2018

Data Storage in Internet of Things: A Proposed Distributed Model

Kostas Kolomvatsos
Department of Informatics and telecommunications National and Kapodistrian University of Athens, kostasks@di.uoa.gr

Panagiota Papadopoulou
Department of Informatics and telecommunications National and Kapodistrian University of Athens, peggy@di.uoa.gr

Stathes Hadjiefthymiades
Department of Informatics and telecommunications National and Kapodistrian University of Athens, shadj@di.uoa.gr

DATA STORAGE IN INTERNET OF THINGS: A PROPOSED DISTRIBUTED MODEL

Kostas Kolomvatsos, Panagiota Papadopoulou, Stathes Hadjiefthymiades

Department of Informatics and telecommunications

National and Kapodistrian University of Athens

Panepistimiopolis, Ilisia, 15784

{kostasks, peggy, shadj}@di.uoa.gr

Track No 8 Internet of Things for Smart Living and Smart Business

Abstract

In Internet of Things (IoT), numerous devices are able to collect and report data while they can execute simple processing tasks to produce knowledge. IoT Nodes exhibit limited computational resources, thus, they can only perform a limited number of tasks and store a short version of the collected data. In this paper, we propose a scheme that focuses on a distributed model for data storage in a group of IoT nodes. Nodes cooperate each other exchanging statistical information for their data. Our work aims to provide a model for the selection of the node where the incoming data should be stored irrelevantly of the node in which they are initially reported. The selection process involves a decision making process that adopts a statistical similarity model of the incoming data with the datasets present in the group, the estimation of the load of each node and the in-network communication cost. All these parameters are fed into a multi-class classification scheme for the final decision. Our aim is to have a view on the statistics of the available datasets beforehand, thus, facilitating the post-processing and the production of knowledge. We report on the evaluation of our scheme and present experimental results towards the presentation of pros and cons of our model.

Keywords: IoT, Data storage, Data management, Recommendation system.

1 Introduction

Several efforts in both, academia and industry, recognize that the massive volume of data generated and collected by nodes present in Internet of Things (IoT) can be as problematic as valuable for current and future IoT-based applications. A number of studies have proposed to remove the computation load from mobile devices to the Cloud or introduce cloudlets as a middle tier between devices and the Cloud. However, IoT data pose storage and computational requirements which are hard to be covered by nodes or Cloud resources. On the one hand, IoT data requirements surpass the limited storage and computational capabilities of nodes; the transfer of data to the Cloud creates latency (Satyanarayanan, 2015) that can often be not acceptable, especially for time critical applications, even with the use of cloudlets. Edge computing (Roman et al., 2018) has been a popular topic in IoT research literature as an alternative solution against Cloud. Edge computing eliminates the delays caused by device-to-cloud round trip. Storing and processing data at the Edge reduces latency and increases responsiveness.

The storage of data involves the decision regarding (a) if the collected data should be stored or they should be discarded; (b) if the data should be stored and processed at the Edge, at the collection tier, or they should be transferred to the Cloud/Fog; (c) if the data not sent to Cloud/Fog are stored locally, in the node where they were collected or they should be transferred to other peer nodes. Ideally, data should be stored and processed on the device where they are reported to minimize the latency in their processing. However, apart from the apparent problems of storage and computation inadequacy of a node, such an approach cannot ensure efficiency in responses to data-related queries. Since a node does not operate in isolation but it is part of a group of nodes associated spatially and logically, this group could be leveraged to improve data storage and processing. Previous studies have focused on storing data to the node that collects them and then transferring them from the node they are stored to the node that needs them for a particular task (D' Andria et al., 2015). Data storage followed by on demand data migration (Tai et al., 2017) entails an important overhead. Data, instead of being stored to the node of collection, could be allocated to the most appropriate node of the group, according to the similarity of the data, the node load and the network latency. This approach allows for a pre-processing of data, so that they can be grouped based on their similarity to the respective node. This allows for efficiency in using data for responding to task requests. In practical terms, having a view on the available data, we can easily support efficient responses to queries asking for analytics. For instance, we can avoid setting a query in the entire network but we can 'guide' the queries to the appropriate nodes. This decision becomes more imperative in the IoT domain where numerous distributed datasets are located in various places. In this aspect, we need an 'indexing' model over the available data to speed up the allocation of queries and the provision of responses. In any case, the allocation of data to IoT nodes remains a topic that is largely unaddressed.

The aim of this paper is to attempt to address this gap, focusing on the storage of data to IoT nodes. We propose a distributed model for solving the problem. The innovation is that our model adopts multiple technologies to deliver the appropriate place where data should be stored. The decision is basically made under the rationale that 'similar' data should be kept at close locations. The model is based on a Decision Making System (DMS) for the selection of the node where the data should be stored. Data storage takes place within a group of nodes and involves the allocation of data among them, irrelatively to the collecting node. The recommended selection of the node to store the data is made taking into account the statistical similarity of the incoming data with the datasets present in the group, the load of each node and the in-network communication cost. The aforementioned parameters are fed into a classification module responsible to deliver the final decision. However, the proposed DMS is not just a simple classification scheme but a complete multidimensional data processing model.

