IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 25, 2019, accepted August 17, 2019, date of publication August 29, 2019, date of current version September 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938443

Mist and Edge Storage: Fair Storage Distribution in Sensor Networks

MARINO LINAJE[1], JAVIER BERROCAL[2], AND ALFONSO GALAN-BENITEZ[3]
[1]Department of Computer and Communication Technologies, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain
[2]Department of Computer and Telematic Systems Engineering, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain
[3]Telefónica Soluciones S.A.U, 28050 Madrid, Spain

Corresponding author: Marino Linaje (mlinaje@unex.es)

This work was supported in part by the 4IE+ Project through the Interreg V-A España-Portugal (POCTEP) 2014–2020 Program under Grant 0499_4IE_PLUS_4_E, in part by the Spanish Ministry of Science, Innovation and Universities under Grant RTI2018-094591-B-I00 (MCIU/AEI/FEDER, UE) and Grant TIN2014-53986-REDT, in part by the Department of Economy and Infrastructure of the Government of Extremadura under Grant GR18112 and Grant IB18030, and in part by the European Regional Development Fund. We thank Sergio Laso for his testing work.

ABSTRACT Sensor/Actuator devices are currently being massively adopted, often as nodes of larger sensor networks. These sensor networks are typically dedicated to context acquisition (e.g., get temperature) as well as providing acting services (e.g., open the blinds). However, regarding their own data storage, data is usually sent to Fog/Cloud servers. Fog/Cloud storage solutions provide several advantages over sensor network storage solutions, but also some drawbacks. For instance, in Cloud environments, privacy and legal issues may appear, while in Fog, additional costly hardware must be purchased and maintained, at least a server with redundant storage or many servers when distributed data storage is required. Nowadays, sensor nodes count in thousands around us, and they have significantly increased their storage and computational capabilities over the past few years. Therefore, traditional Fog/Cloud storage solutions could be combined or even replaced by Mist/Edge storage solutions for many use cases. A principal contribution of this paper is a novel data distribution and replication storage solution for wireless sensor networks, the first to consider sensor node heterogeneity to find the optimal storage replication according to node capabilities. The solution has been carefully planned and implemented to run even in very low-end microcontrollers, that lives in many of our surrounding smart devices. Other contributions include data comparing Mist/Edge and Amazon S3 regular storage, showing that there remains plenty of room for research into Mist/Edge storage, as well as into the industry itself.

INDEX TERMS Mist computing, edge computing, distributed storage, sensor networks.

I. INTRODUCTION

During the last decade, we have been observing a massive adoption of sensor/actuation devices. Their size and cost have shrunk significantly, while their storage and computational capabilities have grown as Moore predicted, but maybe not strictly following Moore's Law [1]. This massive adoption includes the public and private sectors, and is related to ''buzzwords'' such as the Internet of Things (IoT), Internet of Everything, SmartX, Industry 4.0, or Smart Manufacturing, but also Ambient Intelligence, Pervasive Computing, or Ubiquitous Computing when there are various sensor devices communicating to work together towards a common goal.

The associate editor coordinating the review of this article and approving it for publication was Abderrezak Rachedi.

Nowadays, these sensor networks are generally deployed using acting devices to function in the environment (e.g., turn on/off the A/C), sensing nodes to get contextual data (e.g., the current temperature in a meeting room), and a central storage point (i.e., server/s) to store the data generated. This central server is also used to retrieve certain data when required in traditional storage solutions. In such traditional storage solutions, this server can be located in the Fog or in the Cloud, using well-known storage techniques. When redundant distributed data storage is required, there has to be more than one central server, increasing the purchasing and maintenance costs. Even though Fog environments are gaining in attraction to improve system behaviour by reducing processing, communication, and energy requirements [2], in this cases many solutions rely on Cloud servers from third-parties, such as Amazon or Google, just as a service to reduce costs.

At least initially, Cloud computing solutions reduce fail tolerance. Cloud environments are intrinsically very reliable, but every intermediate layer reduces this reliability [3], [4]. In order to counteract this effect, different algorithms and frameworks (such as Hadoop [5]) replicate the same information on different servers, increasing the fault tolerance. But privacy issues are greater when the sensed data are outside the organisation's facilities. For privacy reasons, many organisations and governments have specific regulations about the data that can be stored externally, so that resources must be located and maintained inside the organisation's facilities, requiring human resources to maintain them. Also, while Cloud environments provide a large capacity for computing and storage, the distances involved mean lower real-time responsiveness and location awareness, and the cost and network overhead are usually high [4], [6], [7].

The Fog Computing [8] paradigm partially overcomes this waste of resources by adding the data plane to the Network, thus partially or totally eliminating the currently common approach which is to use Cloud environments outside of the local area where the data is produced. There are various approaches focused on different aspects of this data plane: on defining the topology of the network for correct data storage and replication [9], [10], on load balancing to deal with the overload of the nodes and the network [9], [10], or on providing robust distributed storage [11], [12]. Approaches are also needed to replicate the sensed information so as to increase fault tolerance and select the specific nodes in which these replicates should be stored. These techniques should be fair in selecting the nodes according to their capabilities.

Edge Computing is a promising paradigm for exploiting the computing capabilities of low-end devices to improve compliance with the applications' requirements [13]. And Mist Computing is a paradigm designed to take advantage of the computing and storage capabilities of the nodes, hubs, and gateways deployed in the intermediate layers between the Fog/Cloud and the Edge environments. Some of the aforementioned Fog/Cloud problems could be overcome using Edge and Mist computing and storage capabilities, improving response time, location awareness, general network overhead, and deployment costs [14].

There are so many different use cases, depending on the concrete requirements of the project, using sensor networks that there is as yet no universal computing/storage distribution solution to solve all of them. Some layers can be devoted to specific tasks such as data fusion, data/process mining, machine learning, or business intelligence among others. Therefore, one or more layers of the multi-layer Edge-Mist-Fog-Cloud architecture shown in Figure 1 could well not be used.

In this paper, we present a novel solution for distributed persistent and redundant data storage within the sensor network in the Mist and Edge environments. In this solution, the reduced functionality of the heterogeneous (sensing/acting) nodes is raised when possible, adding to them the possibility of storing and retrieving data from a massive media storage device (e.g., their own flash storage or an additionally added external SD memory card). In the worst case, an SD card memory and reader are necessary but, as we shall show, it is also really cheap to keep all the node information for several years. We define our proposal as ''fair'' because the nodes selected to store the information are chosen on the basis of their features and capabilities. Therefore, a device with, for instance, better storage capabilities would be more likely to be selected to store information than others that otherwise are similar. Node capabilities are dynamical and can change during node lifetime, being the storage selection algorithm affected by these changes. By distributing the data storage fairly among the more capable nodes of the network, we ensure keeping the nodes working as regular sensing devices when possible even if they are battery-operated. As we shall show, our proposal reduces the storage cost when compared to typical Cloud solutions, does not overload the network of messages, and adds location-awareness. It provides a high level of privacy and customizable levels of redundancy and fault tolerance, improving both the user and the developer experience.

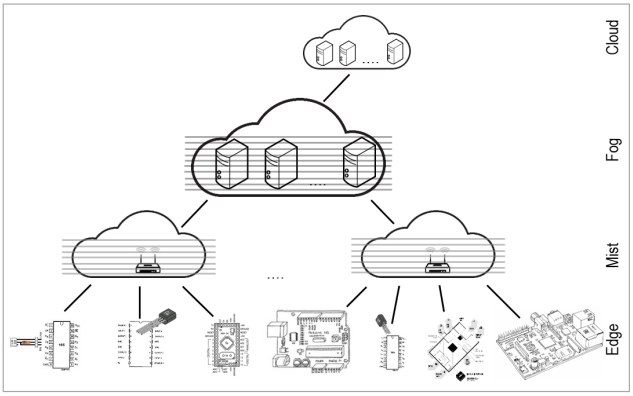

FIGURE 1. Multi-layer edge-mist-fog-cloud architecture.

The proposal presented also includes data retrieval capabilities. But the focus of this communication will be on data storage rather than data retrieval so as to simplify the Evaluation and Related Work sections in what is an already long paper. The rest of the document is structured as follows. Section II describes the motivation of the work. Section III details the mathematical formulae of the Fair Storage Distribution algorithm. Section IV is focused on the implementation of the proposal. Then Section V presents the evaluation of the proposal in different dimensions. The results are discussed in Section VI and related works are analysed and compared in Section VII. Finally, in Section VIII some conclusions are detailed.

II. MOTIVATION AND INSPIRATION

On the one hand, most of the network computational and storage hardware capabilities of Edge and Mist architecture layers are currently wasted [15]. While on the other, we may be paying too much to store information in third-party Clouds

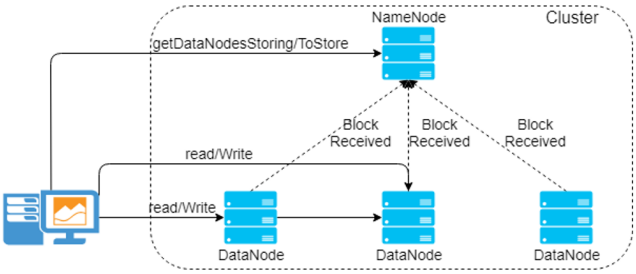

FIGURE 2. HDFS workflow.

at the same time as losing control of our data and having to deal with data privacy issues (information from a sensor can be used to infer when John left his home or when Ann is away on holiday).

During the last few years, different algorithms have been developed to distribute and process large volumes of data, primarily in the Cloud. Even though it has been overtaken by other frameworks, one of the best known is the Hadoop Distributed File System (HDFS) [5], [16] which was our initial inspiration for this work. HDFS provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce [17] paradigm. This framework relies on partitioning and distributing data across thousands of servers to enable the storage capacity to be economically increased by just adding new servers. In addition, this algorithm brings data close to the final applications, reducing the bandwidth and being able to access and compute those data in parallel. HDFS uses two different kinds of servers: one to store the metadata, called NameNode, and many others to store application data, called DataNodes. The NameNode contains information about the files and directories, storing attributes like permissions, modifications, access time, the mapping between files and DataNodes, etc. Each file is divided into blocks which are replicated in multiple DataNodes. The average number of replicates is three, but the user can configure a different number of replicates per file. The usual workflow (Figure 2) when data has to be read is: first, the client contacts the NameNode to get information on the servers storing the files, and then reads the file from the closest server. To write data, the client has to first ask the NameNode for a number of servers with enough capacity to store the information, and then write the information to those DataNodes [18].

