electronics

Edge Computing for Internet of Things

Edited by
Kevin Lee and Ka Lok Man

Printed Edition of the Special Issue Published in *Electronics*

www.mdpi.com/journal/electronics

MDPI

Edge Computing for Internet of Things

Edge Computing for Internet of Things

Editors

Kevin Lee
Ka Lok Man

MDPI • Basel • Beijing • Wuhan • Barcelona • Belgrade • Manchester • Tokyo • Cluj • Tianjin

Editors

Kevin Lee
School of IT
Deakin University
Melbourne
Australia

Ka Lok Man
Department of Computer
Science and Software
Engineering
Xi'an Jiaotong-Liverpool
University
Suzhou
China

Editorial Office
MDPI
St. Alban-Anlage 66
4052 Basel, Switzerland

This is a reprint of articles from the Special Issue published online in the open access journal *Electronics* (ISSN 2079-9292) (available at: www.mdpi.com/journal/electronics/special_issues/edge_computing_IoT1).

For citation purposes, cite each article independently as indicated on the article page online and as indicated below:

LastName, A.A.; LastName, B.B.; LastName, C.C. Article Title. *Journal Name* **Year**, *Volume Number*, Page Range.

ISBN 978-3-0365-4276-8 (Hbk)
ISBN 978-3-0365-4275-1 (PDF)

Contents

electronics

MDPI

Editorial

Edge Computing for Internet of Things

Kevin Lee [1,*] and Ka Lok Man [2]

[1] School of Information Technology, Deakin University, Geelong, VIC 3220, Australia
[2] Department of Computer Science and Software Engineering, Xi'an Jiaotong Liverpool University, Suzhou Dushu Lake Higher Education Town, Suzhou Industrial Park, Suzhou 215123, China; Ka.Man@xjtlu.edu.cn
* Correspondence: kevin.lee@deakin.edu.au

Citation: Lee, K.; Man, K.L. Edge Computing for Internet of Things. *Electronics* **2022**, *11*, 1239. https://doi.org/10.3390/electronics11081239

Received: 8 April 2022
Accepted: 12 April 2022
Published: 14 April 2022

1. Introduction

The Internet of Things (IoT) is maturing and becoming an established and vital technology. IoT devices are being deployed in homes, workplaces and public areas at an increasingly rapid rate. IoT is the core technology of smart homes, smart cities, intelligent transport systems and automated logistics systems. IoT has the potential to optimize travel, improve logistics, reduce energy usage and improve quality of life.

With the increasing use of IoT, the problem of managing the vast volumes of data, wide variety and type of data that are generated, and erratic generation patterns is becoming increasingly clear and challenging. As well as the increasing number of IoT devices conducting traditional sensing, generating more data, there is also an increasing number of cameras, generating large volumes of complex data with stringent processing requirements. The current standard IoT model with Cloud computing is not sustainable, and a new model is needed to improve response time, reduce data transfer and increase processing availability.

This Special Issue is focused on solving this problem through the use of edge computing. Edge computing offers a solution to managing IoT data by processing IoT data close to the location where the data are generated. Edge computing allows for computation to be performed locally, thus reducing the volume of data that need to be transmitted to remote data centers and Cloud storage. It also allows for decisions to be made locally without having to wait for Cloud servers to respond.

2. This Issue

The ten articles in this Issue all present research in the area of edge computing for IoT. They address topics in the areas of resource management and offloading, deployment management, failure recovery, architectures, and algorithms for processing IoT data.

With the increasingly complex demands from IoT applications regarding end nodes and edge computing, there is an increasing need to effectively utilize the available resources. In [1], the authors focus on maximizing the use of available IoT devices by allowing them to collectively execute services using their spare resources. This reduces latency and data transfer to cloud services and improves the overall performance of IoT applications. Offloading computation is needed when the resources for a particular device are not enough to perform a task. In [2], the authors propose a new cooperative offloading method which uses edge-computing resources in rapidly changing environments. In this way, the authors aim to reduce latency and energy use. IoT applications should use all available resources, but it is difficult to decide which computation should run on which available resource. In [3] the authors examine approaches to job scheduling and execution on smart phones, with the goal of enabling the widespread use of dew computing. In [4], the authors aim to use Fog resources efficiently by utilising deep-learning to optimize content caching in edge nodes.

Edge-based IoT deployments need to be managed effectively, efficiently and automatically. IoT applications that involve end-nodes, edge and fog computing are complex to manage. In [5], the authors propose a method for dynamically orchestrating services in a Fog architecture. The approach focuses on heterogeneous devices and the dynamic nature of the end devices. The appropriate placing of these edge resources is the focus of [6], which attempts to use reinforcement learning to optimize placement based on the latency and load of the server. The authors analyze the effectiveness of the proposed solution to attempt to maximize network-wide performance and focus on improving security using this approach.

The issue of automatically detecting and recovering from failures in container-based IoT deployment is the focus of paper [7]. The authors observe the rate at which data are received form the IoT end node, and if this is not as expected, perform the recovery process. The work is focused on container-based deployments, which are increasingly becoming the norm for IoT applications.

For edge computing to work effectively, system architectures that can be deployed in different contexts are needed. One such context is in automated warehouses, to support efficient logistics. In [8], the authors propose a hierarchical edge-based architecture to enable a rapid response in intelligent warehouses. The aim is to take advantage of edge nodes, to offer low latency while also enhancing the reconfiguration abilities of nodes at the edge of the network.

IoT data vary depending on their deployment and application; all data need to be processed in some way. In [9], the authors propose algorithms for processing spacial data that are of interest to the data mining field. They propose region-based frequent pattern growth (RFP-Growth) to search for associations in IoT data. In [10], the authors use deep learning algorithms to classify objects from RGB cameras on IoT end-devices.

Acknowledgments: We would like to thank all the authors who submitted their excellent work to this Special Issue. We are extremely grateful to all those reviewers who took the time to read and provide valuable feedback to authors regarding their submissions. The feedback from reviewers enabled authors to improve the overall quality of all the papers in this Special Issue. Our special thanks go to the editorial board of the MDPI Electronics journal for the opportunity to guest edit this Special Issue, and to the Electronics Editorial Office staff for their hard work ensuring an efficient peer-review schedule and timely publication.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Coelho, J.; Nogueira, L. Enabling Processing Power Scalability with Internet of Things (IoT) Clusters. *Electronics* **2022**, *11*, 81. [CrossRef]
2. Kim, S.; Kang, J.; Yoon, Y. Linked-Object Dynamic Offloading (LODO) for the Cooperation of Data and Tasks on Edge Computing Environment. *Electronics* **2021**, *10*, 2156. [CrossRef]
3. Hirsch, M.; Mateos, C.; Zunino, A.; Majchrzak, T.A.; Grønli, T.M.; Kaindl, H. A Task Execution Scheme for Dew Computing with State-of-the-Art Smartphones. *Electronics* **2021**, *10*, 2006. [CrossRef]
4. Bhandari, S.; Ranjan, N.; Khan, P.; Kim, H.; Hong, Y.S. Deep Learning-Based Content Caching in the Fog Access Points. *Electronics* **2021**, *10*, 512. [CrossRef]
5. Morkevicius, N.; Venčkauskas, A.; Šatkauskas, N.; Toldinas, J. Method for Dynamic Service Orchestration in Fog Computing. *Electronics* **2021**, *10*, 1796. [CrossRef]
6. Kasi, M.K.; Abu Ghazalah, S.; Akram, R.N.; Sauveron, D. Secure Mobile Edge Server Placement Using Multi-Agent Reinforcement Learning. *Electronics* **2021**, *10*, 2098. [CrossRef]
7. Olorunnife, K.; Lee, K.; Kua, J. Automatic Failure Recovery for Container-Based IoT Edge Applications. *Electronics* **2021**, *10*, 3047. [CrossRef]
8. Wang, Y.; Man, K.L.; Lee, K.; Hughes, D.; Guan, S.U.; Wong, P. Application of Wireless Sensor Network Based on Hierarchical Edge Computing Structure in Rapid Response System. *Electronics* **2020**, *9*, 1176. [CrossRef]
9. Jang, H.J.; Yang, Y.; Park, J.S.; Kim, B. FP-Growth Algorithm for Discovering Region-Based Association Rule in the IoT Environment. *Electronics* **2021**, *10*, 3091. [CrossRef]
10. Xu, L.; Xie, J.; Cai, F.; Wu, J. Spectral Classification Based on Deep Learning Algorithms. *Electronics* **2021**, *10*, 1892. [CrossRef]

electronics

MDPI

Article

Enabling Processing Power Scalability with Internet of Things (IoT) Clusters

Jorge Coelho [1,2,*] and Luís Nogueira [1]

[1] School of Engineering (ISEP), Polytechnic of Porto (IPP), 4249-015 Porto, Portugal; lmn@isep.ipp.pt
[2] Artificial Intelligence and Computer Science Laboratory, University of Porto (LIACC), 4099-002 Porto, Portugal
* Correspondence: jmn@isep.ipp.pt; Tel.: +351-228-340-500

Abstract: Internet of things (IoT) devices play a crucial role in the design of state-of-the-art infrastructures, with an increasing demand to support more complex services and applications. However, IoT devices are known for having limited computational capacities. Traditional approaches used to offload applications to the cloud to ease the burden on end-user devices, at the expense of a greater latency and increased network traffic. Our goal is to optimize the use of IoT devices, particularly those being underutilized. In this paper, we propose a pragmatic solution, built upon the Erlang programming language, that allows a group of IoT devices to collectively execute services, using their spare resources with minimal interference, and achieving a level of performance that otherwise would not be met by individual execution.

Keywords: edge computing; computational offloading; orchestration; IoT; functional programming

Citation: Coelho, J.; Nogueira, L. Enabling Processing Power Scalability with Internet of Things (IoT) Clusters. *Electronics* **2022**, *11*, 81. https://doi.org/10.3390/electronics11010081

Academic Editor: Akash Kumar

Received: 22 November 2021
Accepted: 24 December 2021
Published: 28 December 2021

1. Introduction

Given the ubiquity of internet of things (IoT) devices and their strong proliferation [1] many opportunities appear in exploring their potentialities, namely their connectivity and computational power [2]. It is well known that the quantity of data produced by a variety of data sources and sent to end systems to further processing is growing significantly, increasingly demanding more processing power. The challenges become even more critical when a coordinated content analysis of the data sent from multiple sources is necessary. Thus, with a potentially unbounded amount of stream data and limited resources, some of the processing tasks may not be satisfyingly answered by individual devices, guaranteeing a desired level of performance.

Computation offloading is recognized as a promising solution by migrating a part or an entire application to a remote server in order to be executed there. Various models and frameworks have been proposed to offload resource-intensive components of applications for more efficient execution [3–5]. However, these solutions rely on the concept of offloading to the cloud. Due to the increasing hardware capabilities of IoT devices and their proliferation, making it common to have several of these devices in the same area, offloading to the cloud may not always be a necessity, if the available resources of these devices are wisely used. The study of scenarios where heterogeneous nodes with unknown resources are aggregated in a collaborative effort to achieve some goal has been the subject of works such as [6] where some sort of data analysis is needed to estimate each node capacity and distribute work wisely.

Orchestration in distributed systems is a common approach to creating an abstraction layer between the several devices of the system. With the orchestration layer, devices that constitute the distributed system are "hidden" and their details and behavior are managed by the orchestrator (also referred as coordinator), providing a simplified interface to those devices and centered in the use of their resources without the need to know other operational details. This is particularly relevant in service-oriented architectures (SOA) [7] with obvious applications when using clusters of IoT devices [8].

At the same time, functional programming is an established approach to implement parallel and distributed systems [9]. The minimization of the need of a shared state enables code distribution and parallel processing that fosters the development of easily scalable systems. Due to the rise of multicore and distributed systems, functional programming spread its influence through many mainstream languages [10,11] and is used in major cloud infrastructures such as the AWS Lambda [12]. In particular, Erlang [13], due to its simplicity and strong support for fault-tolerant distributed programming, is seen as a promising language for IoT applications [14].

In this paper, we propose an Erlang-based framework for the parallel processing of tasks in a cluster of IoT devices. These devices are able to communicate and report their resource availability, accepting computational tasks for execution. The goal is to have one of the connected devices requesting the offloading of tasks and relying on a module that coordinates all the communication process and balances the scheduling of tasks based on their estimated computational cost and the device's computational power and availability.

Resource allocation is one of the most complex problems in large multi-processor and distributed systems, and in general it is considered NP-hard. Computation platforms now integrate hundreds to thousands of processing cores, running complex and dynamic applications that make it difficult to foresee the amount of load they can impose to those platforms. Elementary combinatorics provides us with evidence of the problem of scale. For a simple formulation of the problem of allocating jobs to processors (one-to-one allocation), one can see that the number of allocations grows with the factorial of the number of jobs and processors.

A static allocation decided before deployment, based on the (nearly) complete knowledge about the load and the platform, is no longer viable. In traditional embedded systems, the workload is usually allocated in terms of its worst-case behaviour, but static allocations that take such characterisation into account tend to produce under-utilized platforms. It is, then, evident that optimal resource allocation algorithms cannot cope with this type of problem, and that lightweight heuristic solutions are needed. A comprehensive survey of the kinds of resource allocation heuristics that can cover different levels of dynamicity, while coping with the scale and complexity of high-density many-core platforms, is available in [15].

To cope with dynamism, a dynamic approach to resource management is the most obvious choice, aiming to dynamically learn and react to changes to the load characteristics and to the underlying computing platform. Linux has a strong momentum in the embedded software industry and has, in the past years, become the prevalent choice of operating system for new platforms. A paradigm, based on resource reservation, can endow applications with timing and throughput guarantees, independently of the good or malicious behavior of other applications, and can be employed across all system resources, including processor cycles, communication bandwidth, disk bandwidth, and storage.

The work presented here is the enhancement of previous work by the same authors [16,17] and the remaining of this paper is organized as follows. In the next section, we introduce the system model with the formal definitions for the network, communication protocol and scheduling behavior along with details of the orchestration process. Then, we describe the implementation of our system, and finally, we evaluate the results and conclude the paper.

2. System Model

We now proceed with the description of our system model by introducing formal definitions along with several considerations about its behavior. It is important to note that it is the programmer's responsibility to identify decomposable problems that can be used in this scenario. In a high level perspective, our system integrates the following features:

Data decomposition and assignment of data to nodes: Work is decomposed in several pieces, where the number of pieces is a function of the number of available nodes, and their size is proportional to each node's performance index.

Communication and failure management: There is a need to send data for processing to chosen nodes, to wait for the results and manage any eventual failures. Whenever a node fails, the work that was not processed is returned to the decomposition phase as a new instance of the process.

Mapping of results: Final result computation and its return to the application.

We use an orchestrator-based approach in order to achieve this. The details will be clarified in the following sections. We now proceed with some definitions and further explanations.

Definition 1 (Task)**.** *We define a task, t_i, as a λ function. By its nature, it will have no side effects and can be executed in parallel with other λ functions.*

In the remaining of this paper we will use the term task and lambda function for describing the same unit of execution and we use the term IoT device and node with the same meaning.

A device that needs to offload tasks to others can rely on a cluster of IoT devices for accomplishing this goal. We now define a cluster, which is the set of nodes currently available, meaning they are currently accepting tasks to execute.

Definition 2 (Cluster of IoT devices)**.** *Given an IoT device, we represent it by a node n_i. A cluster has a number of nodes, which can be variable during the execution of a computationally intensive application and is defined as $\mathcal{S} = \{n_1, \ldots, n_k\}$, where $k \geq 1$ and $n_i \in \mathcal{S}$ is one of the nodes currently available. The nodes can enter and leave the cluster at any time, as a result, for example, of a power failure (in case of leaving) or a new device is turned on (in case of entering).*

A cluster of nodes can be ordered from the more powerful to the less powerful members by evaluating their capabilities in terms of processing power and memory. Our option was to adopt a pragmatic approach, by implementing a simple heuristic function that relates clock speed, available CPU, number of cores, available RAM and available battery life. Details on how we get this data are described in the implementation section. We now define the device performance index.

Definition 3 (Device Performance Index)**.** *We define a function, $\mathcal{P}$, that given a node, n_i, its CPU speed, Cs_{n_i} (measured in Ghz), the number of cores, Cc_{n_i}, the available CPU capacity, Ca_{n_i} (measured in a number between 0 and 1), the available RAM, M_{n_i} (measured in Gigabytes), and the available battery, B_{n_i} (measured in a number number between 0 and 1), returns the value $\mathcal{P}(n_i)$, which is a numerical estimate for n_i performance based on the following formula:*

$$\mathcal{P} = \alpha * (Cs_{n_i} * Cc_{n_i} * Ca_{n_i}) + \beta * M_{n_i} + \delta * B_{n_i}$$

This is an easily computed value that, even if it is a relatively rough approximation, is, nevertheless, enough to distinguish each node's execution capacity without the burden of online benchmarking. It is also the programmer's responsibility to define adequate values for α, β and δ to produce an adequate value for his/her application.

Example 1. *Given a node, n_0, reporting the following data: $Cs_{n_0} = 1.4$, $Cc_{n_0} = 4$, $Ca_{n_0} = 0.6$, $M_{n_0} = 0.37$ and without battery information, and given $\alpha = \beta = 0.5$, the application of the formula results in:*

$$\mathcal{P} = 0.5 * (1.4 * 4 * 0.37) + 0.5 * 0.6 = 1.336$$

Example 2. *Given a node n_1 reporting the following data: $Cs_{n_1} = 1.5$, $Cc_{n_1} = 4$, $Ca_{n_1} = 0.15$, $M_{n_1} = 0.54$, the application of the formula results in:*

$$\mathcal{P} = 0.5 * (1.5 * 4 * 0.54) + 0.5 * 0.15 = 1.695$$

Knowing each node's performance index, we now define how to decompose the problem in order to distribute it in a balanced manner.

Definition 4 (Simple Problem Decomposition)**.** *Given a problem $\mathcal{D}$ and given a cluster of available nodes $\mathcal{S} = \{n_1, \dots, n_k\}$, then the problem must be decomposable in k parts and defined as $\mathcal{D} = \{d_1, \dots, d_k\}$ such that each part's computational cost is proportional to the assigned device performance index.*

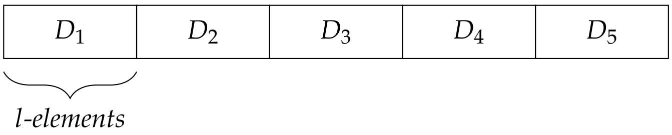
Example 3. *Given nodes $n_0, \dots, n_5$ and a problem of summing 100,000 numbers, the calculated performance index, the percentage of the computational power each node represents and the assigned partition of the problem is presented in the following table:*

Node	*Performance Index ($\mathcal{P}_i$)*	*Percentage of System Power (p_i)*	*Assigned Partition*
n_0	*2.013*	*21%*	*21,000*
n_3	*1.965*	*20%*	*20,000*
n_1	*1.695*	*18%*	*18,000*
n_4	*1.472*	*15%*	*15,000*
n_2	*1.336*	*14%*	*14,000*
n_5	*1.125*	*12%*	*12,000*

A strict decomposition can be a bad solution if the computational cost of processing data is unevenly distributed, since a small interval of data can be harder to process than a larger one. The approach we purpose includes the option to split the work in a bounded number of parts that are processed sequentially by the cluster of nodes. We now define the enhanced problem decomposition.