The structure of the paper is as follows. Section 2 reviews the related work. The third section presents the high-level description of the DMS for data allocation in IoT nodes. Section 4 presents the decision-making process. Section 5 presents our experimental evaluation and results. The paper ends with the conclusions, including our future research plans.

2 Related Work & Contribution

IoT and the management of data emanating from it have received research attention in several works, in conjunction with other topics such as Cloud, Edge and Fog computing. A state-of-the-art review is offered in Escamilla-Ambrosio et al., (2018) with a comparison of the aforementioned technologies against Cloud, complemented with a practical approach of the topic through the presentation of use cases. IoT-based big data storage in Cloud computing is examined by Cai et al. (2017), who provides a review of acquisition, management, processing and mining of IoT big data. Challenges and opportunities in current research for IoT applications associated with big data are also presented. Dolui and Datta (2017) present a comparison of Edge computing implementations, discussing Fog computing, cloudlets and mobile Edge computing. After a comparative analysis of three implementations along the dimensions of node devices, location, architecture, context awareness, proximity, access mechanisms and internode communication, the authors present the parameters, namely, physical proximity, access mediums, context awareness, power consumption and computation time, which can be leveraged for an DMS to decide on the selection of the appropriate implementation for a particular use case.

In addition to overviews and surveys that greatly facilitate our understanding of the current issues in IoT data, previous research with a more specific focus has proposed several frameworks for data storage and management. Ruiz-Alvarez and Humphrey (2012) proposed a model for data storage in Cloud. The decision for data allocation is based on a mathematical model that takes into account resource characteristics such as storage and compute cost and performance factors such as latency, bandwidth and turnaround time. The authors present the algorithm and show that its implementation provides solutions for the optimal assignment of datasets to storage services and of application runs to compute services. Jiang et al. (2015) proposed a data storage framework for enabling efficient storing of massive IoT data, integrating both structured and unstructured data. The framework can combine and extend multiple databases and Hadoop to store and manage diverse types of data. Habak et al. (2015) propose a system for leveraging mobile co-located devices to provide Cloud services at the Edge. Their system provides a multi-device mobile Cloud. Mobile devices can be part of a cluster managed by a controller. The computational capability of each device is estimated to determine that which can be available for sharing as a Cloud service. The controller is responsible for the addition of devices to the cluster and for task allocation to the devices. Shafagh et al. (2017) present a blockchain-based design for storage and sharing of IoT data that allows for secure and resilient distributed access control and data management. They propose the storage of time-series IoT data at the Edge of the network through a locality aware decentralized storage system managed with blockchain technology.

Fu et al. (2018) propose a framework for secure data storage and retrieval in the context of industrial IoT. According to their design, the data collected from physical devices are sent to the Edge server where they are preprocessed. Time-sensitive data are extracted and stored locally, while non-time-sensitive data are sent to the Cloud server for storage and retrieval. The framework integrates Fog and Cloud computing, focusing on security. Xing et al. (2018) propose a distributed model for IoT data storage with a multi-level storage system for Edge computing. The model is based on a multiple factor replacement algorithm which solves the problems of limited data storage space and data loss caused by network instability. Storage levels are composed of devices on the Edge. According to the algorithm, when the storage space of a node is exhausted, part of the data of the node is selected and transferred to the upper level. The selection of data to be uploaded considers use frequency of data and the importance of data.

3 Preliminaries

We consider a set $\mathcal{N}$ of IoT nodes i.e., $\mathcal{N} = \{n_1, n_2, ..., n_{|\mathcal{N}|}\}$ having specific characteristics concerning their computational capabilities and resources. They are capable of observing their environment and performing simple processing tasks on top of the collected data. As their resources are limited, nodes should store only the necessary data for processing. These data are updated while the remaining are sent to the Fog/Cloud. Nodes may execute tasks locally on top of the stored data. The selected data should fit into the available resources and assist in the provision of efficient responses to users/applications. For instance, nodes may decide to keep only a portion of the collected data or exclude data considered as outliers. When data are not selected to be locally stored, they are transferred either to the peer nodes or to the Fog/Cloud. It should be noted that, when nodes rely on the Fog/Cloud for the processing of data, they enjoy increased latency (Satyanarayanan, 2015). In this aspect, we envision an ensemble storage mechanism where the nodes are coordinated to store data that they consider important for further processing.

We propose a distributed data storage scheme applied in IoT nodes. Our model acts proactively and decides the nodes at which the incoming data should be stored. Our aim is to support the real time pre-processing of data streams. An efficient allocation is realized when similar data are gathered to the same node. Having a view on the statistics of data facilitates the provision of efficient query execution plans[1]. Actually, there are two ways to manage data. The first involves the concentration of data in a centralized point and, afterwards, their processing. The second approach deals with the pre-processing of data preparing them for the upcoming step. The former approach suffers from the increased time required to process huge volumes of data. For instance, for delivering analytics, we should eliminate outliers, missing values or any other 'harmful' data that may jeopardize the quality of the final result. However, applying machine learning or computational intelligence on top of huge volumes of data may require increased time and computational resources. In this paper, we focus on the latter approach providing a mechanism that proactively allocates the data to the appropriate node instead of separating them in a centralized system.