HDFS focuses on the distribution of data on different servers. This allows final applications to access data located closer to them (in a nearby server located in the same region), improving computation and response times. However, being a Cloud environment, the times obtained in some cases do not meet the requirements of IoT [19] applications. Analysing the requirements and behaviour of the IoT applications, other researchers have already stated that the scalability, latency, and response times are very limited, and the cost is greater [9]. Furthermore, HDFS was implemented with large volumes of data in mind (files, databases,...) but not the storage needs of IoT (e.g. temperature information needs just few bytes of data, but it is going to be stored many times per hour).

Different data management techniques have been implemented to support these requirements in the Fog Computing paradigm. Examples are Hierarchical Data Aggregation [20] and Fog Storage. Fog Storage can even use virtualisation technology to implement a local repository storing the data in a non-volatile memory [21]. However, Fog solutions still waste Edge and Mist Devices' computing and storage capabilities currently available.

In the sensor networks research field, different partially or fully distributed data storage approaches have been proposed. We shall cover them in some depth in Section VII. As a brief summary of that section, none of these approaches considers the resources and capabilities of each sensor node so as to distribute the data on them fairly.

III. FAIR STORAGE DISTRIBUTION (FSD)

A. FSD FOUNDATIONS

Fair Storage Distribution (FSD) manages the distributed storage and retrieval of information in the different nodes of a sensor network, focusing on the Edge and Mist architecture layers' hardware capabilities. It can work with homogeneous or heterogeneous hardware as well as with wired or wireless sensor nodes.

In order to maximize the potential usage of FSD, from a communications perspective, FSD was conceived to work with the currently most extensively used sensor network architecture. This architecture use the traditional star network topology in which the Edge nodes are connected to a central hub or gateway. Also, one of the commonest communication schemes used to cope with long battery-life sensor nodes is public/subscribe, which we also have adopted to deal with the data distribution while preserving when required the regular wake-sleep periods in the low-end Edge devices.

Thus, in FSD a central communication device is required, as in HDFS. It may belong to either the Mist or the Edge layer and it also serves to temporarily store all the information produced by an Edge node until this information is distributed to the Edge nodes. It also maintain metadata about the sensor nodes in order to apply the fair storage distribution. When the different Edge nodes gather and transmit the data, the FSD enabled node stores the metadata associated with this information (timestamp, source, etc.) and the Edge nodes in which it will be stored. It then distributes the information to the Edge nodes depending on the capabilities of these nodes and the number of replicates. The central node can also be used optionally to store one of the replicates for any of the sensor data that might be produced periodically (e.g., temperature) or not (e.g., smoke in the kitchen). Also, an Edge node with sensing or acting capabilities producing or not producing data can be used or discarded as a storage device, perhaps just because the manufacturer decided to maximize device autonomy.

When designing and implementing FSD central device, it has been taking into account low spec hardware

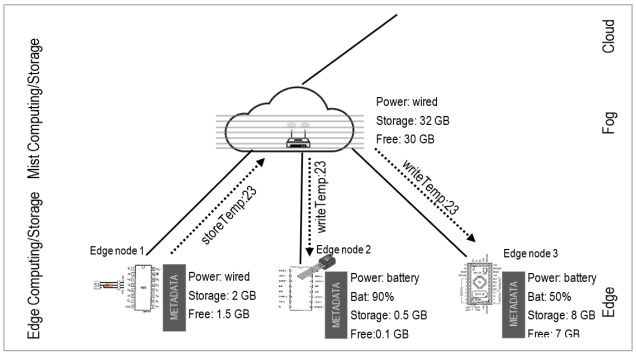

FIGURE 3. Fair storage distribution use case example.

regarding FSD storage and computation requirements, as well as reutilize hardware already typically presented in the sensor network (e.g., hubs and gateways). Thus, this central node continues to serve as a hub or gateway (also known as sink in communications research) while also serving as the central point to receive and distribute, according to the FSD algorithm, all the information produced by the Edge nodes.

Figure 3 shows an example where the Mist node concentrates the data communication coming from the sensor nodes (Temp. 23 from Edge node 1), and distributes it to other nodes in the network to comply with the specified data replication policy (in this case 2 copies/replicates). Among all the sensors in the network, Edge nodes 2 and 3 were selected by FSD according to their capabilities (e.g., node 1 has 2GB capacity and still 1.5GB free) and constraints (e.g., nodes 2 and 3 are battery powered).

Therefore, the storage in the network is fair with regard to the sensor storage selection, while also compliant with one or more objectives/requirements (as will be discussed in Section IV-A) that developers may apply to the algorithm using a set of weights in order to cope with preferences and/or needs for certain projects (e.g., minimize latency).

B. FSD ALGORITHM

To support the fair persistent distributed data storage and retrieval from the nodes, we envisioned an approach responsible for monitoring the features of the network's nodes (including resources and capabilities of interest), such as memory and processing speed, battery levels, etc. Based on this set of features for each node in the network which can be selected to store some replicate, the algorithm selects dynamically, in a very flexible way, the nodes for the common data storage goal.

We tag the algorithm as ''fair'' because each node may or may not be promoted as candidate according to the features it shows, and the choice of the nodes for storage and processing is not done randomly as in other Hadoop-like approaches, but is dependent only on their features.

We tag the algorithm as ''dynamic'' because the central node detects when nodes are added, eliminated or they just changed their capabilities, recomputing the optimal data storage nodes to be used when the next storage operation is required. This dynamism allows any node at any time to re-send its metadata showing that its capabilities have changed over time (e.g., it has stored more information from its own regular operations or has gained energy from a solar panel).

Finally, we tag the algorithm as ''flexible'' because developers can set a goal for the whole sensor network adapting weights in the algorithm that are coupled to specific parameters (as shown below) which can be set in accordance with the project's needs and constrains. E.g., in a project where some battery-powered devices are embedded in the concrete of a building to sense modifications in the structure, these devices could be only used for sensing while avoiding storage in these nodes to maximise their lifetime. However, for other nodes in the network there could also offer storage capabilities to their sensor network, including previous sensors embedded in the concrete. The variability of use cases that we found in the literature was so great that one of our premises was to develop a flexible system able to cope optimally with the specific requirements of quite different projects.

To formalise the FSD algorithm, let $N = \{n_1, ..., n_{|N|}\}$ be the set of nodes in the network. For instance, in a smart-home network, n_3 could be a sensor measuring the temperature.

For each $n_i(\forall i \in N)$ a set of parameters $P = \{p_1, ..., p_{|P|}\}$ is specified. These parameters are used to assess each node's capability for storing data, and thus to identify the optimal nodes for storage in the whole network. For the current implementation, the parameters that are currently evaluated are the available storage, the computing capability, the battery capacity (only when appropriate), the latency with respect to the main node, and the available RAM. All these parameters are normalised as will be detailed below.

Each node in the network is a black box that sends information on the properties that it has to the main node. Considering these parameters, the main node decides which have the best capabilities and conditions to store information from the rest of the nodes of the network. The more information provided by the nodes, the better will be the assessment to identify the optimal ones. Those nodes not sending their information, or sending fewer parameters, score lower in the storage capability equation. The following equation details how the storage capability (st) for one node (n_i) is calculated considering the importance (W) of each parameter for the network designer:

$$n_i^{st} : WxP \rightarrow R \tag{1}$$

The weight of each parameter has a value between 0 and 1, and they sum to 1 (Equation 2). For instance, if only two parameters p_1 and p_2 are available, one combination could be $w_{(p_1)} = 0.3$ and $w_{(p_2)} = 0.7$. The weight of each parameter is fixed by the designer of the network in accordance with the data storage requirements for each deployment.

$$\forall\, p \epsilon\, P, \quad \exists! w(0,1) : \sum_{i=0}^{P} w_{p_i} = 1 \tag{2}$$

A maximum number of replicates (r), i.e., nodes storing simultaneously the same data, must be defined. This property allows network designers to define the redundancy of the system. It is a maximum because a developer might set, for example, 5 replicates, but the sensor network may not even have 5 sensor nodes at all so, that this maximum could not be reached (i.e., the nodes can be selected for storage only once). Therefore, r is a natural number between 1 and the maximum number of nodes within the network (Equation 3). Thus, one can guarantee that at least one node stores the data. Obviously, the greater the number of replicates, the greater the system's redundancy and robustness.

$$r \in \mathbb{N} : 1 \leq r \leq N \tag{3}$$

Finally, we define v_{n_j,p_i} as the element representing the real value (v) of a specific property p_i for a specific node n_j. Therefore, for each node-parameter pair, there exists a value v_{np} detailing the real number that will be used to calculate the storage capacity of the node.

$$\forall\, n_j \in N, \quad \exists\, p_i \in P : v_{n_j,p_i} \in \mathbb{R} \tag{4}$$

These parameters are sent by each node to the main node. For instance, node 5 (n_5) could publish that its storage capacity (p_3) is $v_{n_5,p_3} = 500$ MB. Once this information has been sent to the main node, the algorithm normalises every property so as to be able to identify the storage capability of every node. The following matrix represents the real value for every node-parameter pair.

$$\begin{array}{c} \\ p_0 \\ \vdots \\ p_p \end{array} \begin{array}{c} \begin{array}{ccc} n_0 & \cdots & n_n \end{array} \\ \begin{pmatrix} v_{n_0,p_0} & \cdots & v_{n_n,p_0} \\ \vdots & \ddots & \vdots \\ v_{n_0,p_p} & \cdots & v_{n_n,p_p} \end{pmatrix} \end{array} \tag{5}$$

In order to be able to perform the comparison between nodes, it is necessary to normalise (x) for every parameter of each node. To this end, each parameter is normalised to a value between 0 and 1 in relation to the maximum value of the subset of the values belonging to a single parameter.

$$\forall\, n_j \in N, \quad \exists\, p_p \in P : x_{n_j,p_i} = \frac{v_{n_j,p_i}}{Max\left(v_{n_0,\,p_p}, v_{n_n,\,p_p}\right)} \tag{6}$$

