Efficient Edge Storage Management Based on Near Real-Time Forecasts

Ivan Lujic, Vincenzo De Maio, Ivona Brandic
Institute of Software Technology and Interactive Systems, Vienna University of Technology
Favoritenstrasse 9-11/188, A-1040 Vienna, Austria
{ivan, vincenzo, ivona}@ec.tuwien.ac.at

***Abstract*—Nowadays, data analytics is utilized on edge based systems to perform near real-time decisions in proximity of the user. When performing near real-time decisions on the Edge, we need historical data to perform accurate data analytics. Since storage capacities on the Edge are limited, we are faced with a challenge to balance the quantity of data stored with the quality of near real-time decisions. In this paper, we present a three-layer architecture model for data storage management on the Edge including an adaptive algorithm that dynamically finds a trade-off between providing high forecast accuracy necessary for efficient real-time decisions, and minimizing the amount of data stored in the space-limited storage. We focus on time series data, typical in the context of sensor-based monitoring in IoT environments. By using the proposed approach it is possible to reduce the amount of stored data by an average 80.27% without affecting specified threshold for prediction accuracy.**

***Index Terms*—edge systems, edge storage management, forecast accuracy, real-time decision.**

I. Introduction

Recently, IoT devices are used in a wide set of applications, such as Smart Cities [1], Health-care [2] and Traffic Management [3]. Managing such systems generally requires three steps: (1) collecting data through sensors, (2) processing these data and (3) acting based on the obtained information [4]. In most of existing systems, such processing is done in massive data centers. However, since massive data centers are usually not in the vicinity of the IoT end device producing the data, transferring data from sensors to massive data centers may result in high latency. Since such systems have very strict latency requirements [5], reducing the time for collecting and processing the data of IoT devices is of paramount importance.

Recently, Edge analytics has become one of the most widely used solutions to this problem [6]. In Edge analytics, analysis and collection of data coming from IoT devices is performed in nodes that are closer to them than massive data centers. By placing data analytics close to the source of data, Edge architectures can reduce the amount of data traversing the network, thus minimizing latency and overall costs [7]. Plus, they allow to get fast and accurate responses, thanks to the possibility to analyze batch and streaming measurements while simultaneously moving data analytics to the Edge [8]. However, storing a huge amount of data on the Edge can result in several problems due to the storage limitations on the Edge nodes [9]. As real-time analytics require a lot of historical data to perform accurate predictions, storage efficient real-time analytics becomes a key issue on the Edge [10].

Storage management problem has been discussed by works like [11], [12] but they consider neither limitation of storage capacity on the Edge nor accuracy of near real-time analytics for sensitive Edge systems. Due to the always increasing use of data exchanged by IoT devices [13] and the growing tendency of using Edge nodes for performing real-time analytics on them, proposing a way to reduce the amount of data stored on the Edge nodes without affecting forecast accuracy can bring substantial benefits in this context.

In this paper, we present a three-layer architecture model for efficient data storage management in Edge analytics. The proposed architecture ensures flow, analysis and storage of data in order to cope with limited storage capacity on the Edge. Our solution also supports dynamic analysis of monitoring data to cope with behavior of unpredictable systems requiring constant monitoring of data continuously read by the sensors. Based on the proposed model, we derive an adaptive algorithm that finds a trade-off between reducing the necessary storage space needed and accuracy of forecasts. We utilize different forecast methods and different methods for measuring predictions accuracy to determine the amount of data that need to be stored in a Edge node. Finally, we evaluate the applicability of our approach in an experimental scenario by utilizing different datasets.

This work targets time series data coming from IoT sensors. We evaluate our adaptive algorithm performing simulations with the R forecast package [14], as done by works like [15], [16]. The simulation results show an average potential reduction of stored data amount by an average 80.27%, while keeping accuracies of prediction above a user-defined threshold.

The rest of the paper is organized as follows. Background on time series data, their role and related forecasting methods are covered in Section II. Section III presents the Edge architecture model for Edge storage management. In Section IV the design principles of the algorithm are shown, while Section V gives a detailed description of the proposed adaptive algorithm. In Section VI, we present the evaluation of the proposed approach and discuss the results. Related work is outlined in Section VII and Section VIII concludes the paper.

II. Background on time series forecast

In this section, we discuss methodologies used to define the adaptive algorithm and to implement architecture for efficient

edge storage management. We discuss the data types used in this paper and methods for measuring accuracy of forecasts.

A. Time series data

Generally, data generated by sensor-based monitoring are classified as time series data [17]. A time series is a sequence of data points made over a continuous time interval, where each data point consists of a time stamp and one or multiple values. In this paper we consider only this type of data, in contrast to other IoT data types such as video, audio, status or location data. Analysis and forecasting based on time series has been a very active research area over the past few decades. Thus, based on historical data it is possible to predict the future values of time series data. Determining accuracy of time series forecasting is an essential step in many decision-making processes. For this reason, improving the effectiveness of forecasting models is of great interest for many time-sensitive Edge systems [18]. Based on time series forecasting and analysis of unusual data behavior, life-threatening situations and system failures can be detected even before they occur [17].

B. Forecasting methods and forecast accuracy measure

The most widely used forecasting techniques are the Auto-Regressive Integrated Moving Average (ARIMA) [19] and the Exponential Time Smoothing (ETS) [20]. Both techniques can also deal with unusual time series patterns, estimate parameters and compute forecasts without user intervention [15]. Unusual time series have features such as unusually high seasonality [21]. Thus, both methods are suitable for Edge analytics, where analysis of data has to be performed automatically.

To assess the accuracy of the forecasts, we utilize the Mean Absolute Percentage Accuracy (MAPA) measure. This measure is derived from Mean Absolute Percentage Error (MAPE) [22] accuracy measure and expresses accuracy of prediction based on forecast error. We compute MAPA by calculating the MAPE between our forecasts and the original dataset, as shown in Equation 1.

$$\texttt{MAPA}(Y, \hat{Y}) = 100 - \texttt{MAPE}(Y, \hat{Y}) \tag{1}$$

$\texttt{MAPE}(Y, \hat{Y})$ is equal to $\frac{100}{n} \sum_{i=1}^{n} |\frac{(y_i - \hat{y}_i)}{y_i}|$, where 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 forecast error, and y_i and $\hat{y}_i$ represent respectively the i-th original value of an individual data point and its prediction. The choice of MAPA allows us to have an accuracy measure based on MAPE. MAPE facilitates comparison of forecast accuracy on different time series since it is scale-independent.