Definition 5 (Enhanced Problem Decomposition)**.** *Given a problem, $\mathcal{D}$, and given a cluster of available nodes, $\mathcal{S} = \{n_1, \dots, n_k\}$, then the problem must be decomposable in n parts and defined as $\mathcal{D} = \{D_1, \dots, D_n\}$ and for each $D_i \in \mathcal{D}$, it is possible to decompose it further into k parts and defined as $D_i = \{d_{i1}, \dots, d_{ik}\}$, such that each part's computational cost is proportional to the assigned device performance index. Thus, given a node, n_i, with a percentage of system power, p_i, then the size of the part, D_i, it will process is given by $p_i * sizeof(D_i)$.*

Example 4. *Given the Example 3, if we choose to have five partitions, then we get:*



Here, each node n_i will process $p_i * l$ elements of each D_i corresponding to:

Node	*Performance Index ($\mathcal{P}_i$)*	*Percentage of System Power (p_i)*	*Part D_k Size*
n_0	*2.013*	*21%*	*4200*
n_3	*1.965*	*20%*	*4000*
n_1	*1.695*	*18%*	*3600*
n_4	*1.472*	*15%*	*3000*
n_2	*1.336*	*14%*	*2800*
n_5	*1.125*	*12%*	*2400*

Given the previous definitions we can now define the assigned problem.

Definition 6 (Assigned Problem)**.** *Given a cluster of nodes $\mathcal{S} = \{n_1, \ldots, n_k\}$, where each node n_i has a performance index, $\mathcal{P}_i$, and the problem, $\mathcal{D}$, which is decomposed in k different parts, we define an assigned problem as a set of triples, $\mathcal{A}_P = \{(n_1, \mathcal{P}_1, d_1) \ldots (n_k, \mathcal{P}_k, d_k)\}$.*

Communication between nodes is done using asynchronous message passing. There is a permanent link between the node requesting the work and the nodes executing that work. When this link is broken, it signals a loss of communication and the node is removed from the list of available ones.

Definition 7 (Link set)**.** *Given a cluster of available nodes $\mathcal{S} = \{n_1, \ldots, n_k\}$ we define $\mathcal{L} = \{l_1, \ldots, l_k\}$ as the list of links to the nodes such that the connection to node n_k is done by link l_k.*

Failure during the execution of a task results in rescheduling the unfinished task to the closest available node in terms of performance index. More formally, we define task reassignment.

Definition 8 (Task Reassignment)**.** *Given a cluster of nodes, $\mathcal{S} = \{n_1, \ldots, n_k\}$, where each node, n_i, has a performance index, $\mathcal{P}_i$, the problem, $\mathcal{D}$, decomposed in k different parts proportional to each of the nodes and the assigned problem $\mathcal{A}_P = \{(n_1, \mathcal{P}_1, d_1) \ldots (n_k, \mathcal{P}_k, d_k)\}$. When a link, l_j, assigned to a node, n_j, such that $(n_j, \mathcal{P}_j, d_j) \in \mathcal{A}_P$ fails, then the task, d_j, is reassigned to the node, n_m, such that $(n_m, \mathcal{P}_m, d_m) \in \mathcal{A}_P \setminus (n_j, \mathcal{P}_j, d_j)$ and $\mathcal{P}_m \geq \mathcal{P}_n$ for any $(n_n, \mathcal{P}_n, d_n) \in \mathcal{A}_P \setminus (n_j, \mathcal{P}_j, d_j)$.*

The orchestrator plays a central role in the system. It is responsible for the coordination of the different participants, dealing with the details of each device and providing the programmer with an API that abstracts the use of the distributed system. Thus, its main features are:

- communication between all the participants;
- adding nodes to the cluster and removing nodes from the cluster;
- task distribution; and
- fault tolerance.

The orchestrator relies on the host that needs to offload work to other nodes. We now describe in more detail the concepts behind each of its features and other relevant details will be clarified in the implementation section.

Communication is done by message passing. All the different participants behave like actors [18]. All the messaging relies on the built-in features of the Erlang language that provide high-level approaches to message passing and code distribution, facilitating the whole process.

IoT devices can enter and leave the cluster at anytime. When they enter, they are available to accept tasks to execute. The node starts by sending a registration message to the orchestrator and, after acknowledgment, sends its score, which results from a performance index computation in the node. This will allow the orchestrator to rank that specific node within the cluster. Nodes can leave the cluster in two different scenarios: (i) when the orchestrator is shutdown; and (ii) when they stop, for example, due to power failure. In the first case, a message is sent from the orchestrator to the node, terminating the collaboration process. In the second one, the orchestrator detects the node's failure and removes it from the list of available nodes. Again, these features rely strongly on the built-in features of Erlang.

The API that the orchestrator provides accepts code and data, and returns the result of applying the code to the data. Its main goal is to distribute the data by node. It starts by splitting the data by the different nodes using their rank (given by the score obtained by the computed device performance index) to create partitions of data that each one will process.

This is illustrated in Figure 1, where we have an input to the orchestrator consisting of code and data, it distributes the code by the different nodes $\{N_1, N_2, \ldots, N_n\}$ and data $\mathcal{D}$ = $\{D_1, D_2, \ldots, D_n\}$, such that they can process it locally. For each N_i, the size of D_i varies accordingly with N_i's rank.

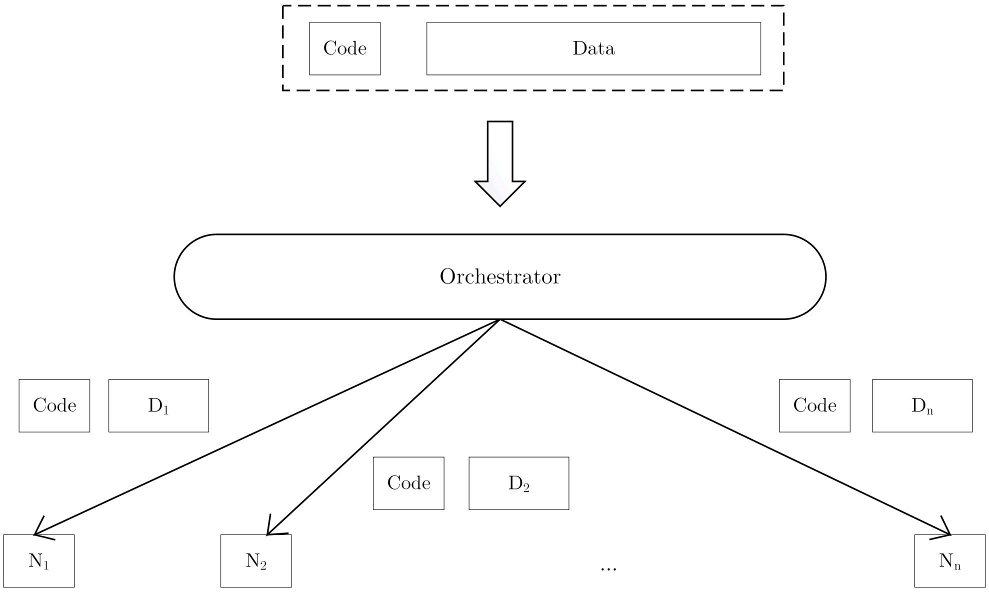


Figure 1. Orchestration process.

It is also the role of the orchestrator to provide fault-tolerance. Here, fault-tolerance consists in guaranteeing that all the data is processed. Since data is split among different nodes, failure in one or more than one node during execution results in losing the corresponding partial result. To guarantee the completion of the designated task, the orchestrator maintains a permanent link with each node in the cluster and detects any failure. In case of failure, the task given to that node is rescheduled for execution in the available node with highest performance index.

3. Data and Code Distribution Algorithm

We now present the core algorithms of this framework. Algorithm 1 describes how data is split and Algorithm 2 describes how data is distributed among the nodes. In Algorithm 1 we start by choosing from the simple problem decomposition of data (Definition 4) or the enhanced problem decomposition (Definition 5). In the first case, we split data by the number of available nodes proportionally to each node's performance index. On the other hand, if the enhanced problem decomposition is chosen, then an additional parameter (here described by the variable k) is provided, allowing a first split of the partition into k parts, then, for each of these parts, the data is split again, now in p parts (given p, the number of available nodes) with each of these parts with a size proportional to the nodes' performance indexes. This allows a more fine-grained distribution to the computational power of nodes with respect to the data being processed, which is particularly useful when the processing data has an uneven processing cost.

Having all the parts of data defined, we proceed with the distribution of the data and the supplied code for processing the data (described as $\mathcal{F}$) by the different nodes as described in Algorithm 2. The result is then stored. In case of a node being unable to complete a task, which translates into a broken link, the associated data must be processed by another node. The approach we use is to give it to the available node with highest performance to minimize further delays. In the case of several nodes failing, the process is repeated and the data is queued to the best available node. The algorithm hides most of the low-level technicalities which will be further discussed in the following sections.

Algorithm 1 Data Split

```
Let $\mathcal{D} := \{D_1, \ldots, D_p\}$ be the set of data which is divided in $p$ parts.
Let $\mathcal{D}' := \{\}$ be an empty set of data pairs.

if Simple Problem Decomposition then
    $p := n$ and the size of partition $D_i$ is adjusted to be proportional to $p_i$.
    $\mathcal{D}' := \mathcal{D}' \cup \{(D_1, n_1), \ldots, (D_p, n_p)\}$
end if
if Enhanced Problem Decomposition then
    A size $k$ is provided, $p := k$ and
    for each $D_j \in \mathcal{D}$ do
        Split $D_j$ in $\{d_{j1}, \ldots, d_{jp}\}$ where each $d_{ji}$ is proportional to $p_i$.
        $\mathcal{D}' := \mathcal{D}' \cup \{(d_{j1}, n_1), \ldots, (d_{jp}, n_p)\}$
    end for
end if
return $\mathcal{D}'$
```

Algorithm 2 Data Distribution

```
Let $\mathcal{S} := \{n_1, \ldots, n_n\}$ be the set of available nodes in the cluster.
Let $\mathcal{L} := \{l_1, \ldots, l_n\}$ be the set of links to nodes in the cluster, where $l_i$ is the link to node $n_i$.
Let $\mathcal{S_P} := \{(n_1, p_1), \ldots, (n_n, p_n)\}$ be the set of pairs of available nodes in the cluster where each $n_i$ is the node name and $p_i$ is node's $i$ performance index.
Let $\mathcal{D}'$ be the result of execution of the previous algorithm.
Let $\mathcal{F}$ be a function to process data in $\mathcal{D}$.
Let $\mathcal{R} := \{\}$ be an empty set of results of processing data in $\mathcal{D}$ by function $\mathcal{F}$.
Let $Success := False$

while Success = False do
    while $\mathcal{D}'$ has data do
        Remove $(d_a, n_b)$ from $\mathcal{D}'$
        Submit data $d_a$ for execution by $n_b$ with code $\mathcal{F}$
    end while
    Wait until all nodes return a response and add them as a tuple $(n_i, r_i)$ to $\mathcal{R}$, where $r_i$ is the value returned by node $n_i$.
    if failed links exist then
        Add unprocessed requests $(d_a, n_b)$ as $(d_a, n_c)$ to $\mathcal{D}'$, where node $b$ is replaced by the best one available, $c$.
    else
        Success := True
    end if
end while
```

4. Implementation

Although the idea is to have a general purpose solution for IoT devices, at this moment, we decided to focus on a specific type of hardware/software to develop a proof of concept with all the properties we believe that are relevant in this domain. Our nodes are all single-board computers, namely Raspberry Pi devices [19]. They all run a Linux distribution and an Erlang virtual machine.

Although single-board computers (SBC) are just one type of IoT device, they enjoy enormous popularity due to their high performance for their price range and the vast number of scenarios where they can be used [19,20]. It is possible to have several Raspberry Pi SBCs in the same area, each with a different purpose. With our framework we enable the optimization of devices that are, often, sitting idle.

4.1. IoT Node Implementation

The IoT node must have installed the Erlang VM to allow code transfer, execution and communication. Whenever an IoT device is available for collaboration, it searches for a registered orchestrator (in Erlang, registered processes are those that have a name associated with them) and uses the Erlang built-in instruction, $net\_adm : ping('orchestrator\_name')$, to register with the orchestrator. On success, the ping will add the IoT device to the list of known neighbors in the orchestrator process. Then, the device will compute its performance index using *sysbench* (https://github.com/akopytov/sysbench, accessed on 22 November 2021) and Linux's integrated *acpitools* and sends it to to the orchestrator. This allows the orchestrator to rank the node. From here on, the node is ready to be used as part of the cluster.

The code deployed to a node is initially minimal and consists of a simple process that executes code as instructed by received messages and is described in Listing 1.

Listing 1. IoT node main code.

```
task_executor()->
receive
{ From,execute ,Mod,Fun,Param } ->
From ! { B,E,Mod:Fun(Param) } ,
task_executor();
_ ->
task_executor()
end .
```

The function, $task\_executor/0$, waits for messages instructing the node to execute code (function Fun from module Mod with parameters $Param$) and the result of the execution is returned to the requesting node.

4.2. Orchestrator Implementation

The orchestrator is executed on the node that needs to offload data. This node, as all the others nodes that may form a cluster, runs an Erlang VM and the orchestrator is activated whenever it needs to offload work to others. After activation, the orchestrator is able to build a cluster of IoT devices and knows each one's performance score. Therefore, it can split tasks accordingly. In more detail, after knowing the list of available nodes $\mathcal{S}$ (nodes that successfully registered and sent their performance index), the first step is to create a ranked list. Given a list of nodes, $\mathcal{S} = \{n_1, \ldots, n_n\}$, they will be ordered in a list from the one with highest performance index to the one with lowest performance index. Their relative computational power will be used to compute the partition they must handle.

The orchestrator then proceeds with the transfer of code, $\mathcal{C}$, and data, $\mathcal{D}$, to nodes in the cluster. Erlang provides an easy way to do this. Given a module, Mod, on the device where the orchestrator is running, the Erlang instruction $c : nl(Mod)$ will transfer it to the nodes in $\mathcal{A}$. After this step, all the nodes have the same version of the code and data and know the partition they will work on, which is also sent by the orchestrator.

Please note that this approach may not escalate well when the problem being decomposed does not have an even distribution of work. It is also unable to split inter-dependent pieces of data. Nevertheless, these type of problems can also be handled by the orchestrator, but data will not be partitioned. Instead, all the application will be transferred to the node with the highest performance index.

After all the work has been distributed, the orchestrator waits for the results and handles failures. Waiting for results means it will wait for an answer from each node with the result of its particular execution. In case there is some failure, for example one node disconnects, it detects this because it relies on the underlying Erlang mechanism that generates a message whenever one element of $\mathcal{S}$ stops working. Thus, it is easy to know if one node stopped working and which part of the data it was processing. In this case, the orchestrator reschedules this exact execution to another node. The criteria implemented is to wait until the end of the current execution and reschedule the uncompleted one to the highest ranked node.

The orchestrator node coordinates the offloading process and, typically, nodes can be both an orchestrator and a node executing tasks. The orchestrator Erlang function is succinctly described in Listing 2.

Listing 2. Orchestrator node main code.

```
orchestrator(Mod,Fun,DataSize,NPart) ->
NodesList = initialize_cluster(),
c:nl(Mod),
Parts_Node = distribute(NodesList,DataSize,NPart),
Results = execute(Mod,Fun,Parts_Node),
resechedule(Results,Parts_Node).
```

The function *orchestrator*/4 receives the name of the module, *Mod*, with the code and data that must be distributed, the main function processing the data, *Fun*, the size of the data being processed, *DataSize*, and the number of partitions, *NPart*, that should be used in the distribution of the work. Next, the function *initialize_cluster*/0 finds nodes in the local network area that are able to collaborate (execute the slave function described before), and adds them to the *NodesList*, establishing a link. Code and data are then distributed to the available nodes with the Erlang builtin function $c : nl/1$ and *distribute*/3 determines which parts of the data partition must be processed by which nodes. In case *NPart* is one, the process is a simple problem decomposition, if $Npart > 1$, then the process is an enhanced problem decomposition. The next step is to send the data for remote execution and gather all the results in the *Results* list. Finally, the function *resechedule*/2 compares the results obtained from nodes with the requests that were made. In case it detects unanswered requests (resulting from nodes failing during execution), tasks related with those requests are rescheduled to available nodes as described in the system model.

Example 5. *Given a module, primes, where a function, sum_primes(B, E, List), returns the sum of the prime numbers in the interval from B to E in List and given a list of 100,000 numbers, summing all the primes in the list can be done using the cluster of IoT devices available, by submitting the following instruction to the orchestrator:*

```
orchestrator:process(primes,sum_primes,1000000,1)
```

By calling this function, all the features previously described are used to create a collaborative effort of handling the problem.

5. Evaluation

We carried several tests using a cluster of four devices connected wirelessly to the same WiFi router. These four IoT devices were in use with different main applications as described in Table 1.

Table 1. Cluster Setup.

Node	Device	CPU	Clock	RAM	Application
n_1	Raspberry Pi 3 B+	quad-core	1.4 GHz	1.0 GB	arcade machine
n_2	Raspberry Pi Zero W	single-core	1.0 GHz	0.5 GB	network add-blocker
n_3	Raspberry Pi 3 A+	quad-core	1.4 GHz	0.5 GB	wireless print server
n_4	Raspberry Pi Zero W	single-core	1.0 GHz	0.5 GB	no application

To evaluate the benefits of our framework, we have developed a battery of tests based on data-decomposable problems. Since results are consistent among the several tests, here, we present one of them, consisting of counting prime numbers in the interval [1, 50,000].

The interval we use in this test is $\mathcal{I} = \{1, \ldots, 50,000\}$ with a total of 5133 prime numbers. Note that the primes are not evenly distributed in this interval and higher ones are considerably more difficult to find than lower ones. We started with one device and added devices to the cluster measuring the gain in performance. Since the problem is not easy to break in balanced partitions we use the enhanced problem decomposition solution and break it in several partitions (1, 5, 10 and 20). Each of the partitions is then split by the available nodes accordingly with their reported performance index. We choose the node n_1 to be the node running the orchestrator although it can also execute tasks. The calculation of the primes on $\mathcal{I}_1$ took an average of 26.07 s. We don't get any advantage in using more than one partition with only one device since the implementation already uses multiple processes to optimize the use of the available cores of the device. The results of distributing data by the nodes are presented in Figure 2.

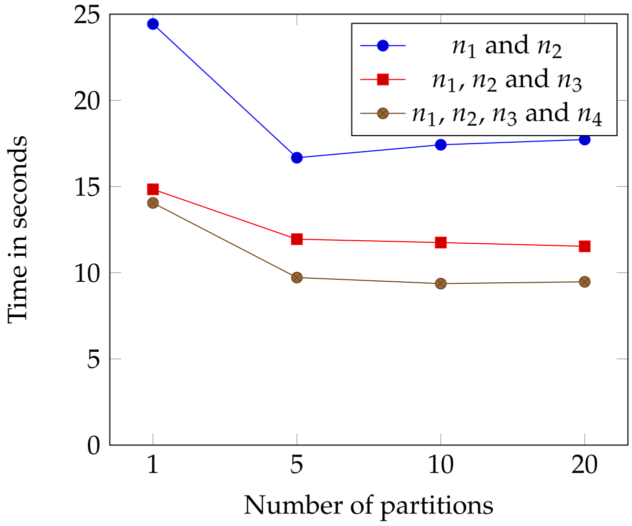


Figure 2. Distributing data by nodes.

Note that, if we add one device (n_1 and n_2), even with only one partition, the time needed to compute all the primes decreases from an average of 26.07 s to an average of 24.43 s. The performance increases as we divide chunks of work by the devices. With five partitions, we achieve the best result of an average 16.67 s. The increase in the number of partitions is not alone a factor of enhancement in performance, since, the more partitions we have, the more messages we need to exchange. When using devices n_1, n_2 and n_3, the performance increases considerably, which seems normal since n_3 is a powerful node in this context. Adding node n_4 also further increases the cluster's performance. In Table 2, one can see the percentage of gain in terms of execution time when adding, one, two and three devices to the cluster. The advantage is evident, although the gain when adding three devices when compared to two devices is marginal. One thing we notice is that problems have, generally, an ideal number of devices in the cluster to get the better trade-off between the size of the problem and the cluster setup and communication overhead.

Table 2. Percentage of gain by adding devices.

Devices Added	Gain When Compared to Single Device
1	55.8%
2	64.1%
3	65.2%

In Figure 3 we present the graphic detailing the gain of having one device (no devices added to help), with one, two and three devices added. These values are the ones for the configuration with best performance in each of the scenarios.