In matrix form, the equation would be:

$$\begin{pmatrix} x_{n_0,\,p_0} & \cdots & x_{n_n,\,p_0} \\ \vdots & \ddots & \vdots \\ x_{n_0,\,p_p} & \cdots & x_{n_n,\,p_p} \end{pmatrix} = \begin{pmatrix} \frac{v_{n_0,\,p_0}}{Max\left(v_{n_0,\,p_0}, v_{n_n,\,p_0}\right)} & \cdots & \frac{v_{n_n,\,p_0}}{Max\left(v_{n_0,\,p_0}, v_{n_n,\,p_0}\right)} \\ \vdots & \ddots & \vdots \\ \frac{v_{n_0,\,p_p}}{Max\left(v_{n_0,\,p_p}, v_{n_n,\,p_p}\right)} & \cdots & \frac{v_{n_n,\,p_p}}{Max\left(v_{n_0,\,p_p}, v_{n_n,\,p_p}\right)} \end{pmatrix} \tag{7}$$

Once all the parameters have been normalised, the previously defined weights come into play to identify the storage capability of each node.

$$\forall\, n_j \in N, \quad \exists\, p_i \in P : y_{n_j,p_i} = x_{n_j,p_i}.w_{p_i} \tag{8}$$

Again, the matrix form of this equation would be:

$$\begin{pmatrix} y_{n_0,\,p_0} & \cdots & y_{n_n,\,p_0} \\ \vdots & \ddots & \vdots \\ y_{n_0,\,p_p} & \cdots & y_{n_n,\,p_p} \end{pmatrix} = \begin{pmatrix} x_{n_0,\,p_0}.w_{p_0} & \cdots & x_{n_n,\,p_0}.w_{p_0} \\ \vdots & \ddots & \vdots \\ x_{n_0,\,p_p}.w_{p_p} & \cdots & x_{n_n,\,p_p}.w_{p_p} \end{pmatrix} \tag{9}$$

The storage capability of a node (n_j^{st}) is calculated using the following equation:

$$\forall\, n_j \in N, \quad \exists! n_j^{st} \in \mathbb{R} : n_j^{st} = \sum_{p=0}^{P} y_{np} \tag{10}$$

Finally, depending on the defined number of replicates (r), the r nodes with the greatest storage capability are selected.

IV. FSD IMPLEMENTATION

The theoretical definition of the distribution system defined above has also been implemented. To that end, we set some specific requirements for the system being developed, the parameters that would be monitored for each node to assess its storage capacity, and the technologies that would be used to implement the system.

The fully developed code, including the code for some hardware prototyping platforms used in the tests, can be found in the shared and public GitHub repository[1].

A. REQUIREMENTS

Table 1 presents the full set of system requirements that we set nearly two years ago when starting the project.

The proposed solution must deal with constrained low-end devices that typically appear in sensor networks (R1), where:

- Each node may have a different power source. They may use batteries, power-line, Power over Ethernet (PoE), inductive or wireless power transfer, to name just a few. Power could be a limitation for some nodes of the network to store data since data storage consumes extra power. So it is essential for the proposed solution to take into account power consumption so as to minimise it in those nodes using batteries so as to prolong their life (R12). Another key factor during the implementation was optimising the processes, so that the data transmission would be performed as efficiently as possible.
- There is a wide range of microcontrollers (MCUs) for sensor nodes sold by different vendors exhibiting different processing power and communication capabilities (the capabilities most interesting for the present work). While many of them do not excel in computing performance, others are capable of quite complex computing.

[1] https://github.com/algalanb/EM-Project

TABLE 1. System requirements.

ID	Short Description	Long Description
R1	Computationally simple	Able to run on heterogeneous very low-end devices
R2	Redundant and distributed	Data must be stored redundantly and distributed
R3	Self-managed	Able to manage itself without human intervention except for the initial setup
R4	Available	At least one node should be capable of storing data
R5	Scalable	Able to add new nodes without re-configuring the main node
R6	Adaptive	Able to add new sensors and therefore new data models
R7	Maintainable	Simple network management
R8	Reliable	Reduced number of possible points of failure
R9	Secure	Communications should be encrypted if the nodes support that feature
R10	In real time	Able to retrieve data instantaneously
R11	Privacy	Prevent attacks from stealing information produced by sensors
R12	Efficient	Reduced consumption and increased system lifetime
R13	Compatible	The system should be able to integrate with other systems
R14	Extensible	Able to adapt to different technologies
R15	Traffic shaping	Able to prioritise certain types of traffic

To select the optimal nodes for data storage, we used benchmarks for MCUs and microprocessors to set the properties of each node, such as those of [21]. Anyway, for new MCUs or those not available in those lists, the engineers responsible for the deployment can set these parameters according to their knowledge and experience.

- It is rare for constrained MCUs and microprocessors to include ''massive'' storage within their IP core. Anyway, the existence of modules and SD cards makes it easy and inexpensive to add massive storage to any node as we will show in Section 5.
- Latency must be as short as possible since a node's power consumption is much greater in RX/TX mode than in idle/sleep mode.

Additionally, the system must support a heterogeneous network of sensors, as may occur in real environments (R1). The current implementation supports many Edge nodes.

Currently, at the implementation level, both wired and wireless devices are supported (R5 and R6). The system developed also requires IP capable devices, so that, among others, CoAP or Zigbee sensor networks lie outside the scope of the implementation, although the proposed algorithm would still be valid. The current implementation relies on a message broker supporting a publish/subscribe mechanism.

B. PARAMETERS MONITORED

The algorithm monitors a set of parameters as metadata individually for each node. We decided to keep this set short to avoid computational complexity (requisite R1 of our system). These parameters are updated periodically by the sensor node and sent to the central node as part of the metadata. The developer can adjust the periodicity of the metadata messages for each sensor node. The set of parameters for FSD are:

- Processing, corresponding to the set of features that make a microprocessor or MCU computationally more powerful than another. Sources such as CoreMark [22] can be used for this classification. In addition, the opinion of the development and deployment engineers should also be relevant in providing details about this parameter since not all the processors are presently available in this and other lists.
- Available/Free Disk, corresponding to the remaining ''massive'' storage capacity.
- Battery, comprising two sub-parameters, one to specify whether the node uses a battery (limited power source), and another to indicate the remaining battery. Again, this is a dynamic value that is sent as metadata.
- Latency, defined as the time that the node takes to respond to an external request. This includes an interesting implicit feature – the frequency at which each sensor node wakes up. Thus, they will be able to reply to any data retrieval request before sleeping again.
- Memory, defined as the free main RAM memory of the node.

C. IMPLEMENTATION TECHNOLOGIES

On the one hand, we tested FSD sensor/actuator node implementation on various ESP8266-based (Lolin nodemcu and wemos), stm32 ARMs (STM32F103C8), and Atmel (ATMega328, ATmega32U4, ATmega2560) development boards. The source code repository also includes the weight of different parameters for some Atmel MCUs as well as for ESP8266. For the FSD firmware implementation to be run on the low-end devices, we chose to support Arduino (which is a set of C and C++ libraries for hardware abstraction and some main function re-modeling) to ensure support for the largest number of 8 to 32 bit MCUs from different vendors. So, any Edge node able to run Arduino code would be instantaneously added with minimal or no effort (R5) as well as any MCU running C.

On the other hand, for the central node a larger MCU or a simple microprocessor is required, since it must be able to run a software stack composed by node.js, the mosquito broker (publish/subscribe is based on MQTT due to its wide-spread adoption), and mongodb. We tested FSD on cheap Intel Atom processors (including intel Edison and Galileo Gen 2), Raspberry Pi 2 and 3 as well as on a commercial home router (Linksys WRT AC1200) with a custom Linux-based openwrt firmware augmente with FSD. All the platforms tested were running Linux.

With respect to the software technologies used, node.js, mosquito, and mongodb have official implementations for different operating systems. As they are also lightweight, we selected them for the development of this project. We checked the best-known c10k [23] capable servers for deployment on these constrained systems, and selected node.js [24] due to its development community and capacities. A no-SQL DBMS was preferred for its flexibility and ability to deal with the adaptivity requirement [25]. Thus, MongoDB [26] was selected in view of its nice integration with node.js. The data format selected was JSON for being lightweight, and coupling not only perfectly with node.js but also with low-end 8-bit MCUs via light-weight C JSON parsers [27].

The communications selection required more effort. We compared AMQP [28], CoAP [29], DDS [30], XMPP [31], and MQTT [32], matching them with the system requirements. We ended up choosing MQTT for being a lightweight publish-subscribe protocol capable of satisfying our security (R9), privacy (R11), extensibility (R14), and traffic shaping (R15) requirements. MQTT includes many Quality of Services levels, making it also suitable for low-end MCUs. It also may includes encryption (TLS/SSL) that can be used natively by some 32-bit MCUs.

A set of communication channels was defined to satisfy requisites R2 (Redundant and Distributed), R3 (Self-managed), and R5 (Scalable), and to keep the system modular to satisfy the maintenance requirement (R7).

D. A SPECIFIC FSD IMPLEMENTATION

For the sake of simplicity, in this subsection we shall explain a specific concrete implementation. We start with the set of nodes that belong (i.e., are connected) to the sensor network. During this first step, each node must register with the central node, sending information about its storage capacity, the information it senses, how that information is transmitted, and so on. To that end, different channels or data flows are defined in order to share different kinds of information with the central node. These channels are implemented as topics (since we are using MQTT) in which a node can publish specific information and to which other nodes can subscribe in order to get that information. The topics defined are: *model* (to share the data model of the information sensed by a node), *model_req* (to request the data model from a specific node), *meta* (to share the FSD parameters of each node), *istate* (to share the sensed data), *query* (to request the central node to send data, e.g., from an external website), *response* (to reply to requests), and *ctrl* (to share control data between nodes, e.g., for an acting node to open a window). Therefore, as Figure 4 shows, for a node to register in the Fog Network, it has to open an MQTT connection and subscribe to the different topics in which it is interested (at least *meta* in order to share its features with the central node, and, if it is a sensor node, *model* in order to share its data model).

In the second phase, once a node has subscribed to different topics, the algorithm must calculate its storage possibilities. Each node in the network sends its features to the central node as a numeric set representing the aforementioned parameters. Figure 5 shows the different features used to calculate the storage capacity of each node, and how each parameter is converted before sending it to the central node.

During this second phase, the central node normalises each node's parameters according to the rest of the values of the same parameter for the different nodes. Parameters not specified by the node are counted as null, decreasing the probability of the node's selection for storage. As will be detailed below, this normalisation consists of dividing each parameter value by the maximum value of that parameter among all the nodes in the network.

In the third phase, each parameter value from the second phase is multiplied by this parameter's generic weight. The weights are in the range 0 to 1 in accordance with the objectives or goals of the whole sensor network, and must sum to 1. The sum of all these parameter values is the corresponding node's score.

