Computers, Materials & Continua
DOI: 10.32604/cmc.2022.029639
Article

Clustered Single-Board Devices with Docker Container Big Stream Processing Architecture

N. Penchalaiah[1], Abeer S. Al-Humaimeedy[2], Mashael Maashi[3], J. Chinna Babu[4,*], Osamah Ibrahim Khalaf[5] and Theyazn H. H. Aldhyani[6]

[1]Department of CSE, AITS, Rajampet, 516126, Andhra Pradesh, India
[2]Software Engineering Department, College of Computer and Information Sciences, King Saud University, P.O. Box 103786, Riyadh, 11616, Saudi Arabia
[3]Software Engineering Department, King Saud University, Riyadh, 11543, Saudi Arabia
[4]Department of Electronics and Communication Engineering, Annamacharya Institute of Technology and Sciences, Rajampet, 516126, Andhra Pradesh, India
[5]Al-Nahrain University, Al-Nahrain Nanorenewable Energy Research Center, Baghdad, 10072, Iraq
[6]Applied College in Abqaiq, King Faisal University, P.O. Box 400, Al-Ahsa, 31982, Saudi Arabia
*Corresponding Author: J. Chinna Babu. Email: jchinnababu@gmail.com
Received: 08 March 2022; Accepted: 31 May 2022

Abstract: The expanding amounts of information created by Internet of Things (IoT) devices places a strain on cloud computing, which is often used for data analysis and storage. This paper investigates a different approach based on edge cloud applications, which involves data filtering and processing before being delivered to a backup cloud environment. This Paper suggest designing and implementing a low cost, low power cluster of Single Board Computers (SBC) for this purpose, reducing the amount of data that must be transmitted elsewhere, using Big Data ideas and technology. An Apache Hadoop and Spark Cluster that was used to run a test application was containerized and deployed using a Raspberry Pi cluster and Docker. To obtain system data and analyze the setup's performance a Prometheus-based stack monitoring and alerting solution in the cloud based market is employed. This Paper assesses the system's complexity and demonstrates how containerization can improve fault tolerance and maintenance ease, allowing the suggested solution to be used in industry. An evaluation of the overall performance is presented to highlight the capabilities and limitations of the suggested architecture, taking into consideration the suggested solution's resource use in respect to device restrictions.

Keywords: Big data; edge cloud; cluster architecture; performance engineering; Raspberry pi; dockers warm; container technology; data streaming

1 Introduction

Virtually everything on Internet of Things (IoT) generates data, and most of that data is stored or processed in the cloud. Furthermore, according to a study conducted by International Data

Corporation (IDC) in 2014 on the Digital Universe, the amount of data generated or gathered is steadily expanding [1]. Local data pre-processing would be better in this circumstance than the present centralized cloud processing strategy. The objective is to develop a light-weight architecture based on the edge cloud computing paradigm for offering low-cost, low environmental clusters at the cloud's edge which might include IoT devices. As a result, the proposed design is small, low cost, low power clusters of Single-Board Computers (SBC) that can pre-process data created at the edge and are linked in a local network. The objective is to present a low-power, scalable model that might be employed in industrial IoT applications.

Big Data processing tools such as Hadoop Cluster as well as Apache Spark are used in the proposed edge cloud architecture [2] which offers comparable attributes of high speed, diversity, and volume when downscaled to suit the limits imposed by IoT devices. Docker, a containerization platform that allows you to coordinate services across a device cluster, is the system's base. To assess the system's performance in respect to the Raspberry Pi's restrictions, a monitoring stack based on Prometheus, as well as a measuring and reporting tool implemented on the cluster, are used to collect system data [3].

[4–6] have already examined into Raspberry Pi-based architectures for IoT and edge contexts, but further research is needed in an industry-relevant setting. Software defined, virtualized architecture's software management boundaries must also be examined.

The proposed methodology may be built utilizing a cluster of Raspberry Pi's SBC as demonstrated in this paper. This paper shows how to create the desired system by installing and orchestrating light-weight containers that run an Apache Hadoop and Spark clusters to analyze data using Docker. This paper also shows how a Prometheus-based monitoring system can be used to collect important system metrics and use the cluster as a test environment for such applications. Finally, this paper assesses the cluster's overall performance as well as its management complexity. Some of the advantages and limitations of SBCs are listed below: Advantages: Easy to use, verified hardware, adaptable, single source, time to market. Limitations: Cost, flexibility, knowledge.

2 Applications Circumstances and Prerequisites

Data is collected and delivered to a centralized system with available storage space and computational capability in a typical cloud computing scenario, with no selection or analysis. Due to increased complexity and the requirement to cope with increasing volumes of data created, centralized cloud architecture is not appropriate in IoT contexts in terms of latency, cost, and dependability.

This paper's goal is to determine if a low-cost, lightweight fog or edge cloud computing architecture can be created. The ability to gather, analyze, and aggregate data locally utilizing low-cost devices is the most important requirement, since it reduces overall bandwidth and eliminates the need for a backup cloud environment. The cost of buying, maintaining, and operating system, as well as its feasibility for use in an industrial context and overall performance given the minimal power available, will be discussed. For software deployment or large data processing, the software platform chosen must reflect an industry-relevant scenario.

The IoT, Self-monitoring like remote local power systems are one of the most common uses for such a system. Besides that, the ability to evaluate data locally using a low power, low cost, device open up new use cases in scenarios where there aren't many processing capabilities on-site but less traffic would be beneficial, such as systems in remote locations. Smaller regional energy grids or analogous dispersed systems in rural regions are examples of prospective uses.

Raspberry Pi and other SBCs may benefit such systems due to their ability to attach sensors to the General Purpose Input/Output (GPIO) of the device as well as its capacity to perform more complex computations [7] As a result, an intelligent sensor network capable of capturing, filtering, and analysing data would be built individual nodes. A large data streaming and analytical platform's cluster of nodes can be connected to spread the burden in the format of containers, with every node accountable for a separate piece of the data pipeline from generation to assessment. As a result, microservices architecture and more flexible software deployment and administration would emerge [8,9].

In the following sections, We'll go through the tools required for building a light-weight clustering infrastructure (Raspberry Pi's), a container based development environment and automation platform (Docker swarms), and a large data streaming application design.

3 Background

Three platform technologies will be discussed: Raspberry Pi's, Docker containers, and Hadoop.

Raspberry pi

The Raspberry pi is a tiny microprocessor firstly introduced in 2013. The computer was intended to be used as an educational tool, but due to its small size and low cost, it immediately caught the interest of developers [10]. Since the introduction of the first model, the platform has been modified multiple times. The Raspberry Pi 2 Model B, which is utilized in this project, was introduced in 2015. Below Tab. 1 shows the lists of specifications for the Raspberry Pi2 Model B.