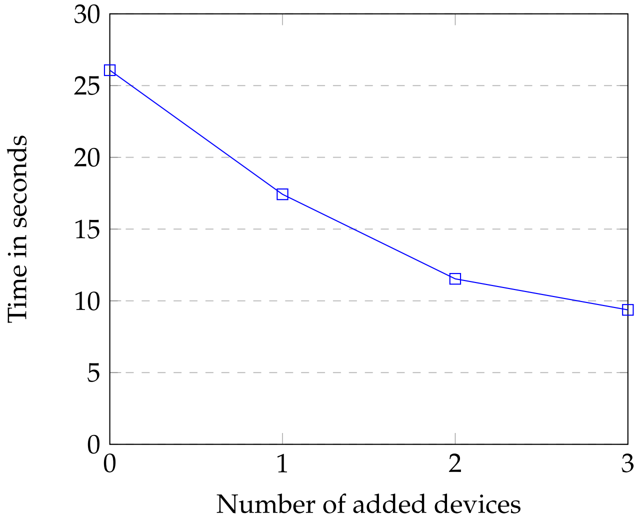


Figure 3. Result of adding devices to n_1.

We also experimented with failure in nodes and consequent rescheduling. The impact of such operation is highly dependent on the capacity of the node or of the nodes failing. Failing node n_3 has a considerable higher impact than failing node n_2, due to their different capacity and thus the amount of work that is distributed to them. Nevertheless, with a small number of failures, the cooperative distributed computation still has a better performance when compared to the single problem solving solution. In terms of time needed for the execution we conclude that given a set of nodes $\{n_1, \ldots, n_k\}$, the time, X, needed to decompose the service, the time, Y, to send code and data to a node, the time, W, to send the result back from a node and, t_{n_i}, the time that node n_i needs to processes its block, the total time, $\mathcal{T}$, needed for a distributed service execution can be determined by:

$$\mathcal{T} = X + k * Y + \sum_{i=1}^{k} \frac{t_{n_i}}{k} + k * W$$

From a general perspective and focusing only in the framework we developed and not the problems it may solve, we are able to draw some conclusions about the scalability of our framework. The division of work is done in two different approaches, one using a simplified split of data based on the performance index of the available nodes, and another based on the split in a given number of partitions and again, the split of each partition given the number of available nodes and their performance index. Both operations have low complexity since they have no relation with the size of the data being processed. In terms of communication and message passing the number of messages needed to setup the framework are equal to the number of available nodes and the number of messages needed to send and receive data are twice the number of partitions assigned to each node. Thus, the complexity of the whole framework setup and distribution of data and code is low. In the case of the enhanced problems assignment and depending on the data being processed, there is a compromise between the number of partitions given to each node and the time of the execution of the code over the data. Sometimes, depending on the data it may be faster to have less partitions, avoiding the communication overhead. General complexity of distributed systems has been previously studied in research such as [21–23].

6. Conclusions

Even though IoT devices are becoming more powerful, the available local resources cannot cope with the increasing computational requirements of resource-intensive applications that can be offered to a large range of end-users. This has created a new opportunity for task offloading, where computationally intensive tasks need to be offloaded to more resource powerful devices. Naturally, cloud computing is a well-tested infrastructure that can facilitate the task offloading. However, cloud computing, as a centralized and distant infrastructure, creates significant communication delays that cannot satisfy the requirements of the emerging delay-sensitive applications.

To this end, in this paper we presented a cooperative framework for IoT devices based in Single Board Computers and the Erlang programming language. The goal is to maximize the collaborative power of these devices with a minimal setup and interference on their main functions. By distributing the computational load across a set of heterogeneous IoT nodes, a cooperative environment enables the execution of more complex and resource-demanding services that otherwise would not be able to be executed on a stand-alone basis or would suffer from unacceptable performance. We intend to add more features to the framework and foresee the creation of a distributed solution for computation that uses available power of simple devices replacing larger systems.

Author Contributions: Conceptualization, J.C. and L.N.; methodology, J.C. and L.N.; software, J.C.; validation, L.N.; investigation, J.C. and L.N.; writing—original draft preparation, J.C.; writing—review and editing, J.C. and L.N.; visualization, J.C. and L.N.; supervision, J.C.; project administration, J.C. and L.N.; funding acquisition, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Artificial Intelligence and Computer Science Laboratory, University of Porto (LIACC), FCT/UID/CEC/0027/2020, funded by national funds through the FCT/MCTES (PIDDAC).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cheng, C.; Lu, R.; Petzoldt, A.; Takagi, T. Securing the Internet of Things in a Quantum World. *IEEE Commun. Mag.* **2017**, *55*, 116–120. [CrossRef]
2. Nogueira, L.; Coelho, J. Self-organising Clusters in Edge Computing. In *Intelligent Systems Applications in Software Engineering*; Silhavy, R., Silhavy, P., Prokopova, Z., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 320–332.
3. Khan, M.A. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *J. Netw. Comput. Appl.* **2015**, *56*, 28–40. [CrossRef]
4. Kumar, S.; Tyagi, M.; Khanna, A.; Fore, V. A Survey of Mobile Computation Offloading: Applications, Approaches and Challenges. In Proceedings of the 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), Paris, France, 22–23 June 2018; pp. 51–58. [CrossRef]
5. Noor, T.H.; Zeadally, S.; Alfazi, A.; Sheng, Q.Z. Mobile cloud computing: Challenges and future research directions. *J. Netw. Comput. Appl.* **2018**, *115*, 70–85. [CrossRef]
6. Meurisch, C.; Gedeon, J.; Nguyen, T.A.B.; Kaup, F.; Muhlhauser, M. Decision Support for Computational Offloading by Probing Unknown Services. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–9.
7. Erl, T. *Service-Oriented Architecture: Concepts, Technology, and Design*; Prentice Hall PTR: Hoboken, NJ, USA, 2005.
8. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A Survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
9. Cesarini, F.; Vinoski, S. *Designing for Scalability with Erlang/OTP: Implement Robust, Fault-Tolerant Systems*, 1st ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2016.
10. Terrell, R. *Concurrency in NET: Modern Patterns of Concurrent and Parallel Programming*, 1st ed.; Manning Publications Co.: Shelter Island, NY, USA, 2018.
11. Warburton, R. *Java 8 Lambdas: Pragmatic Functional Programming*, 1st ed.; O'Reilly Media, Inc.: Greenwich, CT, USA, 2014.
12. Hausenblas, M. *Serverless Ops*; O'Reilly Media, Inc.: Greenwich, CT, USA, 2016.

13. Armstrong, J. A History of Erlang. In Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL III), San Diego, CA, USA, 9–10 June 2007; ACM: New York, NY, USA, 2007; pp. 6-1–6-26. [CrossRef]
14. Kopestenski, I.; Van Roy, P. Erlang as an Enabling Technology for Resilient General-Purpose Applications on Edge IoT Networks. Proceedings of the 18th ACM SIGPLAN International Workshop on Erlang (Erlang 2019), Berlin, Germany, 18 August 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 1–12. [CrossRef]
15. Indrusiak, L.; Dziurzanski, P.; Singh, A. *Dynamic Resource Allocation in Embedded, High-Performance and Cloud Computing*; River Publishers: Delft, The Netherlands, 2016. [CrossRef]
16. Coelho, J.; Nogueira, L. Orchestration of Clusters of IoT Devices with Erlang. In *Software Engineering Perspectives in Intelligent Systems*; Silhavy, R., Silhavy, P., Prokopova, Z., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 585–594.
17. Coelho, J.; Nogueira, L. Collaborative Task Processing with Internet of Things (IoT) Clusters. In *Science and Technologies for Smart Cities*; Paiva, S., Lopes, S.I., Zitouni, R., Gupta, N., Lopes, S.F., Yonezawa, T., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 293–304.
18. Agha, G. *Actors: A Model of Concurrent Computation in Distributed Systems*; MIT Press: Cambridge, MA, USA, 1986.
19. Johnston, S.J.; Basford, P.J.; Perkins, C.S.; Herry, H.; Tso, F.P.; Pezaros, D.; Mullins, R.D.; Yoneki, E.; Cox, S.J.; Singer, J. Commodity single board computer clusters and their applications. *Future Gener. Comput. Syst.* **2018**, *89*, 201–212. [CrossRef]
20. Jabbar, W.A.; Wei, C.W.; Azmi, N.A.A.M.; Haironnazli, N.A. An IoT Raspberry Pi-based parking management system for smart campus. *Internet Things* **2021**, *14*, 100387. [CrossRef]
21. Ranganathan, A.; Campbell, R.H. What is the complexity of a distributed computing system? *Complexity* **2007**, *12*, 37–45. [CrossRef]
22. Fraigniaud, P.; Korman, A.; Peleg, D. Towards a Complexity Theory for Local Distributed Computing. *J. ACM* **2013**, *60*, 1–26. [CrossRef]
23. Cabri, G.; Leonardi, L.; Quitadamo, R. Tackling Complexity of Distributed Systems: Towards an Integration of Service-Oriented Computing and Agent-Oriented Programming. In Proceedings of the 2008 International Multiconference on Computer Science and Information Technology, Wisla, Poland, 20–22 October 2008; pp. 9–15. [CrossRef]

electronics

MDPI

Article

FP-Growth Algorithm for Discovering Region-Based Association Rule in the IoT Environment

Hong-Jun Jang [1], Yeongwook Yang [2], Ji Su Park [1] and Byoungwook Kim [3,*]

[1] Department of Computer Science and Engineering, Jeonju University, Jeonju 55069, Korea; hongjunjang@jj.ac.kr (H.-J.J.); jisupark@jj.ac.kr (J.S.P.)
[2] Division of Computer Engineering, Hanshin University, Osan 18101, Korea; yeongwook.yang@hs.ac.kr
[3] Department of Computer Science and Engineering, Dongshin University, Naju 58245, Korea
* Correspondence: bwkim@dsu.ac.kr; Tel.: +82-61-330-3358

Abstract: With the development of the Internet of things (IoT), both types and amounts of spatial data collected from heterogeneous IoT devices are increasing. The increased spatial data are being actively utilized in the data mining field. The existing association rule mining algorithms find all items with high correlation in the entire data. Association rules that may appear differently for each region, however, may not be found when the association rules are searched for all data. In this paper, we propose region-based frequent pattern growth (RFP-Growth) to search for association rules by dense regions. First, RFP-Growth divides item transaction included position data into regions by a density-based clustering algorithm. Second, frequent pattern growth (FP-Growth) is performed for each transaction divided by region. The experimental results show that RFP-Growth discovers new association rules that the original FP-Growth cannot find in the whole data.

Keywords: FP-Growth algorithm; association rules; frequency pattern analysis

Citation: Jang, H.-J.; Yang, Y.; Park, J.S.; Kim, B. FP-Growth Algorithm for Discovering Region-Based Association Rule in the IoT Environment. *Electronics* **2021**, *10*, 3091. https://doi.org/10.3390/electronics10243091

Academic Editors: Kevin Lee and Ka Lok Man

Received: 28 October 2021
Accepted: 7 December 2021
Published: 12 December 2021

1. Introduction

With the rapid development of mobile devices and sensor technology, various forms and vast amounts of spatial data are being collected in the IoT environment [1]. As the amount of spatial data collected increases in the IoT, the demand for using spatial information is also increasing in fields where spatial information has not been utilized before [2]. However, many IoT applications require short response times and depend on devices with limited resources, so the application of existing data mining techniques is inefficient and limited [3–6]. Research on spatial data mining techniques to obtain knowledge specific to a region using the physical location information of the sensor from which data are collected, is being actively conducted.

Frequent pattern (FP) mining has been extensively studied in the field of data mining. Apriori algorithm has received a lot of attention in the field of data mining [7–9]. However, Apriori-based approaches have the disadvantage that they generate many candidate sets and are expensive due to frequent database scans. In order to overcome this drawback, many papers have proposed a new data structure that calculates frequency itemsets from a transactional database. One of the most popular of these data structures is the FP-Tree structure [10]. FP-Growth algorithm, which is a data mining technique based on FP-Tree, can discover a set of complete frequency patterns. FP-Tree is an extended prefix-tree structure to store important and quantitative information related to frequency patterns, avoiding the shortcomings of the Apriori-based approach. FP-Tree has the advantage of low tree construction cost by creating a tree with two scans of the entire database. Thus, FP-Growth algorithm is faster than the Apriori algorithm. FP-Tree requires two database scans and cannot be applied to a variable database because the frequency of occurrence of items must be obtained through a full database scan before constructing a tree. In order to overcome these disadvantages, many methods of generating frequency pattern trees have been studied. The FP-Stream [11] structure is proposed to apply the existing FP-Tree in the streaming database, and the COFI-tree [12], which allows the conditional tree to be generated to a minimum through pruning during tree generation, and DRFP-Tree [13,14] to use a database instead of memory, and CanTree [15], which constructs a tree using alphabetical specific criteria as a method to reduce database scans to obtain the number of occurrences of items in FP-Tree.

A representative example of association rule mining is market basket analysis [16]. However, in the existing market basket analysis, one transaction only has a list of purchased items, not a region for purchasing the items. Table 1 shows an example of the item purchase region added to the item transaction used in the existing market basket analysis. If the purchase region is not considered in the market basket analysis, the support of beer→diaper is 0.5 (5/10) and the confidence of beer→diaper is 0.5 (4/8). If minimum support and confidence are set higher than 0.5 in order to prevent the generation of massive association rules, beer→diaper will not be derived.

Table 1. An example of item transaction considering the purchase location.

TID	Market	Items
1	A	Bread, Milk, Peanut
2	A	Bread, Diaper, Beer, Eggs, Peanut
3	B	Milk, Diaper, Beer, Cola
4	B	Bread, Milk, Diapers, Beer
5	C	Diaper, Beer, Eggs
6	D	Bread, Beer, Peanut, Eggs
7	D	Beer, Milk, Peanut
8	E	Beer, Peanut, Diaper
9	E	Bread, Milk, Cola, Eggs
10	F	Beer, Milk, Peanut, Cola, Eggs

In Figure 1, Markets A, B, C are close, and Markets D, E, F are close by distance. Depending on the density of the market and house, dense markets form a cluster, e.g., C_1 and C_2, and the cluster can be assumed as one commercial district. We can get the confidence and the support of beer→diaper for each cluster.

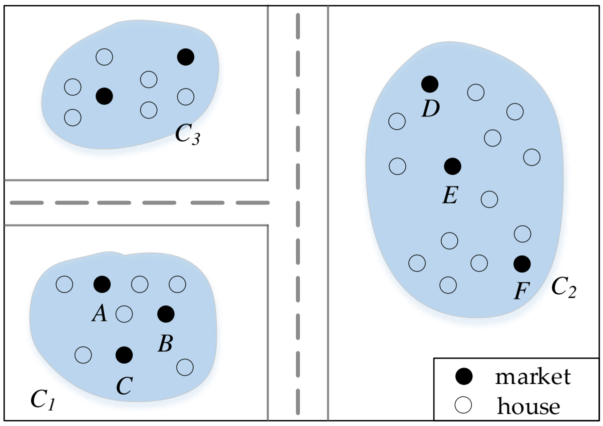


Figure 1. An example of association rule mining considering purchase location.

The support of beer→diaper is 0.8 (4/5), and the confidence of beer→diaper is 1 (4/4) in C_1. The support of beer→diaper is 0.2 (1/5), and the confidence of beer→diaper is 0.25 (1/4). Even if the minimum support and confidence are set to 0.5, we can find the association rule with beer→diaper [sup: 0.8, conf: 1]. For example, data analysts can infer that there is a lot of households with babies around the C_1 commercial district from these association rules. With this knowledge, it is possible to set up a strategy for promoting baby products in the market of the relevant commercial district. In this way, association rules that were not discovered when analyzing the entire data can be discovered in data generated in a specific region. Association rules discovered in a specific region can be used as information to analyze the purchasing patterns or behavioral characteristics of consumers in that region. Until now, many association rule algorithms have been developed to discover association rules in the entire data, but no algorithm has yet been proposed to discover association rules effectively in partial transactions.

In this paper, we propose region-based FP-Growth (RFP-Growth) that discovers frequent patterns for each divided cluster after dividing the entire transaction data into density-based clusters. RFP-Growth algorithm generates FP-Tree with only transactions in those regions when the regions to find the association rule is selected. RFP-Growth discovers discover new frequent rules that were not found in the whole data. The contributions of this paper are summarized as follows.

- We proposed a novel problem of discovering association rules in item transactions considering the item purchase location.
- We proposed RFP-Growth which organizes item transaction data with location data into clusters by dense regions and discovers association rules for each cluster.
- We conducted extensive experiments on the real and synthetic datasets to prove that RFP-Growth discovers the new frequent rules that are not discovered in the analysis of the entire data.

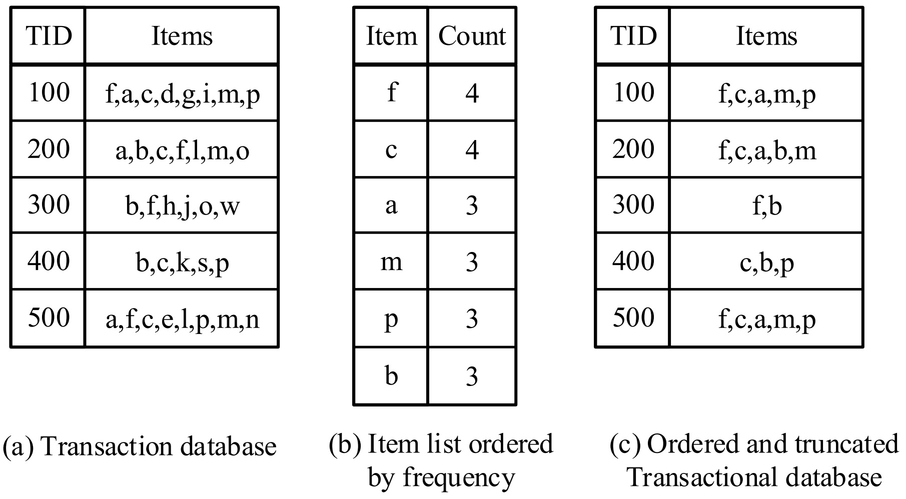
The rest of this paper is organized as follows. Section 2 reviews related works to spatial clustering algorithms with FP-Growth and defines the problem. In Section 3, we describe an overview of RFP-Growth. In Section 4, we present experimental results and their evaluation. In Section 5, we conclude our work and present some directions for future research.

2. Background and Related Works

2.1. Background: FP-Growth

Apriori algorithm is the most representative algorithm for association rules and is a useful algorithm for finding frequent itemsets for binary association rules [9,17]. However, since candidate itemsets are repeatedly generated and the support is calculated while scanning the database, a lot of processing time is consumed. To compensate for this drawback, several studies have been conducted to reduce the number of candidate sets or the number of database scans. FP-Growth algorithm is attracting attention because it can analyze frequent patterns with only two database scans without generating a candidate set [10,18]. Important and quantitative information about frequent itemsets is stored in an extended prefix tree structure called FP-Tree. FP-Tree generates a frequency pattern tree with only two database scans. The priority of items is determined by counting the number of frequent occurrences of each item through the first database scan. Each set of items entered through the second database scan is sorted using the number of frequent occurrences.

A simple example of constructing FP-Tree is shown in Figure 2. Figure 2a is an example transaction database for creating FP-Tree (the minimum support is set to three). Each row is composed of a set of items that occur simultaneously within one transaction and is classified by transaction identification (TID). To construct FP-Tree using this example transactional database, first, we need to find the frequency of the items. The database is scanned once for the first time to count the number of items represented in the database. Then, in order to make a list of frequent items, only items with a minimum support rating or higher are used to create the FP-Tree in the order of the highest frequency. Figure 2b shows items with a frequency greater than or equal to the minimum support in the order of the highest frequency.