In the final phase, these scores and the number of replicates configured in the algorithm are used to select the optimal number of nodes. The number of replicates are configurable in the algorithm to a value between 1 and the number of nodes of the network, since flexibility is one of the approach's requirements.

A video of a laboratory demo testing a collection of heterogeneous hardware is available[2]. It shows the adaptive hot-plugging capacity of the system connecting, disconnecting, and reconnecting devices, changing or leaving unchanged their storage and processing capabilities. The demo video includes a screen recording showing data monitoring and querying using the API that we had developed. Currently, a fork of this implementation is being used in SPILab to manage all the information from the custom sensors and actuators we are running in the Lab as a testbed. This dataset is the only one not shared for privacy reasons.

V. EVALUATION

The present implementation was evaluated in five of its aspects: memory use and cost, fault tolerance, network overload, battery or energy consumption and latency. These five aspects are keys to assessing the feasibility of deploying the proposed distributed storage framework in complex sensor

[2]https://youtu.be/9JbE2f-RJ_U

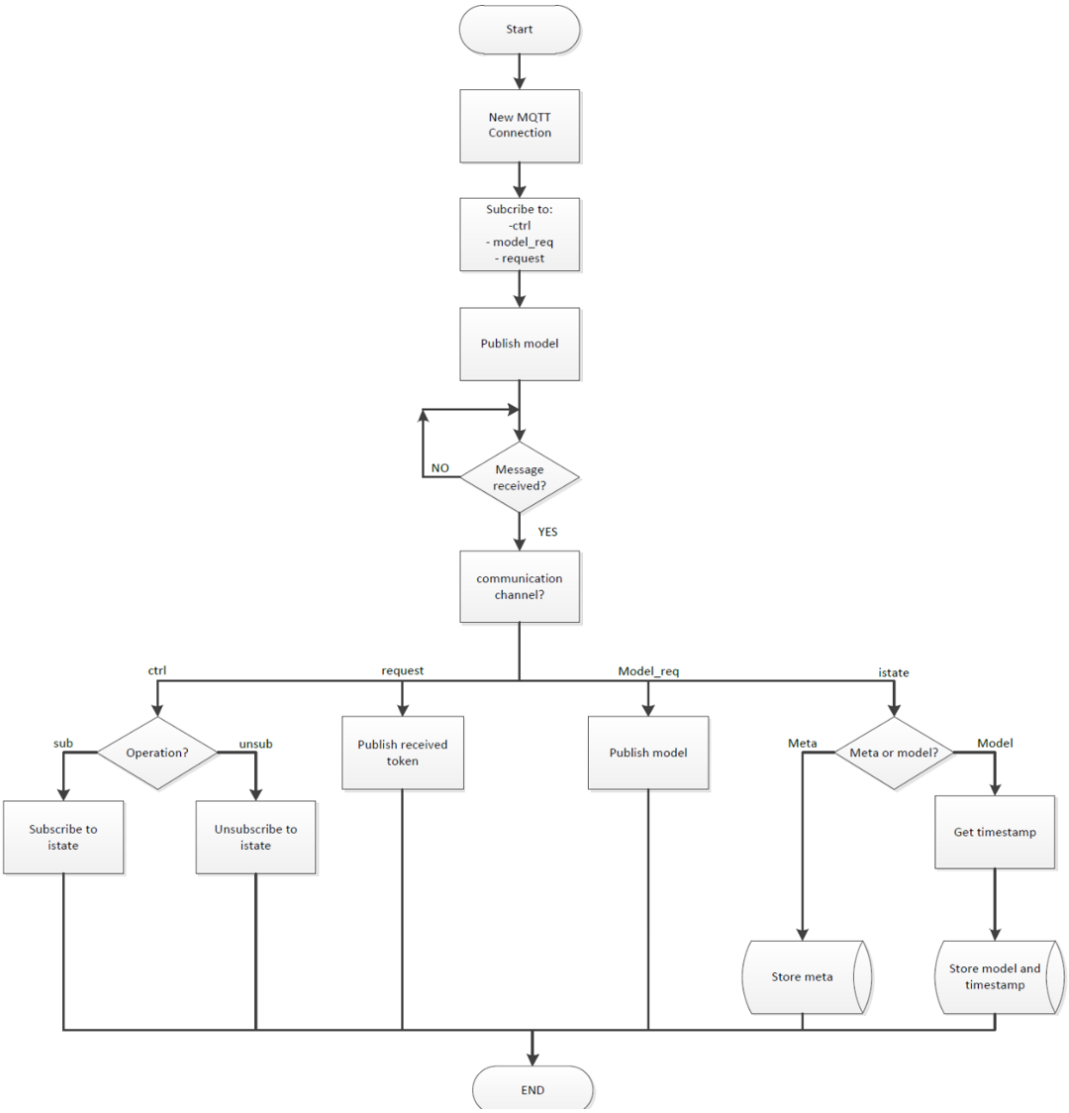

FIGURE 4. Registering and sharing information in a fog network.

network scenarios. The datasets obtained from this evaluation are available as supplementary material to this paper[3].

A. MEMORY USE AND COST

Memory use and cost of data storage is crucial for many IoT applications. As noted in the Introduction, nowadays most solutions are Cloud-based in order to avoid the costs of hardware acquisition and maintenance. We evaluated memory use, regarding disk space and the associated cost, that would be required if the same information were stored in the Cloud environment (using Amazon services, being one of the most common storage options) instead of using FSD in Edge and Mist environments. In this evaluation, we assumed that one wants to store information from a sensor of a room's humidity (percentage) and temperature (to 1 decimal place). The space required to store the sensed information is detailed in Table 2, including some metadata (Node identification and timestamp of the measurements).

TABLE 2. Storage needs.

Information	Space
Node ID	2 bytes
Timestamp	4 bytes
Temperature	2 bytes
Humidity	1 byte
TOTAL	9 bytes

If a developer/user configures the sensor to acquire information every minute, 43 200 entries per month would be generated (525 600 per year). If, however, the Edge node senses the environment more frequently, more entries would

[3]downloadable material (300KB size) available at IEEE Dataport, http://dx.doi.org/10.21227/maqf-3p48

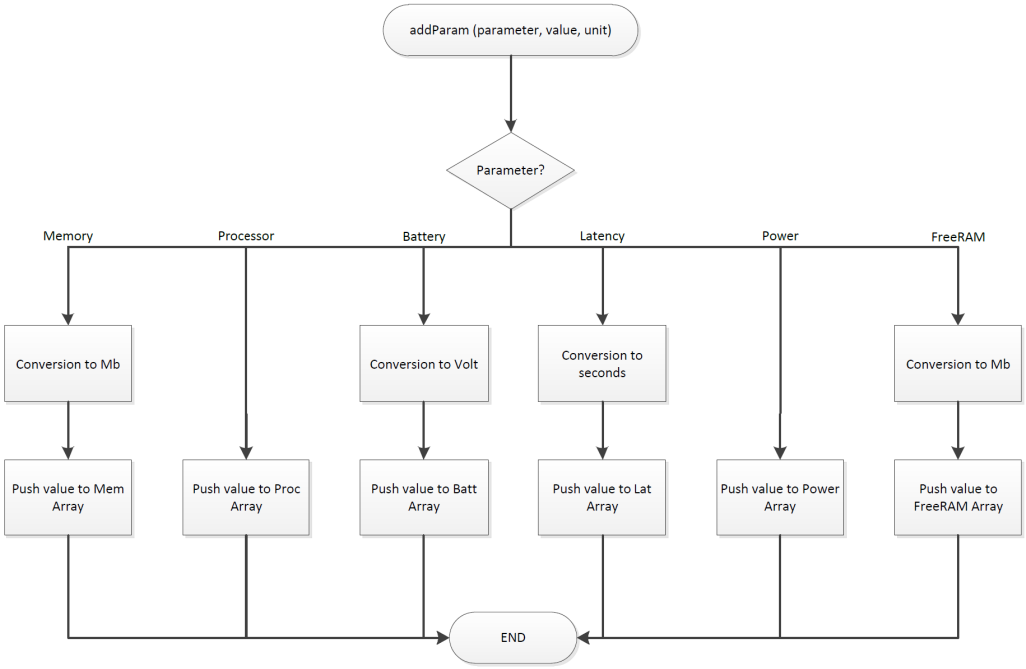

FIGURE 5. Sending the features of each node to the central node.

TABLE 3. Time required to fill a 8GB memory storing the information detailed in table 2.

Number of replicates stored	Time required to fill the memory	
	1 entry/minute	60 entries/minute
Entries from 3 nodes	605 years	10 years
Entries from 5 nodes	363 years	6 years
Entries from 7 nodes	259 years	4 years
Entries from 9 nodes	202 years	3 years

be generated. For instance, a smoke sensor usually senses the environment every second, generating 2 592 000 entries per month (31 536 000 entries per year). E.g., If an Edge node has an 8 GB micro SD, Table 3 gives estimates of the time that would be required to completely fill the memories with data sensed by other different 9 Edge nodes. Obviously, this estimate may depend on the size of the information obtained. Therefore, Table 2 data has been used for this estimation, with a period of one data unit stored every minute from each of the 9 sensors. As can be seen, the memory would be completely full in 202 years. This estimate shows that it seems feasible to store the sensed information in the network itself without any storage capacity issues arising. Even the worst case showed in Table 2 shows 3 years without storage capacity issues. It is also easy and cheap to double the capacity of the SD memories.

Comparing the above estimates with storing the information on Amazon S3, one could identify which architecture has lower cost. To that end, let us assume that on average the node senses just once per minute, that there will be 5 replicates, and that the associated costs are those listed in Table 4. These costs correspond to the prices of Amazon S3 standard [33]. The number of replicas is set to 5 in order to have the fairest possible comparison regarding fault tolerance between FSD and the Amazon cloud storage service as we will discuss during this Section.

Therefore, the cost of Edge storing 5 replicates of the sensed information would be $26.50, since 5 SD cards and SD card reading/writing modules (to access the SD card from a MCU) would be needed.

Figure 6 shows the accumulated cost of the two architectural styles. As can be seen, using the Edge/Mist FSD solution there is a greater cost during the first months. This is because of the need to purchase the extra SD memory cards and modules to handle the proposed FSD algorithm. But, in implementing a Cloud architecture, the system would have to face accumulated costs from the 14th month onwards that are greater than the cost of the Edge solution. It must be taken into account that when one stores e.g., 1GB of data in a third-party server in one month, and then another 1GB the next month, in the second month one must pay for the 2GB total storage used. Also, these storage systems are focused on the transfer of large volumes of data for storage, not on frequent periodic transfers of just a few data that many IoT devices exhibit, as in Table 2 and Table 3. With the price tiers of S3 and similar Cloud services, the major initial cost is the price every 100 entries (Table 4).