We consider that data are received and stored in the form of multivariate vectors i.e., $\vec{x} = [x_1, x_2, ..., x_M]$ where M is the number of dimensions. Let D_{n_i} be the dataset stored in n_i with $D_{n_i} = \vec{x}_i = \{x_{ji}\}, j = 1,2, ..., M$. D_{n_i} should be 'solid' meaning that the distance between the available data should be minimized. Nodes should identify if the incoming data 'match' the local data or any other dataset in the network. In addition, nodes should check if the load of peers and the communication cost makes the migration of the incoming vector disadvantageous. If no dataset is 'similar' to the incoming vector, or the migration is judged as disadvantageous, $\vec{x}$ is transferred to the Fog/Cloud for further processing. At pre-defined intervals, nodes exchange information about the statistics of their data and their load while they calculate the current communication cost. In addition, every node applies a sliding window approach i.e., only W (recent) vectors are stored. When the storage capacity is exhausted and a new vector is decided to be locally stored, the oldest one is evicted and transferred to the Fog/Cloud. This way, nodes rely on 'fresh' data. In addition, the discussed approach facilitates the 'natural' evolution of data obviously affecting their statistics, thus, future decisions as well.

We propose the use of a DMS that is responsible to decide the node (or the Fog/Cloud) where $\vec{x}$ will be allocated. The decision is made on top of the distribution of the local data and the data present in peer nodes as well as their load and communication cost. The proposed DMS considers load balancing

1https://www.brentozar.com/archive/2013/08/query-plans-what-happens-when-row-estimates-get-high/

aspects to secure that no node will be overloaded. When a node is overloaded means that data are quickly refreshed jeopardizing the 'solidity' of the dataset. Our aim is to evenly distribute the appropriate data (if they are similar to the underlying dataset) into the network forming a 'cooperative storage', however, taking into consideration the 'solidity' of the datasets. In an upcoming step, every processing task submitted to the network will be allocated, for execution, to the appropriate node i.e., the node that 'owns' the dataset that matches to tasks' requirements.

Consider that a vector $\vec{y}_k = [y_{1k}, y_{2k}, \ldots, y_{Mk}]$ (k is the discrete time, k=1,2,…) arrives in a node n_i. The corresponding DMS i.e., RS_i, initially, should calculate the similarity of the incoming vector with the available datasets. This will be realized with the adoption of the available statistics and the use of an ensemble similarity model (see next Section). The proposed scheme incorporates into the RS_i the statistical similarity of $\vec{y}_k$ with every dataset. The ideal case is when $\vec{y}_k$ is stored locally where it is reported as no communication (migration) cost will accrue. The second step is to estimate the load of its peers. This will be realized by the use of an ensemble estimation scheme. We rely on a set of estimators and, accordingly, we aggregate their results to produce the final estimation. Based on the similarity with each dataset, the estimated load and the communication cost, n_i produces a set of Nodes' Status Vectors (NSVs) $\overrightarrow{NSV}_j = [s_j, \lambda_j, \kappa_{ij}], j = \{1,2, \ldots, |\mathcal{N}|\}$. It stands true that $\kappa_{ii} = 0$ (in the case of storing locally the incoming vector). On top of the NSVs, n_i adopts a multiclass classification scheme to deliver the node where $\vec{y}_k$ will be allocated. The multiclass classification scheme follows the one-over-all methodology (Han et al., 2012) based on a set of binary classifiers. These classifiers have the form of decisions trees built with the adoption of the C4.5 algorithm.

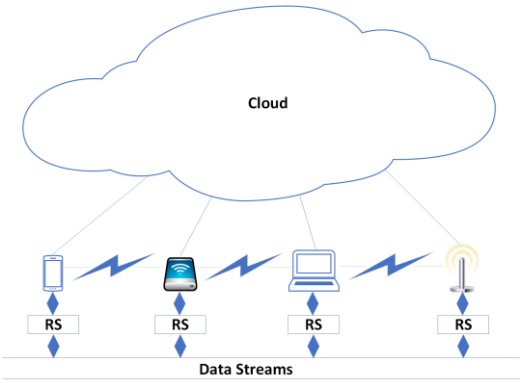

Figure 1. The envisioned architecture.