TID	Items
100	f,a,c,d,g,i,m,p
200	a,b,c,f,l,m,o
300	b,f,h,j,o,w
400	b,c,k,s,p
500	a,f,c,e,l,p,m,n

(a) Transaction database

Item	Count
f	4
c	4
a	3
m	3
p	3
b	3

(b) Item list ordered by frequency

TID	Items
100	f,c,a,m,p
200	f,c,a,b,m
300	f,b
400	c,b,p
500	f,c,a,m,p

(c) Ordered and truncated Transactional database

Figure 2. An example of transaction database and ordered and truncated transactional database.

The next step is to scan the transaction database a second time to construct FP-Tree. Starting from the root, transactions are added one by one to the root subtree in a prefix tree method. After reading each transaction, the items are reordered in reverse order of frequency. Items that do not meet the minimum support are not considered. Figure 2c shows the transaction after omitting items with a support rating of less than three from Figure 2a.

Figure 3 shows that the process of construction of FP-Tree. The process consisted of the four-step to add the five transactions of Figure 2a to FP-Tree. Figure 4 shows the final FP-Tree and its header table for the transaction database. FP-Tree reduces frequent database scans compared to Apriori algorithm. Since FP-Tree does not generate candidate sets, it is useful for finding frequent itemsets from large amounts of data. However, when the depth of the tree increases and the number of nodes increases, the dependence of the memory size is large, and a lot of processing time may be consumed for mining.

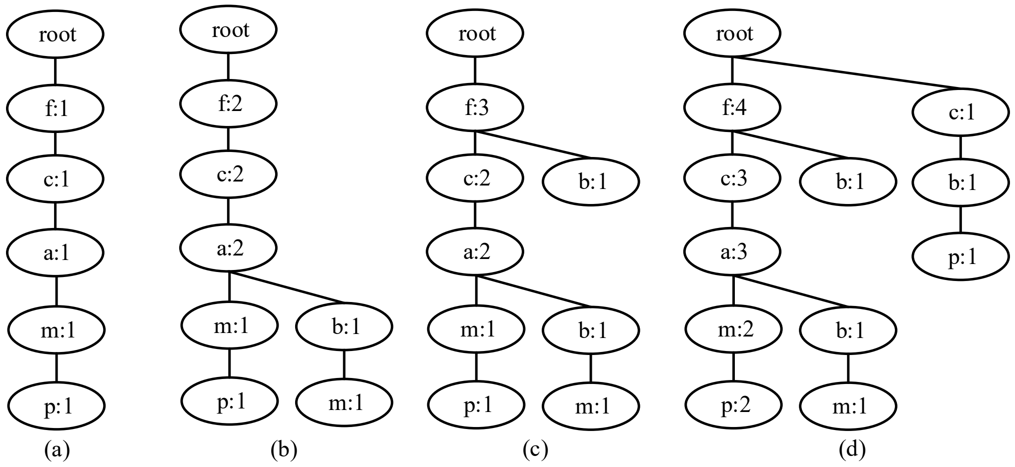
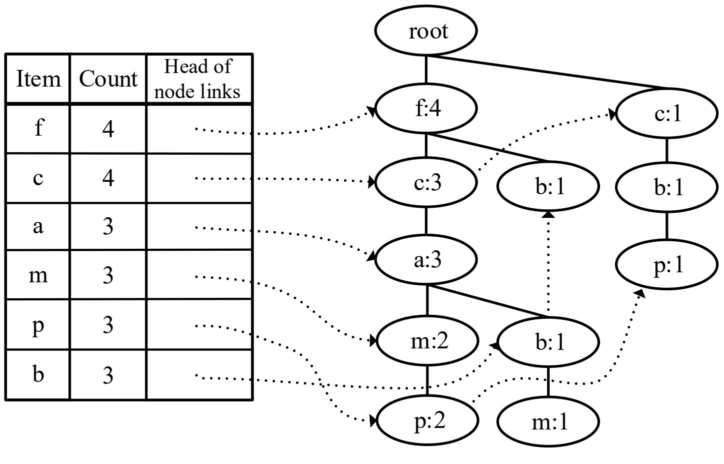


Figure 3. A process of FP-Tree construction. (**a**) Transaction (f,c,a,m,p) is inserted, (**b**) transaction (f,c,a,b,m) is inserted, (**c**) transaction (f,b) is inserted, (**d**) transaction (c,b,p) is inserted.



Item	Count	Head of node links
f	4	
c	4	
a	3	
m	3	
p	3	
b	3	

(a) Header Table

(b) FP-tree

Figure 4. FP-Tree built based on the data in Figure 2a.

2.2. Variants of FP-Growth

Since the introduction of FP-Tree, various tree structures for frequency pattern mining such as COFI-tree [12], DRFP-tree [13,14], CanTree [15], DSTree [19], AFOPT-tree [20], CATS Tree [21], were presented. Since FP-Tree reads the entire database and constructs a tree using the number of frequent items, it is a data mining technique that applies only to a fixed database and cannot be used in a streaming database. In order to overcome the constraints of FP-Tree, DS-Tree for mining association rules in a streaming database is presented. DS-Tree does not use the number of frequent items but sorts the items according to the criteria set by the user so as to fit the characteristics of items such as alphabetical order or lexical order. After sorting, DS-tree is created by reading a batch, which is a set of transactions as much as the window size. DS-Tree is a tree construction method that can be applied not only to a fixed database but also to a data stream environment. While FP-Tree uses the number of item frequency as the item sorting criterion, DS-Tree uses a simple criterion such as alphabetical order or lexicographic order as a criterion for tree construction. Therefore, because the database scan to find the item sorting criteria can be omitted, a tree can be constructed with only one database scan, and a tree can be constructed even in a data stream environment. Since only one data scan is required, the time it takes to construct the tree can be reduced.

2.3. FP-Growth Based on Spatial Data

With the increase in spatiotemporal data, the problem of discovering spatiotemporal association rules in spatiotemporal databases has received considerable attention in the field of frequent pattern discovery. In addition, research on spatial frequent pattern analysis based on FP-Growth is being actively conducted.

Maiti et al. [22] proposed a Map-Reduce-based approach as a method of finding co-location patterns defined with R-proximity measure and conditional probability. The purpose of this approach is to find co-location patterns of all sizes from distributed data. The colocation rule is a model for inferring the existence or nonexistence of spatial features around the item by using the features of the item included in the rule. This approach consists of four algorithms, and FP-Growth algorithm is utilized as one method to find colocation by finding all frequent itemsets.

Lee et al. [23] proposed SFP-Growth algorithms to find spatial frequent patterns from social data. This study divides the entire space into cells on a 2D grid and manages cells hierarchically by dividing the side of a cell by four. The SFP-Growth algorithm extracts spatial frequent patterns of specific locations which explain the relative characteristics of the location. However, in this method, when a frequent pattern is extracted as the sum of distant lower cells included in the same upper cell, the frequent pattern becomes information describing the characteristics of the upper cell. In the upper cell, there are also lower cells that are not involved in the frequent pattern extraction at all. In this case, even cells that do not affect the frequent pattern extraction may be misinterpreted as having the frequent pattern property defined in the upper cell.

Kiran et al. [24] proposed frequent spatial pattern growth (FSP-Growth). This study defined interesting spatial patterns, including not only frequent items that occurred at close distances between two items but also items in which the maximum distance between two items was not greater than the user-specified *maxDis*.

2.4. Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n binary attributes called items. Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions called the database. Let $R = \{r_1, r_2, \ldots, r_k\}$ be a set of k density regions. Each transaction in D has a unique transaction ID, a region where the product was purchased, and contains a subset of the items in I. A rule is defined as an implication of the form:

$$\text{r: X}\rightarrow\text{Y, where X, Y}\subseteq\text{I and r}\subseteq\text{R.} \quad (1)$$

In order to select interesting rules from the set of all possible rules, constraints on various measures of significance and interest are used. The best-known constraints are minimum thresholds on support and confidence.

Let X, Y be itemsets and r be regions, r: X→Y an association rule, and T a set of transactions of a given database.

Definition 1. (**Support**) *Support is an indication of how frequently the itemset appears in the dataset. The support of X with respect to T is defined as the proportion of transactions t in the dataset which contains the itemset X.*

$$\text{supp(X, r)} = |\{t\in T; X\subseteq t\}| / |T|\text{, where T.region = r}$$

The support gives an idea of how frequent an itemset is in all the transactions.

Definition 2. (**Confidence**) *Confidence is an indication of how often the rule has been found to be true. The confidence value of a rule, X→Y, with respect to a set of transactions T, is the proportion of the transactions that contain X which also contains Y.*

$$\text{conf(X}\rightarrow\text{Y, r) = supp(X}\cup\text{Y)/supp(X), where T.region = r} \quad (2)$$

We modified by adding the locality of the transaction to the support and confidence used in the existing association rule. For conciseness of expression, however, it is expressed in the same way as support and confidence.

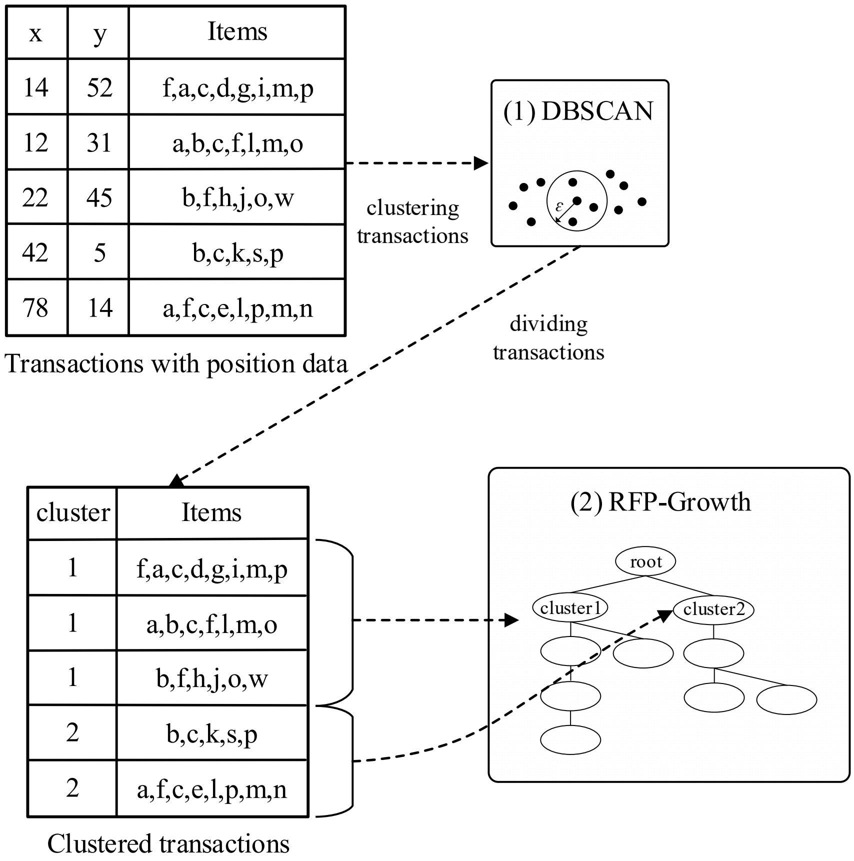
Problem definition. Given a set of transactions, D, containing regions where the items were purchased, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively for user-specified regions.

3. Methods

3.1. Overview of RFP-Growth

The purpose of this study is to prove that, if frequent rules are discovered by classifying transactions by region, frequent rules that cannot be discovered in the entire data can be found. RFP-Growth first divides transactions into density-based regions. The position data could be the name or code of the store where the transaction occurred, or it could be the longitude or latitude where the transaction occurred. We assume that the raw data to be analyzed contains longitude(x) and latitude(y) data where the transaction occurred.

DBSCAN performs the clustering process using only the location data of the raw data. Through the DBSCAN, each transaction is assigned to a cluster. In Figure 5, the cluster column means the cluster number to which each transaction is assigned in the clustered transaction table. For each cluster, RFP-Tree is generated using the transaction assigned to the cluster.



x	y	Items
14	52	f,a,c,d,g,i,m,p
12	31	a,b,c,f,l,m,o
22	45	b,f,h,j,o,w
42	5	b,c,k,s,p
78	14	a,f,c,e,l,p,m,n

Transactions with position data

cluster	Items
1	f,a,c,d,g,i,m,p
1	a,b,c,f,l,m,o
1	b,f,h,j,o,w
2	b,c,k,s,p
2	a,f,c,e,l,p,m,n

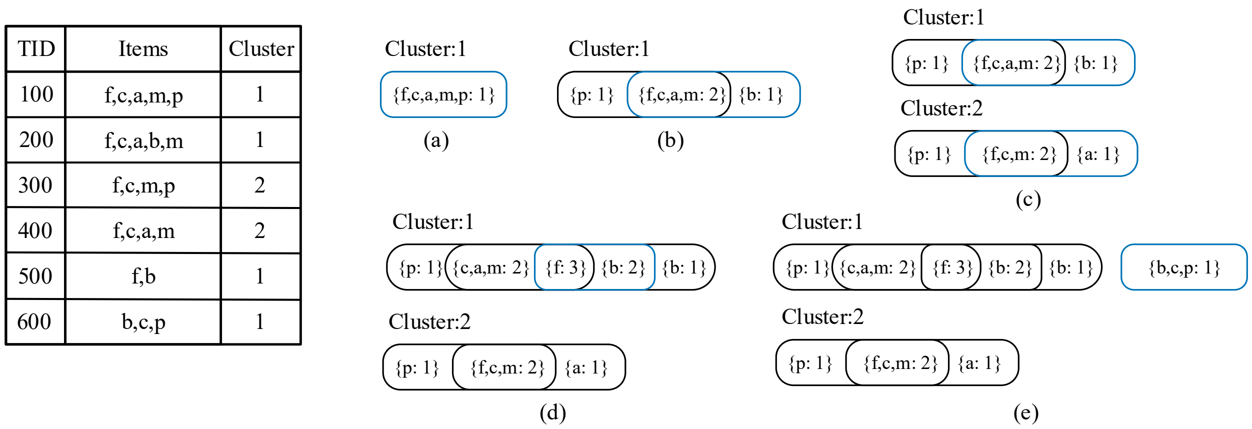
Clustered transactions

Figure 5. Overview of RFP-Growth.

3.2. Intersection-Based FP-Tree

The existing FP-Tree constructs a tree based on the criterion of the frequency of occurrence. However, in this paper, a method of constructing FP-Tree using the intersection is adopted. RFP-Tree based on the intersection is not a method of organizing a tree by sorting items using a specific criterion, but by grouping items generated by intersection each time each transaction is entered. Only one item may be included in one node, or multiple items may be included in one node. There is no need to rearrange the items in the transaction, and there is no need to build a new tree even if a continuous transaction flows in. When a new subtree is created, the item set with the highest frequency including the input transaction among the item sets is made as to the parent node. Therefore, the latest item frequency is continuously applied to the node can be optimized whenever a transaction is entered.

Figures 6 and 7 show a process to construct RFP-Tree using the example of a transactional database in the aggregate expression method and tree structure. In the case of transactions 100 and 200 in Figure 6b, the intersection of two sets {f,c,a,m} was generated twice, and the {p} and {b} item sets were generated once. If this is presented as a tree, it can be expressed as shown in Figure 7b. Whenever a transaction is entered one by one, the item sets are finally grouped using the intersection set similar to Figure 7e. If there are no items intersected with an item of an existing transaction, such as Transaction 600 in Figure 6, a new node is created in a tree. These nodes are not considered in the process of deriving the association rule. In this way, a tree can be constructed each time a transaction is added one by one without the process of scanning the entire transaction once.



TID	Items	Cluster
100	f,c,a,m,p	1
200	f,c,a,b,m	1
300	f,c,m,p	2
400	f,c,a,m	2
500	f,b	1
600	b,c,p	1

Figure 6. Construction of intersection-based FP-Tree (blue circles indicate newly added transactions). (**a**) Transaction (f,c,a,m,p) is inserted, (**b**) transaction (f,c,a,b,m) is inserted, (**c**) transactions (f,c,m,p) and (f,c,a,m) are inserted, (**d**) transaction (f,b) is inserted, (**e**) transaction (b,c,p) is inserted.

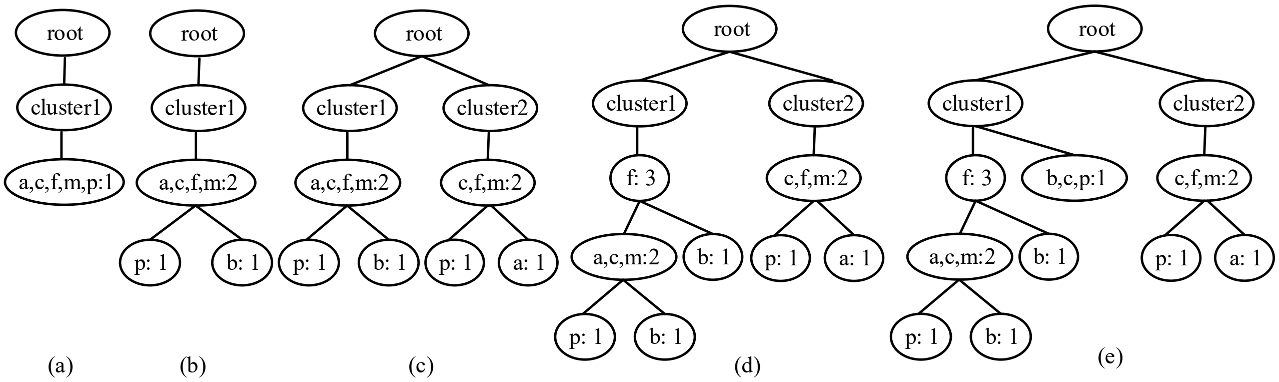


Figure 7. Tree representation of RFP-Tree. (**a**) Transaction (f,c,a,m,p) is inserted, (**b**) transaction (f,c,a,b,m) is inserted, (**c**) transactions (f,c,m,p) and (f,c,a,m) are inserted, (**d**) transaction (f,b) is inserted, (**e**) transaction (b,c,p) is inserted.

A general FP-Tree construction scans the entire database, which is the preprocessing step of tree construction, calculates the number of items, and uses this to rank the items, sort the items in each transaction in the reverse order of the number of items. However, since RFP-Tree does not require the pre-processing step of such tree configuration, it reads the database transaction and proceeds to construct the tree, thus reducing the pre-processing cost. A general FP-Tree composes a tree by reading one transaction from a database and reading items in the transaction one by one, but the proposed FP-Tree reads one transaction and constructs FP-tree in units of transactions, which reduces the cost of time compared to a general FP-Tree.

In general, the study of extracting association rules from spatial data first divides the data into clusters and then applies the traditional association rule mining algorithm to each cluster to find association rules. Instead of creating a tree after the DBSCAN process is finished, a tree can be constructed at the same time as one cluster is constructed in DBSCAN.

A transaction consists of <TID, region, items >where TID means a unique identifying number and region means a market where the consumer purchased the item and items mean the list of items purchased by consumers.

Algorithm 1 shows how RFP-tree is built. Through the entire transaction scan, only transactions with a given region from the user are selected in each transaction. For each transaction, the treeConstruct function is called and an item is passed as a parameter

(Lines 2–4). If the intersection of the item of the child node (*childNode*) and the items of the current transaction is an empty set, items are inserted into the child of the current node (*currentNode*) (Lines 6–7). *childNode* means child node of *currentNode.* If the intersection of the item of the child node (*childNode*) and the items of the current transaction is not an empty set, a new node is added to the FP-Tree (Lines 8–23). If items and items of child node are the same, increase the frequency of child node by 1. If the items are a subset of the items of a *childNode*, insert the result of the difference between *childNode* and items in child of *childNode.* The intersection of *childNode* and items is reinserted in *childNode*. The frequency of childnode is increased by 1 (Lines 11–14). If the items of a *childNode* are a subset of the *items*, the frequency of the *childNode* is increased by 1. The result of the difference between *items* and items of *childNode* is inserted into *restItems*. *treeConstruct* function with *restItem* and *childNode* set as parameters is executed (Lines 15–18). If it is not included in the above three cases, two child nodes are created (Lines 20–21). The difference between *childNode* and *items* is inserted into the first child node (Line 20), and the difference between *items* and *childNode* is inserted into the second child node (Line 21). The intersection of *childNode* and *items* is reinserted in *childNode and* the frequency of *childNode* is increased by 1 (Lines 20–23).