The above numbers are for an average frequency of one read per minute. Other sensors may have a higher frequency. For those casesm Table 5 also presents the accumulated costs for the two architectures (Cloud Computing and Edge/Mist Computing) for two frequencies of the sensors reading and sending their state – 1 entry per minute, and 1 entry

TABLE 4. Cost of storing the sensed information for the example.

	Capacity	Price	Price per 100 entries	Replicates	Entries per month
Edge/Mist (SD card)	8 GB	\$5 SD card; \$0.30 SD card module	\$0	5	216 000
Cloud	unlimited	\$ 0.03 per GB/month	\$0.0054	unknown	43 200

TABLE 5. Accumulated cost for different frequencies.

Year	Accumulated Cost (\$) and Storage Required (GB)			
	Edge/Mist (SD card)		Cloud (Amazon S3)	
	1 entry per minute	60 entries per minute	1 entry per minute	60 entries per minute
$1st$ Year	\$ 45.00 (0.02GB)	\$ 45.00 (1.3 GB)	\$ 18.20 (0.004 GB)	\$ 1 091.8 (0.26 GB)
$2nd$ Year	\$ 45.00 (0.04GB)	\$ 45.00 (2.6 GB)	\$ 51.80 (0.009 GB)	\$ 3 107.45 (0.52 GB)
$3rd$ Year	\$ 45.00 (0.07GB)	\$ 45.00 (3.9 GB)	\$ 85.39 (0.013 GB)	\$ 5 123.07 (0.78 GB)

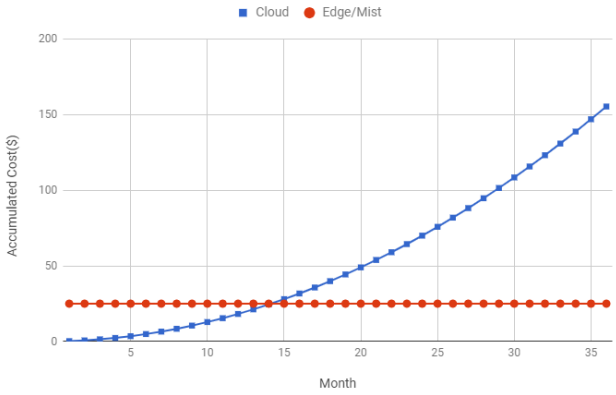

FIGURE 6. Comparison of accumulated costs of Edge/Mist versus Cloud storage for the example.

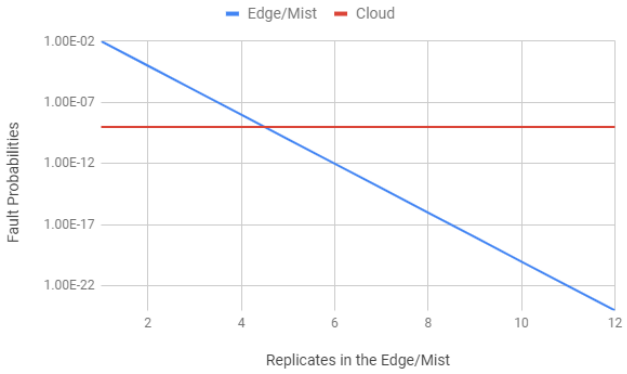

FIGURE 7. Relationship between the number of replicates and the fault tolerance.

per second). While FSD requires to replicate the data, Cloud services do not charge us for this replication, that it is performed in the background and related to the fault tolerance that the Cloud storage service offers.

B. FAULT TOLERANCE

One of the main advantages of a Cloud environment is the reliability of the system. Informally, services such as Amazon S3 claim a fault probability of 10^{-9} [34], while for an SD card, as a hardware component, this probability is only 10^{-2}. However, with defining only 5 replicates per sensed datum, the probability of getting a fault is $10^{-2*r} = 10^{-10}$. Therefore, the presented algorithm not only allows to get a system that is reliable but, flexible (since the number of replicas is adjustable). Figure 7 shows the relationship between the number of replicates in the Edge environment and the fault probability. A lower fault probability implies a higher fault tolerance.

C. NETWORK OVERLOAD

With FSD, the distributed storage implies an increase in the amount of data communicated among the sensor nodes and the central node. In this subsection, we shall compare the network overload produced by FSD with that of a Cloud-based solution.

A Cloud-based solution has many possible ways in which its services might be distributed. In a Cloud infrastructure, the three services (the message broker, the application server, and the DBMS) might be in the same or in different physical/virtual machines, in the latter case, possibly separated by thousands of kilometres. We shall therefore compare FSD with the best and the worst Cloud-based scenarios with respect to the number of messages required. In the best Cloud-based scenario, all the services run in the same physical/virtual machine, while in the worst they run in different ones.

In the best Cloud-based scenario and using the same technology stack described in Section IV-C, the Edge nodes would send their data (message 1) to the message broker. The application's server gets this information and sends it to the mongodb service for storage. If the developer also wants a data schema/model of the information sent by each of the Edge nodes, an initial message from the sensor node to the MQTT broker (message 2) would need to be generated. This model is also useful for the ease of searching and filtering information in an unstructured DBMS such as mongodb.

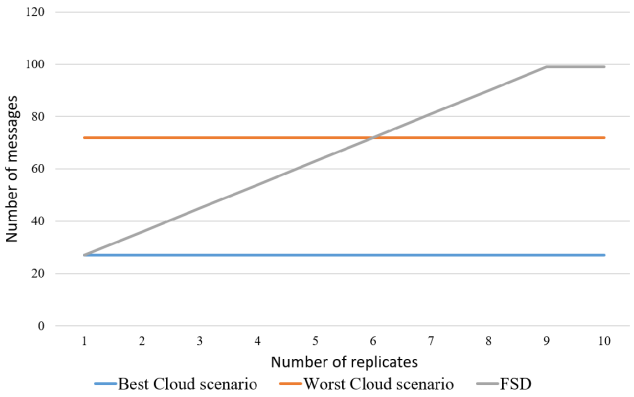

FIGURE 8. Network overload of replication in the cloud and FSD for 9 nodes.

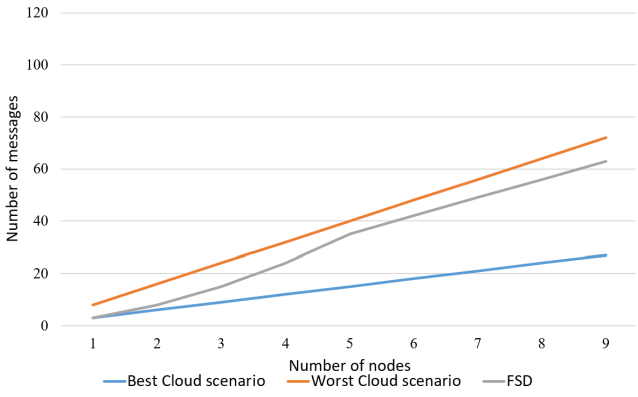

FIGURE 9. Network overload of replication in the cloud and FSD with 5 replicates.

The main network overload problem appears in the worst Cloud environment case. In this case, the three services run on different machines, so that new communications are needed to send the information from one machine to another – for this particular example, a new message from the MQTT broker to node.js (message 3), another from node.js to mondodb to store the sensed data (message 4), and one from the MQTT broker to node.js (messages 5).

When using FSD over Edge/Mist, each sensor node initially sends its model (message 1) to the model channel. Subsequently, the sensor node periodically sends the sensed data (message 2) using the istate channel and a message with the Edge node FSD parameters (message 3) to the meta channel. Only the first is sent internally to mongodb for storage. The major differences arise when replicates are required in order to support distributed storage. For a real comparison, one must consider all the additional messages required to store the replicates on different computing devices. Comparing, for instance, FSD again with Amazon S3 regular storage Cloud service, the latter always uses 2 replicates [34] for each datum sent to be stored. Thus, one can imagine Amazon periodically sending messages with the sensed data to the 2 servers containing the replicates (messages 9 and 10). For FSD to get a slightly better fault tolerance than Amazon S3, 5 replicates are required as explained above, but the system only needs to send the sensed data for replication, not the meta information which is only relevant for the central node to be able to run the FSD selection algorithm. Thus, each sensed data is sent to 5 node channels for storage (messages 4, 5, 6, 7, and 8, let us say).

Figure 8 shows a comparison of how these different storage solutions evolve for a constant number of 9 nodes in the sensor network, a sampling rate of 1 message per minute, and a number of replicates increasing from 1 to 10. One observes in the figure that FSD always leads to more network overload (i.e., it generates more messages) than the best Cloud scenario. Since the S3 solution always stores two replicates, a parameter which we cannot alter, although the worst Cloud scenario produces more network overload than FSD from 1 to 5 replicates, there is then a cross-over point and it produces less overload for 7 replicates or more. Also, above 9 replicates, FSD has a limitation in this scenario since it is only able to store the same replicate in the same node once (i.e., the number of replicates cannot be greater than the number of nodes in FSD). Thus, even if the specification of 10 replicates is given to FSD, if there are just 9 nodes available, the maximum number of replicates (and messages) that would be generated would also be 9. This is the why the FSD line in Figure 8 finishes horizontal in parallel with the Cloud service lines.

Figure 9 plots another interesting comparison as the number of nodes increases from 1 to 10, again for a sampling rate of 1 message per minute, but now with a constant number of 5 replicates (to get a slightly better fault tolerance than S3 as explained above). One observes that, with a small number of nodes (from 1 to the number of FSD replicates which is 5 in this case), the FSD line goes up as a series of steps. The reason is as was noted in the previous paragraph, since the number of replicates in FSD cannot be greater than the number of nodes. FSD always generates more messages than the best Cloud scenario but, fewer messages than the worst Cloud scenario. Also, above 5 nodes, its growth on the plot becomes linear but not quite as steep as that of the worst Cloud case. Indeed, we have also estimated other scenarios with thousands of nodes, finding that FSD continues to separate from the worst Cloud scenario, demonstrating its validity for the future predicted growth of IoT.