4 The Decision Making Process

4.1 Retrieving the Similarity with Datasets

We propose a scheme for calculating the distance between every $\vec{\mu}_j$ and $\vec{y}_k$ ($\vec{\mu}_j$ is the jth vector of means for each variable). We rely on two widely adopted techniques, i.e., the Euclidean distance (Han et al., 2012) and the cosine similarity (Manning et al., 2009). Among all available similarity techniques, we rely on two simple, however, efficient methods to be capable of producing the final outcome immediately. At predefined intervals, nodes exchange information about the statistics of their datasets and their load. In addition, they calculate the communication cost to conclude the NSVs. Let

us consider the behaviour of node n_i. n_i receives vectors with the mean values for each variable, i.e., $\vec{\mu}_j = [\mu_{1j}, \mu_{2j}, \dots, \mu_{Mj}], j = \{1,2, \dots, |\mathcal{N}|\}$. Our aim is to allocate the incoming vector to the most similar dataset keeping the deviation from the mean low. For calculating the similarity between $\vec{y}_k$ and $\vec{\mu}_j$, we adopt a 'combination' of the Euclidean distance and the cosine similarity model.

The Euclidean distance for vectors $\vec{y}_k$ and $\vec{\mu}_j$ is calculated as follows: $d_j^E = \sqrt{\sum_{l=1}^{M}(y_{lk} - \mu_{lj})^2}$. The Euclidean distance looks at the 'direct' distance between the two vectors and is adopted when the magnitude of vectors matters. It performs better with numerical results. To 'transform' the Euclidean distance to a similarity measure, we rely on the inverse of distance, i.e., $s_j^E = \frac{1}{1+d_j^E}$. The cosine similarity is also applied. Hence, our model delivers $|\mathcal{N}|$ similarity values. These values are measures that represent the cosine of the angle between the examined vectors. Based on the vector representation of the mean values and the incoming vector $\vec{y}_k$, n_i calculates similarity scores as follows: $s_j^C = \frac{\vec{\mu}_j \cdot \vec{y}_k}{|\vec{\mu}_j||\vec{y}_k|}$. The numerator represents the dot product of $\vec{y}_k$ and $\vec{\mu}_j$ defined as the simple multiplication of the values contained in each vector, i.e., $\vec{\mu}_j \cdot \vec{y}_k = \sum_{l=1}^{M} y_{lk}\mu_{lj}$. The denominator represents the product of the Euclidean lengths of the two vectors, i.e., $|\vec{\mu}_j| = \sum_{l=1}^{M}\mu_l^2$, $|\vec{y}_k| = \sum_{l=1}^{M} y_{lk}^2$. The cosine similarity is used as a metric for measuring the distance when the magnitude of vectors does not matter. It is more 'generic' technique meaning that it can be efficiently used when we consider e.g., textual data.

The proposed scheme tries to combine the aforementioned techniques and derive the final result taking into consideration many aspects of the examined vectors. We combine two techniques with different characteristics to provide a more efficient result. The Euclidean distance focus on the magnitude of the vectors while the cosine similarity aims to incorporate their orientation into the final outcome. We adopt a linear combination scheme $s_j = \alpha s_E^j + (1-\alpha)s_C^j$, where $\alpha \in [0,1]$ is a fuzzy number that affects the final result. When $\alpha \to 1$, the final outcome is delivered mostly based on the Euclidean distance. When $\alpha \to 0$, the final s_j is delivered mostly based on the cosine similarity. α is affected by the variance S^2 of the mean vectors $\vec{\mu}$. When S^2 is high, the mean vectors are spread from the mean, i.e., there are high fluctuations which lead to the conclusion that datasets present in the nodes are completely different each other. In such cases, we do not want to pay attention in the Euclidean distance and take into consideration the magnitude of the vectors. On the other hand, when the variance is low means that the mean vectors are close to each other, thus, the datasets present in nodes are similar. In this case, we want to focus more on the Euclidean distance and the magnitude of the vectors to carefully select the most appropriate dataset. α is calculated through the use of a sigmoid fuzzy membership function. We consider an upper limit for the variance above which the realization of α is close to zero. Hence, for delivering α the following equation stands true: $\alpha = \frac{1}{1+e^{a(S^2-c)}}$, where a and c are parameters of the sigmoid function.

4.2 Estimating the Load of Nodes

Our scheme tries to evenly allocate the incoming vectors to the entire set of nodes. This decision is based on the load that each node exhibits. The load is depicted by the number of vectors stored in each node. We consider that the load is represented by a value in the interval [0,1]. We also consider that there is a maximum number of vectors U (U differs from W, $U >> W$) that each node could store. Let us consider the discrete time T. At each t a number of vectors are allocated at n_i. When decided that U vectors will be allocated to n_i at t, the load approaches to unity. When decided that no vectors will be allocated to n_i, then the load approaches to zero. The load realizations are recorded and adopted for delivering the future estimation for each node. This future estimation is critical for the conclusion of any allocation in the network. This is because the proposed system incorporates a load balancing aspect in the allocation of the incoming vectors trying to avoid any overloading. Overloading may negatively affect the statistics of datasets as multiple vectors present in a dataset are substituted abruptly.