Algorithm 1 TreeBuilder

```
Input: A transaction DB, a set Q of query keywords, a set R of region
Output: FP-tree
1.  NODE node = null;
2.  for each transaction $t \in$DB do
3.      if $t$.region $\in R$ then
4.          treeConstruct(t.items, node)
5.  Procedure treeConstruct(items, currentNode)
6.      if childNode.items $\cap$ items == $\varnothing$ then
7.          currentNode.child $\leftarrow$ items
8.      else
9.          if childNode.items == items then
10.             childNode.frequency++
11.         else if childNode.items $\supset \bigcap$ items then
12.             childNode.child $\leftarrow$ childNode–items
13.             childNode = childNode $\cap$ items
14.             childNode.frequency++
15.         else if childNode.items $\subset$ items then
16.             childNode.frequency++
17.             restItems = items–childNode.items
18.             treeConstruct(restItems, childNode);
19.         else // split and add node
20.             childNode.child$\leftarrow$ childNode–items
21.             childNode.child$\leftarrow$items–childNode
22.             childNode = childNode $\cap$ items
23.             childNode.frequency++
```

Complexity. Existing FP-Tree construction algorithms require two database scans that need $2n$, where n is the number of transactions. However, the RFP-Tree construction algorithm can create a tree with one database scan. Our proposed algorithms can reduce the number of transactions to n. Whenever a transaction is added one by one, the item comparison operation of the two transactions executes. The complexity of RFP-Tree construction algorithm is dominated by comparing two sets of elements. If the average number of items in one transaction is m, the number of times to compare common items in two transactions is m^2. Thus, the complexity of RFP-Tree construction algorithm is $O(nm^2)$.

4. Results and Discussion

4.1. Experiment Setting

4.1.1. Algorithms

The purpose of this study is to verify whether new association rules are found when the association rule is extracted by dividing a transaction database by region compared to when the association rule is extracted for the entire data. We compared RFP-Growth algorithm with the original FP-Growth, dFIN [25] and negFIN [26]. The FP-Growth algorithm discovers frequent patterns from the transaction that consists solely of items, while RFP-Growth algorithm considers the transaction with spatial data. RFP-Growth algorithm consists of two steps.

(1) The transaction is divided into regions by clustering on spatial data included in a transaction. We used DBSCAN algorithm, a clustering algorithm that allows clusters to have an arbitrary shape because we considered a commercial area with dense stores as one region. (2) FP-Growth algorithm is performed for each transaction divided by region.

All the proposed algorithms were implemented in Java, and the experiments were conducted on an Intel Core i7 at 3.50 GHz with 32 GB memory. The parameters used for the experiments are summarized and default values are shown in boldface in Table 2.

Table 2. Parameters for the experiments.

Parameters	Description	Values
n	no. of objects	1K, 2K, 3K, 4K, 5K, 6K, 7K, 8K, 9K, **10K**
k	no. of clusters	**5**, 10, 15, 20, 25
minsup	minimum support	0.20, 0.22, 0.24, 0.26, 0.28, 0.30, 0,32, 0,34, 0.36, 0.38, 0.40, 0.42, 0.44, 0.46, **0.48**, 0.50, 0.52, 0.54, 0.56, 0.58, 0.60, 0.62, 0.64, 0.66, 0.68, 0.70

"K" represents 1000.

4.1.2. Data Sets

The real-world datasets and synthetic data sets are used for experiments. For a real dataset, we collected sets of purchase items made for an online retail company based in the UK during an eight-month period (https://www.kaggle.com/vijayuv/onlineretail (accessed on 1 December 2021)). The real dataset consists of 25,900 transactions, and there are 4070 items. Table 3 shows samples of real datasets. We removed unnecessary columns for the experiment, i.e., Quantity, Invoice Date, Unit Price, and Customer ID. Since the same transaction is divided into several rows, the rows of the same invoice number are combined into one row. We conducted an experiment assuming the same country as one cluster by using the country column as location information. For the synthetic data, spatial data are generated in the 2D space (0, 100) × (0, 100) to indicate a store's location, and item data are generated among 100 items totally. Items in the synthetic data are generated so that there are no duplicate items in one transaction which has an average of 20 items.

Table 3. Samples of a real dataset.

Invoice No.	Stock Code	Quantity	Invoice Date	Unit Price	Customer ID	Country
536365	85123A	6	01-12-2010 08:26	2.55	17850	UK
536365	71053	6	01-12-2010 08:26	3.39	17850	UK
356365	84406B	8	01-12-2010 08:26	2.75	17850	UK
536370	22728	24	01-12-2010 08:45	3.75	12583	France
536370	22727	24	01-12-2010 08:45	3.75	12583	France
536370	22726	12	01-12-2010 08:45	3.75	12583	France
536389	22941	6	01-12-2010 10:03	8.5	12431	Australia
536389	22941	8	01-12-2010 10:03	4.95	12431	Australia
536389	22941	12	01-12-2010 10:03	1.25	12431	Australia

4.2. The Discovery of New Association Rules

In this section, we evaluate the effect with respect to the *minsup* on the discovery of new association rules. The *Support* means an indication of how frequently the itemset appears in the dataset. *minsup* is the minimum support for an itemset to be identified as frequent. The smaller the *minsup*, the more frequent rules are discovered. If the support of an itemset is low, there is not enough information about the relationships between items. We need to find support that elicits a reasonable number of frequent rules. We conduct experiments with various *minsup* [0.2, 0.7], and set n = 10,000 and k = 5. Since FP-Growth, dFIN, and negFIN algorithms yield the same rules as a result, we compared the results of the FP-Growth algorithm to see if RFP-Growth algorithm discovers new rules.

Figure 8 shows the number of newly discovered frequent rules according to the support level from the synthetic datasets. The left y-coordinate represents the number of the frequent rules found in FP-Growth. The right y-coordinate represents the number of the frequent rules that are not found in FP-Growth and are newly discovered in RFP-Growth. As *minsup* becomes larger, the number of the frequent rules discovered decreases. As *minsup* increases, the number of newly discovered rules decreases. In the section where the number of the frequent rules discovered in FP-Growth is maintained, [0.22–0.3, 0.34–0.44, 0.5–0.68], the number of newly discovered frequent rules in RFP-Growth is low. It can be seen that when the number of the frequent rules discovered in FP-Growth is reduced, the number of the frequent rules newly discovered in RFP-Growth increases. When *minsup* was 0.7, no rules were discovered in FP-Growth, but 20 rules were discovered in RFP-Growth.

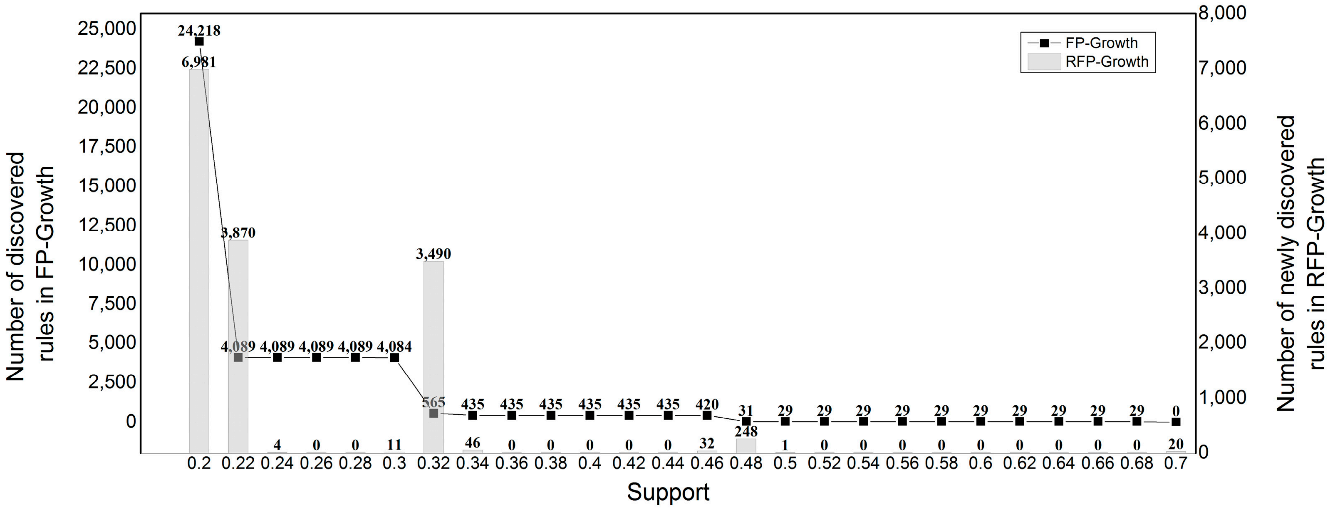


Figure 8. The number of discovered frequent rules in FP-Growth and RFP-Growth from the synthetic data.

Figure 9 shows the number of newly discovered frequent rules according to the support level from the real datasets. The left y-coordinate represents the number of the frequent rules found in FP-Growth. The right y-coordinate represents the number of the frequent rules that are not found in FP-Growth and are newly discovered in RFP-Growth. We conduct experiments with various *minsup* [0.0001, 0.001], and set n = 1,044,000 and k = 50. In the section where the number of the frequent rules discovered in FP-Growth is maintained, [0.0001–0.0002, 0.0006–0.001], the number of newly discovered frequent rules in RFP-Growth is low. It can be also seen that when the number of the frequent rules discovered in FP-Growth is reduced, the number of the frequent rules newly discovered in RFP-Growth increases. When *minsup* was 0.0006, 0.0008 and 0.001, no rules were discovered in FP-Growth, but 206,106, 13,765 and 380 rules were discovered in RFP-Growth respectively.

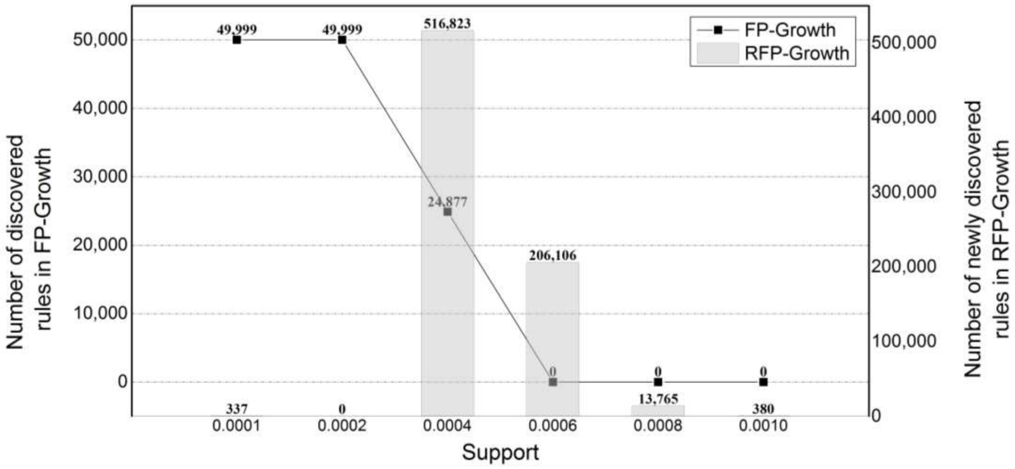


Figure 9. The number of discovered frequent rules in FP-Growth and RFP-Growth from the real-world data.

4.3. Memory Consumption

In this section, we measure the memory consumption with respect to the *minsup* for the real-world datasets and the synthetic datasets. Figure 10 shows that the larger the *minsup*, the smaller the memory usage. This is because, in general, as *minsup* increases, the number of items treated as a frequent pattern decreases. Comparing the memory usage of RFP-Growth and FP-Growth, it was found that the memory usage decreased by 34% for synthetic datasets and 23% for real datasets. This can be explained as follows. RFP-Growth generates an FP-Tree from transaction data through an intersection operation, whereas FP-Growth generates an FP-Tree after creating an ordered and truncated transactional table in Figure 2c. dFIN and negFIN show higher memory usage than FP-Growth for low *minsup*.

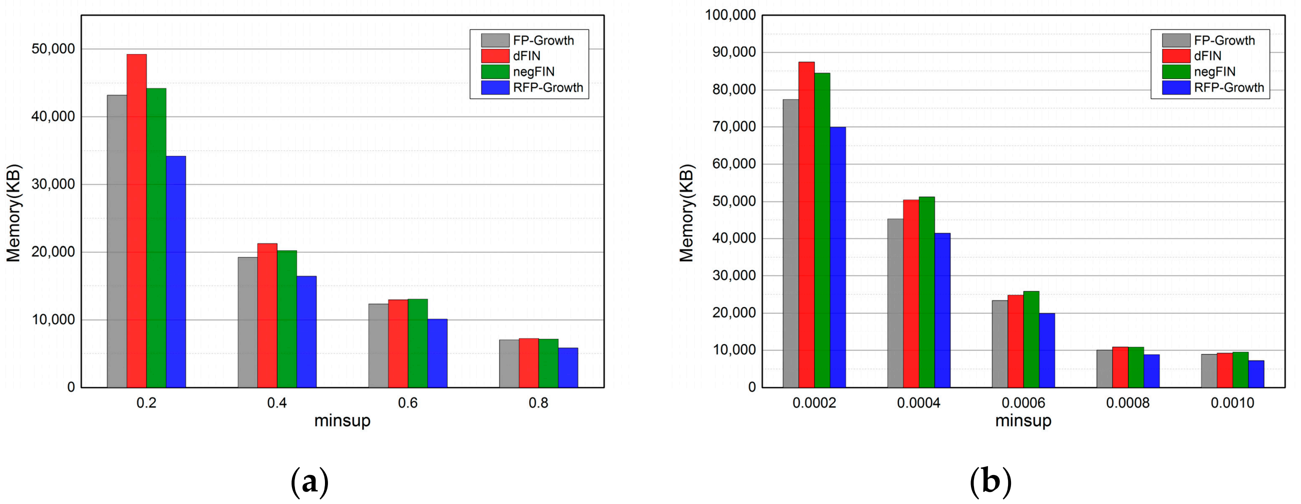
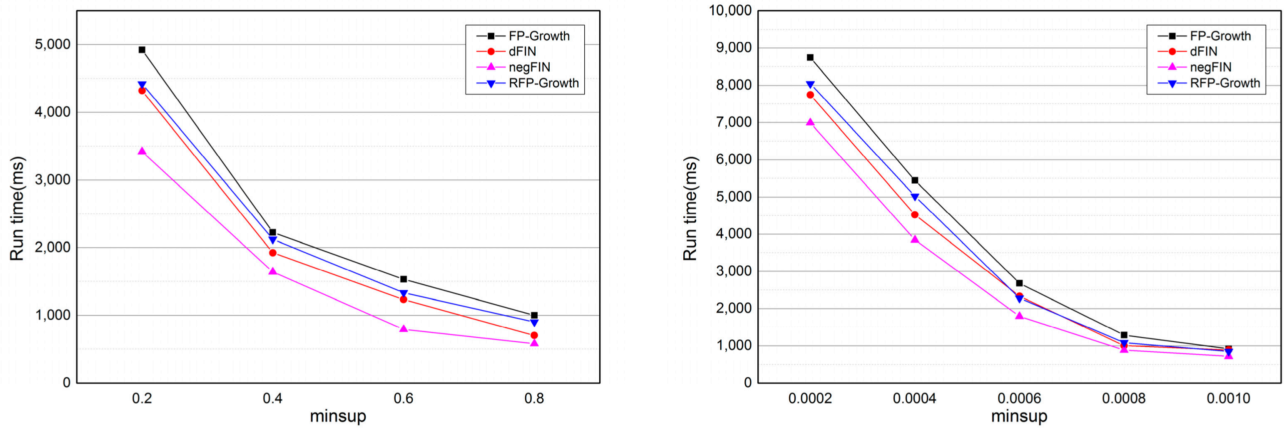


Figure 10. Memory consumption comparison for different datasets, depending on the *minsup*. (**a**) Synthetic datasets, (**b**) real-world datasets.

The runtime comparison of RFP-Growth against FP-Growth, dFIN, and negFIN with respect to the *minsup* is shown in Figure 11. In all algorithms, it appears that the runtime cost decreases as *minsup* increases. The best performance at runtime is negFIN, which is known as the fastest algorithm, followed by dFIN, RFP-Growth, and FP-Growth. RFP-Growth shows better performance than FP-Growth, and the runtime performance of RFP-Growth is slightly lower than dFIN and negFIN. Although RFP-Growth uses less memory, it seems that the set operation to find common items increases the time cost.



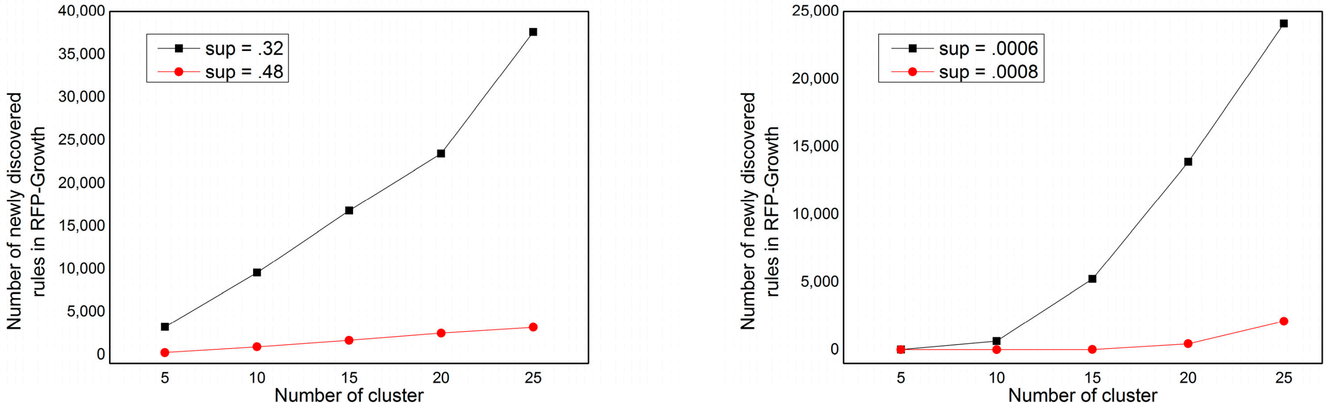
(**a**) (**b**)

Figure 11. Run time cost comparison for different datasets, depending on the minimum support. (**a**) Synthetic datasets, (**b**) real-world datasets.

4.4. The Effect of the Number of Clusters

In this section, we evaluate the effect with respect to the number of clusters for the real-world datasets and the synthetic datasets. We conduct experiments with the various clusters [5, 10, 15, 20, and 25]. In the synthetic datasets, we set n = 10,000, *minsup* = [0.32, 0.48] and k = 5. In the real-world datasets, we set n = 1,044,000, *minsup* = [0.0006, 0.0008] and k = 50.

Figure 12 shows the effects of the number of clusters on the number of newly discovered frequent rules in RFP-Growth. Two graphs are also shown on a linear scale. For each dataset, the number of newly discovered frequent rules in RFP-Growth is approximately proportional to the number of clusters. In both graphs, when *minsup* is high (0.48, 0.006), the number of newly discovered frequent rules gradually increases. On the other hand, when *minsup* is low (0.32, 0.008), the number of newly discovered rules increases rapidly as the number of clusters increases.



(**a**) (**b**)

Figure 12. The number of discovered frequent rules with respect to the number of clusters in RFP-Growth. (**a**) Synthetic datasets, (**b**) real-world datasets.

Figure 13 shows the effects of the number of clusters on the time cost in RFP-Growth. Two graphs are also shown on a linear scale. For each dataset, the time cost in RFP-Growth is approximately proportional to the number of clusters. In the synthetic datasets, the time cost is different for the two *minsup* (0.32 and 0.48), but in the real dataset, there is little difference in the time cost for the two *minsup* (0.0006 and 0.0008). This difference is caused by the difference in scale for the two datasets. What we can see from this result is that the time cost increases as the number of clusters increases.