III. Architecture for Edge storage management

Figure 1 represents the proposed three-layer architecture model for Edge storage management on Edge nodes. The goal of the presented Edge architecture is to devise a self-management process to automatically manage limited Edge data storage without intervention of third party or providers. Each Edge monitoring process includes three software layers, namely: Cloud layer, Edge layer and Gathering layer. Even though the main scope of this paper is the Edge layer, we describe all of them for the sake of completeness.

a) Cloud layer: represents data repository which stores all historical data collected from monitored systems. This layer performs big data analytics and delivers useful information based on entire datasets.

b) Edge layer: manages data storage process based on our proposed adaptive algorithm. Further, this layer performs local analytics and extracts actionable information from available data. Edge layer consists of several components:

- *Monitoring component* collects the data, monitors average amount of incoming data, monitors storage space, sends information on storage capacity and control commands to IoT actuators;
- *Data preparation component* receives data from monitoring component and performs data preparation operations on these data (e.g., filling missing data, removing unnecessary data, normalizing data). After this step, the data are sent either to Edge data storage or to the Cloud layer via mediator component, based on the specification list;

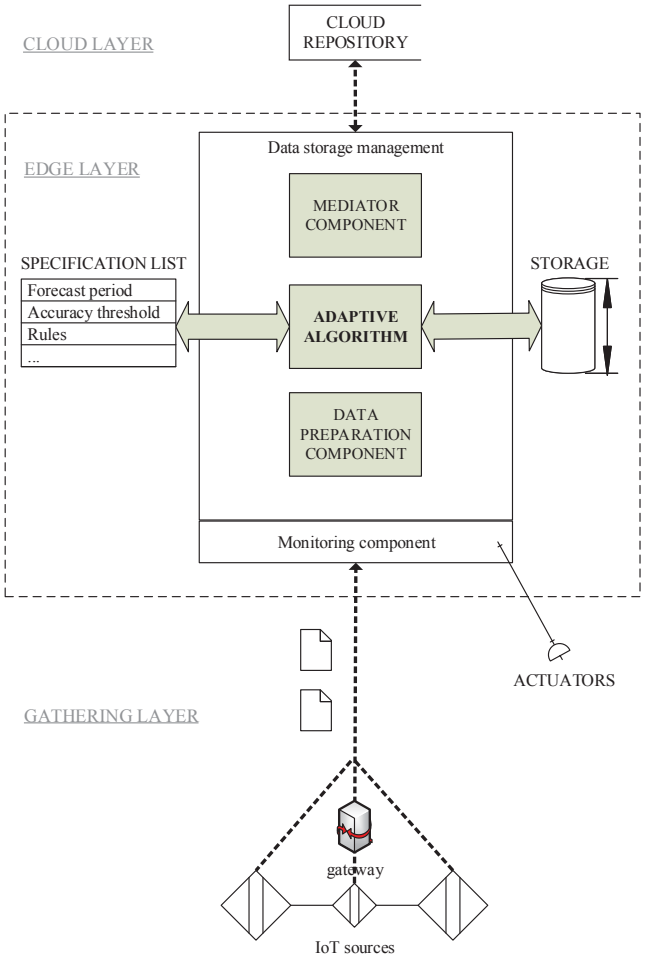

Fig. 1. Architecture for Edge storage management

- *Adaptive algorithm* provides the core of the data storage management. It receives data from the storage, checks the specification list, and implements design principles provided in Section IV;
- *Specification list* consists of user-defined information for monitored data and for adaptive algorithm such as: forecast period specified by client for monitored process or by the edge system whose data will be analyzed, forecast accuracy threshold and other rules (e.g., which method to use for forecasts, which data do not have to be forwarded to the Cloud data repository);
- *Storage* is responsible for (1) storing data received by data preparation component, (2) sending data to the adaptive algorithm, (3) receiving data retrieved by mediator component and (4) storing adaptive algorithm results;
- *Mediator component* requests to the cloud repository the needed range of data and forwards them to the storage.

c) Gathering layer: transmits IoT measurements to an Edge node, either indirectly via gateway (in case of devices that are not capable of sending data through IP) or directly (IP-enabled devices). To reduce communication costs and latency in distributed sensor networks [23], gateways help in aggregation of sensor data by sending them in appropriate message format and size to the monitoring component.

In this work, we focus on the data storage management process, describing the adaptive algorithm and its interaction between specification list and storage, that is described in the following sections. The other layers are left as future work.

IV. Algorithm design

Considering previous architecture model, our algorithm is shown in Figure 2, including following phases: (1) learning phase, (2) validation of the specification list, (3) multiple forecast iteration on available dataset, (4) detection of stable accuracy clusters, (5) data management action, (6) validation of available dataset. The learning phase is executed only once, as it provides information used by all the other phases, that are continuously executed starting from phase 2. We describe now each one of them:

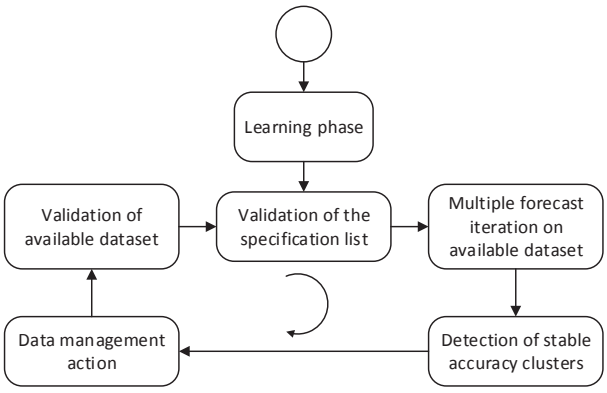

Fig. 2. Algorithm design principles.

a) Learning phase: the aim of the first phase is to derive information about data the first time the algorithm is executed. Such information is used during the other phases. This phase includes actions such as time series pattern recognition, that is useful to determine the most appropriate forecast method, allowing us to select the most appropriate method for that specific pattern. Lin et al. [24] discussed the state-of-the-art techniques for time series pattern recognition used in this paper. Other information, such as periodicity of time series data and seasonality over a certain period, can be useful to set up a forecast method [25].