Table 1: Lists the specifications for the Raspberry pi2 model B.

Architecture	Advanced RISC Machines (ARM) v7
System on chip	BCM2836 Broadcom
Ethernet	100 Mbit's
Memory	1 GB
CPU	ARMCortex-A7 quad-core 900 MHz 32-bit

Docker Architecture

Docker is a freeware project that allows containerized programmes to be executed. It was initially released in 2013. A container is nothing more than an executable version of a Container image, which is a layer template which consists of building instructions. A container is a virtualization layer that encapsulates everything a programme needs to run, such as software, libraries, and services, while also isolating it from the architecture on which it operates. This is accomplished by effectively separating Container procedures and their offspring use the Linux containers (LXC) and lib container technologies, as well as kernel Name spacing and Control Groups (CGroups), can successfully isolate the process from the rest of the system's processes while still utilizing the host kernel and resources. Containers vary from virtual servers in that they share the host kernel and hence do not need their own operating system, resulting in lower overhead and resource usage. Fig.1 shows Architecture of Docker containers. The access to system resources granted to each container may be customized, allowing access to storage as well as, the GPIO on the Raspberry Pi are used to interface with the environment via sensors or actuators.

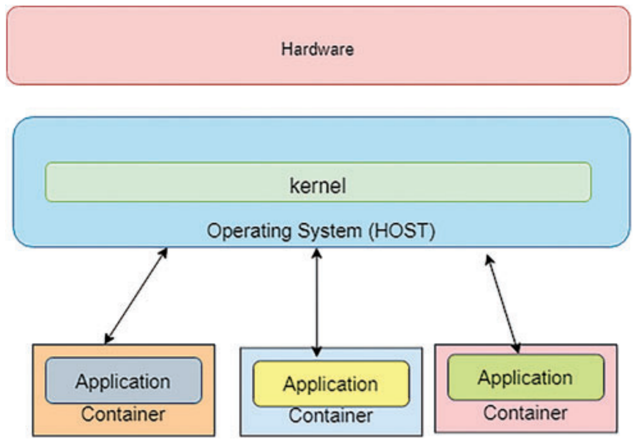

Figure 1: Architecture of docker containers

Docker Virtual Machine

Docker Engine is a client-server system that runs Docker software. The Docker client, which also functions as a host, manages all Docker objects, including images, containers, volumes, and networks. Using a Representational State Transfer (REST), Application Programming Interface (API), a Command Line Interface (CLI) connects to the server.

Docker Swarm

Docker swarm mode allows you to build, manage, and deploy application services to swarms using the Docker CLI alone, without the need for any other orchestration tools. A swarm is made up of many Docker servers operating as management nodes in swarm mode. A host can be both a manager and an employee.

When a worker node expires, the jobs it was given are distributed among other nodes, ensuring fault tolerance. Unlike a single container, a job is a functioning container that the swarm maintains and manages. A Docker swarm is seen in Fig. 2.

Hadoop

Map Reduce

Map Reduce is a software framework for working with large data sets that allows a distributed, parallel algorithm to run on a cluster of machines [11]. The Map Reduce programme has two components: a map approach for sorting and filtering data and a reduction mechanism for executing associative actions. The main goal of the Map reduce is to maximize the throughput and fault-tolerant of the core engine, but also the sustainability and fault-tolerant of the services that use it, despite being influenced by functional programming's map and reduce approaches.

Both the Map and Reduce processes in Map Reduce work with structured data (key, value) pairs. From each pair (K1, V1) in the input, the Map function is used to construct a list of pairings (K2, V2). The Map Reduce architecture then merges all pairings with almost the same value K2, resulting in a single set (K2, list (V2)) for each key. The Reduce function is then applied to each unit in turn, resulting in V3: a value list. The following Fig. 3 depicts the full method:

1 map list (K1, V1) (K2, V2)

2 reduce list (K2, V2) (V3)