D. ENERGY CONSUMPTION

The deployment of this algorithm has an impact in terms of energy consumption, since the MCUs need to do more tasks than the regular sensing and transmission ones. Many IoT devices are battery powered and the deployment of any proposal that impacts negatively on it can lead to shortening their lifetime and impact on the behaviour of the whole system. Therefore, we have performed an analysis of the energy consumed by the FSD algorithm. To that end, we have executed the algorithm at a sampling rate of 60Hz during

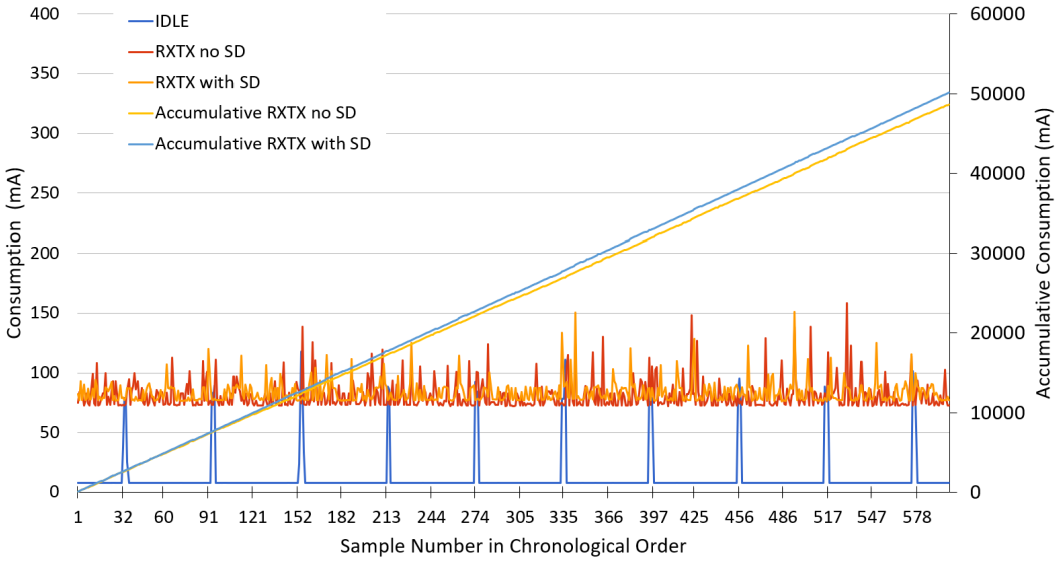

FIGURE 10. Energy consumption at a sampling rate of 60Hz.

10 minutes. For these tests we selected an bare-mounted ESP8266.

As can be seen in Figure 10, we have measured, first, the consumption when the node is idle (blue line). In this mode, the MCU wakes up every minute because it is using a e DTIM beacon interval of 1 minute (DTIM1 according to the MCU datasheet) having a residual consumption around 7.88mA. The red line represent this MCU punctual power consumption while continuously sampling the required sensor, storing and/or sending the sensed data, as well as computing the received requests. This process entails an average consumption around 81.05 mA (291.79 Ah). Finally, we have also measured the consumption when the device is receiving and transmitting extra information from different MQTT channels as well as storing data in the SD to cope with FSD (orange line). This normal node operations plus the extra FSD process entails an average consumption around 83.50 mA (300.61 Ah).

For this test, we assumed that the node is always awaken, avoiding the differences we notice between the different power consumption modes that exhibit different MCUs. Obviously, in a real environment the device will be idle some time to reduce this excessive and unnecessary energy consumption. As can be seen, the consumption is similar in both cases. The FSD algorithm is not power-hungry for the sensor nodes and consumes a small extra amount of energy (around 9.7%).

This increase can better be seen in the accumulative lines drawn in Figure 10 (note that there are two Y axis, one at the left and one at the right). The yellow line depicts the accumulative power consumption when the node does not store any replicates and the blue line depicts the accumulative consumption when the node stores them. In order to better compare these lines, a second left Y-axis was included for these lines. As can be seen, the consumption slightly increases more sharply when the device stores replicates.

The very small difference in the sensor nodes between using or not FSD is because most of the power consumption occurs during the standard connection phase (e.g., getting an IP from the DHCP, connecting to a DNS,...). Due to this phase is already performed by sensor nodes not using FSD, the only extra amount of energy/time required is to send/receive to/from an additional set of channels and store very little information when required. This operations are performed really fast (less than 100ms in average) with the tested MCU 80MHz clock speed.

E. LATENCY

Finally, another dimension that is affected by the FSD algorithm is latency. One of the main advantages of the Fog, Mist and Edge Computing paradigms is the responsiveness, specially in the latter. The presented approach allows developer to store the sensed information in Edge or Mist devices, reducing the responsiveness.

Figure 11 shows the latency obtained for storing a datum in the Cloud environment or in the Edge/Mist using FSD. For evaluating the latency of storing a datum in a Cloud environment, we firstly invoked a Cloud endpoint (deployed in Amazon) for storing the datum. That endpoint was invoked 1 000 times in order to get the average latency. On average it required 83.48 ms to store the datum.

Secondly, we evaluated the time required to store a datum using FSD with different configurations (from 2 to 7 replicates). As can be seen in Figure 11 (blue line), as the number of replicates increases, the latency also increases. Again, in order to get the average latency, the same data was stored 1 000 times. With two nodes, the obtained latency was 13.61 ms, while with seven nodes the average latency

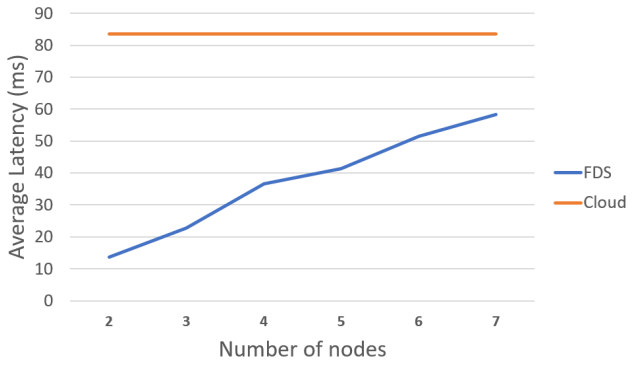

FIGURE 11. RXTX message latency.

was 58.35 ms because of the network overload and the time required for creating and storing every replica. Nevertheless, the obtained latency is always lower than the latency obtained with the Cloud environment. Obviously, with a higher number of replicates, the latency would be higher than the one obtained using a cloud environment. However, we have also discussed before than even with 5 replicas the fault tolerance is slightly better than the Amazon S3 one.

VI. DISCUSSION

In the previous section, we evaluated what in our opinion were the most interesting aspects with which to compare the FSD approach with the state-of-the-art Cloud solutions (currently, maybe the most extensively used development option to store data from sensors). To make a comparison with a real storage system, we chose Amazon S3 regular storage as currently being one of the most used Cloud storage services. Since this service is not in our full control, some details are difficult or impossible to find and/or contrast. We were able to find the parameters required for the evaluation section in the Amazon S3 documentation pages as well as other some informal sites.

Regarding memory usage, Section V-A showed that any Edge/Mist storage approach could be valid for real-world implementations. This is because of the capacity of the SD card memories available nowadays, and the small footprint that produces each datum from a sensor node. If a sensor samples and sends information to the network every second, a regular 8GB memory could store its own information for decades. Even in a scenario with 5 distributed replicates, this SD card could store all the information produced by 5 nodes each second for 6 years. Just doubling the memory to 16GB, quite common storage capacity nowadays, would also easily double this capacity with no further change.

Regarding cost, Section V-A also described a plausible scenario with certain number of replicates, applying the current cost of SD memory cards and the costs associated with Amazon S3 storage. The result was that for the first months it is cheaper to use the Cloud services, but after the 14th month the recurrent costs of the Amazon services surpass the FSD costs. Thus, we can state that for prototypes as well as short-term deployments it is cheaper to use Amazon S3, but for long-term deployment FSD is definitely cheaper. This is logical since memory cards are getting cheaper, and in Amazon as in other third-party Cloud storage solutions one must pay for the accumulative storage as well as for the data transmissions.

Regarding fault tolerance, Section V-B showed that in FSD one has control over the number of replicates, so that fault tolerance can easily be fitted to satisfy specific project requirements. This is not the case for Amazon S3 which has a fixed fault tolerance of 10^{-9}. Section V-B also included the number of replicates that are needed to get this same fault tolerance in FSD. The result was between 4 (to get 10^{-8} fault tolerance) and 5 (to get 10^{-10} fault tolerance). Nonetheless, the delocalization of Cloud solutions has another advantage – less possibility of suffering a disaster in two locations at the same time. Due to the data locality of FSD, a disaster at the place where the sensor network is located could physically affect all the devices. To mitigate this problem, we think that FSD could be combined with other layer architecture storage solutions (e.g., just to back up). To avoid privacy issues in this situation, a clear advantage when using FSD over Amazon, FSD could be extended in the future to support federation, to store data of one sensor network in other Edge/Mist devices of the same organisation but in some other location. As we have shown, storage capacity and cost should not be a problem.

Regarding network overload (addressed in Section V-C), the evaluation showed that the number of messages is not excessive in our opinion. With FSD, one can flexibly adapt the fault tolerance for a specific project or deployment, balancing the network overload and fault tolerance as required. Summarising, we could say that FSD requires an intermediate number of messages when compared with the best and worst Cloud-based scenarios as the number of nodes grows (Figure 9). This statement concerning relationships can also be maintained when replicates are increased.

Energy consumption (Section V-D) is one of the key aspects affecting the deployment of FSD. Many IoT devices shaping the sensor network are battery-powered and any increase in the energy consumption can reduce the applicability of the system. Concretely, we have identified that using the sensor node also to receive and store replicates lead to an increase of 9.7% in the battery consumption. This increase, although it is significant, can be perfectly assumed by many nodes in order to reduce the operational Cloud cost or even to increase the fault tolerance.

Regarding the latency, one of the assumptions of using an Edge/Mist solution is the increase in the responsiveness. The evaluation showed that the latency for storing a sensed data increases as the number of nodes and replicates growth. This fact makes sense since the task, the network overloads, etc. also increase. The results showed that the latency when 7 replicates is still lower that the average latency of storing that same data in a Cloud environment. In addition, with just 5 replicates the reliability of the system would be similar

to a cloud environment (see Section V-B) while the average latency would be the half.

Finally, regarding the implementation, some of the examples provided generate even more messages because they use some FSD features that are beyond the scope of this paper, whose focus has been on the storage capabilities of FSD for Edge and Mist environments. An example of these additional features is a timestamp channel updated by the central node in order to support a common clock for all the Edge nodes when the project requires it.