b) Validation of the specification list: in this phase the specification list defined by a client is validated. During the execution of the proposed algorithm, clients can make some changes in the specification list anytime, such as setting forecast accuracy threshold, a new forecast period or different forecast methods. Any changes made by the client can affect the whole Edge storage management. Thus, there is the need of checking this list each time a cycle of execution starts.

c) Multiple forecast iteration on available dataset: this phase takes one of the forecasting methods (in our case ETS or ARIMA) with specified accuracy threshold and forecast period from the specification list. The available dataset is separated into two datasets representing training and test data. Test data are always fixed and equal to the number of data points specified by the client in the specification list as they are used only for accuracy evaluation purposes. The amount of training data is reduced in each iteration by a certain amount of data in order to find parts of dataset that results in required forecast accuracy. At the end of each iteration, forecast accuracy measures are added in a vector to be used in the next phase.

d) Detection of stable accuracy clusters: the aim of this step is to find stable clusters of accuracy values within the vector returned by the multiple forecast iteration phase.

Definition 1. *A **stable cluster** is a set of subsequent data points in the vector whose standard deviation for contained values is less than a given percentage s of the standard deviation of entire vector.*

Accordingly, in order to provide reliable information regarding future system behaviors our predictions must be stable. When the forecast iteration process is finished, a technique for 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 in Sections V-A and V-B.

e) Data management action: this step either releases storage or queries for more data to mediator component. There are three possible cases: (1) We can reach stable cluster with desired accuracy with fewer amount of data. In this case, all data not needed to obtain the observed accuracy cluster are deleted; (2) Forecast accuracy of stable cluster is higher with increased amount of training data, e.g. forecast based on all available data from the storage, then mediator component can

send a request to cloud data repository for more data (e.g., for seasonal pattern it can retrieve one more period of data); (3) None of resulting accuracy of stable clusters meet the specified accuracy threshold by the client. In this case data management action will select the one with highest forecast accuracy but obtained with less data points.

f) Validation of available dataset: algorithm checks storage for new data that are collected while the data management action is completed. In the next cycle both data received from data preparation and from mediator component are included.

The adaptive algorithm is continuously repeated and consequently, it tries to keep the lower quantity of data on the Edge, while trying to keep a level of accuracy that is equal to the forecast accuracy threshold value in the specification list. 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 more data from the past, and amount of current stored data will soon exceed storage limitation, then this kind of data analytics will be performed in the cloud environment. Thus, the approach requires to constantly monitor the evolution of the accuracy rate for performed forecasts.

V. Adaptive Algorithm

According to the steps that we defined in Section IV, we develop an algorithm that is capable of reducing the amount of monitored data without reducing forecast accuracy. All these steps are performed in three main algorithms: the `FindStableClusters` (Section V-A), the `FindAppropriateClusters` (Section V-B) and the `AdaptiveAlgorithm` (Section V-C). Each one of these is detailed in the following sections. Table I describes the notation used.

TABLE I
Definitions

Symbol	Definition
γ	Vector containing forecast accuracies from the forecast iteration phase.
S_{factor}	Scaling factor dividing standard deviation of γ used to determine threshold for finding clusters.
sd	Standard deviation of γ.
CL_{th}	Threshold in identifying stable forecast accuracy clusters.
$temp_v$	Temporal vector that contains standard deviations calculated from the sampled vector γ.
$\mathcal{C}$	Matrix containing recognized stable clusters.
acc_{th}	Forecast accuracy threshold specified by the client in the specification list.
CL_{ap}	Appropriate cluster that has stable forecast accuracy.
f_p	Forecast period specified in the specification list by client.
$Sdata_1, Sdata_2$	Available dataset in storage.
$periodicity$	Period contained in seasonal data.
d_{factor}	Decrement factor that decreases available dataset $Sdata$ during the forecast iteration phase.

A. Find stable clusters

Detection of smooth behaviors for consecutive forecast accuracies previously calculated due the forecast iteration phase, represents the cornerstone of our algorithm. There are many clustering techniques [26] 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 less as possible number of clusters based on Definition 1, 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 in clusters of fixed length and standard deviation is calculated per cluster. A cluster contains at least three members. Third, obtained deviations are compared to the baseline based on the previously calculated threshold. Consequently, stable clusters are used to show where the forecast accuracies are stable. Pseudo-code for the method of the cluster accuracy detection is presented in Algorithm 1. The method requires vector γ consisting of forecast accuracy measures (MAPA) from the forecast iteration process and scaling factor S_{factor}, that is used for threshold calculation.

In line 1, algorithm calculates standard deviation sd of the entire vector γ and divides the result in line 2 by scaling factor S_{factor} for the purpose of setting a threshold CL_{th} for finding clusters. The threshold CL_{th} differs between different datasets, because each measurement has its own scale of values with unpredicted volatility. By default, scaling factor is always equal to 5 in the first attempt of stable clusters detection. This means that only stable clusters with s equal to 20% (See Definition 1) of the baseline standard deviation will be selected. Howeever, 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, i.e. since forecast accuracies show greater dispersion, threshold is increased and the process is repeated. Decreasing the scaling factor from 5 to 4, the s becomes 25% of the baseline for detecting stable clusters, by setting in Algorithm 3. Further, in line 3 standard deviation will be calculated for each of grouped iteration results in sliding window in vector γ and then stored in temporal vector $temp_v$. Before searching for stable clusters, algorithm initializes two counters and creates one empty matrix in lines 4-6, namely, counter i will count clusters in $temp_v$ and counter j will denote discovered stable clusters in matrix $\mathcal{C}$ including appropriate attributes such mean value of forecast accuracies in cluster, and corresponding range of cluster indexes. Lines 7-27 show loop for detecting stable clusters. The algorithm starts from the beginning of $temp_v$ (line 7) and checks if standard deviation for the first cluster is below threshold CL_{th} (line 8). If cluster is recognized as a stable, corresponding data will be added in a new row of matrix $\mathcal{C}$ (lines 9-12). In some cases, stable clusters can be wider, so it is necessary to check the neighbor cluster (lines

Algorithm 1: `FindStableClusters`

Input: Vector of iteration results γ, int S_{factor}
Output: Matrix $\mathcal{C}$ that represents stable accuracy clusters with corresponding information