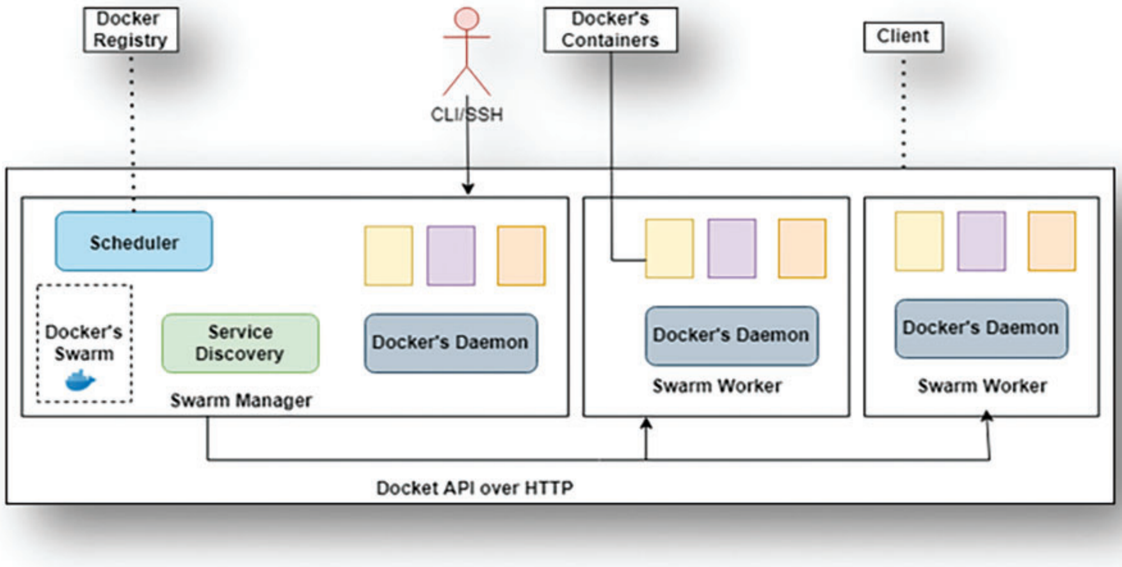

Figure 2: A sample dockers warm configuration

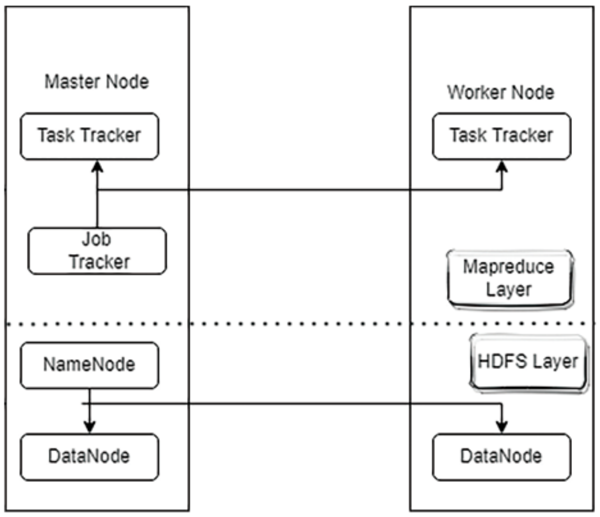

Figure 3: Hadoop cluster

HADOOP Ecosystem

Hadoop Ecosystem is a freeware package that uses the Map reduce programme paradigm to compute huge datasets on a decentralized basis [12,13]. It scales from a single node to clusters of thousands of devices [14]. In the standard Apache Hadoop library; you'll find the following modules:

- Hadoop Common–libraries and utilities that all Hadoop users have access to.
- Hadoop Distributed File System (HDFS) is a distributed file system designed to store large amounts of data across nodes in a cluster;

- Hadoop YARN is a framework for managing and scheduling computer resources as well as Hadoop-based applications.
- Hadoop Map reduce is a big data analytics platform that makes use of the Map Reduce programming model.

One K-Nearest Neighbor (KNN) algorithm and multiple worker nodes made formed a simple Hadoop cluster. The slave nodes act as map tasks and Data Nodes, while the central server serves as just a work and job trackers, as well as a Name Node (Data Index) and Data Node (Data Repository). Fig. 3 shows a cluster like this.

HDFS

The Hadoop framework is separated into two parts: the HDFS and the Map reduce processing element. Data Nodes receive files divided into data blocks. Hadoop employs the idea of data locality by moving a heavily loaded application to the same nodes. Hadoop enables quicker and more efficient dataset processing than more typical supercomputer systems because the nodes alter the data they have access to [15].

Apache Spark

Apache Spark is just a distributed software platform which provides a graphical user interface for executing programmes on clusters [12,13]. The architecture is built on the foundation of a resilient distributed dataset (RDD), which is a global and fault-tolerant collection of read-only data elements. Sparks RDDs are a subtype of Hadoop distributed shared memory that extends the Map Reduce paradigm for distributed programmes. Apache Spark enables the construction of programmes that can cut latency by orders of magnitude when opposed to an Apache Hadoop implementation, the Map reduce framework enables for less enforced dataflow.

Sparks native, Hadoop Ecosystem YARN, but also Apache Mesos clusters are now all supported, as well as a distributed storage system. HDFS, Cassandra, and Amazon S3 are all supported, but a bespoke solution may also be developed. Furthermore, as an adjunct to the transaction management architecture of Apache Hadoop, The interface for doing streaming analytics is provided by Apache Spark. This involves processing data in smaller batches on a continual basis. Spark takes streaming data from TCP/IP connections, distributed storage systems, and a variety of data feed sources including Twitter and Kafka.

4 A State of the Art Analysis

Containers have been increasingly popular in recent years, thus the suggested lightweight virtualization technique is built on them. They're perfect for computing at the edge because of their small weight, the data out from IoT layer's still has to be processed:

Because of kernel capabilities like as namespaces and control groups, Docker container virtualization has been found to have no significant overhead when compared to the traditional methods. [16,17] formalizes process segregation and resource distribution, such as I/O devices, memory, and CPU.

Practical implications of Docker's IoT/Edge Cloud Docker, according to recent study, is an exciting alternative for IoT Edge Cloud applications, owing the limits imposed by low power devices, offering light-weight virtualization and facilitating the construction of distributed systems [18,19]. Docker may also be used to supply Hadoop and Pachyderm is two big data systems, making them easier to install, manage, and run [20].

Hadoop and other similar systems have been proved to be a viable option for preparing massive amount of data on tiny clouds with limited processing capabilities and networking [21].The need for low-cost single-processor computers allows for the creation of low-power, low-cost clusters for IoT applications, allowing for sensor data pre-processing while lowering acquisition and maintenance costs [22]. As evidenced by studies conducted at the University of Glasgow in the Smart campus project, smart infrastructures and smart sensor networks benefit enormously from devices like the Raspberry Pi, Wearable sensors may be connected to its GPIO pins.

Docker containers technologies like Kubernetes and Docker Swarm may be used on Raspberry Pi's to construct highly available clusters, according to the developers of Hypriot OS; a Linux based operating system meant to bring the Docker ecosystem to ARM and IoT devices [23–25].

The limitations of operating a containerized huge data streaming application on lightweight cluster architecture, on the other hand, are yet to be discovered. As a consequence, we constructed one and assessed its cost, setup, and performance [26].

5 Data Processing Platform with Lightweight Edge

On base of a Raspberry Pi cluster, the suggested system is built. The devices are comprised of Docker containers, which employ Docker's container orchestration simplicity across several devices. The Apache Hadoop and Apache Spark clusters, Docker containers are used to execute the Prometheus surveillance stack as well as the apps that mimic data collecting, making deployment and management easy, even if hardware fails. An Apache Stream processing application uses the HDFS as a data source. The data is delivered via a Nodes application that distributes files to HDFS using the API. The architecture showed in Figs. 4 and 5. Fig.4 shows the test implementation architecture and data flow, and Fig. 5 shows a resource distribution map on the Raspberry Pi used in the experiment.

