INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING
DOI: 10.46300/9106.2021.15.193 Volume 15, 2021

IoT Cloud Computing Middleware for crowd monitoring and evacuation

Alexandros Gazis[1, *], Eleftheria Katsiri[1,2]
[1]Democritus University of Thrace, Department of Electrical and Computer Engineering, Xanthi, 67100, Greece
[2] Institute for the Management of Information Systems, Athena Research & Innovation Center in Information Communication & Knowledge Technologies, Marousi, 15125, Greece
[*]agazis@ee.duth.gr, ekatsiri@ee.duth.gr

Received: June 8, 2021. Received: November 28, 2021. Accepted: December 9, 2021. Published: December 23, 2021.

Abstract— Map-Reduce is a programming model and an associated implementation for processing and generating large data sets. This model has a single point of failure: the master, who coordinates the work in a cluster. On the contrary, wireless sensor networks (WSNs) are distributed systems that scale and feature large numbers of small, computationally limited, low-power, unreliable nodes. In this article, we provide a top-down approach explaining the architecture, implementation and rationale of a distributed fault-tolerant IoT middleware. Specifically, this middleware consists of multiple mini-computing devices (Raspberry Pi) connected in a WSN which implement the Map-Reduce algorithm. First, we explain the tools used to develop this system. Second, we focus on the Map-Reduce algorithm implemented to overcome common network connectivity issues, as well as to enhance operation availability and reliability. Lastly, we provide benchmarks for our middleware as a crowd tracking application for a preserved building in Greece (i.e., M. Hatzidakis' residence). The results of this study show that IoT middleware with low-power and low-cost components are viable solutions for medium-sized cloud computing distributed and parallel computing centres. Potential uses of this middleware apply for monitoring buildings and indoor structures, in addition to crowd tracking to prevent the spread of COVID-19.

Keywords— middleware, covid-19, Internet of things (IoT), Map-Reduce, wireless sensor networks (WSN), distributed fault-tolerant sensing system, crowd monitoring, crowd tracking, crowd evacuation, preserved building middleware, cultural buildings middleware.

I. INTRODUCTION

SCIENTISTS and experts have focused their efforts on advancing traditional computing infrastructure and techniques via the use of "Industry 4.0". More specifically, this term is used to describe the fourth industrial revolution, which involved breakthroughs in manufacturing, artificial intelligence, machine learning, and data science in general. In between our era and the future is the so-called "Internet of Things (IoT)", i.e., the development of interconnected devices capable of communicating and processing information typically arising from sensory networks [1].

On the one hand, recent developments in low-cost sensor technologies enabled the use of crowdsourcing techniques and IoT for the collection and management of measurements with wide spatiotemporal coverage. An increasing number of applications in the fields of embedded computing [2], education [3], wildlife [4], environment [5], network monitoring [6], business analytics [7], robotics [8], smart sensing [9], etc. rely increasingly on sensor networks. Because of the limited capabilities of sensor devices, computation often needs to be outsourced to a powerful computing infrastructure. IoT offers a common infrastructure which sensors can use to forward their data to more capable computing devices for analysis, aggregation, and storage, using standardized protocols such as 6LoWPAN [10] over IPv6 [11] or LoRaWAN [12]. These protocols are important serving as a communication tool for large data streams that are generated from sensor devices. This data is often severely limited by the available bandwidth and latency of communication protocols in low-power and wide-area networks, such as NB-IOT [13], making in-network processing necessary.

On the other hand, the large amount of data generated by sensors requires simple computations to be distributed across hundreds or thousands of nodes in a reasonable amount of time. The main issues we faced where how to parallelize the computation, distribute the data, and handle failures. To deal with the ever-increasing volume of high-data information, Map-Reduce was introduced.

Map-Reduce was originally developed by Google, based on parallel and distributed processing principles via Apache's Hadoop open-source project. It soon became the de facto solution for big data applications. Map-Reduce aims to facilitate data parallelization, load balancing, and data distribution through flexible, simple, and scalable processing. It can process both structured and unstructured data, while its characterized by its fault tolerance capabilities, because tasks leading to failed nodes must be restarted [14]. This model resembles the Message Passing Interface standard, presented in the early 1990s, which introduced reduce and scatter operations [15] and it ensures that applications can process large data sets via the use of distributed capabilities and data processing [16].

Moreover, Map-Reduce is a thoroughly examined research subject in regard to cloud computing applications. It became widely available and used by academics and industry, due to Hadoop [17] and, more recently, Spark [18]. These open-source projects provide a complete software platform for making computations in computer clusters. While both platforms have similar functionality, they greatly vary in performance [19], [20], [21], depending on the case scenario (e.g., machine learning [22], cloud computing [23], or networking [24]. Hadoop excels at batch processing tasks [25], whereas Spark has a higher performance in real-time data streams [26]. In-fault detection is typically handled via the "Monitor-Analyse-Plan-Execute" protocol [27].

In our study, we emphasized on developing a distributed and parallel processing system to address one of the main weaknesses of the Hadoop architecture, i.e., how the system operates in the event of the master's node failure. To deal with this problem, the Hadoop framework, uses checkpointing where, in case of failure, the system's information is stored in the remote repository. Afterwards, this information is used to restore the system to its previous operating status. Unfortunately, even though this method is highly effective, it cannot be implemented in the case of sensor networks due to their characteristics. Analytically, the sensor nodes are typically low-power and low-cost computing devices that have severe limitations from SD cards failure, due to the re-writes to computation resources (either CPU/RAM capabilities or the risk of increasing the devices' temperature). As a result, implementing this solution is a task which demands more resource-intensive machines thus, increasing the overall cost during both the day-to-day operation of the systems and future upgrades in case of virtual scaling of the master or horizontal scaling of the system.

II. AIMS AND OBJECTIVES

The aim of this article is to provide a top-down approach to a WSN cloud-based system focusing on its architecture and applications. Moreover, we present a crowd tracking middleware application that implements a distributed algorithm. This application can be used as a tool to monitor the total number of visitors in indoor structures. Analytically, in the midst of the COVID-19 pandemic and in an effort to monitor a potential exposure location or follow COVID-19 positive cases, this method could be used as a tool to study and measure the crowd's density, thus reducing the virus' spread. Specifically, most of the existing solutions focus on developing such systems as evacuation tools [28]), while our work focuses on tracking, locating and monitoring the total visitor count per room.

The objective of this article, similarly to [29], is to use the suggested middleware as both a COVID-19 tool and a means to track the rooms which attract the most interest among the public. The main advantage of our novice middleware architecture is that it focuses on providing a cloud computing solution that is less resource intensive than the tasks in recent bibliography which mainly rely on image/video analysis [30]. Moreover, our middleware uses low-power and low-cost components which can effectively handle small to medium sized data and network load tasks thus making it ideal for rapid prototyping applications.

The outline of this article is as follows: firstly, we provide a literature review of the most used distributed principles in WSN, emphasizing on map-reduce algorithms implementations in middleware applications. Secondly, we explain the Map-Reduce algorithm which inspired this current work. Thirdly, we provide details on the building used as a test case scenario for this application. Fourthly, we suggest each step of our novel approach in detail, as well as its architecture for a fault-tolerant algorithm on distributed computer networks. We also present said system and provide the database properties and Java classes used for its development. Lastly, we discuss the results and benchmarks for the proposed system and algorithm on low power and low-cost computing devices (i.e. Raspberry Pi), as well as future works of this publication.

III. RELATED WORK

Before designing and applying our system, we examined related work regarding middleware. Specifically, for hardware (sensors or WSNs) and software to interact, there must be a middle layer responsible for coordinating, triggering, and orchestrating all necessary services and processes to achieve optimal functionality. In other words, middleware acts as a bond that unites various domains, services, or applications into one single entity to handle a specific task [30].

According to recent studies, most middleware applications are WSN-centric [31]. This is because WSNs offer a simple, fast, and reliable network of information which combines different technologies. This is the reason middleware is responsible for various tasks, including processing information from multiple sources, monitoring system connectivity, scaling system resources, and coordinating computer nodes [32].

In a recent bibliography, middleware-based applications focus on multimedia [33], intelligent service processing [34], high performance computing [35], mobile edge computing [36], fog computing [37], data transmission [38], automotive industry [39], big data [40], web ontology [41], context-awareness [42], semantics [43] and service-oriented [44], microservice-oriented [45] or software-orientated [46] architectures. Since middleware is case specific, several studies have been conducted regarding design and implementation applications in IoT [47], environmental monitoring [48], and urban activities [49]. Other examples include home automation [50], healthcare [51], parking systems [52], sensor systems [53], farming [54], robotics [55], blockchain [56], urban pollution [57] and sound monitoring [58], autonomous driving [59], mobile sensing and social networks [60].

