data sources.

REFERENCES


Ivan Lujic received his bachelor’s and master’s degree in Computer Science from the University of Split, Croatia. He is currently a PhD student and research assistant at the Institute of Information Systems Engineering at Vienna University of Technology, Austria. His research interests include monitoring, data management and data analytics in emerging Cloud/Edge computing technologies. He is currently employed by the RUCON project (http://rucon.ec.tuwien.ac.at/).

Vincenzo De Maio received his PhD in 2016 at the University of Innsbruck, Austria. His research in the area of parallel and distributed systems comprises energy-aware Cloud computing and scheduling. Since 2017, he is a postdoctoral researcher at the Institute of Information Systems Engineering of the Vienna University of Technology. He authored different conference and journal publications on the topic of energy efficiency and modeling for Cloud and Edge computing.

Ivona Brandic is a Professor at Vienna University of Technology. In 2015 she was awarded FWF START prize, the highest Austrian award for young researchers. She received her PhD degree in 2007 from Vienna University of Technology. In 2011 she received the Distinguished Young Scientist Award from the Vienna University of Technology for her project on the Holistic Energy Efficient Hybrid Clouds. Her main research interests are cloud computing, large scale distributed systems, energy efficiency, QoS and autonomic computing.