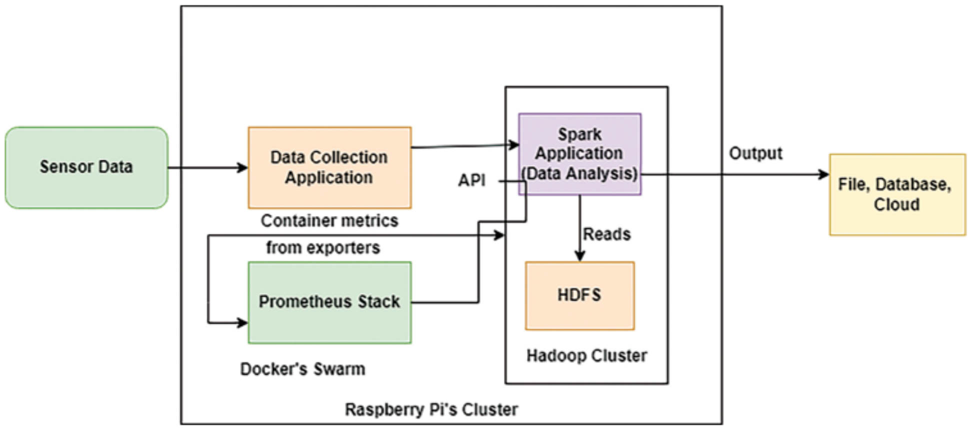

Figure 4: Design of a system

Hardware Architecture

Eight Raspberry Pi 2 Model B computers are employed in the cluster; each comes with an 8 GB flash Drive for installing the operating system. The Raspberry Pi's are linked to a Veracity Cam switch 8 Mobile switch4, which provides power to the devices through Power over Ethernet (POE). Seven of the ten 10 or 100 Mbit/s connections on the switch have 802.3af POE outputs. Category 5E SFTP cables are used to connect the Raspberry Pi's to the switch. Although it is neither necessary nor cost effective, POE has the benefit of being more ecologically friendly since it removes the need for a separate power supply for Raspberry Pi, resulting in fewer cables crossing the network. Because the Raspberry Pi's lack sufficient connection for POE, the devices should be equipped with a separate POE module connected to the Raspberry Pi's' GPIO ports.

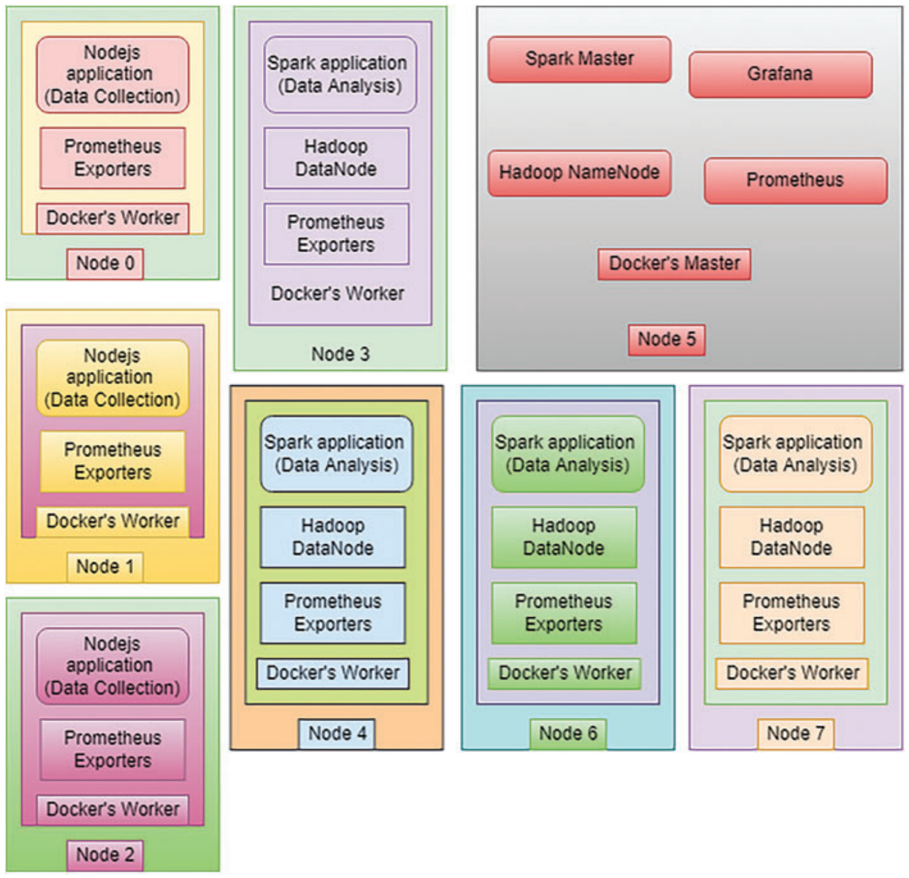

Figure 5: Shows how service containers are distributed throughout the nodes

Hypriot OS

Hypriot OS is a specialized debian distribution that runs on the Raspberry Pi's. Docker is already installed, as well as a rudimentary operating system designed specifically for Docker on ARM platforms. The Hypriot OS image may be flashed onto a micro SD card and used right away. The operating system comes with a pre-boot configuration file that, among other things, you can specify a host name using Avahi's local host name server.

The operating system comes with a preinstalled Secure Shell (SSH) service that may be accessed using the credentials you provide. All nodes have been configured to utilize public key authentication instead of password authentication due to the vulnerability of password authentication. A configuration management tool, notably Ansible6, since it provides for the defining of hosts and the automation of configuration actions through SSH, to automate the setup of the nodes, such as establishing automated disc mounting and authentication, or installing extra software.

Docker Setup and Control of Swarms

Using the Docker engine CLI, start and configure the swarm. To begin, one of the nodes creates a single node swarm, with that node serving as its manager. The manager stores join tokens, which may be used to add new computers to the swarm. Different tags may also be set on each node using the Docker CLI to limit service deployment to certain nodes. Other swarm administration operations, including as promotions, demotions, and node membership management, may be performed by giving Docker engine commands to any of the swarm managers.

Service Provisioning

Docker stacks provisioning can be used to provide services to the swarm, allowing it to deploy a whole application stack. Docker uses a stack description in the form of a Compose file to describe the stack, which consists of several services (version 3). Each stack service can have its own networks, mounted volumes, service names, ports, replicas and registry, as well as deployment limitations such as Docker nodes tag. When Docker is deployed on a management node, it distributes each service in the stack throughout the swarm's nodes according to constraints and requirements.

Although the tasks outlined in this paper were done manually or with any other scripting language, The compose files was sent to the swarm management and the nodes were set up using Ansible scripts, which included building configuration files and mounting directories.

Apache Spark (Hadoop) despite the fact that Hadoop may be used to build native computer clusters, both Hadoop and Apache Spark are installed in Docker containers to simplify the deployment process and eliminate the requirement for software installation and maintenance on every device.

To deploy Hadoop and Apache Spark in a container and set up the environment, a customized Docker image is utilized. One master node and four autonomous worker nodes make up the final design. Because the worker node act as the cluster's Data Nodes, each worker node container has ample storage, which is provided by a common 3.5 inch 1TB storage device.

6 Data Edge Monitoring and Processing

Data collection and analysis

Two apps will be used to put the architecture to the test: one will mimic data gathering on a subset of the nodes, and the other will be installed on an Apache Spark cluster to analyze the data. The apps are also used to gather system performance statistics, such as the amount of time it takes to process provided data.