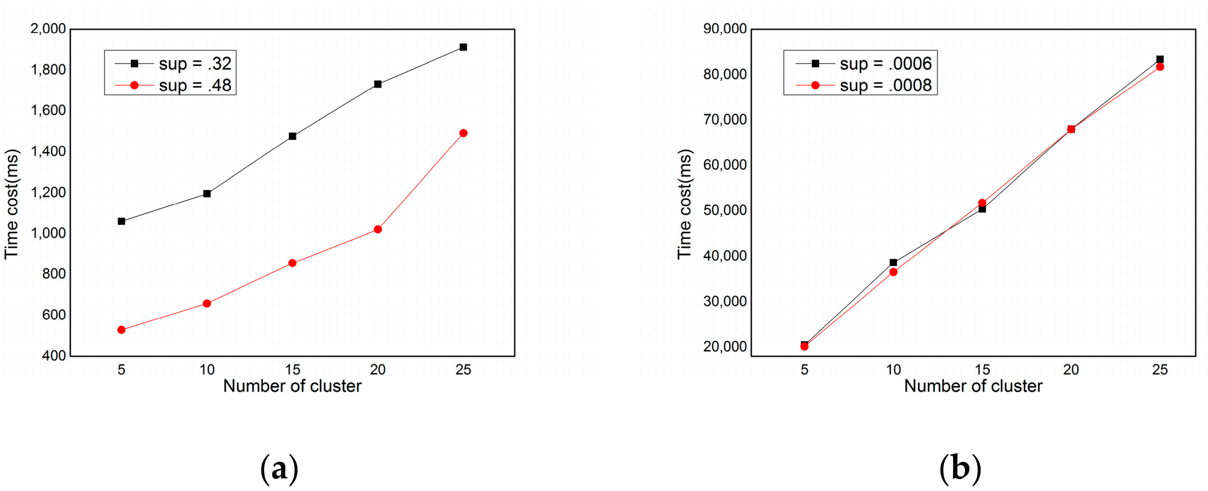


Figure 13. Time cost with respect to the number of clusters in RFP-Growth algorithm. (**a**) Synthetic datasets, (**b**) real-world datasets.

4.5. The Effect of the Size of Data

To illustrate scalability, we vary the number of objects from 1K to 10K for the synthetic datasets. Other parameters are given their baseline values (Table 2). Figure 14 shows the effect of the number of objects in FP-Growth, dFIN, negFIN, and RFP-Growth. This graph is shown on a linear scale. For each algorithm, the runtime is approximately proportional to the number of objects. As shown in Figure 14, the time cost of RFP-Growth algorithm is lower than the time cost of FP-Growth. It can be seen that processing the divided data after dividing the data into several groups is faster than processing the entire data at once.

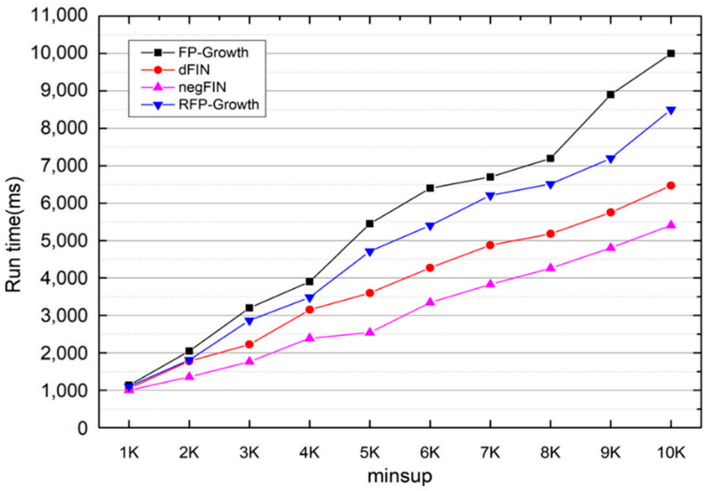


Figure 14. Effect of the size of datasets.

5. Conclusions

In this paper, we proposed the noble problem of discovering association rules by regions. Toward this goal, we proposed RFP-Growth, which organizes item transaction data with location data into groups and discovers association rules for each cluster. RFP-Growth divides item transaction included position data into regions by a density-based clustering algorithm. FP-Growth is performed for each transaction divided by region. The experimental results show that RFP-Growth discovers new association rules that the original FP-Growth cannot find in the whole data. RFP-Growth has a disadvantage, however, that the performance decreases as the number of clusters increases. In future work, we plan to improve the performance of RFP-Growth, so even if the number of clusters increases, the performance is stabilized.

Author Contributions: Conceptualization, H.-J.J., Y.Y. and B.K.; methodology, B.K.; software, B.K.; validation, H.-J.J.; investigation, Y.Y.; data curation, B.K.; writing—original draft preparation, H.-J.J. and B.K.; writing—review and editing, J.S.P.; visualization, J.S.P.; supervision, B.K.; project administration, B.K.; funding acquisition, B.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by industry-academic Cooperation R&D program funded by LX Spatial Informaion Research Institute(LXSIRI, Republic of Korea) [Project Name: A Study on the Establishment of Service Pipe Database for Safety Management of Underground Space/Project Number: 2021-502) and this research was funded by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Korean Government (MSIT) (No. 2020R1F1A1077369) and by the Korean Government (MSIT) (No. 2021R1F1A1049387).

Data Availability Statement: The spatial transaction dataset is on a public dataset and available at " https://www.kaggle.com/vijayuv/onlineretail" (accessed on 1 December 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alshammari, H.; El-Ghany, S.A.; Shehab, A. Big IoT Healthcare Data Analytics Framework Based on Fog and Cloud Computing. *J. Inf. Process. Syst.* **2020**, *16*, 1238–1249.
2. Qi, J.; Yang, P.; Hanneghan, M.; Tang, S.; Zhou, B. A Hybrid Hierarchical Framework for Gym Physical Activity Recognition and Measurement Using Wearable Sensors. *IEEE Internet Things J.* **2019**, *6*, 1384–1393. [CrossRef]
3. Wang, T.; Qiu, L.; Sangaiah, A.K.; Liu, A.; Alam Bhuiyan, Z.; Ma, Y. Edge-Computing-Based Trustworthy Data Collection Model in the Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4218–4227. [CrossRef]
4. Al-Shargabi, A.; Siewe, F. A Lightweight Association Rules Based Prediction Algorithm (LWRCCAR) for Context-Aware Systems in IoT Ubiquitous, Fog, and Edge Computing Environment. In Proceedings of the Proceedings of the Future Technologies Conference (FTC) 2020, Vancouver, BC, Canada, 5–6 November 2020. [CrossRef]
5. Lee, C.H.; Park, J.S. An SDN-Based Packet Scheduling Scheme for Transmitting Emergency Data in Mobile Edge Computing Environments. *Hum. Cent. Comput. Inf. Sci.* **2021**, *11*. [CrossRef]
6. He, Y.; Tang, Z. Strategy for Task Offloading of Multi-user and Multi-server Based on Cost Optimization in Mobile Edge Computing Environment. *J. Inf. Process. Syst.* **2021**, *17*, 615–629.
7. Lee, J.-H. Next Task Size Prediction Method for FP-Growth Algorithm. *Hum. Cent. Comput. Inf. Sci.* **2021**, *11*. [CrossRef]
8. Agrawal, R.; Imieliski, T.; Swami, A. Mining association rules between sets of items in large databases. In Proceedings of the ACM SIGMOD Record, New York, NY, USA, 1 June 1993; Volume 22, pp. 207–216.
9. Agrawal, R.; Srikant, R. Fast algorithms for mining association rules in large databases. In Proceedings of the 20th International Conference on Very Large Data Bases, San Francisco, CA, USA, 12–15 September 1994; pp. 487–499.
10. Han, J.; Pei, J.; Yin, Y. Mining Frequent Patterns without Candidate Generation. *ACM SIGMOD Rec.* **2000**, *29*, 1–12. [CrossRef]
11. Giannella, C.; Han, J.; Pei, J.; Yan, X.; Yu, P.S. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In *Data Mining: Next Generation Challenges and Future Directions*; Kargupta, H., Joshi, A., Sivakumar, K., Yesha, Y., Eds.; AAAI Press: Palo Alto, CA, USA, 2004.
12. Zaïane, O.R.; El-Hajj, M. COFI Approach for Mining Frequent Itemsets Revisited. In Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Paris, France, 13 June 2004; pp. 70–75.
13. Adnan, M.; Alhajj, R. DRFP-tree: Disc-resident frequent pattern tree. *Appl. Intell.* **2009**, *30*, 84–97. [CrossRef]
14. Adnan, M.; Alhajj, R. A bounded and Adaptive Memory-based Approach to Mine Frequent Patterns from Very Large Databases. *IEEE Trans. Syst. Man Cybern. Part B* **2011**, *41*, 154–172. [CrossRef] [PubMed]
15. Leung, C.K.-S.; Khan, Q.I.; Hoque, T. CanTree:A Tree Structure for Efficient Incremental Mining of Frequent Patterns. In Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05), Houston, TX, USA, 27–30 November 2005; pp. 274–281.
16. Ünvan, Y.A. Market basket analysis with association rules. *Commun. Stat. Theory Methods* **2021**, *50*, 1615–1628. [CrossRef]
17. Mar, Z.; Oo, K.K. An Improvement of Apriori Mining Algorithm using Linked List Based Hash Table. In Proceedings of the 2020 International Conference on Advanced Information Technologies (ICAIT), Yangon, Myanmar, 4–5 November 2020. [CrossRef]
18. Pan, Z.; Liu, P.; Yi, J. An Improved FP-Tree Algorithm for Mining Maximal Frequent Patterns. In Proceedings of the 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Changsha, China, 10–11 February 2018. [CrossRef]

19. Leung, C.K.-S.; Khan, Q.I. DSTree: A tree structure for the mining of frequent sets from data streams. In Proceedings of the Sixth International Conference on Data Mining (ICDM'06), Hong Kong, China, 18–22 December 2006; pp. 928–932.
20. Liu, G.; Lu, H.; Yu, J.X.; Wang, W.; Xiao, X. AFOPT: An efficient implementation of pattern growth approach. In Proceedings of the ICDM Workshop, Melbourne, FL, USA, 19–22 November 2003.
21. Cheung, W.; Zaiane, O.R. Incremental mining of frequent patterns without candidate generation or support constraint. Database Engineering and Applications Symposium. In Proceedings of the Seventh International Database Engineering and Applications Symposium, Hong Kong, China, 18 July 2003; pp. 111–116.
22. Maiti, S.; Subramanyam, R.B.V. Mining co-location patterns from distributed spatial data. *J. King Saud Univ. Comput. Inf. Sci.* **2020**, *33*, 1064–1073. [CrossRef]
23. Lee, Y.; Nam, K.W.; Ryu, K.H. Fast mining of spatial frequent wordset from social database. *Spat. Inf. Res.* **2017**, *25*, 271–280. [CrossRef]
24. Kiran, R.U.; Shrivastava, S.; Fournier-Viger, P.; Zettsu, K.; Toyoda, M.; Kitsuregawa, M. Discovering Frequent Spatial Patterns in Very Large Spatiotemporal Databases. In Proceedings of the SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, 3–6 November 2020; pp. 445–448. [CrossRef]
25. Deng, Z.-H. DiffNodesets: An Efficient Structure for Fast Mining Frequent Itemsets. *Appl. Soft Comput.* **2016**, *41*, 214–223. [CrossRef]
26. Aryabarzan, N.; Minaei-Bidgoli, B.; Teshnehlab, M. negFIN: An efficient algorithm for fast mining frequent itemsets. *Expert Syst. Appl.* **2018**, *105*, 129–143. [CrossRef]

electronics

MDPI

Article

Automatic Failure Recovery for Container-Based IoT Edge Applications

Kolade Olorunnife *, Kevin Lee * and Jonathan Kua *

School of Information Technology, Deakin University, Geelong, VIC 3220, Australia

* Correspondence: kolorunnife@deakin.edu.au (K.O.); kevin.lee@deakin.edu.au (K.L.); jonathan.kua@deakin.edu.au (J.K.)

Abstract: Recent years have seen the rapid adoption of Internet of Things (IoT) technologies, where billions of physical devices are interconnected to provide data sensing, computing and actuating capabilities. IoT-based systems have been extensively deployed across various sectors, such as smart homes, smart cities, smart transport, smart logistics and so forth. Newer paradigms such as edge computing are developed to facilitate computation and data intelligence to be performed closer to IoT devices, hence reducing latency for time-sensitive tasks. However, IoT applications are increasingly being deployed in remote and difficult to reach areas for edge computing scenarios. These deployment locations make upgrading application and dealing with software failures difficult. IoT applications are also increasingly being deployed as containers which offer increased remote management ability but are more complex to configure. This paper proposes an approach for effectively managing, updating and re-configuring container-based IoT software as efficiently, scalably and reliably as possible with minimal downtime upon the detection of software failures. The approach is evaluated using docker container-based IoT application deployments in an edge computing scenario.

Keywords: Internet of Things (IoT); edge computing; failure recovery

Citation: Olorunnife, K.; Lee, K.; Kua, J. Automatic Failure Recovery for Container-Based IoT Edge Applications. *Electronics* **2021**, *10*, 3047. https://doi.org/10.3390/electronics10233047

Academic Editor: George Angelos Papadopoulos

Received: 9 November 2021
Accepted: 2 December 2021
Published: 6 December 2021

1. Introduction

The past decade has seen the rapid development and adoption of the Internet of Things (IoT). IoT refers to an ecosystem where billions of physical devices/objects are equipped with communication, sensing, computing and actuating capabilities [1,2]. In 2021, there is an estimated 12.3 billion of active IoT endpoints and it is forecast that the number of active IoT endpoints will reach 27 billion in 2025 [3]. IoT-based systems have been extensively deployed across many industries and sectors that impact our everyday lives, ranging from smart homes, smart cities, smart transport and smart logistics. Business opportunities and new markets abound in IoT, with new IoT applications and use cases constantly evolving. However, IoT applications are increasingly complex, commonly with a large number of end devices with many sensors and actuators and computing spread across end-nodes, edge and cloud locations. There has been substantial effort in designing architectures and frameworks to support good IoT application design [4,5]. An effective and efficient IoT system requires optimal architectural, communication and computational design. The heterogeneity of device hardware, software, communication protocols has made achieving the performance requirements of specific IoT services challenging [6–8]. Service requirements for IoT systems have motivated the development of new computational paradigms such as edge and fog computing to facilitate IoT data computation and analysis, bringing data intelligence and decision-making processes closer to IoT sensors and actuators, thus improving their performance by reducing service latency [9–12]. This is particularly important for time-sensitive tasks, such as those in autonomous vehicles, manufacturing and transport industries, where minute delays in services can have serious safety consequences [13].

There are significant trade-off considerations when deciding on the computational paradigm for IoT systems [14]. Cloud computing offers higher reliability, availability and capabilities, while edge computing offers low-latency services. However, like all complex computer systems, edge-based IoT deployments can suffer from failures [15]. This is potentially more problematic for IoT deployments due to the nature of these often being in an embedded or hard to reach physical locations. IoT devices are usually deployed at scale and has cheaper chipsets and hardware, which make them more prone to faults. It is challenging to manage faults, especially when these devices are deployed across a large area in physically hard-to-reach environments. Solving this issue can be through fault-tolerance [16,17] attempting to prevent errors in the first place, or through failure recovery techniques that attempt to recover from problems [18,19]. In addition, IoT applications are increasingly deploying software on cloud services which have no physical access making fault-tolerance necessary and difficult [20]. A potential solution to this problem is to use virtualisation technologies for resource optimisation in heterogeneous service-oriented IoT applications [21]. Redundancy is another common approach which has been extensively studied, particularly providing fail-overs in the IoT sensing, routing and control processes [22]. In addition to fault-tolerance, approaches using anomaly detection and self-healing techniques have also been explored for building more resilient IoT and Cyber-Physical Systems (CPS), addressing the various risks posed by threats at the physical, network and control layers [23].

Among the many fault-tolerance approaches studied and proposed, container-based approaches are still lacking. The aim of this paper is to investigate the feasibility of automatically detecting failures in container-based IoT edge applications. Specially, this paper investigate if this technique be used in scenarios with IoT software deployed in embedded or hard to reach scenarios, with no or difficult physical access. The proposed approach can automatically diagnose faults with IoT devices and gateways by monitoring the output of IoT applications. When faults are discovered, the proposed approach will reconfigure and redeploy container-based deployments. The experimental evaluation analyses the impact of error rate, redeployment time and packed size on the recovery time for the IoT application.

The contributions of the paper are (i). An evaluation of approaches for failure recovery for IoT applications (ii). A proposed framework for enabling failure recovery for IoT applications (iii). Experiments to evaluate the flexibility, resilience and scalability of the proposed approach.

The structure paper is as follows. Section 2 provides background of edge computing and failure recovery for IoT applications. Section 3 proposes the architecture of a framework for automatic failure recovery of IoT applications. Section 4 describes the experimental setup for this paper and Section 5 presents results of the evaluation of the framework. Section 6 concludes the paper and discuss future work.

2. Background

This section presents some background information on edge computing, IoT software management and failure recovery in IoT.

2.1. Edge Computing

With billions of IoT sensors and devices generating a large amount of data and exchanging communication and control messages across complex networks, there arise a need for more efficient computational paradigm, rather than merely relying on cloud computing infrastructure. Edge computing is designed and proposed to address the complex challenges resulting from the large amount of data generated by IoT systems, such as resource congestion, expensive computation, long service delays which negatively impact the performance of IoT services. Edge computing aims to be a distributed infrastructure and perform data computation and analysis closer to the sensors that collect the data and actuators that act upon the decisions [9]. This significantly reduce service response time and is particularly important for IoT applications that requires real-time or time-sensitive services [13].

Put simply, an edge device is a physical piece of hardware that is a bridge between two given networks. In IoT, an edge device would commonly receive data from end devices which include sensors such as temperature sensors, moisture sensors or radio-frequency identification (RFID) scanners. The data from the sensors would be passed onto a edge device that then forwards the data to the cloud or minor data processing occurs at the edge which is then forwarded to the cloud [24].

Figure 1 illustrates a general IoT and edge-based architecture. On the left there are various sensors connected to a singular edge device which in this case is an embedded device. The edge device in Figure 1 is connected to an actuator which could possibly flip on a light switch or turn on an air conditioning unit. The edge device will forward the sensor data to an internet gateway. The internet gateway will pass this data onto an application server which will then handle the processing of this data or store it. The relationship between an edge device to edge gateway can be many to many. We can have many edge devices connect to a singular edge gateway or have many edge devices to connect to one of many edge gateways. Processing of data can happen at any stage of this architecture [25].

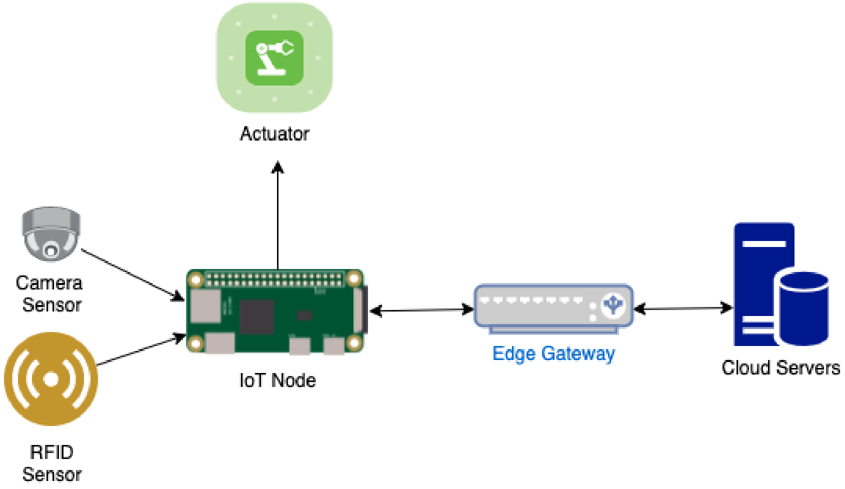


Figure 1. Edge Computing Architecture.