1 **Calculate** $sd \leftarrow$ standard deviation of entire vector γ
2 **Calculate** $CL_{th} \leftarrow \frac{sd}{S_{factor}}$
3 **Create** and **fill** vector $temp_v$ applying standard deviation on sliding window of length 3 on vector of iteration results γ
4 **Set** counter i $\leftarrow 1$
5 **Set** counter j $\leftarrow 1$
6 **Create** empty matrix $\mathcal{C}$ with 3 columns representing mean value, start and end index
7 **while** $i < length(temp_v)$ **do**
8 **if** $temp_v[i] < CL_{th}$ **then**
9 **Add** cluster in $\mathcal{C}$ such that
10 $\mathcal{C}[j,1] \leftarrow$ mean value of corresponding range in γ;
11 $\mathcal{C}[j,2] \leftarrow$ start index of corresponding data from storage;
12 $\mathcal{C}[j,3] \leftarrow$ end index of corresponding data from storage;
13 **Increment** i
14 **if** $temp_v[i] < CL_{th}$ **then**
15 **while** $temp_v[i] < CL_{th}$ **do**
16 **Update** mean value $\mathcal{C}[j,1]$ for the extended cluster;
17 **Update** end index $\mathcal{C}[j,3]$ for the extended cluster;
18 **Increment** i
19 **end**
20 **Increment** j
21 **else**
22 **Increment** i
23 **Increment** j
24 **end**
25 **else**
26 **Increment** i
27 **end**
28 **end**
29 **Return** $\mathcal{C}$

13-14) and if the new one is recognized as stable then it will continue to check other neighbors (line 15). For each new cluster in a row recognized as stable, algorithm extends existing cluster updating its corresponding mean value (line 16) and end index (line 17) and check the next cluster (line 18). When there are no more stable clusters in a row, a place for new stable cluster is prepared increasing counter j (line 20). In case the next cluster is not recognized as stable (line 21), algorithm will simply close the existing cluster and check the next cluster (lines 22-23). For each cluster that is not recognized as stable (line 25), algorithm will increment counter (26) and loop back to line 7. Finally, matrix $\mathcal{C}$ will be returned in line 29. Each row in the matrix represents one cluster with the corresponding attributes.

B. Find appropriate cluster

Since it may happen that Algorithm 1 returns more than one stable cluster, we need to define how we select the most appropriate one. Stable clusters can differ in mean value and in amount of used data. Therefore, it is needed to set priorities for the selection. We propose a two-stage process for cluster selection and a corresponding algorithm. First priority is to satisfy the specified threshold defined by client. A stable cluster is **appropriate** if its mean value is the closest to the threshold value in specification list, as defined in Equation 2,

$$CL_{ap} = \arg\min_{\mathcal{C}[i]} \left(|\mathcal{C}[i]^{mean\_value} - acc_{th}|\right) \tag{2}$$

where $\mathcal{C}[i]^{mean\_value}$ is the mean value of forecast accuracies included in cluster $\mathcal{C}[i]$, and acc_{th} denotes forecast accuracy threshold specified by client in the specification list. The appropriate cluster CL_{ap} becomes the one with the minimum absolute difference between acc_{th} and $\mathcal{C}[i]^{mean\_value}$. Second priority is to find a cluster that has higher accuracy but using less data. If such stable cluster exists, it will be the newly selected cluster.

Algorithm 2 describes the cluster selection process. First, it starts in line 1 checking if there are one or more 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 needs to find appropriate cluster (lines 2-10). Considering the 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}, i.e. 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), one of them that includes less data, i.e. which has the lowest start index (line 9), will be selected as a new appropriate cluster CL_{ap}. Finally, appropriate cluster CL_{ap} is returned in line 14.

Algorithm 2: `FindAppropriateCluster`

Input: Matrix $\mathcal{C}$ that contains discovered stable clusters, int acc_{th}
Output: Appropriate cluster CL_{ap}

1 **if** $\mathcal{C}$ *has more than 1 cluster* **then**
2 **Compute** CL_{ap} using Equation 2
3 **for** *each cluster* $i \in \mathcal{C}$ **do**
4 **if** $\mathcal{C}[i]^{mean\_value} > CL_{ap}^{mean\_value}$ *AND* $\mathcal{C}[i]^{start\_index} > CL_{ap}^{end\_index}$ **then**
5 **Add** $\mathcal{C}[i]$ to temporary matrix $\mathcal{A}$
6 **end**
7 **end**
8 **if** $\mathcal{A}$ *is not empty* **then**
9 $CL_{ap} \leftarrow \mathcal{A}_i$ with minimum starting index
10 **end**
11 **else**
12 $CL_{ap} \leftarrow \mathcal{C}[0]$
13 **end**
14 **Return** CL_{ap}

C. Adaptive algorithm

The proposed algorithm is based on calculating prediction accuracies including the detection of clusters of stable accuracy values. Algorithm 3 requires forecast period f_p and accuracy threshold acc_{th} that are specified in the specification list, and arrays $Sdata_1$ and $Sdata_2$ which denote all

Algorithm 3: `AdaptiveAlgorithm`

Input: int f_p, int acc_{th}, array $Sdata_1$, array $Sdata_2$

1 $periodicity \leftarrow$ `findPeriodicity`$(Sdata_1)$
2 **if** $(periodicity > 1)$ **then**
3 $\quad d_{factor} \leftarrow periodicity$
4 **else**
5 $\quad d_{factor} \leftarrow \frac{f_p}{2}$
6 **end**
7 **while** $length(Sdata_1) > f_p$ **do**
8 $\quad$ **Perform** method $(Sdata_1, periodicity, f_p)$
9 $\quad$ **Calculate** MAPA (See Equation 1)
10 $\quad$ **Add** MAPA to vector γ
11 $\quad Sdata_1 \leftarrow Sdata_1$ decreased for d_{factor}
12 **end**
13 **Set** $S_{factor} \leftarrow 5$, i.e. set threshold on $\frac{1}{5}$ (20% of overall standard dev.)
14 $\mathcal{C} \leftarrow$ `FindStableClusters`(γ, S_{factor})
15 **while** $\mathcal{C}$ *is empty* **do**
16 $\quad$ **Decrease** S_{factor}
17 $\quad \mathcal{C} \leftarrow$ `FindStableClusters`(γ, S_{factor})
18 **end**
19 $CL_{ap} \leftarrow$ `FindAppropriateCluster`$(\mathcal{C})$
20 **Release** storage data from array $Sdata_2$ in range between the oldest index and the central index of the appropriate cluster CL_{ap}
21 **Replace** data in storage by data in array $Sdata_2$