Obtaining information A Nodejs application transfers an Hyper Text Transfer Protocol (HTTP) PUT requests to the HDFS API with the data content to mimic data collection, which results in the creation of a file in a set length of time. The data gathering application's is installed just on cluster's final three nodes using a Compose file.

The experiment's outcome is a set of sample documents that the software can access and read via the internet, enabling the document's size or content to be altered without having to update and re-deploy the application.

Investigate the data. A Python application is used to analyze the data. Every second, the programme searches for freshly produced files and counts how many times each phrase appears in those files. With minimal modifications, the software is based on an example application provided with the Apache Spark engine.

As previously stated, the used technique uses the usual Map Reduce paradigm to generate a list of (Key, Value) pairs of the kind (Word, 1) every each time a sentence appears. The Reduce step is done to each group, providing the total of each word's occurrences. The last two steps are handled by Reduce by key, a single function. The approach is shown in pseudo code below, using a sample application of Map Reduce solution.

```
Lines
Count :=
Lines.FlatMap(lambda(line){line.split("")})
.map(lambda(word){(word, 1)})
.reduceByKey(lambda(Val, acc){Val + acc})
counts.pprint()
```

To send the application among the master node, use the Spark CLI tool spark submit, which will then distribute it to the cluster's workers.

Monitoring

To assist system monitoring, a Prometheus based monitoring stack's with three components is deployed to the cluster. Prometheus is a free software monitor and/or alert system's that uses a time series database and offers a variety of interfaces to other systems.

HAProxy, the ELK stack (Elastic search, Log stash, and Kibana), and Docker are examples of such technologies. Second, because Prometheus' operational philosophy is to extract data from several services, the stack includes several "exporters" that gather metrics and expose them for Prometheus to collect.

Grafana is a popular tool for analysing and visualizing data. Grafana is a web-based tool that lets users construct dashboards for tracking and AnalysingData. For a use case requiring the tracking of Docker container on different Linux servers, this solution was chosen since it required minimum configuration. As a consequence, a virtually finished implementation for Raspberry Pi's was employed. The stack is deployed to the Docker cluster with the help of a custom Compose file that makes use of Docker containers. Advisor as well as Node exporters are installed on each Raspberry Pi, while Prometheus as well as Grafana are installed on one of the nodes in the final setup.

7 Evaluation Process

Considerations for Set-Up and Usability

The practical efforts necessary to establish and manage the cluster design of choice is evaluated in this area. Neither the hardware setup nor the software preparation needs anymore special skills.

MicroSD cards may be flashed with a special programme that runs on all major platforms. The maintainers of the Hypriot OS provide information on their website, this contains numerous instructions for setting up an SSH connection using public key authentication, as well as the pre-boot configuration file for the node's operating system. Although all of the operations may be accomplished manually, the Docker swarm was set up and then managed to automate time-consuming duties.

Because installation necessitates all nodes doing the same tasks, such as installing vital dependencies or joining a newly formed swarm, it's worth looking into using a configuration management solution or writing shell scripts. After the clusters is up and running, you can use the Docker documentation to control the swarming and deliver services to it. These services are delivered using Docker composition files on each of the master nodes.

- On the primary assessment criteria, the following comments are permissible: Operational simplicity and maintainability, the utilization of containerized services on a Dockers swarm is critical for the solution's maintainability and convenience of use.
- Fault tolerance refers to the capacity to accept defects. Container that fail due to hardware and/or software faults can be instantaneously restart on any other node in the cluster, reducing overhead and increasing fault tolerance. As a result, Docker is a great way to achieve high availability and fault tolerance.
- The amount of time it takes to create an image and how long it takes to create a picture. It's worth mentioning that for programmes like Apache Hadoop and Apache Spark, a Docker image for the ARM architecture may not always be available.

To build the requisite images, we used Gitlab's Continuous deployment and delivery (CI/CD) solution. Because the containers that create the images utilize different instruction sets than the Raspberry Pi, the Docker files employed a system emulator called Quick Emulator (QEMU) to cross-build the images. Despite the fact that this approach has some disadvantages, Because of the longer build times, it was chosen over building the files on several of the clusters, such as the Java Virtual Machine (JVM) reporting issues during builds, necessitating the manual execution of several preparation tasks on previously deployed services After accounting for the time it takes to distribute the image to the nodes, Tab. 2 compares the development timings of a few of the deployed images on ashared computer to the identical image on the cluster's Raspberry Pi. The picture was retrieved from the Gitlab's registry in roughly 6 min.

Table 2: The construction times of Docker engine

Create architecture	Intended architecture	Construction time
Armv7	x86	6m 1s
Armv7	armv7	14m33

Performance

The major goal of the performance evaluation will be to test data processing time and resource utilization utilizing input data file sizes that are representative of real-world IoT scenario's and common small-to-middle size data producers.

The resources provided to the Spark executors had to be fine-tuned for the test programmes, which was impacted by the Raspberry Pi's' limits. Because supplying less memory to each executor would result in out-of-memory errors, and giving more memory to each executor would starve one of the Hadoop/Spark cluster's components, giving 500 MB of RAM to each executor gave the only meaningful results. This memory allocation may be changed after a task has been submitted.

File Processing Times

Using time-stamps to compare the submission time to the time the output was written to the submission shell's standard output stream, the time difference between uploading an image folder and the completion of the analysis process was calculated (stdout).

Tab. 3 illustrates the findings for files delivering 500 B and 1 KB of data per second, demonstrating a typical small to medium data creation volumes. In both cases, the data analysis tool requested fresh files per second, and files were sent one at a time.

The test cases were built to see how the Raspberry Pi-based container cluster architecture will perform in various IoT and edge computing situations. Tab. 3 shows delays that are excessively long for various real-time processing needs, such as any system that relies on a quick reaction time for quick actuation and are notably expensive when considering the quantity of data and submission rate employed during the trial runs. If simply storing and evaluation are required without quick response, or if the sensors only create a modest amount of data, this level will suffice (such as humidity and temperature sensors).

Table 3: File processing time begins when the file is created

File size	Polling time	Delay
1.04 KB	1 s	236 s
532 B	1 s	61 s

Resource Utilization

The monitoring stack's data shows that the quantity of resources required for file submissions and analysis had remained constant.

CPU: Fig. 6 illustrates the CPU spending time (per each node) throughout test application's evaluation, whereas Fig. 7 depicts the same data divided by containers. The graphs show that the Central Processing Unit (CPU) demand on all Apache Spark nodes has grown as expected, notably on the cluster's worker nodes. Despite the fact that data collection began 2 min later, at the 120 s mark, as evidenced by the growing CPU usage of the Spark master node, the analytic application was transmitted through node 3 (spark worker2. At 420 and 480 s, respectively, data collection and analysis were halted. Tab. 3 lists the records in the 532 Bytes test file.