VII. RELATED WORK

In this section, we shall summarise the main storage techniques applied to sensor networks (Edge and/or Mist layers). We want to highlight that the Edge devices comprise in the literature a wide range of devices, from low-end devices, such as the focus of this paper, to computer servers at the Edge layer. Therefore, not all the research works have the same target devices. Anyway, storage in sensor networks can be categorised as centralised or distributed [35], [36]. In those networks using centralised storage, data is sent to a single node responsible for storing data or is sent outside the network. In those networks using distributed storage, the node keeps the information after sensing the data and sends it to another node or to a set of them to replicate the information. We discarded centralised storage approaches for this Related Work section, since our proposal is a fully distributed data storage solution. Distributed storage can be classified into two categories: fully distributed storage and partially distributed storage.

A. FULLY DISTRIBUTED DATA STORAGE

In the fully distributed category, all nodes contribute equitably to generate and store information. All nodes attempt to store the values taken from sensors locally, and if their memory is full, the task of storing the new values is delegated to another node.

ProFlex [9] and SUPPLE [37] require a tree or mesh topology to work with. ProFlex supports heterogeneous networks with a central mobile node. It is a flexible storage scheme that builds multiple data replication structures. Firstly, it creates several storage structures in a tree where master nodes are the ones with the best features. Secondly, it sends to the central node of each tree the importance factor of each node for data storage. Finally, the central node receives the node information and sends it to the storage nodes. Its main disadvantage is its problem in ensuring data security. SUPPLE consists of three phases. In the first, it creates a logical structure in the form of a tree, where the central node is responsible for receiving data from and replicating it to the network. In the second, it assigns weights to the nodes representing their probability of storing data. And in the third, data is sent to the central node. Its main drawbacks are the central node's high power consumption and the high message overhead.

C&R-DS [38] and S&D-DS [39] are approaches with an emphasis on data security and privacy. The goal of C&R-DS is to prevent attacks stealing information produced by sensors. This type of network consists of three types of nodes: sensor, storage, and master nodes. Communications are encrypted, so that storage nodes need at least a certain good level of computing power and main memory capacity. The sensors periodically send data to the storage nodes. The main disadvantages of this approach are related to data loss and storage node failures. S&D-DS security is based on the use of a shared secret key and Reed-Solomon code [40]. This approach is resistant to the existence of compromised nodes, and uses a technique to verify the integrity of the distributed data. Drawbacks are related to the storage, communications, and computing overhead.

C-Storage [9] and DSforIOT [41] are focused on load balancing, improving the problem of low memory capacity in sensor nodes. C-Storage uses data compression techniques to store information. It employs fewer transmissions than other algorithms. Its main disadvantage is related to security issues when a node is compromised. DSforIOT's main contribution is in providing a low complexity mechanism for data replication in a distributed manner. However, control over the number of replicates is lost so that it has a low data availability.

TinyDSM [11] and DSforCDA [42] focus on reliability, providing robust distributed storage so that data can always be recovered despite possible node failures. TinyDSM ensures data availability by introducing data redundancy. Its main disadvantages are security problems and load balancing. DSforCDA introduces sufficient redundancy at minimal network cost to make it possible to recover data after a failure. The replicate management system presented in this solution brings with it high data availability. Its main disadvantage is that it needs intensive computational processing, hindering its use in constrained embedded nodes and in environments requiring instantaneous results.

DDAS4AN [43] is an approach defining an scheme for selecting the data that should be stored locally in the IoT nodes. With this scheme, nodes collect data from their environment in the form of multidimensional streams and decide, in real time, which data should be stored locally for further processing. To efficiently store the data, first, this approach identifies whether the sensed datum is an outlier by applying a consensus scheme. If the datum is not an outlier, the approach identifies the nodes where it should be replicated by using a heuristic model to identify the top-k nodes where the datum should be replicated taking into account the correlation with the data already stored by each node. Again, in order to identify the outliers and the correlated nodes to store the sensed information, the required computational capabilities are higher.

RBCNS [44] defines a local dissemination and a region-based reconstruction algorithm. This approach focuses on those situations in which users may only need information stored in regions where the monitoring events are sensed, and recovering the global data field is actually not necessary. To that end, the authors use a t-hop local dissemination

TABLE 6. Comparison of the approaches analysed against the FSD requirements.

Req.		ProFlex	SUPPLE	C&R-DS	S&D-DS	C-Storage	DS for IoT	Tiny DSM	DS for CDA	DDAS4AN	RBCNS	ElfStore
R1	Computationally simple	Yes	N/A	N/A	N/A	Yes	Yes	N/A	No	No	Yes	No
R2	Distributed	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R3	Self-managed	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R4	Available	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
R5	Scalable	Yes	Yes	No	Yes	Yes	Yes	Yes	No	N/A	Yes	N/A
R6	Adaptive	No	No	No	No	Yes	Yes	Yes	No	N/A	N/A	N/A
R7	Maintainable	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R8	Reliable	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R9	Secure	No	No	Yes	Yes	No	No	No	No	N/A	N/A	N/A
R10	In real time	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes
R11	Privacy	No	No	Yes	Yes	No	No	No	No	N/A	N/A	N/A
R12	Efficient	Yes	No	No	Yes	Yes	No	No	Yes	N/A	N/A	N/A
R13	Compatible	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
R14	Extensible	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R15	Traffic shaping	No	No	No	No	No	No	No	No	No	No	No

strategy in which each sensor reading is only transmitted by t hops, and does not have to be disseminated throughout the network. To increase the algorithm efficiency, a lazy-encoding algorithm to exploit the correlation among sensor readings is proposed, avoiding the encoding of the readings from irrelevant regions. This approach limit the network overhead and the data encoding, but although an inter-region data recovery algorithm is proposed the complexity of recovering this information is increased.

ElfStore [45] is a low overhead distributed/federated storage for the Edge and Fog layers. For comparison with our approach, it stores not unique measurements from a sensor but streams of data blocks instead of single measurements. It also select the appropriate nodes for replicates storage automatically according to their reliability. Two virtual test have been performed to see the behaviour of the framework but, no real ones. For the scope of ElfStore a device like a Raspberry Pi, with 4 IP cores, is an Edge node. The same device using FSD could be used at the Edge or Mist layers (as a sensor/actuator node or as a hub/gateway with extended capabilities). Thus, no low-end microcontrollers inside most IoT products are the focus of this proposal, that requires more computation capabilities.

B. PARTIALLY DISTRIBUTED DATA STORAGE

In the partially distributed category, some nodes handle the storage of certain information while the rest only acquire information and send it to its destination.

SDS [2] and KDDCS [46] require a tree-based node topology. The SDS storage algorithm is based on spatiotemporal data similarity, providing a search service. It distributes the events so that the inverse of the distance between neighbours stands for the similarity of the data stored on them. However, there is an important lack of security, and the algorithm causes delays. KDDCS is a K-D tree refinement with the number of sensors on each side of the partition being approximately equal. This avoids there being any hot spots caused by an uneven distribution of sensors or events. The disadvantages are a longer delay when making queries, insecurity, and poor behaviour in large networks.

The focus of PDCS [47] and DS-FBA [48] is on security and privacy. PDCS encrypts every communication between nodes, needing computation with the keys to decrypt the data stored in a node. It assumes that the sensor network is divided into cells, with each pair of nodes in neighbouring cells being able to communicate with each other. It provides efficient processing of queries, but its main disadvantage is low data availability. DS-FBA tries to ensure that an attacker could not relate encrypted and original data. It has low overload and computational cost at the expense of increased overall system complexity, but it is quite limited in other required features.

DLB [10] and ASR-DCS [49] focus on load balancing. DLB provides a grid-based solution that dynamically sets the number of storage nodes. It also presents a set of thresholds for each grid so that the load is shared between all nodes. However, it has lower data availability and less security than other solutions. ASR-DCS provides the network with scalability at the same time as the ability to auto-adapt to network conditions. It replicates the data in multiple storage nodes. Its disadvantages are poor security and that it does not completely solve the problem of hot spots.

ADCS [50] and D-DCS [12] focus on reliability to provide a robust system and increase data availability. ADCS applies a hybrid method that dynamically determines network conditions. Based on these conditions, the parent node information will be stored where events are sent. It uses decision methods to select the most appropriate way to store data (centralised or distributed). The approach has been reported

to cause security and availability problems as well as having problems with primary node failures. D-DCS provides distributed storage, periodically changing storage nodes. This allows querying the storage nodes prior to the consultation of past events because the data of a storage node is not overwritten until this node has been chosen several times and in different time windows, thus increasing the temporal availability. The main problems include high cost and loss of information resulting from poor management of the windows.

Table 6 presents our conclusions in matching the FSD system requirements with the current proposals described above. No approach covers a full column as FSD does. C-Storage is the one which covers most requirements, but even then, only 10 out of the 15 are covered, leaving 1/3 of them uncovered.

VIII. CONCLUSION AND FUTURE WORK

The combination of the technologies selected for the implementation, including a heterogeneity of constrained devices, presented a level of performance that surprised us pleasantly. Even though this is an ongoing project, we consider it mature enough to be shared with the developer and research community to help advance Edge/Mist Computing distributed storage with low-end/constrained devices.

We currently work on improving the current synchronisation system and API, and the security aspects of the system. We have also thought about deploying a rule service like Drools to add alarms and improve the general management of the system. In addition, we are evaluating how to integrate FSD with in a Edge, Fog and Cloud infrastructure in order to provide support to projects that have certain specific requirements or potential problems due to excessive locality and, also, to get the maximum benefit from each paradigm.

Many issues remain open, such as locality problems that could affect fault tolerance of the whole system, and should be addressed in the future by the research community.

REFERENCES

[1] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, April 19, 1965, pp. 114 ff," *IEEE Solid-State Circuits Soc. Newslett.*, vol. 11, no. 3, pp. 33–35, Sep. 2006.

[2] H. Shen, L. Zhao, and Z. Li, "A distributed spatial-temporal similarity data storage scheme in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 10, no. 7, pp. 982–996, Jul. 2011.

[3] A. Rahmani, P. Liljeberg, J.-S. Preden, and A. Jantsch, Eds., *Fog Computing in the Internet of Things: Intelligence at the Edge*. Cham, Switzerland: Springer, 2018. [Online]. Available: https://www.springer.com/gp/book/9783319576381

[4] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *Proc. 20th Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Jul. 2013, pp. 43–46.