available data in storage. Lines 1-6 represent learning phase from Section IV. First, finding periodicity (line 1) becomes a necessity for determining the seasonality and thereby to make better forecast. To this end, we use the method described in [27]. In line 2, the algorithm checks the periodicity and in case it exists, the decrement factor d_{factor} is set to the same value as periodicity (line 3). Otherwise, if no periodicity is found (line 4), periodicity is 1, that means these data are no-seasonal. Consequently, decrement factor d_{factor} is set to half the forecast period f_p or the whole f_p that is specified in the specification list (line 5). Mentioned decrement factor d_{factor} will decrease storage data $Sdata_1$ in forecast iteration phase (second phase in the model). Forecast iterations (lines 7-12) will continue until amount of data in array $Sdata_1$ becomes less than forecast period (line 7). Appropriate forecast method uses storage data $Sdata_1$ and calculated periodicity (line 1) to make forecast for defined forecast period f_p and calculates mean absolute percentage accuracy MAPA (lines 8-9). At the end of each iteration the forecast measure is stored in vector γ and a certain amount of old data is removed (line 11) based on decrement factor d_{factor}. For the purpose of the phase for accuracy cluster detection (lines 13-18), scaling factor S_{factor} is set in line 13 to number 5 representing the impact of 20% in determining the threshold for finding stable clusters in algorithm 1. If any stable cluster is recognized, the matrix $\mathcal{C}$ gets corresponding data (line 14) such as: mean value, start and end index of cluster. In case that matrix $\mathcal{C}$ does not have data, i.e. clusters cannot be found, the algorithm will decrease the S_{factor} and keep looking for the clusters (lines 15-18). Line 19 finds appropriate cluster CL_{ap}. Finally, appropriate cluster CL_{ap} includes mean value that is stable and closest to required forecast accuracy in corresponding indexes. Accordingly, data in array $Sdata_2$ are released in range between the oldest index and the index calculated by median of indexes from the appropriate cluster CL_{ap} (line 20). Data in storage are replaced by data in array $Sdata_2$ (line 21). Adaptive algorithm repeats itself based on demands in the specification list.

D. Complexity analysis

Time complexity is related to the selected forecasting method. Considering Algorithm 3, its complexity is $O(n^2)$, where n is the size of the dataset. Since ARIMA method (line 8) has $O(n)$ complexity and the outer $while$ loop iterates whole dataset until n is equal to forecast period. In the worst case, it is decreased by 1 at each iteration, leading to a $O(n)$ complexity. Further, both Algorithm 1 and Algorithm 2 have complexity of $O(n)$. Considering Algorithm 1, we analyze the $while$ loop in line 7, that iterates over the size of the dataset. Algorithm 2 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 $O(n)$ complexity, resulting into an overall $O(n^2)$ time complexity. Such complexity may be reduced by using less accurate forecast methods. Even though a $O(n^2)$ is not suitable for big datasets, it can provide acceptable response time in this scenario, since we target small datasets due to the storage space limitations on the Edge.

VI. Evaluation

First, we implement the algorithms in R language, using the R forecast package. Then, we evaluate the results of the approach by evaluating both the accuracy of the forecasts and the amount of data that have to store on the Edge node. Datasets are obtained by UMass Trace Repository [28] and contain traces coming from the Smart* project [29] for purpose of designing sustainable homes. These traces represent real data and contain potential information for IoT actuators, that makes these datasets appropriate samples for our experiments. We have selected three datasets targeting different characteristics of datasets, to show the applicability of our algorithm. Table II shows some characteristics of used datasets such as size (number of data points), periodicity, and range of values.

The first test case shows experimental scenario and application of the adaptive algorithm on loaded dataset, focusing on recognition and selection of clusters with stable forecast accuracy. The second test case shows the benefits of implementing our algorithm in the Edge storage data management.

A. Experimental scenario

To explore the full potential of the adaptive algorithm, we used the dataset with seasonality pattern that will have more than one cluster with stable forecast accuracy. In this subsection, we provide a step by step evaluation of the algorithm. For the experiments, we manually defined some rules to simulate the specification list. Forecast period is set to 24, the same as periodicity of dataset. Forecast period stays fixed in a current cycle, while a client can change the desired forecast period in the specification list anytime. Then, we

TABLE II
INFORMATION ABOUT DATASETS.

Dataset	Size	Periodicity	Range of values
1	336	24	10.06 - 31.33
2	600	1	65.2 - 79
3	7000	1	56.30 - 100.2

TABLE III
STABLE CLUSTERS OF FORECAST ACCURACY.

#	Mean value	Range of cluster indexes		Interval of available data points	
		Start	End		
1	92.44903	12	48	300	264
2	94.13858	132	192	180	120

consider high threshold for forecast accuracy of 90%, since we expect high forecast accuracy results due to the small forecasting period in comparison with the available dataset size.

In Figure 3, the upper graph represents the original dataset, while the lower graph represents the result after applying ETS forecast method validating the principle for multiple forecast iterations on available dataset. Original dataset is composed of 336 data points. As we already defined forecast period, the last 24 data points become test data and the others will be used in different amounts to predict the forecast period.

Bold blue points in lower graph indicate forecast accuracies for corresponding number of used data, i.e. number of data that is artificially decreased in each iteration in order to capture different forecast values estimating the same forecast period. For example, the first blue point shows us that in forecast process 312 data points are used and the forecast accuracy is slightly below 93%. In lower graph of Figure 3, some stable behaviors of forecast accuracies are visible, and their automatic recognition is performed by Algorithm 1. Two stable clusters are selected and shown in Table III. The resulting table provides mean values of accuracies in each cluster and corresponding indexes of data points from the storage for each cluster. Additionally, interval of available data points shows how many data points remain between start/end index of cluster. Sum of the corresponding cells gives the total number of test data.