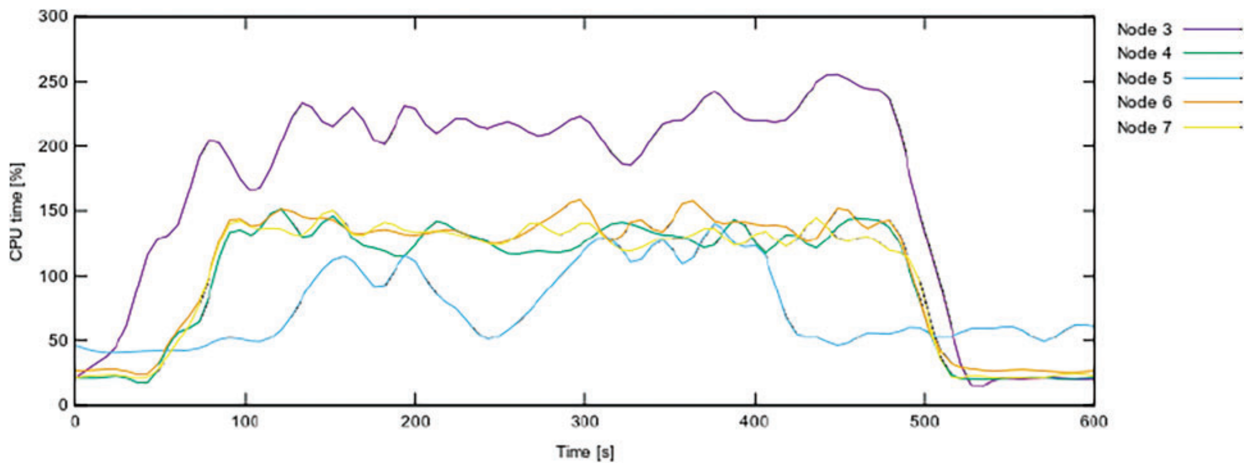

Figure 6: Containers CPU time per node

Memory: Figs. 8 and 9 shows the amount of memory consumed by the Spark containers and the monitor stacks during the experiments, broken down by container and node.

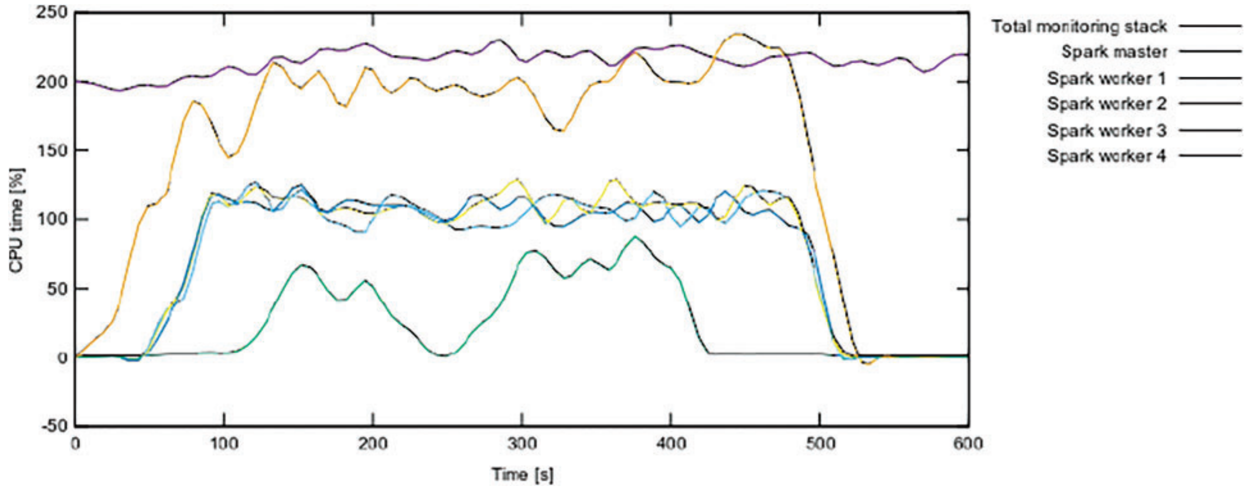

Figure 7: CPU time used by container

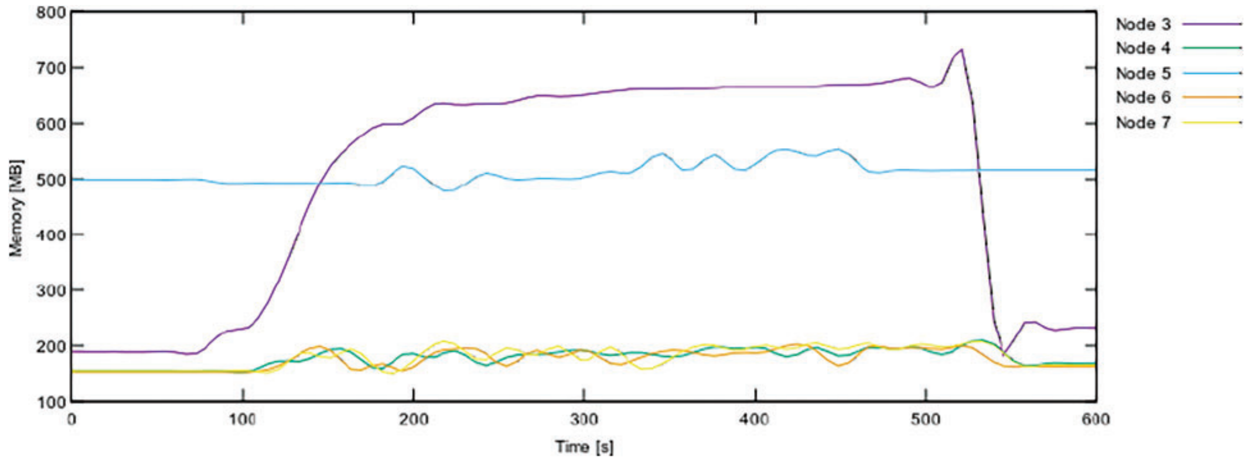

Figure 8: Depicts the consumption of container memory per node

As predicted, the memory use of the Spark node that submitted the application, which also serves as the execution controller and collects data from the other nodes, is larger than that of the other nodes.

The remaining system components (such as system processes and Docker daemons) were consistently used during the test, with CPU time ranging around 2% and memory use hovering around 120 Mb on each node.