Moreover, widely used technologies in middleware solutions are message brokers such as RabbitMQ [61], ActiveMQ [62], ZeroMQ [63], and notably Kafka [64]. While these solutions vary in performance [65], [66], [67], they are extensively used as a tool to "glue" together different computer components. In addition, there are several middleware technologies used in cloud computing [68], most notably open-source solutions, such as Berkley's University OpenWSN [69] and OpenIoT [70].

Furthermore, we studied Google engineers' article on Map-Reduce [71] and we focused on developing a variation of the word count example presented. The algorithm studied provides a platform for splitting and processing considerable data sets simultaneously, thereby providing faster service. This results in data sets breaking into several tasks and processes to capitalise on distributed computing [72].

Typical examples of applications using Map-Reduce are social network applications (counting of words, users, posts, or reactions) [73], [74], web page ranking and indexing, geolocation applications [75], [76], financial services (analytics, risk assessment, investment and trading algorithms) [77], IoT [78], etc. Specifically, implementing the Map-Reduce concept for a truly distributed architecture such as a WSN is not straightforward, as several challenges emerge [79] in the areas of fault tolerance [80], data distribution [81], load balancing [82], reliability [83] and energy efficiency [84]. Map-Reduce and WSN have been used in many scientific fields, such as IoT [85], structures [86], smart grid [87], climate [88], and big data processing [89].

Finally, there are many interesting projects that implement the Map-Reduce paradigm. Hadoop is one of the most widely used projects, which enables programmers to develop and execute data-intensive applications for processing data. Similarly to Hadoop, POSUM uses message-based event handling for real-time decisions. Based on the system's feedback, it uses an architecture of three entities: the data master, the simulator, and the orchestrator [90]. Furthermore, other notable distributed Map-reducing projects are Hsim [91], MRPerf [92], MRSG [93], and Yarnsim [94].

IV. BACKGROUND AND METHODS

This section presents the main components of the Map-Reduce algorithm paradigm and provides a detailed explanation of the existing algorithm. Moreover, we review a real case study used to develop our IoT middleware. Lastly, we review the key contributions of our new algorithm and how it was implemented to count the visitors in the indoor structures of a preserved building (M. Hatzidakis' residence).

A. Map-Reduce

In the mid-1990s, there were many large operations—mainly data centres—that upgraded their systems (e.g., to more resilient and more power-intensive solutions), due to the advances in hardware components. Specifically, businesses needed to process a large quantity of data, which were either "noisy" or sent in different formats and from different sources, or the sampling rates were too high to analyse.

The first steps were to increase the overall computing power of their overall clusters (i.e., tightly connected computers that work together and usually simultaneously for a task). However, after they reached a certain point, it became obvious that it was inefficient to use raw computing power to achieve a computing task, since the key to success was optimising the processes. Subsequently, among the most important solutions introduced to address this issue was Map-Reduce.

The operating principle behind Map-Reduce is straightforward as it consists of two functions, i.e., mapping and reduction. The first allows values to be denoted using a specific key, while the second analyses and combines these key-value pairs into different and smaller pairs.

B. Map-Reduce Algorithm

The Map-Reduce model is used to structure (mapping) and divide (reduction) a computation task on many computers. It uses multiple data centres, assigning less power-intensive tasks to weaker benchmarked computers and ultimately modelling operations to effectively distribute tasks so as not to waste of CPU time and usage (clock ticks). This is important because when a Map-reducing "job" is submitted to a cluster, the code runs simultaneously in multiple computer devices.

Map-Reduce is also known for Hadoop, its most popular open-source implementation for computations in clusters. The Map-Reduce paradigm, as presented in Figure 1, is characterised by the following steps [95]):

1. <u>Map</u>: reception of input data and mapping them to be processed.

$$\mathbf{key, value\ pairs} \rightarrow \mathbf{f}(\mathbf{key, values})$$

2. <u>Shuffle</u>: analysis of the map's stage data and sorting into a predetermined output(s).

$$\mathbf{shuffle\ operation} \rightarrow \mathbf{g}[\mathbf{f}(\mathbf{key, values})] \rightarrow \mathbf{yields}\text{: } \mathbf{k} \rightarrow \boldsymbol{k}(\boldsymbol{G}_{\boldsymbol{key\_value\_pair}})$$

3. <u>Reduce</u>: reception of the output results, aggregation and incorporation to a task that compounds of several smaller inputs of data.

A simple mathematical explanation regarding a Map-Reduce operation is:

- Step 1: abstract data as key value pairs[1] to a map function as follows:

$$(\boldsymbol{key}, \boldsymbol{value}) = (\boldsymbol{k_A}, \boldsymbol{V_A}) = (\boldsymbol{k_{A1}}, \boldsymbol{V_{A1}})(\boldsymbol{k_{A2}}, \boldsymbol{V_{A2}}) \ldots$$

$$(\boldsymbol{key}, \boldsymbol{value}) = (\boldsymbol{k_B}, \boldsymbol{V_B}) = (\boldsymbol{k_{B1}}, \boldsymbol{V_{B1}})(\boldsymbol{k_{B2}}, \boldsymbol{V_{B2}}) \ldots$$

.

.

.

$$(\boldsymbol{key}, \boldsymbol{value}) = (\boldsymbol{k_N}, \boldsymbol{V_N}) = (\boldsymbol{k_{N1}}, \boldsymbol{V_{N1}})(\boldsymbol{k_{N2}}, \boldsymbol{V_{N2}}) \ldots$$

- Step 2: sort/group the function output as follows:

$$(\boldsymbol{k_{A1}}, \boldsymbol{V_{A1}})(\boldsymbol{k_{A2}}, \boldsymbol{V_{A2}}) \ldots \rightarrow \boldsymbol{k}(\boldsymbol{V_A}, \boldsymbol{V_B}) = \boldsymbol{k_N V_N}$$

- Step 3: reduce a set of data points with the same key pairs into a new single value.

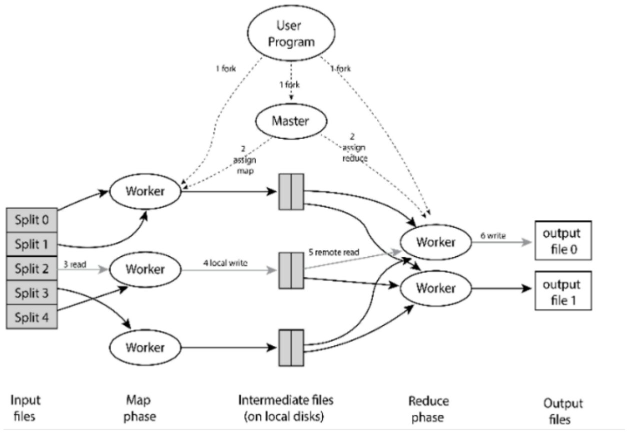

Fig. (1). Map-Reduce execution overview.

Furthermore, regarding steps 2 and 3, i.e., where processes are grouped and divided into multiple partitions (e.g., nodes, mini-computers, clusters) [96], the mathematical equations for the frequencies of the keys are the following:

$$\boldsymbol{Data}_{\mathrm{Locality_{min}}} = \frac{\sum_{n=1}^{\kappa} \boldsymbol{Min}_{\boldsymbol{Frequency}(\boldsymbol{k_N})}}{\sum_{n=1}^{\kappa} \mathbf{FK}_j^n} = \frac{\sum_{n=1}^{\kappa} \mathbf{min}_{1\le j\le n} \mathbf{FK}_n^j}{\sum_{n=1}^{\kappa} \mathbf{FK}_j^n}$$

$$\boldsymbol{Data}_{\mathrm{Locality_{max}}} = \frac{\sum_{n=1}^{\kappa} \boldsymbol{Max}_{\boldsymbol{Frequency}(\boldsymbol{k_N})}}{\sum_{n=1}^{\kappa} \mathbf{FK}_j^n} = \frac{\sum_{n=1}^{\kappa} \mathbf{max}_{1\le j\le n} \mathbf{FK}_n^j}{\sum_{n=1}^{\kappa} \mathbf{FK}_j^n}$$

Lastly, regarding the complexity of Map-Reduce Algorithm, this topic is hard to define, as it depends heavily on many factors, including the hardware, the network, the variation of the algorithm's execution, in addition to the total size, time and memory allocation of the proposed system.

[1] The key value pairs are immutable objects, meaning their value cannot change in any point.

C. Case Study: Hatzidaki's House

We implemented our application in a preserved building in the city of Xanthi, Greece, as presented in Figure 2. The building was constructed in 1829, it has a Baroque-style with neo-classical elements and it is consisted of three floors, covering 1.317 sq.m. (see details in Figure 3) [97]. Specifically, the building was the residence of M. Hatzidakis, one of the most prominent Greek music composers. Due to the status of M. Hatzidakis, his residence was converted to a cultural centre after his death where various exhibitions (music, theatre, etc) are hosted.