Selection of appropriate cluster is done according to Algorithm 2. The forecast accuracy threshold for the current

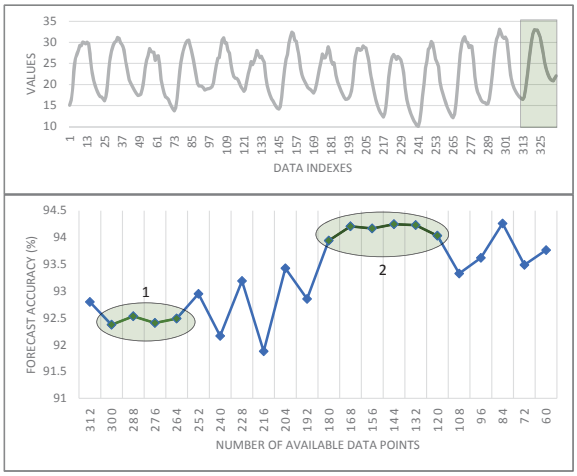

Fig. 3. Observed dataset and forecast accuracy observation with stable clusters detection.

observations is compared with mean value of each cluster. Thus, first cluster is selected at first as an appropriate cluster. Further, checking the conditions for a better accuracy value with smaller dataset, the second cluster replaces the old one and becomes new selected cluster.

Finally, at the end of each cycle, data management action will release data points starting from the oldest data point, i.e. index 1, until middle of appropriate cluster, i.e. index 162, leaving 174 available data points in storage. Using our approach, it is possible to reduce the amount of stored data by 48% in one cycle, while keeping the accuracy of our predictions above specified threshold.

B. Test case for storage management process and discussion

After describing the results of our algorithm, to assess its benefits for Edge storage management, we compare two other approaches with our contribution, the third approach.

a) Naive approach: The storage unit stores all upcoming data until it reaches its limit. Then, the new data replace the oldest data in the next iterations. Forecast period is estimated based on all available data in storage unit.

b) Adaptive algorithm selecting most accurate cluster: only stable clusters with highest forecast accuracy are considered. An unlimited storage capacity is assumed in this case.

c) Adaptive algorithm selecting most appropriate cluster: based on Algorithm 2, a trade-off between stable forecast accuracies and amount of used data is found.

In each approach, we simulate process of Edge storage management under fixed amount of upcoming data. The aim is to evaluate the effects of our algorithm, showing that with our adaptive algorithm is possible to have less data in storage and accurate forecasts for managing time-sensitive edge systems. The used dataset is shown in Figure 4 and contains 600 data points. In all cases of our experiment, we set the upcoming amount of data on 100 data points per turn.

First, we consider the simulation based on our adaptive algorithm. Considering upcoming amount of data, we have six cycles (Figure 5) applying the proposed algorithm. Two graphs per cycle are shown, where upper graph shows dataset that is available in storage at that moment. Vertical dashed line represents calculated marginal index as a separation between data that will be released from storage (left side) and data that remain in the storage (right side). The right side of the vertical line repeats at the beginning of the next cycle. This line corresponds to the middle of indexes of detected stable clusters. In lower graphs, only appropriate clusters are marked as a result of the proposed algorithm.

We consider first cycle in Figure 5. We should have 100 available data points, but precisely we have 96, because of test data used as a defined forecast period from the specification list. The accuracy threshold is set to 99%, and the algorithm has found appropriate cluster in range between 24 and 36 available data points, what corresponds to cluster between data indexes 60 and 72 in our original dataset. Middle index of that cluster indicates that data management action will release data points in range 1-66, i.e. indexes in range 67-100 will be kept in storage. The process repeats for each cycle.

In Table IV all approaches are compared. In the naive and the third approach we assume that storage is limited to 300 data points. Using the naive approach, storage unit stores 100 data points in each cycle, considering the space limitation, resulting in using full storage capacity at the third iteration. Therefore it starts to replace the oldest 100 data points with the newest. In each cycle, forecast accuracy is calculated based on total amount of data available in the storage. The results have shown that in each cycle we have forecast accuracy of 99.89% in average, while using the whole capacity after a while.

In second approach, releasing data based on the selection of most accurate cluster results in forecast accuracy of 99.91% in average, having 170 available data points in average per cycle.

In the third approach, that uses our adaptive algorithm, storage space available is kept around 89% in average, while satisfying the required forecast accuracy. Hereby is shown that using our adaptive algorithm, we have 0.014713% of forecast accuracy higher than in the naive approach, but 0.01041% of forecast accuracy less than in the second approach. However, in the third approach, in average 23.53% less data than in second approach are used, and the amount of data decreases by an average 73.02% in each cycle, without affecting specified threshold for prediction accuracy, and allowing Edge nodes to use excess capacity for other purposes. The reason for high forecast accuracy values in all cases is caused from the small forecast period. In Table V we compare the results of the three approaches for a larger dataset. In this test case we consider that storage is limited to 4000 data points, for the first and the third approach. Further, we consider forecast period of 30 data points. For the naive approach, results show that in each cycle we have forecast accuracy of 99.30% in average, while using the whole capacity after fourth cycle. For the second approach, data are released according to the selection of most accurate cluster. This results in forecast accuracy of 99.52% in average, 1692 available data points in average and the reduction of the amount of data stored by an average 51.35% in each cycle. In the third approach, based on our adaptive algorithm and according to the simulated storage limitation, storage space available is kept around 94.11% in average. Further, it results in forecast accuracy of 99.32% in average, 1182 available data points in average and a reduction of the amount data by an average 80.27% per cycle, showing that in both test cases our algorithm achieves the desired accuracy and substantially reduces the amount of data needed on the Edge.

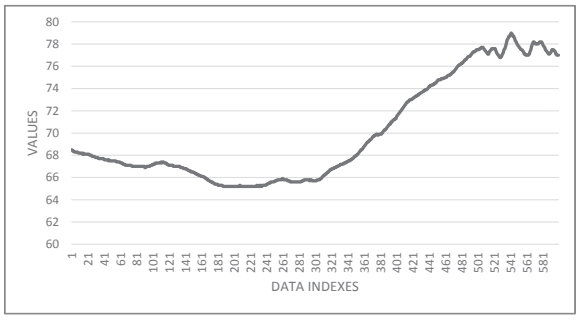

Fig. 4. Observed dataset for simulation of edge storage management.