Discussion and Analysis

The graphs reveal that system resources are not being utilized to their full potential, with CPU and, in particular, device memory space being underutilized. The delay is caused by how HDFS manages the distribution and replication of tiny files 16, resulting in an inefficient use of Spark's streaming capabilities and system resources, according to the study.

Aside from the comparatively significant delays and inefficient resource usage, the acquired data reveals that the Spark master container's CPU usage is uneven, peaking at 0% around 250 s and remaining continuously erratic across all test runs.

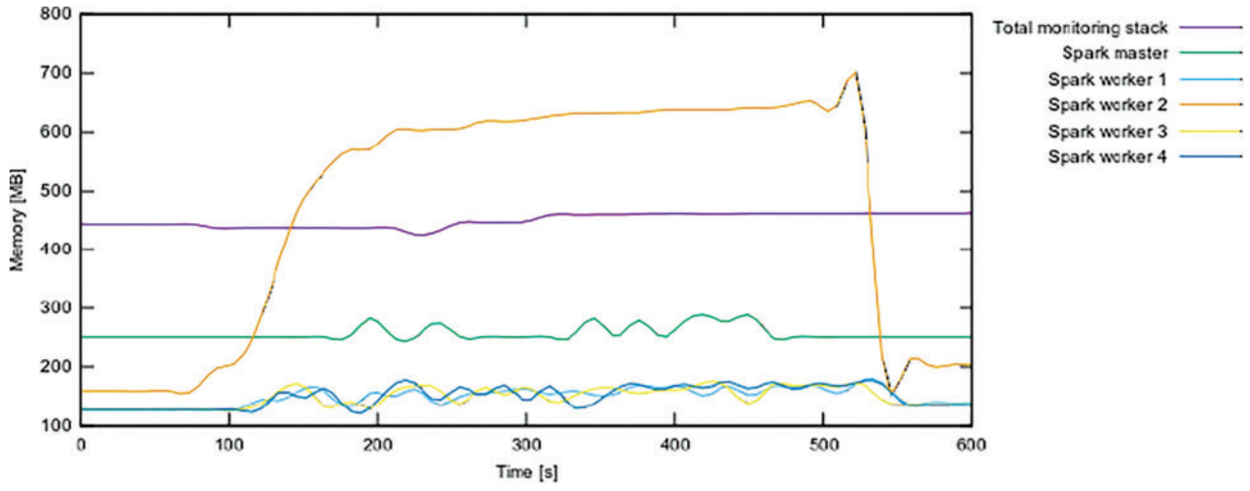

Figure 9: Container memory usage by container

It's possible that the file system, specifically HDFS, is starving the process owing to lengthy I/O delays. Additional study from a number of sources might help to better understand and corroborate the causes. The Raspberry Pi3 may be a better alternative than the Pi2 in terms of prospective hardware performance improvements. While the CPU's speed is merely increased by 300 MHz, its architecture is improved from Cortex-A7 to Cortex-A53, resulting in a 32-bit to 64-bit architectural improvement and a factor 2 to 317 performance boost. The Raspberry Pi 3's RAM speed is 900 MHz, compared to 450 MHz on the Raspberry Pi 2.

Finally, the goal of this investigation was to discover some of the infrastructure's limitations. However, depending on the application, our configuration could suffice or might be developed with better hardware and/or software. Furthermore, a controller that allows for self-adaptation to tackle platform configuration performance and resource utilization anomalies might be a future solution to poor resource usage [27–29].

8 Conclusion

For a variety of edge data processing applications in the Internet of Things, a small containerization clusters platform might be used. Docker's fault-tolerant and controlled edge computing architecture is made possible by its lightweight containerization and container orchestration features. Docker delivers excellent service availability, allowing the cluster to self-heal while also facilitating scaling, thanks to the Docker's swarm constantly monitoring the condition of the services.

Single-board devices have a minimal energy and cost effect while yet being able to run complicated infrastructures via clustering, which bodes well for lower total infrastructure costs even when you're deal with a lot of information. Due to the fact that a Big Data system may be constructed on device cluster with severe networking and computing resource restrictions, such as Raspberry Pi's, our prototype system have limitations. Because the HDFS was used as a streaming data source, the prototype solution suffered from performance issues.

In practice, most of the graphics in our platform setup were built from scratch or extensively modified from previous implementations to match our needs. Despite this, the number of ARM-related initiatives is increasing. For example, the monitoring stack was made available as a fully prepared

Docker construct file that could be installed to the clusters with little configuration modifications, allowing us to use the system as a performance test bed for the implementation and apps.

For a slew of new applications, lightweight edge designs are necessary. Self-driving cars, for instance, require mobile edge clouds. 5G mobile networks, for example, are examples of modern telecommunications technology, are commonly used to help automobiles coordinate their operations. Local edge cloud and central clouds, along with onboard computing, must be leveraged to satisfy the low latency requirement. Clusters of SBCs are a type of edge computing infrastructure.

In the future, the investigation might go deeper into Apache Spark's numerous functions, such as using the network as a raw data. This method might make standard IoT data transit protocols like Message Queue Telemetry Transport Protocol (MQTT) easier to use, while pre-processed and aggregated data could be stored in Hadoop and HDFS. Furthermore, the model used to assess the implementation might be enhanced to test the system's real-world performance across a broader range of circumstances. Future research in this area may look into alternative data stream systems, these systems will be develop and evaluate in multiple clouds.

Another option is to go beyond performance engineering to address issues like security and trust in IoT and edge cloud environments [30–31].

Acknowledgement: All contributors are met the criteria for authorship.

Funding Statement: This research project was supported by a grant from the "Research Center of College of Computer and Information Sciences", Deanship of Scientific Research, King Saud University.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] K. A. Gepreel, A. M. S. Mahdy, M. S. Mohamed and A. Al-Amiri, "Reduced differential transform method for solving nonlinear biomathematics models," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 979–994, 2019.

[2] F. Fowley, C. Pahl, P. Jamshidi, D. Fang and X. Liu, "A classification and comparison framework for cloud service brokerage architectures," *IEEE Transactions on Cloud Computing*, vol. 6, no. 2, pp. 358–371, 2018.

[3] S. N. Alsubari, S. N. Deshmukh, A. A. Alqarni, N. Alsharif and O. I. Khalaf, "Data analytics for the Identification of fake reviews using supervised learning," *Computers, Materials & Continua*, vol. 70, no. 2, pp. 3189–3204, 2022.

[4] O. I. Khalaf and G. M. Abdulsahib, "Frequency estimation by the method of minimum mean squared error and p-value distributed in the wireless sensor network," *Journal of Information Science and Engineering*, vol. 35, no. 5, pp. 1099–1112, 2019.