Our tests were applied on the first floor of the building which hosts the majority of events. Moreover, the design principles of the IoT middleware suggest a novice, effective and reliable method to monitor visitors' number and location. Specifically, after studying both the floor pans presented in Figure 3 and the scheduled events (prior to COVID-19 movement restrictions), we used simulated data to test our application and track its visitors.

Fig. (2). Outside view of M. Hatzidakis' residence.

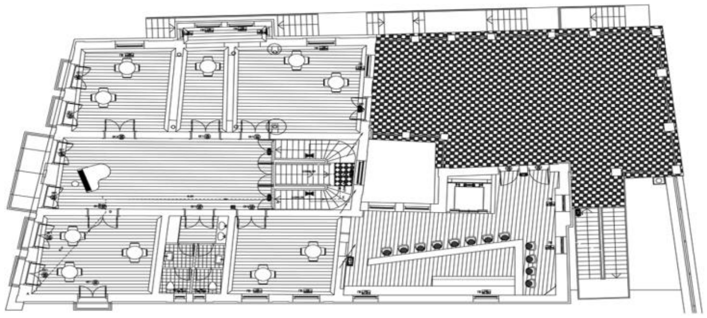

Fig. (3). Floor plan of Hatzidakis' residence for the 1st floor.

D. Description of Map-Reduce use

The algorithm proposed is a variation of the Map-Reduce paradigm to provide a solution on connectivity issues between the master (i.e. server) and the workers (i.e. clients). The IoT architecture described in the following sections provides not only safety and continuation of operation, but also high availability times for the systems. Our application aims at visitor monitoring and tracking in indoor structures of the historical building of our study (Figure 2). Specifically, our rationale is as follows: when visitors buy a ticket to participate in an event, they will be provided with a radio frequency identification (RFID) tag, based on their sex. Subsequently, each room will have several low cost and power devices that will gather information from said tags. These devices will both store data locally and remotely, the as well as perform the Map-Reduce algorithm and provide useful insights regarding each room and visitors' interests.

The Map-Reduce algorithm developed was tested on different operating systems (Unix/Windows) and in a broad spectrum of computer capabilities. Focus was placed on the Raspberry Pi, which was the main component used to process the available information. Specifically, it is a mini-computer device used extensively in academia and real-life situations to teach programming. In recent years, it is also used for robotics and IoT applications. The algorithm proposed was successfully implemented on a wireless computer network consisting of several affordable mini computers, such as Raspberry Pi models 3, 3B, and 4.

The system proposed offered the following operations and subservices: the workers and the master were message-driven and were simulated with finite-state machines that moved from one state to the other (Check Status, Check Node Connectivity, Check Leader, etc.). The communication between master workers (server-client) was performed via UDP messages, and the role exchange part was implemented via a remote database storing the computer devices' unique ids and network properties (e.g., IP addresses). Moreover, to overcome Hadoop constraint of check pointing in case of failure, we have also developed a simple algorithm that elected a leader during each software cycle, based on the connectivity status of the network's node. Specifically, each server must be able to connect with at least 60% of the available systems nodes in order to retain its status as a server; otherwise, another device (client) is elected as leader (server). Lastly, we achieved similar results to other word count applications in recent bibliography such as [98] by implementing Map-Reduce middleware to low-power and low-cost computer components. Analytically, the data complexity for a dataset of S size, for N documents, M words and $f_1,\ldots f_M$ word frequencies is measured as:

$$Complexity_{Key} = O(max_n f_n) \text{ , } \mathrm{Complexity}_{\mathrm{Sequential}} = O(S)$$

V. SYSTEM AND ARCHITECTURE

The first step in building our system was to achieve a stable and continuous connection of each available computer device to our network. To achieve this, we focused solely on remote wireless solutions. Concerning the successful connection of multiple computer devices to a local network, each device must identify the IP/Physical (MAC) address of each available device on the network beforehand. The information required for remote access can be acquired easily through either terminal emulators or remote desktop software.

During our first test, we used virtual network computing (VNC) as it was straightforward and provided a simple and quick way to connect remotely to each of the available PCs with a graphical interface. Official documentation of Raspberry Pi —one of the most known series of small single-board computers—suggests the use of VNC [99]. Our test cases consisted of microcomputers, such as Raspberry Pi Model 3b-4-4a and portable computers running Unix operating systems. More specifically, Raspberry Pis had either Raspbian (the official Raspberry Pi OS) or New Out Of The Box (NOOBS) installed and various hardware portable computers that were dual booted running Windows 10 / Ubuntu 18.4 LTS.

Subsequently, we wanted to reduce the central processing unit (CPU), the random-access memory (RAM), and generally benchmark of our application. For this purpose, we switched from a GUI based solution to a terminal-based application using secure shell (SSH). The benefit of this choice was that we could communicate remotely with each PC by typing its credentials and IP address, thus providing a system-agnostic solution regarding the operating system.

Initially, we implemented static IP addresses on the computer systems so that their addresses do not change each time they connect to the local network. During the first step, we used the Address Resolution Protocol (ARP) command of Unix system command-line tools to find IP / MAC addresses. More specifically, ARP has a list (cache) which is used to acquire IP-MAC address as well as the type of connection entries (dynamic/static). This solution was not ideal because it was not optimal. For example, if we logged in another network or provided Internet access via another router, many settings changed.

In the final stage of our experiments, we decided to shift our focus to an approach that would ping devices connected to our network to find the IP addresses of our system's computers. To achieve that, we decided to use Java's Socket API application— protocol that provides full-duplex, bi-directional, real-time client/server efficient communication between two peers over a computer network protocol.

A. Local Test Implementation

We locally developed a multithreaded application to test and construct all necessary components of our system piece by piece. Our application aims to connect several devices to the same wireless network. More specifically, the application requested the following:

- user's input regarding the number of clients that would be connected and
- IP and Port number for the server's first execution.

The rationale of this system was to start a server capable of accepting many client connections simultaneously based on the user's input. The user would provide all necessary input via the terminal, such as socket's information (IP and Port), the number of clients to be connected, and general settings (e.g., generating a Test Data Set for sampling or using an existing one). A simple execution of this program is the following:

Step 1: The user starts the server. Accordingly, we execute the Client class to create our first connection.

Step 2: The user types information regarding the system, such as IP or Port, using an existing dataset or defining the size of a random data set file generation.

Step 3: The terminal waits for the user to type the number of clients to be created (threads) until the user stops the execution. The IP and port information are not requested as the application's server is hosted locally.

The main issue with the above-mentioned application was that most of the server properties, e.g., IP, port numbers, etc. were "hard-coded." This term is used when there was no validation check. We provided fixed (pre-set) parameters to most of our programs during their execution. Additionally, even though in principle the sockets were created correctly and the connection of the server client was stable, we had yet to connect several clients in real-time conditions.

B. Step 1: Server-Client Properties

As presented, step 1 has several "hard-coded" parameters, thus making it difficult to develop a remote application. To facilitate the progress regarding the necessary properties requested in Step 1, we executed remotely (via SSH) one of the following network commands (ARP, ifconfig, ipconfig) to find the unique IP address needed for execution.

Our middleware automatically gathers this information. The users need to connect it to the local Wi-Fi or a cellular network to communicate with the remote database. After booting, the Raspberry Pi inserts its network properties and a random node ID number is assigned. During the first execution, the master server is selected randomly and other nodes are assigned lower id numbers. The system selects a node's id to act as a master and checks the server's port availability. The port is constant and does not change throughout system's execution. Regarding this decision, for a client to ping all available devices on our network, it must check port numbers 0 to 65535, which is a slow process, i.e., more than 15 minutes which is the average time for a guided tour in our case study. As a result, this decision was made to ameliorate the execution time.

Afterwards, during execution, if several nodes have the same id, the master-server node retains its status. Connectivity is checked periodically for clients and servers alike. If a node cannot be connected to at least two-thirds of the other nodes in our system (the total number derives from DB's data), then it cannot be considered as a master node. In case no node meets the connectivity criterion, the system halts its execution by design.

C. Step 2: Role Exchanging Parameters

After establishing the unique address of each computer connected to our network, the next step regarding our application's flow was to launch the application and handle the role exchanging between server-client. Initially, the node ID number was randomly selected, but later on we correlated it with a connectivity parameter. Specifically, the master node checks the connectivity with other devices to address the node ID number. We decided to correlate each IP address with a specific node number that would be calculated based on each node's id connectivity in accordance with the overall system. During the first execution of the application, this number would be randomly assigned or the user could select which computer device will act as a server.