For deriving the estimation of the future load, we rely on an ensemble scheme. Ensemble estimation is a common way to improve the performance of an estimation compared to single models. An ensemble of individual estimators performs better, in average, than a single estimator (Shafagh et al., 2017). The proposed scheme tries to deal with errors arising from the uncertainty associated with the dynamics of the environment. The uncertainty is related to the number of vectors allocated in every node. Different estimators exhibit different characteristics and performance related to the estimation error. In our work, we adopt the solution presented in Kolomvatsos et al., (2015). We consider $|\mathcal{P}|$ estimators where $\mathcal{P} = \langle p_1, p_2, \dots, p_{|\mathcal{P}|} \rangle$ that adopt different methodologies for deriving the final result. These estimators are: *linear*, *polynomial*, *cycle*, *exponential*, *moving average* and *seasonal naive*. Every p_i takes into consideration the historical load data exchanged though the above discussed messages and results a value in the interval [0,1] that represents the future estimated load. The proposed ensemble estimation scheme consists of the integration of multiple estimators. The ensemble forecasting model is based on a fusion function, i.e., $\hat{\lambda} = f(\boldsymbol{\lambda})$ where $\hat{\lambda}$ is the final estimated value and $f(.)$ is the fusion function applied on the vector of estimators results $\boldsymbol{\lambda}$. In our work, we consider that (for delivering a fast response) $f(.)$ is the mean of the estimations. $\hat{\lambda}$ is a mapping that derives the final estimated value i.e., $\hat{\lambda}: \boldsymbol{\lambda} \rightarrow [0,1]$ (without loss of generality, we consider all values in the interval [0,1]).

4.3 Classifying the Incoming Data

We focus on a binary classifier i.e., a classifier that decides if $\vec{y}_k$ should be allocated in a node n_j. Recall that we consider that $\vec{y}_k$ arrives in n_i. The classification part of the proposed DMS should decide the efficient location where $\vec{y}_k$ will be allocated. Our classifier is based on a decision tree built with the C4.5 algorithm. The input of the algorithms is a training dataset containing vectors as follows: $D_T^l = [s_l, \hat{\lambda}_l, \kappa_l], l = 1,2, \dots$. Each vector represents a combination of the similarity between $\vec{y}_k$ and a potential host (i.e., a peer node), the estimated load and the communication cost. For each variable i.e., s, λ, κ, the algorithm calculates the gain ratio to decide the formation of the decision tree. The algorithm tries to calculate the 'split information' of each variable as follows:

$SI_{s,\lambda,\kappa}(D_T) = -\sum_{l=1}^{|D_T|} \frac{D_T^l}{|D_T|} \cdot log_2 \frac{D_T^l}{|D_T|}$. The discussed value represents the potential information generated when we split the dataset into D_T partitions for a specific variable. The number of training vectors where a variable has a specific value is normalized against the total training vectors. The final gain ratio is calculated as follows: $GR_{s,\lambda,\kappa} = \frac{G_{s,\lambda,\kappa}}{SI_{s,\lambda,\kappa}(D_T)}$, where $G_{s,\lambda,\kappa}$ is the information gain for a variable defined by: $G_{s,\lambda,\kappa} = I(D_T) - I_{s,\lambda,\kappa}(D_T)$. In the aforementioned equation, we have: $I(D_T) = \sum_{m=1}^{2} p_m \, log_2(p_m)$ and $I_{s,\lambda,\kappa}(D_T) = \sum_{m=1}^{|D_T|} \frac{|D_T^m|}{|D_T|} Info(D_T^m)$. In our model, for the binary classifier, we consider two classes i.e., $C_1 = \{1: \text{store at } n_i\}$ and $C_2 = \{0: \text{no store at } n_i\}$.

For the delivery of the node that will store $\vec{y}_k$, we apply the one-over-all (OVA) methodology (Han et al., 2012). It consists of a widely adopted technique for multiclass classification. In our scenario, $|\mathcal{N}|$ classes are available, one for each node in the network. We adopt $|\mathcal{N}|$ binary classifiers. Actually, we apply $|\mathcal{N}|$ times the same binary classifier for deciding in which nodes the incoming vector could be allocated. The training set contains vectors for similarity, load and cost and the corresponding class C_1 or C_2. The classification of the unknown vector $\vec{y}_k$ concerns a voting scheme. If the classifier positively predicts the jth node for $\vec{y}_k$ (based on the similarity, load and cost values) then the node gets a vote. If the result is negative all the remaining nodes except j get a vote. The node with the most votes is selected to host $\vec{y}_k$.

5 Experimental Evaluation

We report on the performance of the proposed scheme aiming to reveal if it is capable of correctly identifying the appropriate nodes of the incoming vectors. We provide a synthetic training dataset where we record various combinations of the discussed parameters (i.e., similarity, load and cost) and the corresponding class (i.e., C_1 or C_2). The class is delivered by C_1=1 when the adopted parameters (similarity, load, cost) do not violate a set of pre-defined thresholds, otherwise we assign the C_2=0 class. With this setting, we adopt a 'strict' scenario (a conjunctive clause) where all the parameters together should not violate the thresholds. Each training vector consists of the realization of the aforementioned parameters. For the evaluation of the proposed model, we rely on two synthetic datasets adopting two different distributions, i.e., the Gaussian and the Uniform. Through the adoption of the Gaussian, we try to simulate scenarios where the collected vectors are around the pre-defined mean. Actually, the adoption of the Gaussian aims to identify how 'stable' data affect the performance of our scheme. Through the adoption of the Uniform, we simulate a scenario where data change continually and there is no any guarantee that consecutive vectors will exhibit a low statistical difference.