[5] L. Alhenak and M. Hosny, "Genetic-frog-leaping algorithm for text document clustering," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 1045–1074, 2019.

[6] B. Hu, F. Xiang, F. Wu, J. Liu, Z. Sun *et al.,* "Research on time synchronization method under arbitrary network delay in wireless sensor networks," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 1323–1344, 2019.

[7] O. I. Khalaf, G. M. Abdulsahib, H. D. Kasmaei and K. A. Ogudo, "A new algorithm on application of block chain technology in live stream video transmissions and telecommunications," *International Journal of e-Collaboration*, vol. 16, no. 1, pp. 16–32, 2020.

[8] O. I. Khalaf, C. Andrés Tavera Romero, A. AzhaguJaisudhanPazhani and G. Vinuja, "VLSI implementation of a high-performance nonlinear image scaling algorithm," *Journal of Healthcare Engineering*, vol. 2021, no. 5, pp. 1–10, 2021.

[9] H. Kyu Shin, Y. Han Ahn, S. Hyo Lee and H. Young Kim, "Digital vision based concrete compressive strength evaluating model using deep convolutional neural network," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 911–928, 2019.

[10] D. Von Leon, L. Miori, J. Sanin, N. El Ioini, S. Helmer *et al.,* "A lightweight container middleware for edge cloud architectures," *Fog and Edge Computing: Principles and Paradigms*, vol. 1, pp. 145–170, 2019.

[11] S. Rajendran, O. I. Khalaf and Y. Alotaibi, "Map reduce-based big data classification model using feature subset selection and hyper parameter tuned deep belief network," *Scientific Reports*, vol. 11, pp. 24138, 2021.

[12] S. Tabatabaei, "A novel fault tolerance energy-aware clustering method via social spider optimization (sso) and fuzzy logic and mobile sink in wireless sensor networks (wsns)," *Computer Systems Science and Engineering*, vol. 35, no. 6, pp. 477–494, 2020.

[13] A. Gumaei, M. Al-Rakhami, H. AlSalman, S. M. M. Rahman and A. Alamri, "Dl-har: Deep learning-based human activity recognition framework for edge computing," *Computers, Materials & Continua*, vol. 65, no. 2, pp. 1033–1057, 2020.

[14] O. I. Khalaf and G. M. Abdulsahib, "Design and performance analysis of wireless ipv6 for data exchange," *Journal of Information Science and Engineering*, vol. 37, pp. 1335–1340, 2021.

[15] B. Ahmed, B. Seghir, M. Al-Osta and G. Abdelouahed, "Container based resource management for data processing on IoT gateways," in *The 16th Int. Conf. on mobile Systems and pervasive Computing (MobiSPC), Procedia Computer Science*, Halifax, NS, Canada155, pp. 234–241, 2019.

[16] G. M. Abdulsahib and O. I. Khalaf, "An improved algorithm to fire detection in forest by using wireless sensor networks," *International Journal of Civil Engineering and Technology*, vol. 9, no. 11, pp. 369–377, 2018.

[17] P. Jamshidi, C. Pahland and N. C. Mendonca, "Managing uncertainty in autonomic cloud elasticity controllers," *IEEE Cloud Computing*, vol. 3, no. 3, pp. 50–60, 2016.

[18] O. I. Khalaf, G. M. Abdulsahib and M. Sadik, "A modified algorithm for improving lifetime wsn," *Journal of Engineering and Applied Sciences*, vol. 13, pp. 9277–9282, 2018.

[19] O. I. Khalaf, G. M. Abdulsahib and B. M. Sabbar, "Optimization of wireless sensor network coverage using the bee algorithm," *Journal of Information Science and Engineering*, vol. 36, no. 2, pp. 377–386, 2020.

[20] M. Rajalakshmi, V. Saravanan, V. A. Arunprasad and O. I. Khalaf, "Machine learning for modeling and control of industrial clarifier process," *Intelligent Automation & Soft Computing*, vol. 32, no. 1, pp. 339–359, 2022.

[21] R. Surendran, O. I. Khalaf and C. Andres, "Deep learning based intelligent industrial fault diagnosis model," *Computers, Materials & Continua*, vol. 70, no. 3, pp. 6323–6338, 2022.

[22] L. Alhenak and M. Hosny, "Genetic-frog-leaping algorithm for text document clustering," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 1045–1074, 2019.

[23] R. Morabito, I. Farris, A. Iera and T. Taleb, "Evaluating performance of containerized iot services for clustered devices at the network edge," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 1019–1030, 2018.

[24] C. Pahl, P. Jamshidi and O. Zimmermann, "Architectural principles for cloud software," *ACM Transactions on Internet Technology*, vol. 18, no. 2, pp. 1–23, 2018.

[25] H. Kyu Shin, Y. Han Ahn, S. Hyo Lee and H. Young Kim, "Digital vision based concrete compressive strength evaluating model using deep convolutional neural network," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 911–928, 2019.

[26] M. Shoaib Arif, A. Raza, W. Shatanawi, M. Rafiq and M. Bibi, "A stochastic numerical analysis for computer virus model with vertical transmission over the internet," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 1025–1043, 2019.

[27] N. O. García, M. F. D. Velásquez, C. A. T. Romero, J. H. Ortiz and O. I. Khalaf, "Remote academic platforms in times of a pandemic," *International Journal of Emerging Technologies in Learning*, vol. 16, no. 21, pp. 121–131, 2021.

[28] O. I. Khalaf, Carlos Andrés Tavera Romero, A. Azhagu Jaisudhan Pazhani and G. Vinuja, "VLSI implementation of a high-performance nonlinear image scaling algorithm," *Journal of Healthcare Engineering*, vol. 2021, no. 5, pp. 1–10, 2021.

[29] R. ldar, "Increasing FPS for single board computers and embedded computers in 2021 (Jetsonnano and YOVOv4-tiny). Practice and review," arXiv preprint arXiv: 2107. 12148, 2021.

[30] V. Raghupathy, O. I. Khalaf, C. Andrés, S. Sengan and D. K. Sharma, "Interactive middleware services for heterogeneous systems," *Computer Systems Science and Engineering*, vol. 41, no. 3, pp. 1241–1253, 2022.

[31] G. M. Abdulsahib and O. I. Khalaf, "Accurate and effective data collection with minimum energy path selection in wireless sensor networks using mobile sinks," *Journal of Information Technology Management*, vol. 13, no. 2, pp. 139–153, 2021.