The main issue we came across was how we would configure the application to send the paired data (IP-Port). The first idea was to use a text file (e.g., JSON, XML, and YAML) which would contain all necessary information that would be parsed dur-ing each flow. The text files are generally small-sized (less than 100 KB), thus they require neither high bandwidth nor high volume storage. Additionally, their size means that we could store the file on each device as well as iterate during each application rerun (endless mode) to update the file. This idea was implemented during our beta prototype, but we decided to store this information in a database for various reasons [100] (mainly due to the speed of execution and future scalability).

During each execution, our system has two classes: one for the server (master) and one for workers (clients). After the initial connection, which defines a server node, the system stores all computer device information (IP-node ID) locally and it checks their connectivity. If the server is not fit for operation due to the connectivity, it connects to DB and sets its node ID to zero.

D. Step 3: Process and Functionalities

The proposed system consists of several processes that are distributed evenly across the wireless network we presented earlier. First, since the master is the single point of failure in the Hadoop architecture, we have provided fault tolerance capabilities to our system, as presented in Figure 4. The following generic operations can be implemented by any computer connected to our network:

1. Check Workers: The server pings all clients of our application to validate whether it communicates with at least two-thirds of the total nodes.
2. *Election*: If a client satisfies the Check Worker condition, the system checks the client node ID.
3. *Notify Leadership*: If the election status is true, then it notifies clients regarding the new leader candidate.
4. *Check Leadership*: If the election status is false, it checks if the current leader is alive. If not, it notifies clients regarding the new leader candidate.
5. *Check Alone*: if a new Server is isolated, it re-executes the parallel WSN's steps.

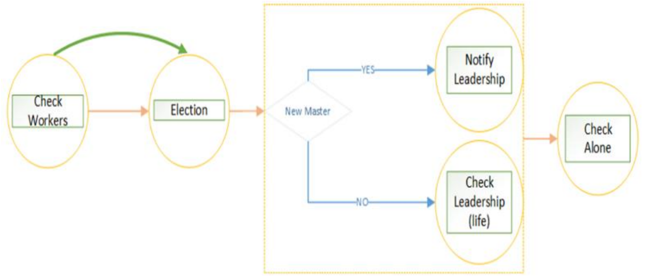

Fig. (4). Fault tolerance capabilities of the application.

E. Step 4: Endless Execution

After Step 3, a number is assigned to each PC and we set up a timeframe for each node to elect a leader server. More specifically, we set a time duration of 1 to 2 minutes following Step 3's execution for our system to communicate with the database and, based on the algorithm output number, check the status of our system.

VI. DATABASE PROPERTIES & JAVA CLASSES USED

In this subsection, we provide information regarding Step 3 of the previous algorithm, i.e., the processes and functionalities of the proposed system. First, we present the database used to store the client-server information and the sensors' measurements. Second, we provide a diagram flow and explain the logic behind each of the classes used to develop our system. Additionally, a sequence diagram, a UML diagram, and a class diagram are presented.

A. *Database Information*

The remote database selected was MySQL due to its use in academia and industry. Moreover, during the early stages of the system's design, we used Xampp, i.e., an open-source cross-platform providing the necessary components to host an online database. Xampp is a software distribution that takes an "all battery included" approach, providing the Apache web server, MySQL database (MariaDB) and Php/Perl/Python (as command-line executables and Apache modules). However, this solution became obsolete the moment we tested our system outside our local area network. Initially, we opted for this solution to quickly test our progress regarding the middleware components. Specifically, we used the local area network to host a local network to test the early stages of our application. Although this allows users to immediately review the necessary time for computers to connect to the DB, as well as the query performance and execution, this solution had its disadvantages. Analytically, in terms of scaling and evaluating the performance of our DB, we were restricted due to the hardware capabilities of the server computer. Moreover, since we were testing our application to a local area network, we were not able to monitor potential issues in the actual network's bandwidth. Lastly, due to the small size network, we were unable to accurately measure the execution times from committing an SQL query until it transferred its output to the actual DB of the cloud provider that would eventually host our remote DB.

Consequently, to develop a cloud solution that would work in all networks, we decided to implement our database remotely. There were several cloud providers available but—since the aim of this article was not to benchmark the database responsiveness—we decided to use a free, open-source database provider. Therefore, we used the latest version of the MySQL Server. The database properties included a MySQL 8.0 server running on the default port 3306 and we connected to the DB remotely using JDBC. Additionally, to successfully compile and execute the Java project developed, we added the MySQL Connector Jar file as provided in Maven's repository.

Our DB consists of two databases: one stores computer device properties (IP address and node IDs) and the other stores samples (sensors' data) used to monitor the visitors of the preserved building. The second table consists of 3 fields:

I. *Room* indicating where the computer device was placed. Our tests were based on the 1st floor of Hatzidakis' residence; thus, only 5 rooms were simulated.
II. *Visitor*, who can be a man, a woman, or other species generated via an RFID tag provided upon entrance.
III. *Timestamp* providing the exact date and time for each measurement.

B. *Java Classes and Functionality*

In this subsection, we provide a brief description of the Java classes used in our system which are:

- *Database Functionality Server Client Db*: it connects to the computer devices' property db and it performs select or insert or update operations.
- *Database Functionality Sensor Properties Db*: it connects to the sensor measurements db and it can perform, select or insert operations.
- *Get IP for Remote Connection*: it finds the local IP of each machine, pings the other IP address, and checks the connectivity, i.e., if they are online (or not).
- *Sensor Data Parser*: it can generate random data (in a specific mode) and parse data from an input file/data stream.
- *Map-Reduce*: it was implemented to count the visitors of the museum. The algorithm was initially developed using structural programming, i.e., arrays and logic. Afterwards, we selected to store the room–visitor count with a hashmap to use built-in Java functionality.
- *Connect With Cloud Database*: it uses JDBC to connect to the remote database's tables.
- *Client*: it takes an input data stream (e.g., a text file or values from the database) generated for each room and executes the Map-Reduce algorithm (either the initial version using arrays or HashMap) to count the number of visitors. Additionally, after map-shuffle-reduce operation, it opens a socket and sends a message to the server via a UDP packet.
- *Server*: it acts as a server, i.e., it receives sockets from clients while similarly executing the Map-Reduce algorithm locally for its own input data stream. We used datagram sockets to send responses and receive UDP packets from clients.
- *System Information*: it is a generic implementation of our overall system. Specifically, it connects to the database via JDBC, gets the local IP address and checks the client-server table for the specific local address. If this address is not present, it is added in the table. When it exists, it checks the node ID and if said id is greater than the current value, it updates it. Afterwards, it checks connectivity of the system (i.e., it pings other addresses provided by the DB). Lastly, if it is connected to more than two-thirds of the system's nodes (i.e., it pings the IP address from DB and receives a response), it executes client or

server class based on the node ID.

VII. RESULTS

Our proposed system inputs the data stream provided from the RFID tags by implementing the Map-Reduce paradigm. Specifically, the output results are twofold, as presented in Figures 5 and 6. On the one hand, it counts all visitors of the residence by identity (man, woman, and other, e.g., dog). On the other hand, it counts the total number of visitors in each room. This means that, depending on the need, this system can be used as a means to count the total number of visitors in a COVID-19 case to track the virus spread or else, if an infected person was located in a specific location -room(s)-, it can be used to track down his/her whereabouts and the number of visitors s/he came in contact with.

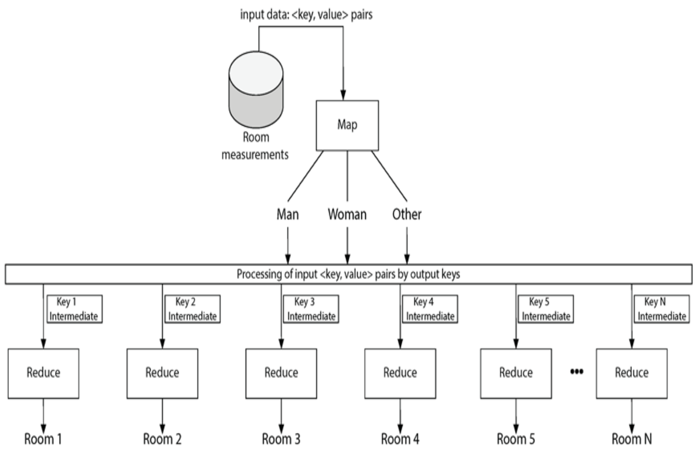

Fig. (5). Flaw diagram of our proposed middleware's operating principles.