The increase in computational power for embedded devices allows for complex data processing on the edge devices before they even reach the cloud service [26]. Fog computing is where the majority of the processing is done at the gateway rather than the edge or the cloud. For IoT applications, there are strong benefits of using edge computing or fog computing. One such benefit is that latency for time-sensitive IoT applications is minimised due to the fact that the processing occurs at the edge rather than transferring data to and from the cloud through a gateway. Lower latency allows for real-time applications to become a forefront within the consumer, industrial and commercial space. In edge architectures, it is also possible that the end device and edge device work together to perform edge computing in order to make informed decisions or trigger an action. In edge architectures, real-time operations become a more feasible option because of lower latency due to the fact that no communication with a cloud service is needed. The sensor(s) embedded within the edge device can process the data as they are collected and then act upon this information without minimal impact to quality of service [9].

To understand the reliability and failure characteristics of IoT application deployments its useful to discuss the architecture. IoT can be thought of as an evolution of traditional sensor networks; however, there is an inherent and growing need for resources to e.g., process video. The traditional solution to this has been to use Cloud Computing resources. This has distinct advantages, such as access to almost unlimited cheap resources, but distinct disadvantages such as high-latency and a lack of control. If there is a failure of communication to the cloud resources or a failure of the cloud computing resource itself, there is often no ability to recover from this. Edge computing offers a compromise, with the introduction of powerful resources at the edge of the network, often in full or partial control of the application developer. In an edge computing architecture, reliability of the IoT application can potentially be better than just using a cloud service as the system does not need to rely entirely on the cloud service for it to fully function. In particular, communications are much simpler for IoT node to edge node, than that of to the Cloud Computing resources which will have many hops.

However, edge computing reaps several benefits as previously mentioned. They still suffer from a common issue, namely, network instability, which plagues any system or application that requires a steady connection to the internet or some form of secondary device. A device network stability has a correlation to consumed power. The greater the amount of consumed power, the higher the chance of instability possibly due to heavy computation on the edge device for processing a large volume of data, which is why segmenting and de-identifying the sensor data for privacy reasons to then send to the gateway for pre-processing is a new challenge.

The need for fault tolerant IoT systems and application have been on the steady incline and also the need for fault tolerant within IoT has been made apparent. The need for fault tolerant IoT systems is due to the possibility of intermittent long distance network connectivity problems, malicious harming of edge devices, or harsh environments physically affecting the devices performance [27]. As IoT systems become increasingly large, the effectiveness of having software frameworks automatically manage the different components within an IoT application is needed [28].

2.2. IoT Software Management

Recently, software defined network (SDN) technologies have been considered as a dominant solution for managing IoT network architecture. Dang et al. propose incorporating SDN-based technologies with over the air (OTA) programming to design a systematic framework which allows for remotely reprogramming heterogeneous IoT devices. This framework was designed for dynamic adaptability and scalability within IoT applications [29].

Soft-WSN is a software defined wireless sensor network architecture in an effort to support application aware service provisioning for Internet of Things systems. Soft-WSN proposed architecture involves a software controller, which includes two management procedures, device management and network management. Device management allows users to control their devices within the network [30]. The network configurations is controlled by the network management policy, which can be modified in run time to deal with dynamic requirements of IoT applications.

UbiFlow, is a software defined IoT system for mobility management in urban heterogeneous networks. UbiFlow adopts multiple controllers to divide software defined networks into different partitions and achieve distributed control of IoT data flows. UbiFlow mostly pertains to scalability control, fault tolerance and load balancing. The UbiFlow controller differentiates scheduling based on per device requirements. Thus, it can present an overall network status view and optimize the selection of access points within the heterogeneous networks to satisfy IoT flow requests, and guaranteeing network performance for each IoT device [31].

2.3. Fault Tolerance in IoT

Providing fault tolerance support to Internet of Things systems is an open field, with many implementations utilising various technologies like artificial intelligence, reactive approaches and algorithmic approaches [32].

A plug-able micro services framework for fault tolerance in IoT devices can be used to separate the work flow for detecting faults. The first micro service utilises complex event processing for real time reactive fault tolerance detection, whereas the second micro service uses cloud-based machine learning to detect fault patterns early and is a proactive strategy for fault tolerance. The reactive fault tolerance that uses complex event processing only initiates recovery protocols upon the detection of an error. This sort of strategy is only effective for systems that have a low latency connection to the faulty device. Whereas the proactive strategy that uses machine learning initiates recovery protocols before errors occur using predictive technologies. The main process behind a proactive strategy is to temporally disable or isolate IoT devices that will cause an error or harmfully impact the system before it occurs [33].

A mobile agent-based methodology can be utilised to build a fault-tolerant hierarchical IoT-cloud architecture that can survive the faults that occur at the edge level. In the proposed architecture, the cloud is distributed across four separate levels which are cloud, fog, mist and dew. The distribution is based on the processing power and distance from the edge IoT devices. This makes for a reliable system by redirecting the application onto an alternate server when faults occur within any level of the system [34].

Whereas, utilising a bio-inspired particle multi-swarm optimization routing algorithm to ensure connections between IoT devices remain in a stable condition is also a feasible methodology for fault tolerant IoT networks. The multi-swarm strategy determines the optimal directions for selecting the multipath route while exchanging messages from any positions within the network [35].

In deployment scenarios where wireless technologies are used, such as those in WSNs, virtualisation technologies for resource optimisation can be used to assist heterogeneous service-oriented IoT applications [21]. There are many different redundancy and fail-over strategies across the IoT stack and ecosystem. The paper in [22] provides a comprehensive survey, particularly in the IoT sensing, routing and control processes [22]. Another paper [23] presents a comprehensive roadmap for achieving resilient IoT and CPS-based systems, with techniques encompassing anomaly detection and self-healing techniques to combat various risks and threats posed by internal/external agents across the physical, network and control layers.

2.4. Theories, Metrics and Measurements for System Reliability

Today's technological landscape requires high system availability and reliability. System downtime, failures, or glitches can result in significant revenue loss and more critically, compromising the safety of systems. Hence, measurement metrics for system reliability are used by companies to detect, track and manage system failures/downtime. Some commonly used metrics are Mean Time Before/Between Failure (MTBF), Mean Time to Recovery/Repair (MTTR), Mean Time to Failure (MTTF) and Mean Time to Acknowledge (MTTA). These metrics allow the monitoring and management of incidents, including tracking how often a particular failure occurs and how quickly can the system recover from such failure. For a more detailed explanation and derivation of these metrics, we refer the readers to [36].

There are many approaches presented in the literature to minimise system failures and manage incidents more effectively. These approaches span multiple industry applications, with most techniques focusing on optimising the MTBF. However, there is currently limited work in applying these techniques to IoT and edge computing. For example, Engelhardt et al. [37] investigated the MTBF for repairable systems by considering the reciprocal of the intensity function and the mean waiting time until the next failure; Kimura et al. [38] looked at MTBF from an applied software reliability perspective by analysing software reliability growth models as described by non-homogenous Poisson process; in two separate works, Michlin et al. [39,40] performed sequential MTBF testing on two systems and compared their performance; Glynn et al. [41] proposed a technique for efficient estimation of MTBF in non-Markovian models of highly dependable systems; Zagrirnyak et al. [42] discussed the use of neuronets in reliability models of electric machines for forecasting the failure of the main structural units (also based on MTBF); Suresh et al. [43] unconventionally applied MTBF as a subjective video quality metric, which makes for an interesting evaluation of MTBF in other application areas other than literal system failures.

Reliability curve is also an important metric that is used widely across many applications and industries [44]. Variants of reliability curves have been recently applied to IoT, edge computing and Mobile Edge Computing (MEC) in the advent of innovations in 5G. For example, Rusdhi et al. [45] performed an extensive system reliability evaluation on several small-cell heterogeneous cellular networks topologies and considered useful redundancy region, MTTF, link importance measure and system/link uncertainties as metrics to manage failure incidents; Liu et al. [46] propose a MEC-based framework that incorporates the reliability aspects of MEC in addition to the latency and energy consumption (by formulating these requirements as a joint optimisation problem); Chen-Feng Liu et al. [47,48] proposed two (related by separate) MEC network designs that also considered the latency and reliability constraints of mission-critical applications (in addition to average queue lengths and queue delays, by using Lyapunov stochastic optimization and extreme value theory); Han et al. [49] proposed a context-aware decentralised authentication MEC architecture for authentication and authorisation to achieve an optimal balance between operational costs and reliability.

Link importance measure is another important aspect for measuring system reliability. There are several techniques that considered link importance measure in the context of IoT and edge computing. For example, Silva et al. [50] developed a suite of tools to measure and detect link failures in IoT networks by measuring the reliability, availability and criticality of the devices; Benson et al. [51] proposed a resilient SDN-based middleware for data exchange in IoT edge-cloud systems, which dynamically monitors IoT network data and periodically sends multi-cast time-critical alerts to ensure the availability of system resources; Qiu et al. [52] proposed a robust IoT framework based on Greedy Model with Small World (GMSW), that determines the importance of different network nodes and communication links and allows the system to quickly recover using “small world properties” in the event of system failures; Kwon et al. [53] presented a failure prediction model using a Support Vector Machine (SVM) for iterative feature selection in Industrial IoT (IIoT) environments which calculates the relevance between the large amount of data generated by IIoT sensors and predict when the system is more likely to experience downtime, Dinh et al. [54] explores the use of Network Function Virtualisation (NFV) for efficient resource placements to manage hardware and software failures when deploying service chains in IoT Fog-Cloud networks.

3. Proposed Framework for IoT Failure Recovery

The aim of this paper is to propose a failure recovery framework for IoT applications. This framework focuses on the problem of failures in IoT applications that are deployed on end nodes and edge nodes that utilise container deployment techniques. This is a relatively new potential problem, as it is only recently that end nodes and edge nodes have the resources to use containerization. The proposed framework assumes that failures occur in IoT deployments, due to corruption or configuration in the end node or edge gateway [55]. The framework will passively monitor communications from the IoT node and gateway to detect potential failures. On discovery of a potential failure, the framework will deploy known good applications in containers. The general aim of the framework is to minimise downtime due to application failure.

Figure 2 presents the overall architecture of the proposed framework which would be bolted on to an existing IoT edge-computing deployment. The deployment controller is a monitoring agent which can send action requests to any IoT gateway or device. An action request is one of two things, either a reconfiguration request to reconfigure one to many IoT gateways or devices, or a redeployment request in which the code-base for the IoT gateways or devices can be updated. The following describes how each device of the IoT-edge deployment interacts with the proposed framework.

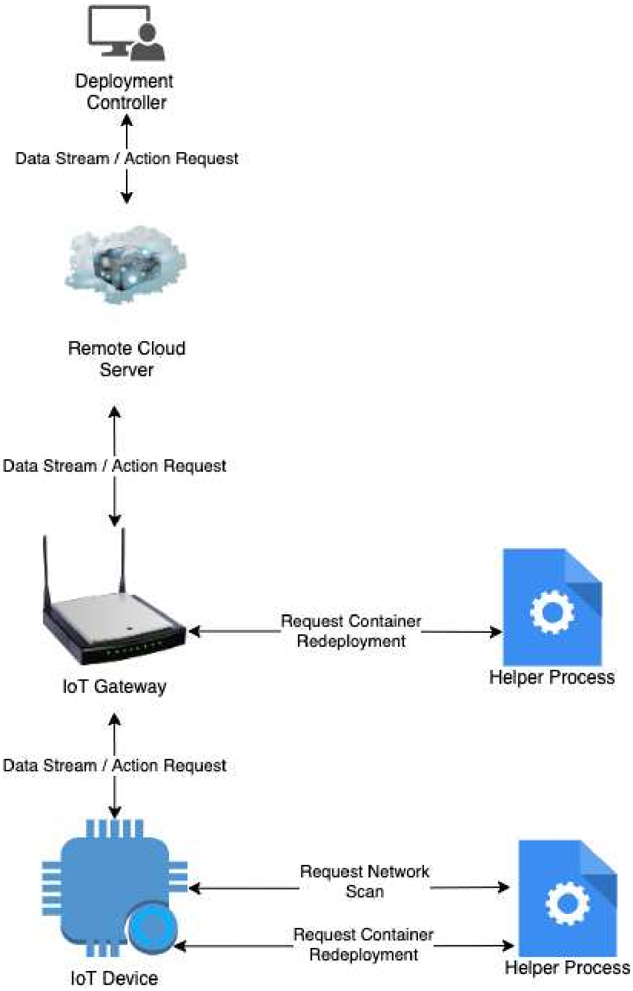

Deployment Controller
Data Stream / Action Request
Remote Cloud Server
Data Stream / Action Request
IoT Gateway
Request Container Redeployment
Helper Process
Data Stream / Action Request
IoT Device
Request Network Scan
Request Container Redeployment
Helper Process

Figure 2. Proposed framework for IoT failure recovery.

The IoT Device receives action requests from the gateway. If the action request is a redeployment request, the helper process is notified and proceeds to handle managing, rebuilding, restarting and deleting of containers and images for a seamlessly migration to the new version. The IoT Device can also request from the helper process to perform a network scan for any active gateways that are connected to the same network as the IoT device. The IoT device sends the collected data to the gateway.

The IoT Gateway receives action requests from the cloud server and will route the request to their corresponding target devices or ignore the request entirely if it is required to execute it. The IoT gateway will communicate with the helper process if the gateway is the target for the redeployment request. When the helper process is notified it will proceed to handle managing, rebuilding, restarting and deleting of containers and images for a seamlessly migration to the newer version. The IoT gateway receives the data stream and forwards it to the cloud server.

The Cloud Server receives action request from the deployment controller and will propagate the request to all connected gateways. The cloud server receives the data stream from the IoT gateway and forwards it to the deployment controller.

The Deployment Controller receives the response of an action request completion or failure from the cloud server which is sent from the IoT devices through the IoT gateway. The deployment controller initiates of action requests and receive the data stream from the cloud server and display them.

IoT-edge Deployments are described through *YAML* configuration files. Redeployment requests must have the *build.yml* file at the root of the update folder. The updated folder is located at /update of the current working directory of the deployment controller. The structure of the build file is illustrated in Figure 3.

```
target:
group: "*"
type: iot-gateway

actions:
- DELETE cache.json
- OVERWRITE Dockerfile
- OVERWRITE device-helper.js
- OVERWRITE gateway.js
- MKDIR unit-tests
```

Figure 3. Example build.yml.

The build file is split into two configuration sections. The target section indicates to the IoT gateways whether this build package is for a gateway or IoT device(s). The first path within the target section, target.group is the identifier of the device to update. Specifying the * key, like in the example, will address all devices of target.type to update. The second path of target within the target section, target.type specifies if the build package should be deployed on the IoT gateway or IoT device.

The second configuration section is the actions section which will tell either the IoT device or gateway what actions to perform to unpack the build. There are only 4 action commands that are usable within the build file actions configuration section. OVERWRITE (filename) will create a new file or overwrite the existing file, MKDIR (dir) will create a directory if it does not exist, DELETE (filename) will remove a file, finally REBUILD will rebuild the device's image. The REBUILD action will not run any subsequent actions and should always be at the bottom of the actions list.

The build.yml refers to a small number of files, which allow the building and rebuilding of the node software. The first of these, *cache.json* is a JSON file on the IoT gateways that store the time data was last received from an IoT device. This allows the IoT gateway to know when data was last sent even if the IoT gateway restarts. *Dockerfile* is the docker configuration file that describes how the image should be built.

device-helper.js is a JavaScript file that executes outside the docker container environment and is solely responsible for interacting and modifying the behaviour of the docker containers when requested to do so. The device-helper primary role is to delete and rebuild images to the new specifications, attempt to start the new image and if a failure occurs attempt failure recovery protocols. The device-helper script is always in continuous communication with the active running docker container.

gateway.js is the main JavaScript file that runs within the docker container and is responsible for keeping track of all the currently connected IoT devices and automatically removing any disconnected IoT device from the list. *gateway.js* has the critical responsibility of relaying data and requests to the required devices at any given point in time.

4. Experiment Setup

To evaluate the effectiveness and robustness of the proposed framework, a series of experiments have been performed with varying configurations. A testbed configuration was setup using common hardware platforms widely used for IoT nodes and gateways. The proposed architecture as described in Figure 2 using several Raspberry Pi small board computers. Raspberry Pis are commonly used in IoT testing and deployment due to their versatility. Their computational capabilities are typically suitable for most IoT deployment scenarios.

The experimental testbed setup consists of (i) a laptop as the deployment controller, (ii) a Raspberry Pi 4 as an IoT node and a (iii) Raspberry 3B+ as an edge node. The Raspberry Pi 4 Model B consists of a 4 GB LPDDR4-3200 SDRAM, Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC at 1.5 GHz. The Raspberry Pi 3 Model B+ consists of 1 GB LPDDR2 SDRAM Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC at 1.4 GHz. The laptop consists of an Intel i7-8565U at 2.00 GHz, 16 GB RAM, Geforce GTX 1050 GPU. Figure 4 illustrates the different devices in the experimental setup.

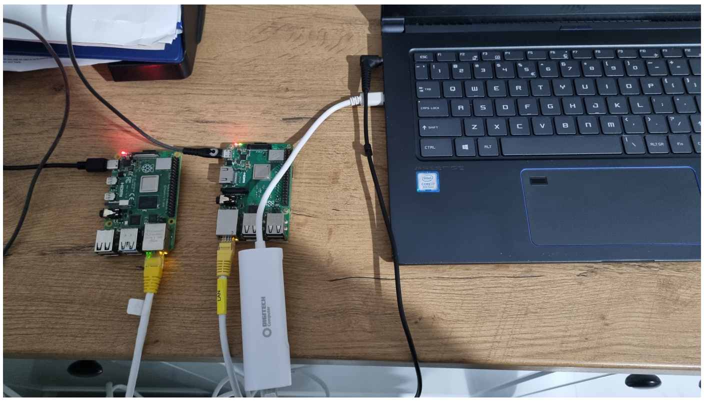


Figure 4. Experiment setup with IoT devices/nodes (RPi 4B), IoT gateway (RPi 3B+), deployment controller (laptop).

There are multiple stages of the IoT redeployment process. The deployment controller will first read the *build.yml* file and send it to the IoT gateway. Upon sending the *build.yml* file, the deployment controller will open a stream pipe to the IoT gateway and start streaming a large data payload. While the segmented data are being received at the IoT gateway, the *build.yml* is stored in memory and the large payload for the image rebuild process is written to the Raspberry Pi's disk space. After the streaming process has completed, the IoT gateway will again read the *build.yml* file and verify if the redeployment request must be forwarded to an IoT devices or the gateway should apply the update to itself.

If the redeployment request must be forwarded to an IoT device, the IoT gateway will open a direct stream pipe to the IoT device and proceed to send the *build.yml* first and then stream the payload that previously saved to the IoT gateway's disk. When each bit of the segmented stream data is received at the IoT device, the IoT device will write the file to the disk on the Raspberry Pi. After the stream process is complete on the IoT device, the IoT device will automatically incorporate the large payload to as a part of the build process for the new image.

The IoT device build process starts with the helper process as shown in Figure 2. It will first terminate the active container, rebuild the new Docker image with the included payload, run the new Docker image and when successfully starts delete the previous image and record the elapsed time for that process of building a new image and then starting the new container and forward the results back to the IoT gateway.

5. Experimental Evaluation

This section presents an experimental evaluation of the proposed framework by investigating five distinct scenarios using the same testbed setup described in Section 4: (i) IoT device redeployment; (ii) IoT gateway redeployment; (iii) IoT device sensor fault detection; (iv) IoT device redeployment failure detection; and (v) IoT gateway redeployment failure detection.