We aim to evaluate the DMS's performance, thus, we rely on widely adopted metrics like precision π, recall ρ and F-measure ϕ. These metrics are defined as follows: $\pi = \frac{TP}{TP+FP}, \rho = \frac{TP}{TP+FN}, \phi = 2\frac{\pi\rho}{\pi+\rho}$ where T refers to 'true', P refers to 'positive', F refers to 'false', and N refers to 'negative'. Hence, TP refers to true positive events, i.e., identified events that had to be identified, FP refers to false positive

events, i.e., identified events that had to not been identified, and so on. In addition, we adopt the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE). MAE is defined as follows: $MAE = \frac{1}{F}\sum_{f=1}^{F}|\epsilon_f - \hat{\epsilon}_f|$ where F is the number of the vectors in the dataset, ϵ_f is the actual class and $\hat{\epsilon}_f$. MAE measures the average magnitude of the classification errors. RMSE is defined as follows: $RMSE = \sqrt{\frac{1}{F}\sum_{f=1}^{F}(\epsilon_f - \hat{\epsilon}_f)^2}$. RMSE is similar to MAE, however, RMSE assigns large weight on large errors. RMSE is more useful when high errors are undesirable. We perform the evaluation of our scheme for $|\mathcal{N}| = \{10,100,500,1000\}$ and $M \in \{5,20,50\}$. We study how $|\mathcal{N}|$ and M affect the proposed model. Based on these experimental scenarios, we reveal the scalability of the system and see if the proposed scheme retains the classification performance at high levels.

We report on performance results for $M \in \{5,20,50\}$. π, ρ and ϕ are equal to 1.000 except the scenario where $|\mathcal{N}| = 10$. In this case, we get the values presented in Table 1. We observe that the proposed technique exhibits high performance with π, ρ and ϕ equal to 1.0. The 'worst' performance is observed when $|\mathcal{N}| = 10$. The increased number of nodes leads to an increased performance with as our model eliminates false negatives and false positives events. We compare our model with a Naive Bayesian Classifier (NBC) applied for the same datasets. The performance of the NBC is presented in Table 2. We observe that NBC's performance is worse than the proposed scheme. The NBC eliminates the false positives events, however, it suffers from increased false negatives. This leads to low ρ and ϕ, respectively. Concerning the NBC, the average π, ρ and ϕ are 0.978, 0.581 and 0.726, respectively. The proposed model results 0.999, 0.999 and 0.999 for π, ρ and ϕ, respectively. In Figure 2, we plot the false positive rate vs the true positive rate (Receiver operating characteristics - ROC) curve for comparing the proposed model with the NBC for the same experimentation parameters (M = 5 and $|\mathcal{N}| = \{10,100,500,1000\}$).

	Gaussian			**Uniform**		
M	π	ρ	ϕ	π	ρ	ϕ
5	0.997	0.994	0.995	0.997	1.000	0.999
20	0.997	1.000	0.999	0.997	0.992	0.994
50	0.996	1.000	0.998	0.998	0.987	0.993

Table 1. Classification Performance for |N = 10|.

	Gaussian			**Uniform**		
|N|	π	ρ	ϕ	π	ρ	ϕ
10	0.964	0.697	0.809	0.995	0.529	0.691
100	0.979	0.494	0.656	0.999	0.503	0.669
500	0.984	0.578	0.728	0.999	0.401	0.572
1,000	0.985	0.553	0.709	0.998	0.294	0.454

Table 2. Classification Accuracy of a Naive Bayesian Classifier for M=5.

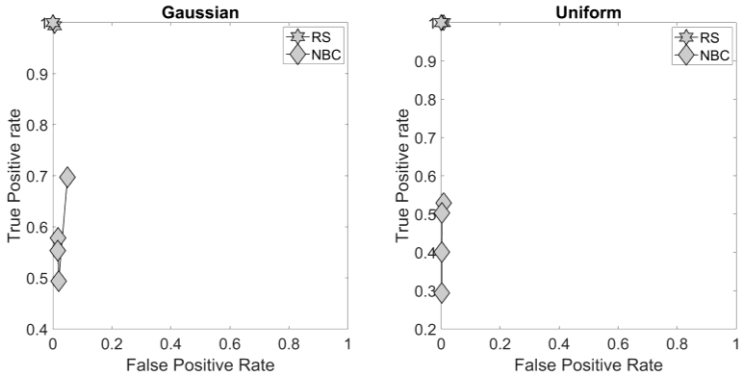

Figure 2. The ROC curves comparison between the proposed model and NBC.