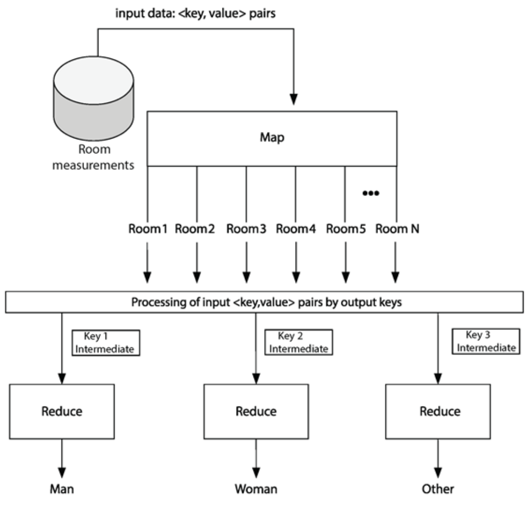

Fig. (6). Map-Reduce algorithm for case 2: room monitoring (room 1 to N).

Additionally, to test this scenario, we have selected to use the preserved building analyzed in the previous section, i.e., M. Hatzidakis' residence. Analytically, we have chosen to use low-cost and low-power devices, such as Raspberry Pi model 3 and 4 consisting of dual and quad-core CPUs and approximately 1 GB of RAM. Our rationale was to place one device in each room (excluding the WC) thus measuring the passers' activities on the floor. Furthermore, after studying Hatzidakis' residence (case study), we proposed a sampling rate of a minimum of 17–20 minutes since 2–5 minutes were adequate for end-to-end communication. As such, for this timeframe, if we consider a guided tour of 4000 persons, the execution time of our system is presented in Figure 7. The results of our test showcased that our system is capable of handling up to 1500-2000 visitors before the execution time begins to rapidly increase. Moreover, Table 1 shows how an input stream of visitors' RFID tags will be processed, and Table 2 and 3 provided information regarding the power consumption and the general benchmarks of our proposed middleware on a Raspberry Pi model 3b machine.

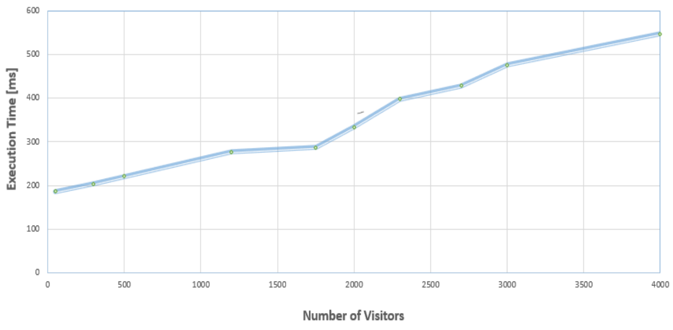

Fig. (7) Map-Reduce algorithm for room tracking a set of visitors.

Table 1. Map-Reduce algorithm for room monitoring and visitor counting application.

```
$ Total Count: <Room1:100, Room2:37,
Room3:108, Room4:131, Room5:50 >

1: < Man: 31, Woman: 68, Other: 1 >
2: < Man: 11, Woman: 26, Other: 0 >
3: < Man: 43, Woman: 64, Other: 1 >
4: < Man: 53, Woman: 78, Other: 0 >
5: < Man: 44, Woman: 4,   Other: 2 >
```

Table 2. Server-Client communication benchmarks.

		Server	Client
CPU	[%]	86	53
Memory	[MB]	212,6	198,4
Power Consumption	[A]	0,51	
	[V]	5,34	

Table 3. Benchmarks for an entire software cycle.

Visitors	10	50	100	250	500
CPU [%]	40,12	44,6	48,20	57,83	71,3
Memory [MB]	810	830	850	874	970

Power Consumption [A]	0,80	0,80	0,81	0,81	0,83

Lastly, the execution times of our novice algorithm are illustrated in Figure 8 where, as evident, the cloud tier of our application illustrates the time between client-server requests.

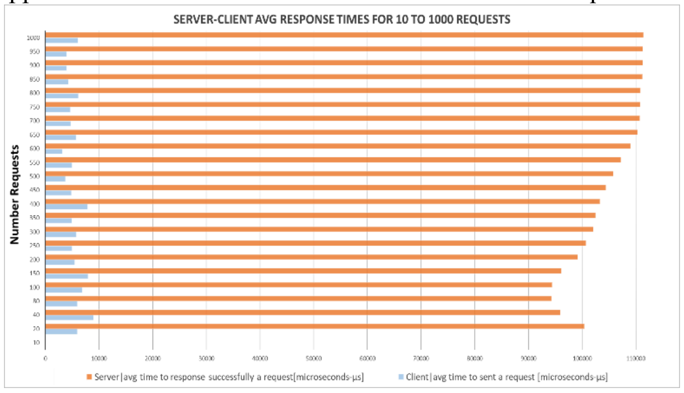

Fig. (8). Average time (in μsecs) for server-client response for 10 to 1000 requests.

VIII. CONLUSIONS

In this publication we have presented an IoT cloud computing middleware application that uses WSN data to monitor the crowd in indoor structures. The application of our system can be both for tracking the visitor count and the crowd's interest on a specific room/location/exhibition and most importantly as a tool to track visitors and minimize a virus' spread (e.g. COVID-19). As evident from above, the output results of our system consisting of several low power, size and cost devices such as Raspberry Pi provide a cloud-based service system that is reliable, even during large number of visitors and connectivity loads (e.g. 1500 requests). Moreover, it is notable that even though most of our experiments consisted of Raspberry Pi version 4, the same response rate and availability was achieved with version 3B, i.e. with less recent hardware devices. This is important as it reduces the overall cost of the system and provides ample opportunity of easy scaling if needed in the future.

Initially, we were not certain whether setting up Raspberry Pis was the optimal choice, since these devices are not manufactured to be continuously executed [101]. During our tests, we used these electronic devices as a headless server/client and we concluded that the endless execution mode is achieved by default. This is the case due to the lack of power management settings which prevents the machine from switching to hault (sleep mode).

Additionally, we selected the Java programming language to develop our application, as it is the de facto enterprise language of cross-platform systems. Although we only included Raspberry Pi using Unix systems in our tests, we believe that similar behaviour can be achieved in Windows-Unix or Windows-Windows operating systems. Finally, similarly to most applications executing Map-Reduce algorithms, our proposed application had high availability and short downtime between operations.

In future steps, we will focus on expanding the parameters and setting the leader election algorithms. Specifically, our work does not currently consider every parameter to elect a server leader. During our initial execution, an available computer is elected and it remains the head computer, based on the connectivity of other nodes. The role exchanging part (client-server) must be further developed to take network speeds, node availability, power consumption of devices, availability of devices, etc. under consideration. The role exchanging part should also provide the opportunity for a client for "bully election", i.e., to stop our proposed algorithm and state that they can be "fit for server". Moreover, an interesting issue is implementing a variation of the Byzantine fault tolerance algorithm in machine-to-machine communication.

Lastly, if a leader election algorithm is implemented, we could use modern AI techniques, like Tiny ML to merge Map-Reduce operations together with the leader election and create new suggestions for a leader server, based on other crucial factors of real case scenarios.

ACKNOWLEDGMENT

This article details our recent work, as part of an MSc course on distributed systems, at the School of Engineering Department of Democritus University of Thrace, Greece to apply the parallel programming to WSNs. Lastly, the authors are grateful to the civil engineer E. Chamalidou for her assistance in the analysis and the presentation of the floor plans of M. Hatzidakis' residence, as presented in this publication

References

[1] Ghosh, J. (2019) '5G Services and IoT Challenges', *International Journal of Sensors, Wireless Communications and Control*, Vol.*9* No.*4*, pp.417-418. https://doi.org/10.2174/221032790904190917101935

[2] Mansour, K., Saeed, A. (2019) 'Implementation of a Low-power Embedded Processor for IoT Applications and Wearables', *International Journal of Circuits, Systems and Signal Processing*, Vol.*13*, pp. 625-636. https://www.naun.org/main/NAUN/circuitssystemssignal/2019/b722005-ald.pdf

[3] Quaratulain, Q., Basit, I., Bakhsh, K., Hafeez, M. (2021) 'Adult Learning Theories and their Role in Instructional Design, Curriculum Development and Educational Technology', *Wseas Transactions on Environment and Development*, Vol. *17,* p.1149-1159. www.doi.org/10.37394/232015.2021.17.106

[4] Gazis, A., Katsiri, E. (2020) 'A wireless sensor network for underground passages: Remote sensing and wildlife monitoring', *Engineering Reports*, Vol.*2*, No.*6*, p.e12170. https://doi.org/10.1002/eng2.12170

[5] Prauzek, M., Konecny, J., Borova, M., Janosova, K., Hlavica, J., Musilek, P. (2018) 'Energy harvesting sources storage devices and system topologies for environmental wireless sensor networks: A review', *Sensors*, Vol. *18*, No.*8*, p.2446. https://doi.org/10.3390/s18082446

[6] Wu, B., Yao B., Yang, Y., Zhou, C., Zhu, N. (2021) 'Interference Suppression and Resource Allocation Strategies Based on IoT Monitoring', *International Journal of Circuits, Systems and Signal Processing*, Vol.15, pp. 1005-1014. https://doi.org/10.46300/9106.2021.15.108