TABLE IV
SIMULATION 1 - RESULTS FOR 600 POINTS DATASET

	Naive approach		Selecting most accurate cluster		Selecting most appropriate cluster	
Cycle	Available data	Accuracy [%]	Available data	Accuracy [%]	Available data	Accuracy [%]
I	100	99.81	100	99.88	100	99.88
II	200	99.97	134	99.98	134	99.98
III	300	99.85	219	99.87	142	99.88
IV	300	99.92	214	99.95	122	99.80
V	300	99.90	193	99.95	136	99.97
VI	300	99.89	162	99.84	148	99.92
Average		99.89	170	99.91	130	99.90

TABLE V
SIMULATION 2 - RESULTS FOR 7000 POINTS DATASET

	Naive approach		Selecting most accurate cluster		Selecting most appropriate cluster	
Cycle	Available data	Accuracy [%]	Available data	Accuracy [%]	Available data	Accuracy [%]
I	1000	99.32	1000	99.28	1000	98.83
II	2000	99.22	1791	99.23	1145	99.24
III	3000	99.42	2686	99.89	1230	99.35
IV	4000	99.75	1525	99.73	1180	99.89
V	4000	98.67	1805	99.08	1220	98.68
VI	4000	99.41	1470	99.78	1200	99.61
VII	4000	99.29	1570	99.64	1300	99.62
Average		99.30	1692	99.52	1182	99.32

VII. RELATED WORK

The problem of reducing data transmission on the Edge has been discussed by several works. In the work [11], a solution for network-edge data reduction targeting IoT devices is presented. However, this work does not consider storage. Paper [12] proposes a dynamic compression-based technique for sensor datasets. Other existing works, like [30], focus instead on data storage structure, memory allocation strategy and data compression in order to efficiently use storage capacity. All these works try to reduce datasets size by maximizing compression of generated data while reducing information loss, but do not consider the impact on real-time needs for sensor-based monitoring systems and corresponding actuators.

Existing works, like [31], [32] propose data staging for Edge nodes including a control loop where data staging manager takes care of retrieving and caching necessary data from the Cloud. Proposing predefined fetching algorithm, these approaches are suitable for a scenario where data are available

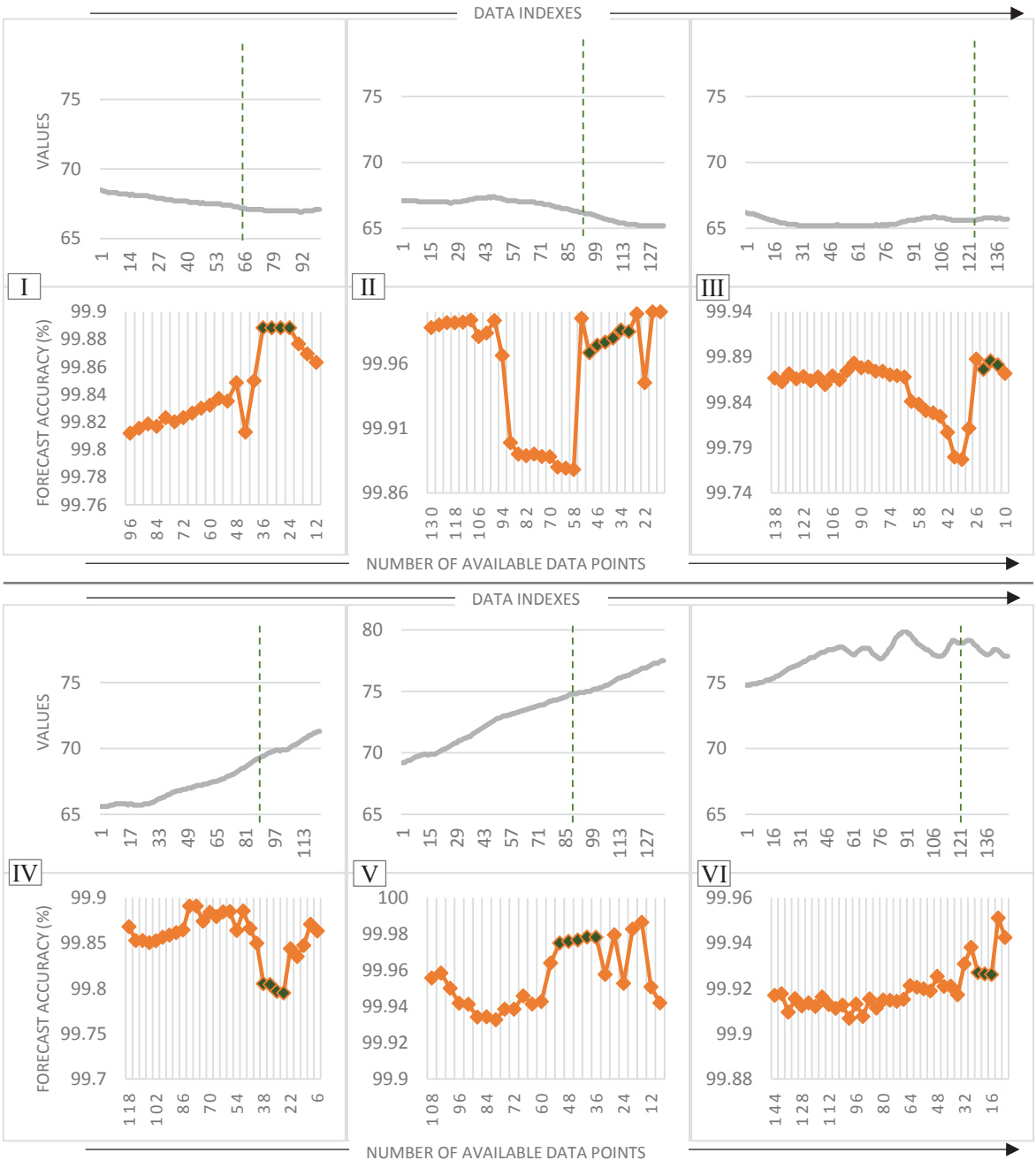

Fig. 5. Evaluation of adaptive algorithm - cycles I-VI with available dataset and corresponding appropriate clusters.

to the mobile devices when it is needed, but it is not suitable for our case, where we need to obtain fast and accurate responses. Plus, their contributions do not consider neither the performance of storage limited Edge nodes, nor the possibility of targeting actuators with reliable predictions.