[5] T. White, *Hadoop: The Definitive Guide*. Newton, MA, USA: O'Reilly Media, 2009. [Online]. Available: http://shop.oreilly.com/product/0636920033448.do

[6] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the Internet of Things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug. 2016.

[7] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive Mobile Comput.*, vol. 52, pp. 71–99, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119218301111

[8] OpenFog Consortium. (Feb. 2019). *OpenFog Reference Architecture for Fog Computing*. [Online]. Available: https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf

[9] A. Talari and N. Rahnavard, "CStorage: Distributed data storage in wireless sensor networks employing compressive sensing," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2011, pp. 1–5.

[10] W.-H. Liao, K.-P. Shih, and W.-C. Wu, "A grid-based dynamic load balancing approach for data-centric storage in wireless sensor networks," *Comput. Elect. Eng.*, vol. 36, no. 1, pp. 19–30, Jan. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790609000573

[11] K. Piotrowski, P. Langendoerfer, and S. Peter, "tinyDSM: A highly reliable cooperative data storage for wireless sensor networks," in *Proc. IEEE Conf. Publication*, May 2019, pp. 225–232. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5067485

[12] N. Cuevas, M. Uruena, G. de Veciana, R. Cuevas, and N. Crespi, "Dynamic data-centric storage for long-term storage in wireless sensor and actor networks," *Wireless Netw.*, vol. 20, no. 1, pp. 141–153, Jan. 2014. doi: 10.1007/s11276-013-0598-5.

[13] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015. [Online]. Available: http://doi.acm.org/10.1145/2831347.2831354

[14] M. Uehara, "Mist computing: Linking cloudlet to fogs," in *Computational Science/Intelligence & Applied Informatics* (Studies in Computational Intelligence), R. Lee, Ed. Cham, Switzerland: Springer, 2018, pp. 201–213. doi: 10.1007/978-3-319-63618-4_15.

[15] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: Architecture, key technologies, applications and open issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 27–42, Nov. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517302953

[16] J. Venner, *Pro Hadoop*. New York, NY, USA: Apress, 2009.

[17] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Opear. Syst. Design Implement. (OSDI)*, vol. 6, 2004, p. 10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264

[18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.

[19] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[20] K. Vatanparvar and M. A. Al Faruque, "Control-as-a-service in cyber-physical energy systems over fog computing," in *Fog Computing in the Internet of Things*. Cham, Switzerland: Springer, 2018, pp. 123–144. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-57639-8_7

[21] (Feb. 2019). *OpenStack Open Source Cloud Computing Software*. [Online]. Available: https://www.openstack.org/

[22] Embedded Microprocessor Benchmark Consortium. (2019). *CPU Benchmark MCU Benchmark CoreMark EEMBC*. Accessed: Jan. 4, 2019. [Online]. Available: https://www.eembc.org/coremark/

[23] D. Kegel, "The C10K problem," Tech. Rep., 2006. [Online]. Available: http://www.kegel.com/c10k.html

[24] (2019). *Node.js Foundation*. [Online]. Available: https://nodejs.org/en/

[25] T. Li, Y. Liu, Y. Tian, S. Shen, and W. Mao, "A storage solution for massive IoT data based on NoSQL," in *Proc. IEEE Int. Conf. Green Comput. Commun. (GreenCom)*, Nov. 2012, pp. 50–57.

[26] (2019). *Open Source Document Database*. Accessed: Jan. 4, 2019. [Online]. Available: https://www.mongodb.com/index

[27] K. Maeda, "Performance evaluation of object serialization libraries in XML, JSON and binary formats," in *Proc. 2nd Int. Conf. Digit. Inf. Commun. Technol. Appl. (DICTAP)*, May 2012, pp. 177–182.

[28] Oasis. (Feb. 2019). *AMQP—Advanced Message Queuing Protocol*. Accessed: Apr. 24, 2019. [Online]. Available: https://www.amqp.org/

[29] CoAP. (Feb. 2019). *CoAP—Constrained Application Protocol*. Accessed: Apr. 24, 2019. [Online]. Available: http://coap.technology/

[30] DDS. (Feb. 2019). *DDS Portal—Data Distribution Services*. Accessed: Apr. 24, 2019. [Online]. Available: http://portals.omg.org/dds/
[31] P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core*, document RFC 3920, 2011. [Online]. Available: http://www.rfc-editor.org/info/rfc6120
[32] MQTT. (Feb. 2019). *MQTT—Message Queue Telemetry Transport*. Accessed: Apr. 24, 2019. [Online]. Available: http://mqtt.org/
[33] Amazon Simple Storage Service. (2019). *Cloud Storage Pricing S3 Pricing by Region*. Accessed: Jan. 9, 2019. [Online]. Available: https://aws.amazon.com/s3/pricing/
[34] (2019). *Amazon S3 Reduced Redundancy Storage*. Accessed: Jan. 15, 2019. [Online]. Available: https://aws.amazon.com/s3/reduced-redundancy/
[35] N. M. Nair. (2013). *Survey on Distributed Data Storage Schemes in Wireless Sensor Networks*. [Online]. Available: https://paper/survey-on-Distributed-Data-Storage-Schemes-in-Nair/9f24a56bcdcb9b8b336ecea59f2a65337433d2db
[36] X. Ma, J. Liang, R. Liu, W. Ni, Y. Li, R. Li, W. Ma, and C. Qi, "A survey on data storage and information discovery in the WSANs-based edge computing systems," *Sensors*, vol. 18, no. 2, p. 546, Feb. 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/2/546
[37] A. C. Viana, T. Herault, T. Largillier, S. Peyronnet, and F. Zaïdi, "Supple: A flexible probabilistic data dissemination protocol for wireless sensor networks," in *Proc. 13th ACM Int. Conf. Modeling, Anal., Simulation Wireless Mobile Syst. (MSWIM)*, 2010, pp. 385–392. [Online]. Available: http://doi.acm.org/10.1145/1868521.1868586
[38] K. V. Kusuma and M. R. Prasad, "Confidential and reliable data storage in WSN," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 4, pp. 148–151, Apr. 2013.
[39] W. Ren, Y. Ren, and H. Zhang, "Secure, dependable and publicly verifiable distributed data storage in unattended wireless sensor networks," *Sci. China Inf. Sci.*, vol. 53, no. 5, pp. 964–979, 2010. [Online]. Available: https://link.springer.com/article/10.1007/s11432-010-0096-7
[40] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*. Hoboken, NJ, USA: Wiley, 1999.
[41] P. Gonizzi, G. Ferrari, V. Gay, and J. Leguay, "Data dissemination scheme for distributed storage for IoT observation systems at large scale," *Inf. Fusion*, vol. 22, pp. 16–25, Mar. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253513000444
[42] S. Yulong, X. Ning, P. Qingqi, M. Jianfeng, X. Qijian, and W. Zuoshun, "Distributed storage schemes for controlling data availability in wireless sensor networks," in *Proc. 11th Int. Conf. Comput. Intell. Secur.*, Dec. 2011, pp. 545–549.
[43] K. Kolomvatsos, P. Oikonomou, M. G. Koziri, and T. Loukopoulos, "A distributed data allocation scheme for autonomous nodes," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Oct. 2018, pp. 1651–1658.
[44] S. Zhou, Y. He, S. Xiang, K. Li, and Y. Liu, "Region-based compressive networked storage with lazy encoding," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1390–1402, Jun. 2019.
[45] S. K. Monga and Y. Simmhan, "ElfStore: A resilient data storage service for federated edge and fog resources," 2019, *arXiv:1905.08932*. [Online]. Available: https://arxiv.org/abs/1905.08932
[46] M. Aly, K. Pruhs, and P. K. Chrysanthis, "KDDCS: A load-balanced in-network data-centric storage scheme for sensor networks," in *Proc. 15th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2006, pp. 317–326. [Online]. Available: http://doi.acm.org/10.1145/1183614.1183662
[47] M. Shao, S. Zhu, W. Zhang, G. Cao, and Y. Yang, "PDCS: Security and privacy support for data-centric sensor networks," *IEEE Trans. Mobile Comput.*, vol. 8, no. 8, pp. 1023–1038, Aug. 2009.
[48] H. Liu, H. Wang, and Y. Chen, "Ensuring data storage security against frequency-based attacks in wireless networks," in *Distributed Computing in Sensor Systems* (Lecture Notes in Computer Science), R. Rajaraman, T. Moscibroda, A. Dunkels, and A. Scaglione, Eds. Berlin, Germany: Springer, 2010, pp. 201–215.
[49] P. Hejazi and H. H. Amin, "An adaptive method for structured replication data-centric storage in wireless sensor networks," in *Proc. 5th Int. Conf. Inf. Technol. Multimedia (ICIMU)*, Nov. 2011, pp. 1–5.
[50] S. Babaei and M. Sabaei, "Adaptive data-centric storage in wireless sensor networks," in *Proc. 3rd Int. Conf. Comput. Res. Develop.*, vol. 1, Mar. 2011, pp. 163–167.

MARINO LINAJE received the Ph.D. degree in computer science, in 2009. He is currently a Teacher and a Researcher with the University of Extremadura. He is also the Co-Founder of two companies (MultipleCaracter and Homeria Open Solutions). Since 2010, he has been researching and teaching Ubiquitous Computing, including trending topics, such as the Internet of Things, wearable computing, ambient intelligent, crowdsensing, and close related ones. He has published several articles related to these topics in top level conferences as well as international journals, since 2004. The aim of his research is to automatize process, including the creation and deployment of complex systems and systems of systems.

JAVIER BERROCAL received the Ph.D. degree in computer science from the University of Extremadura, Spain, in 2014. From 2010 to 2016, he was an Assistant Professor with the Department of Informatics and Telematics System Engineering, University of Extremadura, where he obtained an associate position, in 2016. He is also the Co-Founder of the software consulting company Global Process and Product Improvement S.L. (Gloin), in 2010. He has coauthored numerous peer-reviewed articles in international journals, workshops, and conferences. His research interests include mobile computing, context awareness, pervasive systems, crowd sensing, the Internet of Things, and fog computing.

ALFONSO GALAN-BENITEZ received the bachelor's and master's degree in telecommunications engineering from the University of Extremadura. He is currently serving services with the Security Operation Center (SOC) of Telefónica Soluciones S.A.U. Their job is to provide cyber security solutions for large companies, doing network planning and deployment of security equipment (firewalls, proxies, correlators, and so on). On the other hand, it is passionate about innovative technologies, focused on the Internet of Things (IoT).