[7] Gazis, A., Gazi, T. (2021) 'Big data applications in industry fields', *OUP ITNOW*, Vol.*63*, No.*2*, pp.50-51. https://doi.org/10.1093/itnow/bwab056

[8] Turcu, C., Turcu, C., Gaitan, V. (2012) 'Integrating robots into the Internet of Things', *International Journal of Circuits, Systems and Signal Processing*, Vol.*6(6)*, pp.430-437. http://www.naun.org/main/NAUN/circuitssystemssignal/16-658.pdf

[9] Gazis, A., Katsiri, E. (2021) 'Smart Home IoT Sensors: Principles and Applications - A Review of Low-Cost and Low-Power Solutions', *International Journal on Engineering Technologies and Informatics*, Vol.*2(1)*, pp.19-23. https://doi.org/10.51626/ijeti.2021.02.00007

[10] Mulligan, G. (2007) 'The 6LoWPAN architecture', *Proceedings of the 4th workshop on Embedded networked sensors*, pp.78-82. https://doi.org/10.1145/1278972.1278992

[11] Ordabayeva, G.K., Othman, M., Kirgizbayeva, B., Iztaev, Z.D., Bayegizova, A. (2020) 'A Systematic Review of Transition from IPV4 To IPV6', *Proceedings of the International Conference on Engineering & MIS*, pp.1-15. https://doi.org/10.1145/3410352.3410735

[12] Haxhibeqiri, J., De Poorter, E., Moerman, I., Hoebeke, J. (2018) 'A survey of LoRaWAN for IoT: From technology to application', *Sensor*, Vol.*18*, No.*11*, p.3995. https://doi.org/10.3390/s18113995

[13] Sinha, R.S., Wei, Y., Hwang, SH. (2017) 'A survey on LPWA technology: LoRa and NB-IoT', *ICT Express*, Vol.*3*, No.1, pp.14-21. https://doi.org/10.1016/j.icte.2017.03.004

[14] Lin, J., Dyer, C. (2010) 'Data-intensive text processing with MapReduce', *Synthesis Lectures on Human Language Technologies*, Vol.*3*, No.*1*, pp.1-77. https://doi.org/10.2200/S00274ED1V01Y201006HLT007

[15] Clarke, L., Glendinning, I., Hempel, R. (1994) 'The MPI message passing interface standard', *Programming environments for massively parallel distributed systems*, pp.213-218. https://doi.org/10.1007/978-3-0348-8534-8_21

[16] Maleki, N., Rahmani, A.M., Conti, M. (2019) 'MapReduce: an infrastructure review and research insights', *The Journal of Supercomputing*, Vol.*75*, No.*10*, pp.6934-7002. https://doi.org/10.1007/s11227-019-02907-5

[17] Apache Hadoop | Github. Available online: https://github.com/apache/hadoop (accessed on 22/12/2021)

[18] Apache Spark | Github. Available online: https://github.com/apache/spark (accessed on 22/12/2021)

[19] Hazarika, A.V., Ram, G.J., Jain, E. (2017) 'Performance comparison of Hadoop and spark engine', *Proceedings of the International Conference on I-SMAC*, pp.671-674. https://doi.org/10.1109/I-SMAC.2017.8058263

[20] Samadi, Y., Zbakh, M., Tadonki, C. (2018) 'Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks', *Concurrency and Computation: Practice and Experience*, Vol.*30*, No.*12*, p.e4367. https://doi.org/10.1002/cpe.4367

[21] Makrani, H.M., Homayoun, H. (2017) 'Memory requirements of hadoop spark and MPI based big data applications on commodity server class architectures', *Proceedings of the International Symposium on Workload Characterization*, pp.112-113. https://doi.org/10.1109/IISWC.2017.8167763

[22] Mostafaeipour, A., Jahangard-Rafsanjani, A., Ahmadi, M., Arockia-Dhanraj, J. (2020) 'Investigating the performance of Hadoop and Spark platforms on machine learning algorithms', *The Journal of Supercomputing*, Vol.*77*, pp.1273–1300. https://doi.org/10.1007/s11227-020-03328-5

[23] Mavridis, I., Karatza, H. (2017) 'Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark', *Journal of Systems and Software*, Vol.*125*, pp.133-151. https://doi.org/10.1016/j.jss.2016.11.037

[24] Glushkova, D., Jovanovic, P., Abelló, A. (2019) 'Mapreduce performance model for Hadoop 2.x', *Information systems*, Vol. *79*, pp.32-43. https://doi.org/10.1016/j.is.2017.11.006

[25] Verma, A., Mansuri, A.H., Jain, N. (2016) 'Big data management processing with Hadoop MapReduce and spark technology: A comparison', *Proceedings of the Symposium on Colossal Data Analysis and Networking (CDAN)*, pp.1-4. https://doi.org/10.1109/CDAN.2016.7570891

[26] Aziz, K., Zaidouni, D., Bellafkih, M. (2018) 'Real-time data analysis using Spark and Hadoop', *Proceedings of the International Conference on Optimization and Applications (ICOA)*, pp.1-6. https://doi.org/10.1109/ICOA.2018.8370593

[27] Memishi, B., Ibrahim, S., Pérez, M.S., Antoniu, G. (2016) 'Fault tolerance in MAPREDUCE: A survey' *Resource Management for Big Data Platforms*, pp.205-240. https://doi.org/10.1007/978-3-319-44881-7_11

[28] Zhou, M., Dong, H., Ioannou, P.A., Zhao, Y., Wang, F.Y. (2019) 'Guided crowd evacuation: approaches and challenges', *IEEE/CAA Journal of Automatica Sinica*, Vol.3, No.*6(5)*, pp.1081-1094. https://doi.org/10.1109/JAS.2019.1911672

[29] Ding, X., He, F., Lin, Z., Wang, Y., Guo, H., Huang, Y. (2020) 'Crowd density estimation using fusion of multi-layer features', *IEEE Transactions on Intelligent Transportation Systems*, pp.66-71. https://doi.org/10.1109/TITS.2020.2983475

[30] Mishra, L., Varma, S. (2021) 'Middleware Technologies for Smart Wireless Sensor Networks towards Internet of Things: A Comparative Review', *Wireless Personal Communications*, Vol.*116*, No.*3*, pp.1539-1574. https://doi.org/10.1007/s11277-020-07748-7

[31] Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S. (2016) 'Middleware for internet of things: a survey', *IEEE Internet Things Journal*, Vol.*3*, pp.70–95. https://doi.org/10.1109/JIOT.2015.2498900

[32] Pradeep, P., Krishnamoorthy, S., Vasilakos, A.V. (2021) 'A holistic approach to a context-aware IoT ecosystem with Adaptive Ubiquitous Middleware', *Pervasive and Mobile Computing*, Vol.*72*, pp.101342. https://doi.org/10.1016/j.pmcj.2021.101342

[33] Pereira, D.M., Silva, F.J., Salles, C.S.N., Santos, D.V., Coutinho, L.R., Guedes, A.L. (2021) 'An ontology-based approach to integrate TV and IoT middlewares', *Multimedia Tools and Applications*, Vol.*80*, No.*2*,

pp.1813-1837. https://doi.org/10.1007/s11042-020-09645-4

[34] Park J.H. (2020) 'Intelligent Service and Metadata Management for Smart IoT Middleware', *International Journal of Sensors, Wireless Communications and Control*, Vol.*10*, No.*5*, pp.763-771. https://doi.org/10.2174/2210327910999200624123333

[35] Gankevich, I., Gaiduchok, V., Korkhov, V., Degtyarev, A., Bogdanov, A. (2017) 'Middleware for big data processing: test results', *Physics of Particles and Nuclei Letters*, Vol.*14*, No.*7*, pp.1001-1007. https://doi.org/10.1134/S1547477117070068

[36] Carrega, A., Repetto, M., Gouvas, P., Zafeiropoulos, A. (2017) 'A middleware for mobile edge computing', *IEEE Cloud Computing*, Vol.*4*, No.*4*, pp. 26-37. https://doi.org/10.1109/MCC.2017.3791021

[37] Aazam, M., Huh, E.N. (2016) 'Fog computing: The cloud-iotVioe middleware paradigm', *IEEE Potentials*, Vol.*35*, No.*3*, pp.40-44. https://doi.org/10.1109/MPOT.2015.2456213

[38] Xu, H., Dharmendra, S. (2007) 'Effective Middleware for Efficient XML Data Transmissions on Networks', *International Journal of Circuits, Systems and Signal Processing*, Vol.*1*, pp. 189-193. https://www.naun.org/main/NAUN/circuitssystemssignal/cssp-33.pdf