In comparison with traditional batch data analysis, data stream classification has many open issues. In [33], holdout accuracy is estimated using prequential accuracy. In other works, two types of methods are used: a sliding window with the most recent observations or fading factors that weigh observations using a certain decay factor [34], [33]. However, these approaches do not consider prediction of upcoming data stream together with historical data. Our approach proposes an adaptive algorithm that constantly checks new received data and then together with remaining old data performs proposed principles for edge storage management. A comprehensive analysis of the advantages of having the intermediate layer data centers close to the network edge has been done by Mehta et al. [35].

VIII. Conclusion

Moving the analytics close to the source of data helps to provide near real-time decisions for time-sensitive edge systems. Providing an architecture model for data storage management, we have made a step toward the developing of algorithms able to cope with constant streams of data. The approach is able to continuously monitor forecast accuracy and to capture stable forecast accuracy clusters in order to support qualitative decisions and in the same time manage the space-limited storage at runtime using an adaptive algorithm.

Our simulation results show that our adaptive algorithm can reduce the amount of data by an average 73.02% and 80.27% in each cycle for the two datasets respectively, while satisfying demands for forecast accuracy and thereby showing potential for saving limited storage space.

In future work, we would like to extend our solution in different ways. It would be interesting to introduce methods to deal with multiple sources of data and corresponding issues, such as missing data and outliers. Furthermore, we would like to evaluate our application with different datasets, to show a wide applicability of the proposed approach. Finally, it would be interesting to take into account implications of involved computation resources during the runtime and decrease of complexity by improving algorithms.

Acknowledgment

The work described in this paper has been funded through the Haley project (Holistic Energy Efficient Hybrid Clouds) as part of the TU Vienna Distinguished Young Scientist Award 2011 and Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015.

References

[1] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 112–121, 2014.

[2] C. Doukas and I. Maglogiannis, "Bringing iot and cloud computing towards pervasive healthcare," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, pp. 922–926.

[3] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*. Springer International Publishing, 2014, pp. 169–186.

[4] T. Mastelic and I. Brandic, "Data velocity scaling via dynamic monitoring frequency on ultrascale infrastructures," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 422–425.

[5] S. Krishnan and T. Balasubramanian, "Traffic flow optimization and vehicle safety in smart cities," vol. 5, pp. 7814–7820, May 2016.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. ACM, 2012, pp. 13–16.

[7] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[9] M. Taneja and A. Davy, "Poster abstract: Resource aware placement of data stream analytics operators on fog infrastructure for internet of things applications," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, 2016, pp. 113–114.

[10] C. Aggarwal, N. Ashish, and A. Sheth, "The Internet of Things: A Survey from the Data-Centric Perspective," in *Managing and Mining Sensor Data*, C. C. Aggarwal, Ed. Springer US, 2013, pp. 383–428.

[11] A. Papageorgiou, B. Cheng, and E. Kovacs, "Real-time data reduction at the network edge of internet-of-things systems," in *2015 11th International Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 284–291.

[12] A. Ukil, S. Bandyopadhyay, and A. Pal, "Iot data compression: Sensor-agnostic approach," in *2015 Data Compression Conference*, April 2015, pp. 303–312.

[13] "Ericsson mobility report, june 2016," http://www.ericsson.com/res/docs/2016/ericsson-mobility-report-2016.pdf, 2016, [Online; accessed 11-January-2017].

[14] "R forecast package," "https://cran.r-project.org/web/packages/forecast/forecast.pdf", 2016.

[15] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: the forecast package for R," *Journal of Statistical Software*, vol. 26, no. 3, pp. 1–22, 2008. [Online]. Available: http://www.jstatsoft.org/article/view/v027i03

[16] R. J. Hyndman, R. A. Ahmed, G. Athanasopoulos, and H. L. Shang, "Optimal combination forecasts for hierarchical time series," *Computational Statistics & Data Analysis*, vol. 55, no. 9, pp. 2579–2589, 2011.

[17] V. Gularnik and J. Srivastava, "Event detection from time series data," in *In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 33–42.

[18] G. Zhang, "Time series forecasting using a hybrid {ARIMA} and neural network model," *Neurocomputing*, vol. 50, pp. 159 – 175, 2003.

[19] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[20] P. McSharry, "Stream analytics for forecasting," *Foresight: The International Journal of Applied Forecasting*, no. 24, pp. 7–12, 2012.

[21] R. J. Hyndman, E. Wang, and N. Laptev, in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*.

[22] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, pp. 679–688, 2006.

[23] J. L. Wong, R. Jafari, and M. Potkonjak, "Gateway placement for latency and energy efficient data aggregation [wireless sensor networks]," in *29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 490–497.

[24] J. Lin, S. Williamson, K. Borne, and D. De Barr, *Pattern Recognition in Time Series, Advances in Machine Learning and Data Mining for Astronomy*. Chapman and Hall, 2012.

[25] X. Wang, K. Smith-Miles, and R. Hyndman, "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series," *Neurocomput.*, vol. 72, no. 10-12, pp. 2581–2594.

[26] T. W. Liao, "Clustering of time series dataa survey," *Pattern recognition*, pp. 1857–1874, 2005.

[27] X. Wang, K. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data mining and knowledge Discovery*, vol. 13, no. 3, pp. 335–364, 2006.

[28] "Umass trace repository," http://traces.cs.umass.edu/index.php/Main/Traces/, 2017, [Online; accessed 11-January-2017].

[29] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, p. 112, 2012.

[30] Q. Yan and Y. Y. Wang, "A kind of efficient data archiving method for historical sensor data," in *ICISCE '16*, July 2016, pp. 44–48.

[31] G. A. Lewis and P. Lago, "A catalog of architectural tactics for cyber-foraging," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, ser. QoSA '15. ACM, 2015, pp. 53–62.

[32] J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanarayanan, "Data staging on untrusted surrogates." in *FAST*, vol. 3, 2003, pp. 15–28.

[33] J. a. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. ACM, 2009, pp. 329–338.

[34] J. Read, A. Bifet, G. Holmes, and B. Pfahringer, "Streaming multi-label classification." in *WAPA*, 2011, pp. 19–25.

[35] A. Mehta, W. Trneberg, C. Klein, J. Tordsson, M. Kihl, and E. Elmroth, "How beneficial are intermediate layer data centers in mobile edge networks?" in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Sept 2016, pp. 222–229.