In Figures 3, 4 and 5, we plot the MAE and RMSE for the Gaussian and the Uniform distributions and for $M \in \{5,20,50\}$. When $M = 5$, the increased number of nodes (i.e., $|N|$) positively affects the performance as MAE and RMSE decrease. In average, the adoption of the Gaussian distribution results in higher error compared to the Uniform case (0.0248 for the RMSE-Gaussian combination and 0.0179 for the RMSE-Uniform combination). A high number of nodes, e.g., $|N| = 1{,}000$ results MAE equal to 0. An increased number of variables/dimensions leads to low MAE and RMSE. In average, we take MAE $\in$ {0.0006, 0.0009} for the Gaussian and MAE $\in$ {0.0028, 0.0043}. In addition, we get RMSE $\in$ {0.0189, 0.0206} for the Gaussian and RMSE $\in$ {0.2150, 0.0315}. In such cases, the proposed model has the opportunity to combine multiple values when calculating the final similarity between vectors as already described.

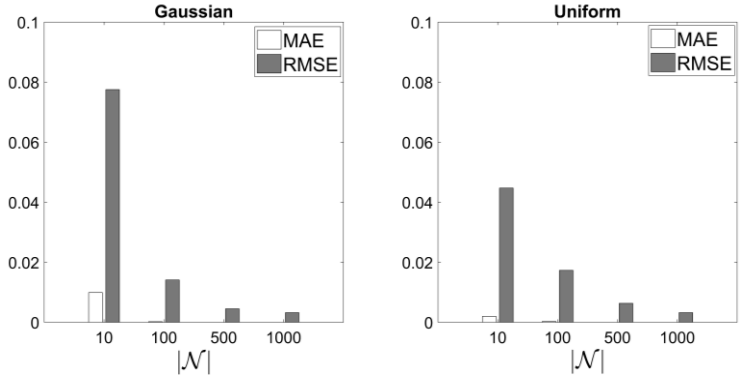

Figure 3. MAE and RMSE for M = 5.

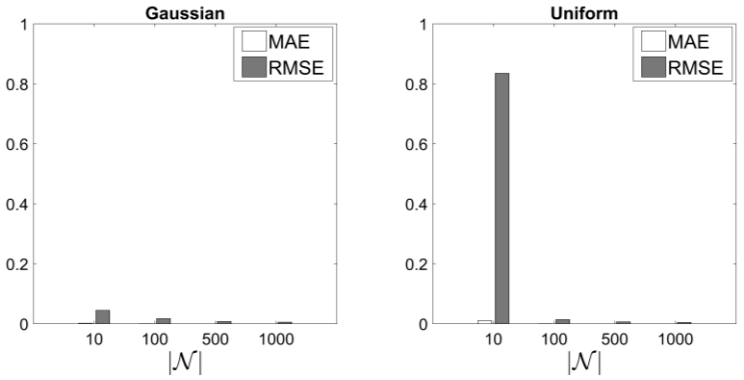

Figure 4. MAE and RMSE for M = 20.

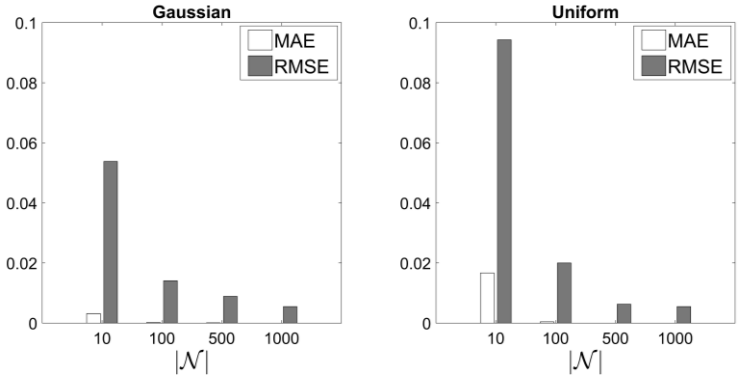

Figure 5. MAE and RMSE for M = 50.

Finally, concerning the required time for training the adopted classifier, we get a mean time equal to 0.0027, 0.0119, 0.0475 and 0.0981 seconds for $|N| \in \{10, 100, 500, 1000\}$, respectively. We observe that the proposed mechanism requires training time below 0.1 second which is capable of supporting applications demanding for real time responses.

6 Conclusions and Future Work

This paper proposes a distributed model for data storage in IoT. Our focus is on an IoT setting where multiple nodes could be adopted for collecting data and executing various processing tasks. We aim to create a storage scheme where every node stores a specific part of the incoming data that exhibit similar statistical characteristics. We propose the real time processing of the incoming data that arrive in the form of vectors. Every time a vector arrives in a node, the node selects the host of the data based on a decision mechanism that combines statistical measures, forecasting techniques and a classification model. We present our model and provide experimental results which show that our scheme is capable of exhibiting high performance in the classification process. We also compare the model with another classification scheme and present comparative results. The advantage of the proposed model is that it can deliver the final decision in the minimum time, thus, it can support real time applications. On the other hand, the scheme requires the presence of a training dataset. Our future research plans involve the inclusion of outlier detection techniques in nodes' decision making. The discussed techniques will identify if a vector is an outlier compared to the entire set of data stored in the nodes.