[39] Kohani, S., Zong, P., Yang, F. (2021) 'Design Coverage Optimization Based on Position of Constellations and Cost of the Launch Vehicle', *Wseas Transactions on Environment and Development*, Vol.*17*, pp. 1160-1190. www.doi.org/10.37394/232015.2021.17.107

[40] Yang, X., Hou, Y., He, H. (2019) 'A processing-in-memory architecture programming paradigm for wireless internet-of-things applications', *Sensors*, Vol.*19*, No.*1*, p.140. https://doi.org/10.3390/s19010140

[41] Caballero, V., Valbuena, S., Vernet, D., Zaballos, A. (2019) 'Ontology-Defined middleware for internet of things architectures', *Sensors*, Vol.*19, No.5,* p.1163. https://doi.org/10.3390/s19051163

[42] Temdee, P., Prasad, R. (2018) 'Context-Aware Middleware and Applications', *Context-Aware Communication and Computing, Applications for Smart Environment*, pp.127-148. https://doi.org/10.1007/978-3-319-59035-6_6

[43] Lanza, J., Sánchez, L., Gómez, D., Santana, J.R., Sotres, P. (2019) 'A Semantic-Enabled Platform for Realizing an Interoperable Web of Things', *Sensors*, Vol.*19*, No.*4*, p.869. https://doi.org/10.3390/s19040869

[44] Mesmoudi, Y., Lamnaour, M., El Khamlichi, Y., Tahiri, A., Touhafi, A., Braeken, A. (2018) 'A middleware based on service oriented architecture for heterogeneity issues within the internet of things (MSOAH-IoT)', *Journal of King Saud University-Computer and Information Sciences*, Vol.*32*, No.*10*, pp.1108-1116. https://doi.org/10.1016/j.jksuci.2018.11.011

[45] Li, Z., Seco, D., Sánchez Rodríguez, A.E. (2019) 'Microservice-oriented platform for internet of big data analytics: A proof of concept', *Sensors*, Vol.*19*, No.*5*, p.1134. https://doi.org/10.3390/s19051134

[46] Asif, S., Webb, P. (2015) 'Software system integration-Middleware-an overview', *Foundation of Computer Science*, Vol.*121*, No.*5*, pp.27-29. http://dx.doi.org/10.5120/21538-4547

[47] Bansal, S., Kumar, D. (2020) 'IoT ecosystem: A survey on devices gateways operating systems middleware and communication', *International Journal of Wireless Information Networks,* Vol.*27,* pp.340-364. https://doi.org/10.1007/s10776-020-00483-7

[48] Kelly, S.D., Suryadevara, N.K., Mukhopadhyay, S.C. (2013) 'Towards the implementation of IoT for environmental condition monitoring in homes', *IEEE sensors*, Vol.*13*, No.*10,* pp.3846-3853. https://doi.org/10.1109/JSEN.2013.2263379

[49] Aguilar, J., Jerez, M., Mendonça, M., Sánchez, M. (2020) 'Performance analysis of the ubiquitous and emergent properties of an autonomic reflective middleware for smart cities', *Computing,* Vol.*102*, No.*10*, pp.2199-2228. https://doi.org/10.1007/s00607-020-00799-5

[50] Vujović, V., Maksimović, M. (2015) 'Raspberry Pi as a Sensor Web node for home automation', *Computers & Electrical Engineering*, Vol.*44,* pp.153-171. https://doi.org/10.1016/j.compeleceng.2015.01.019

[51] Zahra, S.R., Chishti, M.A. (2020) 'A collaborative edge-cloud internet of things based framework for securing the indian healthcare system', *International Journal of Sensors, Wireless Communications and Control*, Vol.*10*, No.*4*, pp.440-457. https://doi.org/10.2174/2210327910666191218144157

[52] Ji, Z., Ganchev, I., O'Droma, M., Zhao, L., Zhang, X. (2014) 'A cloud-based car parking middleware for IoT-based smart cities: Design and implementation', *Sensors*, Vol.*14*, No.*12,* pp.22372-22393. https://doi.org/10.3390/s141222372

[53] Babu, K.R., Remesh, Prathap Vishnu, M., Samuel, P. (2019) 'Context aware reliable sensor selection in IoT', *International Journal of Intelligent Systems Technologies and Applications*, Vol.18, No.1-2, pp.34-51. https://dx.doi.org/10.1504/IJISTA.2019.097746

[54] Symeonaki, E., Arvanitis, K., Piromalis, D. (2020) 'A context-aware middleware cloud approach for integrating precision farming facilities into the IoT toward agriculture 4.0', *Applied Sciences,* Vol.*10, No.3,* p.813. https://doi.org/10.3390/app10030813

[55] Yin, Y., Zou, C., Sun, J. (2020) 'Robot communication system based on OIO middleware', *Proceedings of the International Conference on Systems Man and Cybernetics*, pp.1937-1942. https://doi.org/10.1109/SMC42975.2020.9283445

[56] Alam, T. (2020) 'Design a blockchain-based middleware layer in the Internet of Things Architecture', *JOIV: International Journal on Informatics Visualization*, Vol.*4*, No.*1*, pp.28-31. http://dx.doi.org/10.30630/joiv.4.1.334

[57] Longo, A. Zappatore, M., De Matteis, A. (2020) 'An osmotic computing infrastructure for urban pollution monitoring', *Software: Practice and Experience, Vol.50*, No.*5*, pp.533-557. https://doi.org/10.1002/spe.2721

[58] Vidaña-Vila, E., Navarro, J., Borda-Fortuny, C., Stowell, D., Alsina-Pagès, R.M. (2020) 'Low-Cost Distributed Acoustic Sensor Network for Real-Time Urban Sound Monitoring', *Electronics,* Vol.*9*, No.*12*, p.2119. https://doi.org/10.3390/electronics9122119

[59] Jiang, K., Wang, Y., Kou, S., Yang, D. (2020) 'A Lightweight and Multi-OS Compatible Middleware Designed for Autonomous Driving', *Proceedings of the International Conference of Transportation*, pp.544-555. https://doi.org/10.1061/9780784483053.047