The paper contributes to IoT data storage research by proposing a system that extends previous works for identifying the appropriate nodes for allocating IoT data based on dataset similarity, node workload and data classification. Its research contribution lies in the maintenance of the 'solidity' of each dataset enabling the efficient definition of query response plans adopted to support intelligent analytic in IoT.

The paper has also implications for practice as it can easily be used to support various applications in diverse domains. The model can be adopted for receiving immediate response to real-time business analytics queries. As an example, a data demanding field in terms of storage and processing in which the proposed system can be adopted is supply chain management. With the increasing use of IoT in supply chain management, multiple IoT nodes in a supply chain can collect data related to supply chain services such as the monitoring of shipments. The collected data can be processed and transferred from the collecting node to the node selected as appropriate for storage based on our proposed system. Decisions regarding data allocation to nodes could be facilitated by a model that shows the appropriate locations to be used for decision making. Supply chains and shipments in particular, can be tracked in real time through a combination of sensors, tracking devices and communication networks which enable the exchange of data that are processed and stored in nodes. The proposed system for data allocation and storage can effectively promote the optimization of supply chain operations and management, allowing for the efficient provision of analytics at every stage of the supply chain.

7 Acknowledgment

This work is funded by the European Commission (H2020 FIRE+) that aims to provide research, technological development and demonstration under the grant agreement no 645220 (RAWFIE).

References

Cai H., Xu B., Jiang L. and Vasilakos, A.V., 'IoT-based big data storage systems in Cloud computing: Perspectives and challenges', IEEE Internet of Things Journal, 2017, 4(1): 75-87.

D' Andria, F., et al., 'Data Movement in the Internet of Things Domain', in Proc. of the European Conference on Service-Oriented and Cloud Computing, Taormina, Italy, 2015, pp. 243–252.

Dolui, K. and Datta, K. S., 'Comparison of edge computing implementations: Fog computing, Cloudlet and mobile edge computing', IEEE 2017 GIoTS, 2017.

Escamilla-Ambrosio P.J., et al., 'Distributing Computing in the Internet of Things: Cloud, Fog and Edge Computing Overview', In: Maldonado Y., Trujillo L., Schütze O., Riccardi A., Vasile M. (eds), Studies in Computational Intelligence, vol 731. Springer, 2018.

Fu, J.-S., Liu, Y., Chao, H.-C., Bhargava, B. K. and Zhang, Z.-J., 'Secure Data Storage and Searching for Industrial IoT by Integrating Fog Computing and Cloud Computing', IEEE TII, 2018.

Habak, K., Ammar, M., Harras, K.A. and Zegura, E., 'Femto Clouds: leveraging mobile devices to provide Cloud service at the edge, In IEEE 8th CLOUD, 2015, pp. 9–16.

Han, J., Kamber, M., Pei, J., 'Data Mining, Concepts and Techniques', Morgan Kaufmann Publishers, 2012.

Jiang, L., et al., 'An IoT-Oriented Data Storage Framework in Cloud Computing Platform', IEEE Transactions on Industrial Informatics, 2015, 10(2), 1443-1451.

Kolomvatsos, K., Panagidi, K., Hadjiefthymiades, S., 'A Load Balancing Module for Post Emergency Management', Elsevier Expert Systems with Applications, 42(1), 2014, pp. 657-667, 2015.

Manning, C. D., Raghavan, P. Schutze, H., 'An Introduction to Information Retrieval', Cambridge University Press, 2009.

Roman, R., Lopez, J., Mambo, M., 'Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges', Future Generation Systems, 78(2), 2018, pp. 680–698.

Ruiz-Alvarez, A. and Humphrey, M. (2012). A Model and Decision Procedure for Data Storage in Cloud Computing. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 13-16, Ottawa, ON, Canada.

Satyanarayanan, M., 'A brief history of cloud offload: A personal journey from Odyssey through cyber foraging to cloudlets', Mobile Computing Communications, 18(4), 2015, pp. 19–23.

Shafagh, H., Burkhalter, L., Hithnawi, A. and Duquennoy, S., 'Towards Blockchain-based Auditable Storage and Sharing of IoT Data', 9th ACM Cloud Computing Security Workshop, 2017.

Tai, J., Sheng, B., Yao, Y., Mi, N., 'Live data migration for reducing sla violations in multi-tiered storage systems', In Proc. of the 2014 IEEE Int. Conf. on Cloud Engineering, 2017, pp. 361–366.

Wichard, J., Ogorzalek, M., 'Time Series Prediction with Ensemble Models applied to CATS Benchmark', Neurocomputing, vol. 70(13-15), 2007, 2371-2378.

Xing, J., Dai, H. and Yu, Z., 'A distributed multi-level model with dynamic replacement for the storage of smart edge computing', Journal of Systems Architecture, 83, 2018, pp. 1-11.