[60] Mehrotra, A., Pejovic, V., Musolesi, M. (2014) 'SenSocial: a middleware for integrating online social networks and mobile sensing data streams', *Proceedings of the International Middleware Conference,* pp.205-216. https://doi.org/10.1145/2663165.2663331
[61] RabbitMQ | Github. Available online: https://github.com/rabbitmq (accessed on 22/12/2021)
[62] ActiveMQ | Github. Available online: https://github.com/apache/activemq (accessed on 22/12/2021)
[63] ZeroMQ | Github. Available online: https://github.com/zeromq (accessed on 22/12/2021)
[64] Kafka | Github. Available online: https://github.com/apache/kafka (accessed on 22/12/2021)
[65] Ionescu, V.M.. (2015) 'The analysis of the performance of RabbitMQ and ActiveMQ', *RoEduNet Proceedings of the International Conference-Networking in Education and Research*, pp.132-137. https://doi.org/10.1109/RoEduNet.2015.7311982
[66] Dobbelaere, P., Esmaili, K.S. (2017) 'Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper', *Proceedings of the ACM International Conference on Distributed And Event-Based Systems*, pp.227-238. https://doi.org/10.1145/3093742.3093908
[67] Estrada, N., Astudillo, H. (2015) 'Comparing scalability of message queue system: ZeroMQ vs RabbitMQ', *Proceedings of the Latin American Computing Conference*, pp.1-6. https://doi.org/10.1109/CLEI.2015.7360036
[68] Farahzadi, A., Shams, P., Rezazadeh, J., Farahbakhsh, R. (2018) 'Middleware technologies for cloud of things: a survey', *Digital Communications and Networks*, Vol.*4*, No.*3*, pp.176-188. https://doi.org/10.1016/j.dcan.2017.04.005
[69] Watteyne, T., Vilajosana, X., Kerkez, B., Chraim, F., Weekly, K., Wang, Q., Glaser, S., Pister, K. (2012) 'OpenWSN: a standards- based low- power wireless development environment', *Transactions on Emerging Telecommunications Technologies*, Vol.*23*, No.*5*, pp.480-493. https://doi.org/10.1002/ett.2558
[70] Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J.P., Riahi, M., Aberer, K., Jayaraman, P.P., Zaslavsky, A., Žarko, I.P., Skorin-Kapov, L. (2015) 'Openiot: Open source internet-of-things in the cloud', *Interoperability and open-source solutions for the internet of things*, pp.13-25. https://doi.org/10.1007/978-3-319-16546-2_3
[71] Dean, J., Ghemawat, S. 'MapReduce: Simplified data processing on large clusters', Google. https://static.usenix.org/publications/library/proceedings/osdi04/tech/full_papers/dean/dean.pdf
[72] Karun, A.K., Chitharanjan, K. (2013) 'A review on hadoop—HDFS infrastructure extensions', *Proceedings of the Conference on Information & Communication Technologies*, pp.132-137. https://doi.org/10.1109/CICT.2013.6558077
[73] Dean, J., Ghemawat, S. (2008) 'MapReduce: simplified data processing on large clusters', *Communications of the ACM*, Vol.*51*, No.*1*, pp.107-113. https://doi.org/10.1145/1327452.1327492
[74] Liu, G., Zhang, M., Yan, F. (2010) 'Large-scale social network analysis based on mapreduce', *IEEE Proceedings of the International Conference on Computational Aspects of Social Networks,* pp.487-490. https://doi.org/10.1109/CASoN.2010.115
[75] Braun, P., Cuzzocrea, A., Jiang, F., Leung, C.K., Pazdor, A.G. (2017) 'MapReduce-based complex big data analytics over uncertain and imprecise social networks', *Proceedings of the International Conference on Big Data Analytics and Knowledge Discover,* pp.130-145. https://doi.org/10.1007/978-3-319-64283-3_10
[76] Aghbari, A.Z., Bahutair, M., Kamel, I. (2019) 'Geosimmr: A mapreduce algorithm for detecting communities based on distance and interest in social networks', *Data Science Journal*, Vol.*18*, No.*1,* p.13. http://doi.org/10.5334/dsj-2019-013
[77] Uygun, Y., Erboy, M.O., Aktas, M.S., Kalipsiz, O., Aykurt, I. (2018) 'Technical Analysis on Financial Time Series Data Based on Map-Reduce Programming Model: A Case Study', *International Congress on Big Data Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, pp.92-97. https://doi.org/10.1109/IBIGDELFT.2018.8625357
[78] Bostani, H., Sheikhan, M. (2017) 'Hybrid of anomaly-based and specification-based IDS for Internet of Things using unsupervised OPF based on MapReduce approach', *Computer Communication*, Vol.*98*: pp.52-71. https://doi.org/10.1016/j.comcom.2016.12.001
[79] Kobo, H.I., Abu-Mahfouz, A.M., Hancke, G.P. (2015) 'A survey on software-defined wireless sensor networks: Challenges and design requirements', *IEEE access*, Vol.*5*: pp.1872-1899. https://doi.org/10.1109/ACCESS.2017.2666200
[80] Mohapatra, H., Rath, A.K. (2020) 'Fault Tolerance in WSN Through Uniform Load Distribution Function', *International Journal of Sensors, Wireless Communications and Control*, Vol.10, No.1, pp.385-394. https://doi.org/10.2174/2210327910999200525164954
[81] Vazquez-Olguın, M., Shmaliy, Y.S., Ibarra-Manzano, O., Marquez-Figueroa, S. (2021) 'Distributed UFIR Filtering with Applications to Environmental Monitoring', *International Journal of Circuits, Systems and Signal Processing*, Vol.5, pp. 349-355. http://doi.org/10.46300/9106.2021.15.38
[82] Neghabi, A.A., Navimipour, N.J., Hosseinzadeh, M., Rezaee, A. (2018) 'Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature', *IEEE Access*, Vol.*6*, pp.14159-14178. https://doi.org/10.1109/ACCESS.2018.2805842
[83] Yue, Y.G., He, P. (2018) 'A comprehensive survey on the reliability of mobile wireless sensor networks: Taxonomy challenges and future directions', *Information Fusion*, Vol.*44*, pp.188-204. https://doi.org/10.1016/j.inffus.2018.03.005
[84] Guleria, K., Verma, A.K. (2019) 'Comprehensive review for energy efficient hierarchical routing protocols on wireless sensor networks', *Wireless Networks*, Vol.*25*, No.*3*, pp.1159-1183. https://doi.org/10.1007/s11276-018-1696-1
[85] Sasirekha, S.P., Priya, A, Anita, T., Sherubha, P. (2020) 'Data Processing and Management in IoT and Wireless Sensor Network', *IOP Journal of Physics: Conference Series*, Vol.*1712*, No.*1*, p.012002. https://doi.org/10.1088/1742-6596/1712/1/012002

[86] Abdulkarem, M., Samsudin, K., Rokhani, F.Z., A., Rasid, M.F. (2020) 'Wireless sensor network for structural health monitoring: A contemporary review of technologies, challenges, and future direction', *Structural Health Monitoring*, Vol.*19*, No.*3*, pp.693-735. https://doi.org/10.1177%2F1475921719854528
[87] Qureshi, N.M., Siddiqui, I.F., Unar, M.A., Uqaili, M.A., Nam, C.S., Shin, D.R., Kim, J., Bashir, A.K., Abbas, A. (2019) 'An aggregate mapreduce data block placement strategy for wireless IoT edge nodes in smart grid', *Wireless personal communications*, Vol.*106*, No.*4*, pp.2225-2236. https://doi.org/10.1007/s11277-018-5936-6
[88] Manogaran, G., Lopez, D., Chilamkurti, N. (2018) 'In-Mapper combiner based MapReduce algorithm for processing of big climate data', *Future Generation Computer Systems*, Vol.*86*, pp.433-445. https://doi.org/10.1016/j.future.2018.02.048
[89] Rios, L.G. (2014) 'Big data infrastructure for analyzing data generated by wireless sensor networks', *International Congress on Big Data*, pp.816-823. https://doi.org/10.1109/BigData.Congress.2014.142
[90] Voinea, M.A., Uta, A., Iosup, A. (2018) 'POSUM: A Portfolio Scheduler for MapReduce Workloads', *Proceedings of the International Conference on Big Data*, pp.351-357. https://doi.org/10.1109/BigData.2018.8622215
[91] Liu, Y., Li, M., Alham, N.K., Hammoud, S. (2013) 'HSim: a MapReduce simulator in enabling cloud computing', *Future Generation Computer Systems*,Vol.*29*, No.*1*, pp.300-308. https://doi.org/10.1016/j.future.2011.05.007
[92] Wang, G., Butt, A.R., Pandey, P., Gupta, K. (2009) 'Using realistic simulation for performance analysis of mapreduce setups', *Proceedings of ACM workshop on Large-Scale system and application performance*, pp.19-26. https://doi.org/10.1145/1552272.1552278
[93] Kolberg, W., Marcos, P.D., Anjos, J.C., Miyazaki, A.K., Geyer, C.R., Arantes, L.B. (2013) 'Mrsg–a mapreduce simulator over simgrid', *Parallel Computing*, Vol.*39*. No.*4-5*, pp.233-244. https://doi.org/10.1016/j.parco.2013.02.001
[94] Liu, N., Yang, X., Sun, X.H., Jenkins, J., Ross, R. (2015) 'Yarnsim: Simulating hadoop yarn', *Proceedings of the IEEE/ACM International Symposium on Cluster Cloud and Grid Computing*, pp.637-646. https://doi.org/10.1109/CCGrid.2015.61
[95] Talattinis, K., Sidiropoulou, A., Chalkias, K., Stephanides, G. (2010) 'Parallel collection of live data using Hadoop', *Proceedings of the Panhellenic Conference on Informatics*, pp.66-71. https://doi.org/10.1109/PCI.2010.47
[96] Yufei, G., Yanjie, Z., Bing, Z., Lei S., Jiacai Z. (2017) 'Handling Data Skew in MapReduce Cluster by Using Partition Tuning', *Journal of Healthcare Engineering*, Article No.1425102, pp.1-12. https://dx.doi.org/10.1155%2F2017%2F1425102
[97] Gazis, A., Stamatis, K., Katsiri, E. (2018) 'A Method for Counting Tracking and Monitoring of Visitors with RFID sensors', *Proceedings of the Panhellenic Electrical and Computer Engineering Students Conference (ECESCON)*, Vol.*10*, pp.199-204 http://www.doi.org/10.5281/zenodo.3549417
[98] Goel, A., Munagala, K. (2012) 'Complexity measures for map-reduce, and comparison to parallel computing', *arXiv preprint*, pp1-5. https://arxiv.org/abs/1211.6526
[99] Remote access software for desktop and mobile | RealVNC. Available online: https://www.realvnc.com/en/ (accessed on 22/12/2021)
[100] Gazis, A., Katsiri, E. (2019) 'Web Frameworks Metrics and Benchmarks for Data Handling and Visualization', *Theoretical Computer Science and General Issues Book Chapter: Algorithmic Aspects of Cloud Computing Lecture Notes in Computer Science*, Vol.*1140*, pp.137-151 https://doi.org/10.1007/978-3-030-19759-9_9
[101] Vujović, V., Maksimović, M. (2014) 'Raspberry Pi as a Wireless Sensor node: Performances and constraints', *Proceedings of the International Convention on Information and Communication Technology Electronics and Microelectronics (MIPRO)*, pp.1013-1018. https://doi.org/10.1109/MIPRO.2014.6859717