5.1. Scenario 1: IoT Device Redeployment

In this scenario, the aim is to calculate the time required for a redeployment request to be sent from the deployment controller and for the IoT device to successfully fulfill the request and send a response back to the corresponding IoT gateway. The data collected from this experiment will be used as a base value to determine on average how long the deployment controller should wait with no response from the IoT devices. IoT devices that exceed this threshold will be considered to have possibly encountered an error and further diagnostics on that device may be required.

Table 1 contains the results of 3 consecutive redeployment requests.

Table 1. IoT device redeployment duration.

Test Results	
Test Number	**Elapsed Time (secs)**
1	56.6
2	102.3
3	61.8
Average	73.6

The *Test Number* refers to the number of that test and *Elapsed Time* is the overall duration for the following actions to occur.

The redeployment processes/steps across multiple IoT devices are described as follows (presented in Figure 5):

1. The deployment controller sends a redeployment request for an IoT device to the Cloud server.
2. The Cloud server forwards the request to the IoT gateways.
3. The IoT gateway check what IoT devices to forward the request to.
4. The IoT device receives and completes the redeployment request.
5. The IoT device sends the response of success or error message back to the IoT gateway.
6. The IoT gateway then passes the message back to the cloud server and which then passes it to the deployment controller and the timer is stopped and time taken is displayed.

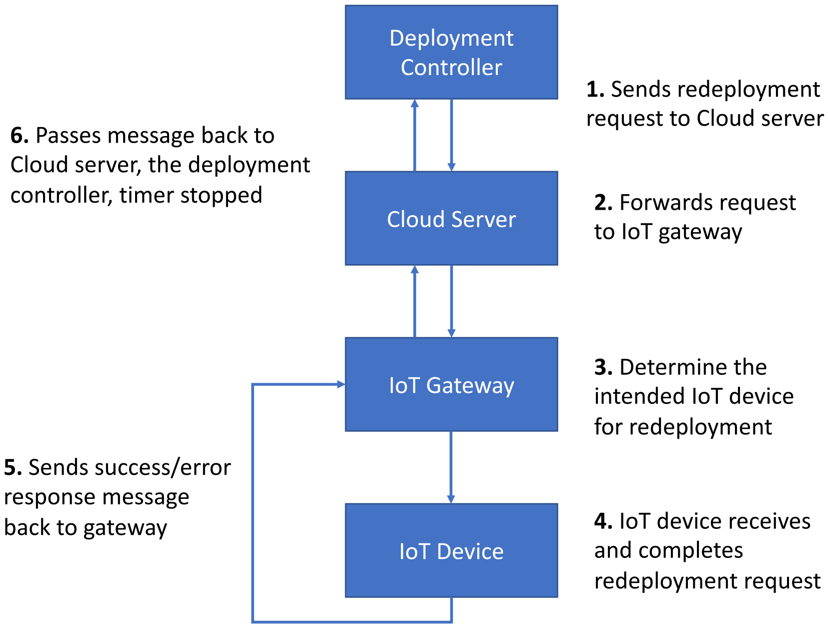

Deployment Controller
1. Sends redeployment request to Cloud server
6. Passes message back to Cloud server, the deployment controller, timer stopped
Cloud Server
2. Forwards request to IoT gateway
IoT Gateway
3. Determine the intended IoT device for redeployment
5. Sends success/error response message back to gateway
IoT Device
4. IoT device receives and completes redeployment request

Figure 5. Scenario 1: Redeployment process across multiple IoT device(s).

Scenario 1 explores the average time in seconds for a redeployment request to be successfully fulfilled. The results from this experiment can be used as a benchmark to determine a time frame window for how long the deployment controller should wait for requests from the IoT device or devices. If the response time exceeds the time frame, the system will assume that the device was unable to migrate to the new version and run further diagnostics as to determine if the device is fully offline or requires another redeployment request.

Further exploration of Scenario 1 led to varying the size of the redeployment package by 50 MB chunks and simultaneously recording the time data as inactive in the system while the IoT devices migrate to the new build.

Figure 6 displays a linear trend that a larger build package sent across the network will increase the time it takes for the redeployment request to be completed. At a 50 MB redeployment package, it takes approximately 150 s for a successfully redeploy and a 100 MB redeployment package on average requires 250 s. The time difference between each 50 MB increase in the redeployment package size is about 100 s.

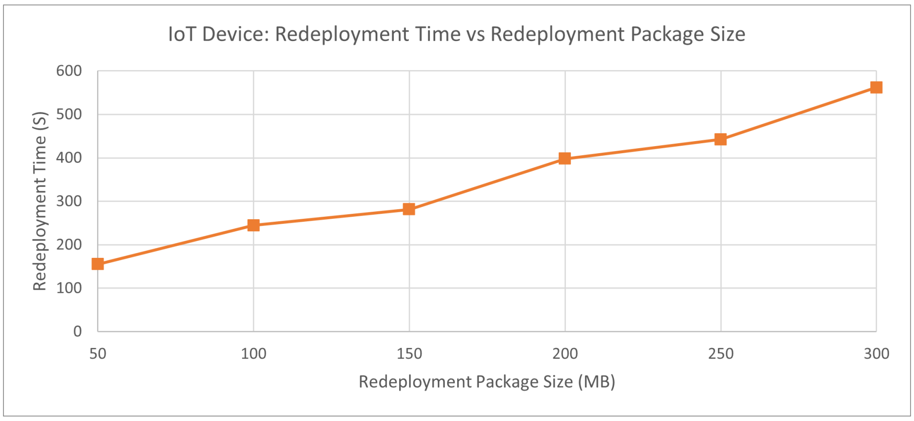


Figure 6. Redeployment Time vs. Redeployment Package Size.

Figure 7 displays a relatively linear trend where the larger the redeployment package, the greater time, no data are sent from the IoT devices to the IoT gateways. The time difference between each 100 MB redeployment package size is greater than 40 s, whereas the jump between 50 MB is relatively stagnant.

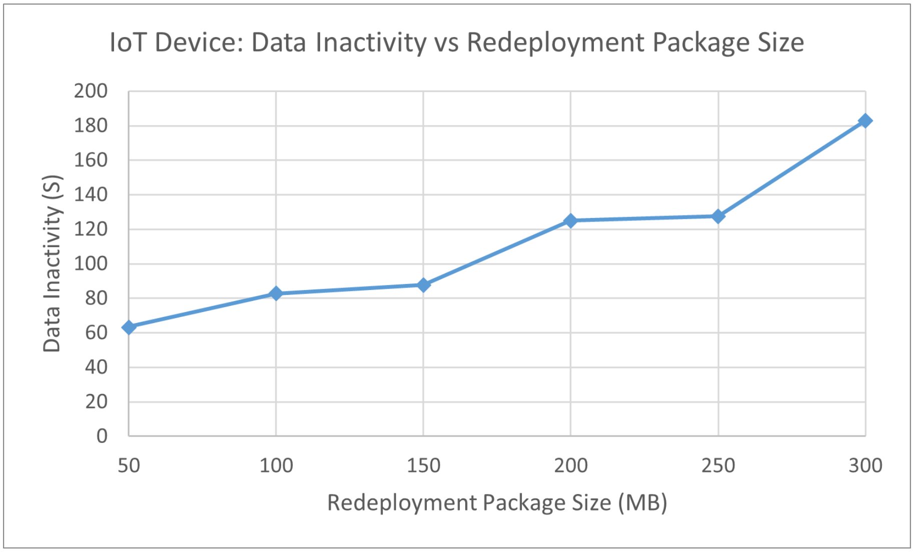


Figure 7. Data Inactivity vs. Redeployment Package Size.

Although Scenario 1 gives a benchmark for the average time it would take for a redeployment request to be successfully fulfilled, it does not take into account when a IoT device does successfully migrate to the new version but the time it took exceeded the response timeout window. In this scenario, the deployment controller assumes something is wrong with the IoT device and will resend the redeployment request which in effect will cause the already updated IoT device to forcefully update again. Scenario 1 results do not take into account build packages that are large or very small, which can greatly vary the average time required for the completion of a redeployment request.

5.2. *Scenario 2: IoT Gateway Redeployment*

Scenario 2 explores the average time in seconds for a redeployment request to be successfully fulfilled. The results from this experiment can be used as a benchmark to determine a time frame window for how long the deployment controller should wait for requests from the IoT gateway(s). If the response time exceeds the time frame, the system will assume that the gateway was unable to migrate to the new version and run further diagnostics as to determine if the gateway is fully offline or requires another redeployment request.

Table 2. IoT gateway redeployment times.

Test Results	
Test Number	**Elapsed Tim (secs)**
1	28.39
2	92.27
3	112.8
Average	77.6

Table 2 contains the results of 3 consecutive gateway redeployment requests. The *Test Number* refers to the number of that test and *Elapsed Time* is the overall duration for the following actions to occur.

The redeployment processes/steps across multiple IoT gateways are described as follows (presented in Figure 8):

1. The deployment controller sends a redeployment request for an IoT device to the Cloud server.
2. The Cloud server forwards the request to the IoT gateways.
3. The IoT gateway(s) check if the request is intended for them.
4. The IoT gateway(s) complete the redeployment request.
5. The IoT gateway(s) send the response whether success or error message back to the Cloud server.
6. The Cloud server passes it to the deployment controller and the timer is stopped and time taken is displayed.

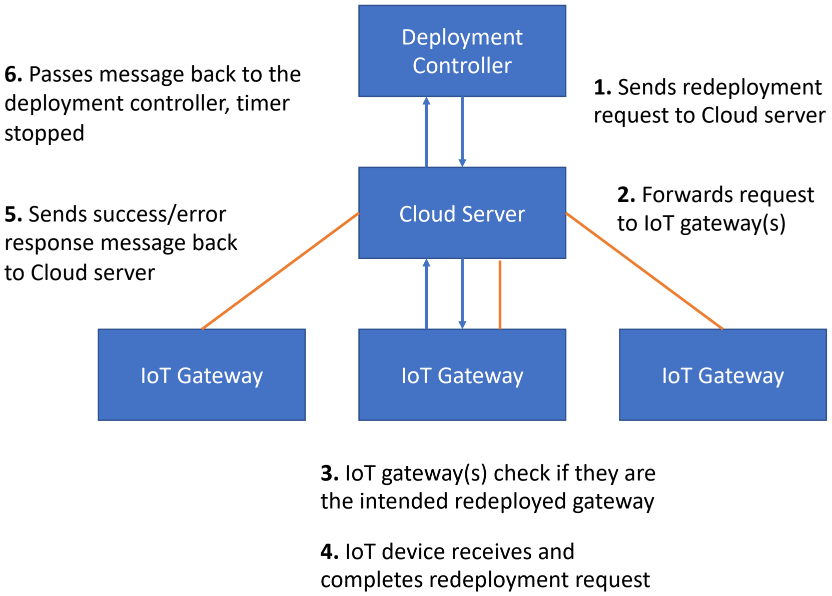

Deployment Controller
6. Passes message back to the deployment controller, timer stopped
1. Sends redeployment request to Cloud server
Cloud Server
5. Sends success/error response message back to Cloud server
2. Forwards request to IoT gateway(s)
IoT Gateway
IoT Gateway
IoT Gateway
3. IoT gateway(s) check if they are the intended redeployed gateway
4. IoT device receives and completes redeployment request

Figure 8. Scenario 2: Redeployment process across multiple IoT gateway(s).

Scenario 2 explores the average time in seconds for a redeployment request to be successfully fulfilled. The results from this experiment can be used as a benchmark to determine a time frame window for how long the deployment controller should wait for requests from the IoT gateway or gateways. If the response time exceeds the time frame, the system will assume that the gateway was unable to migrate to the new version and run further diagnostics as to determine if the gateway is fully offline or requires another redeployment request.

Further exploration of Scenario 2 lead to varying the size of the redeployment package by 50 MB chunks and simultaneously recording the time data as inactive in the system while the IoT devices migrate to the new build.

Figure 9 displays a very linear trend that a larger build package sent across the network will increase the time it takes for the redeployment request to be fulfilled. At a 50 MB redeployment package, it takes approximately 110 s for a successful redeploy and a 100 MB redeployment package on average requires 185 s. The time difference between each 50 MB increase in the redeployment package size is approximately 80 to 90 s.

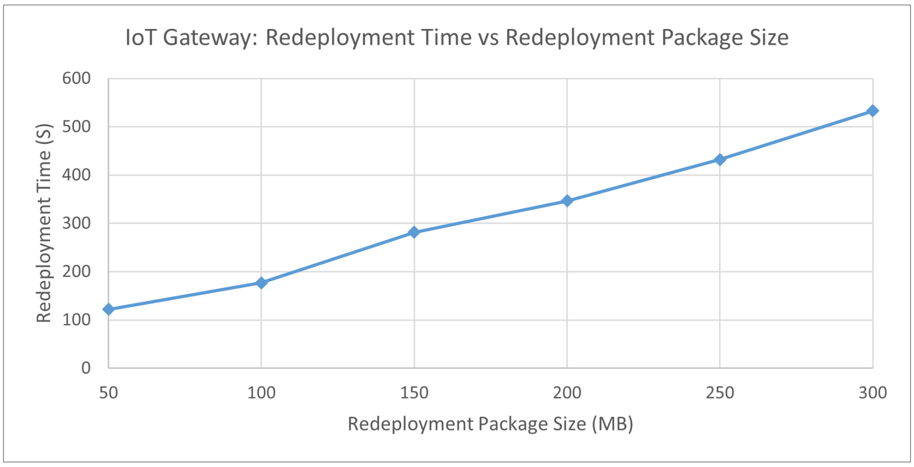


Figure 9. Redeployment Time vs. Redeployment Package Size.

Figure 10 displays a linear trend where a larger build package that is sent across the network increases the time no data will be received from the IoT devices. The time difference between each 100 MB redeployment package size is not consistent by a certain amount, but starts to sharply increase when the redeployment package is above 200 MB.

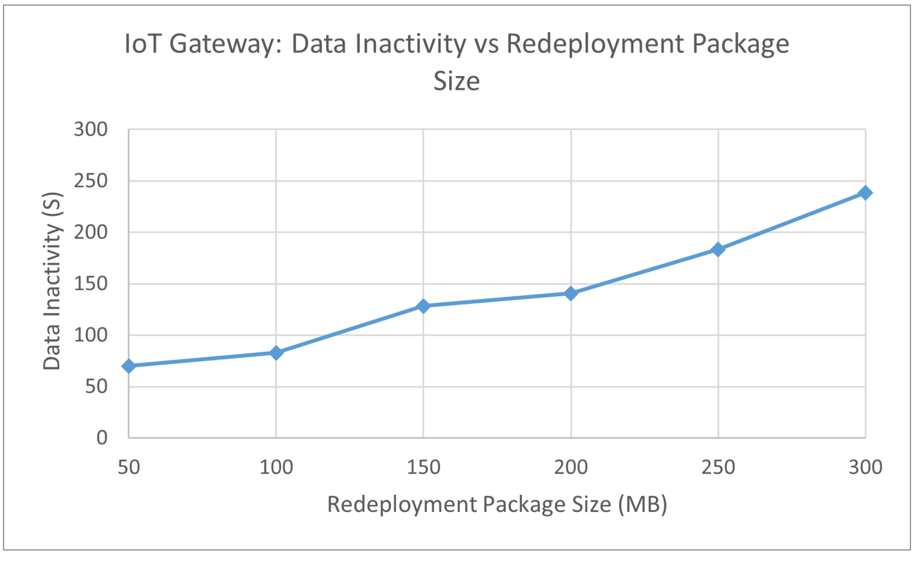


Figure 10. Data Inactivity vs. Redeployment Package Size.

Although Scenario 2 gives a benchmark for the average time it would take for a redeployment request to be successfully fulfilled, it does not take into account when a IoT gateway does successfully migrate to the new version and the time it took exceeded the response timeout window. In this scenario the admin panel assumes something is wrong with the IoT gateway and will resend the redeployment request which in effect will cause the already updated IoT gateway to forcefully update again. Scenario 2 results do not take into account build packages that are large or very small which can greatly vary the average time required for the completion of a redeployment request.

5.3. Scenario 3: IoT Device Sensor Fault Detection

Scenario 3 aims to test how long it takes a gateway to detect that an IoT device has stopped sending sensor data and how long the IoT device spends to recover and reinitialise data sending to the IoT gateway. For Scenario 3, each message sent from the IoT device had a 60 percent chance of failing and each request sent from the IoT gateway to the IoT device to recover had a 40 percent chance of success.

Scenario 3 explores the average time in seconds for a IoT device to recover from a sensor fault. The results from this experiment can be used as a benchmark to determine a time frame window for how long on average an IoT device will take to recover one of its critical functions.

Exploration of Scenario 3 lead to varying the frequency at which data are sent from 1 to 10 s and recording the elapsed time for the IoT gateway to detect an error and request for the IoT device to self-heal. The IoT gateway was configured to check for errors every 5 s.

Figure 11 is a demonstration of the time increasing for the IoT device to recover from sensor failure when the time between messages is increased with 10 s having the largest spread of varying recovery times. The 5 s interval for Figure 11 contains the smallest spread due to the fact that messages are being sent every 5 s to the IoT gateway and the IoT gateway itself was configured to check for problems with the IoT device every 5 s. There is a clear distinction that increasing the frequency at which data are sent allows for a much quicker recovery time with a smaller spread.

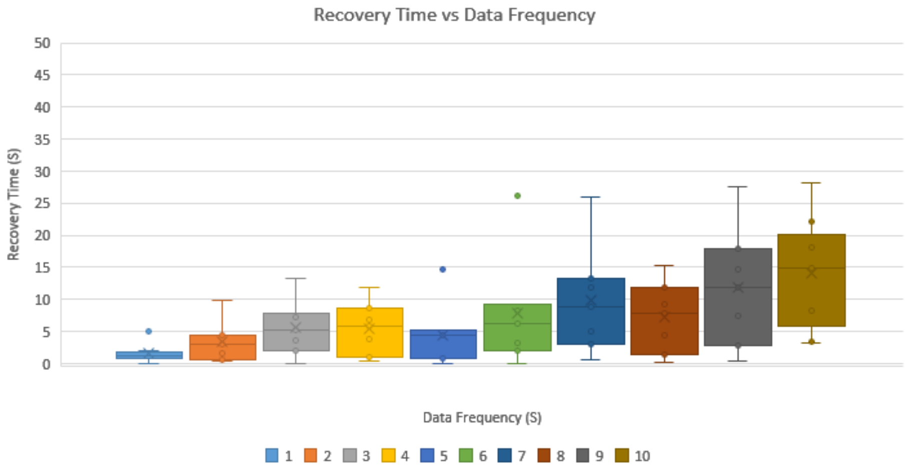


Figure 11. Recovery time vs. Data Frequency.

Although, Scenario 3 explores the average time in seconds for a IoT device to recover from a sensor fault. It is not a true measurement of what would happen in the real world as no physical sensors were attached and forced to fail and then recover. The sensor failures in this experiment are purely software-based and there is a fixed chance for sensor failure and recovery. Due to the software constraints imposed to test sensor recovery the results can only be regarded as a theoretical time constraint.

5.4. Scenario 4: IoT Device Redeployment Failure Detection

Scenario 4 explores the average time in seconds for a redeployment request to successfully recover after encountering failure on the IoT devices. The results from this experiment can be used as a guide to determine on average number of attempts until the redeployment is successful and as well as the elapsed time from failure detection to recovery.

Exploration of Scenario 4 led to varying the chance of and triggering an error during a redeployment request and measuring the elapsed time and attempts that is required for the system to recover and update the IoT device.

Redeployment time is the overall duration for the following actions to occur: deployment controller sends redeployment request to the IoT gateway. IoT gateways verifies the IoT devices to forward the request to. IoT device receives and complete the redeployment request. IoT device sends success of failure message back to the IoT gateway. IoT gateway checks if the redeployment request was successful and if it has failed it will request for the IoT device to perform the redeployment request again. Upon success of the redeployment request the IoT gateway will pass the message back to the deployment controller and the timer is stopped and time taken is displayed.

Figure 12 shows that the redeploy time is not greatly affected when the chance of a build error occurring is lower than 30 percent. The redeployment time begins to have a major increase from 60 percent chance of build error.

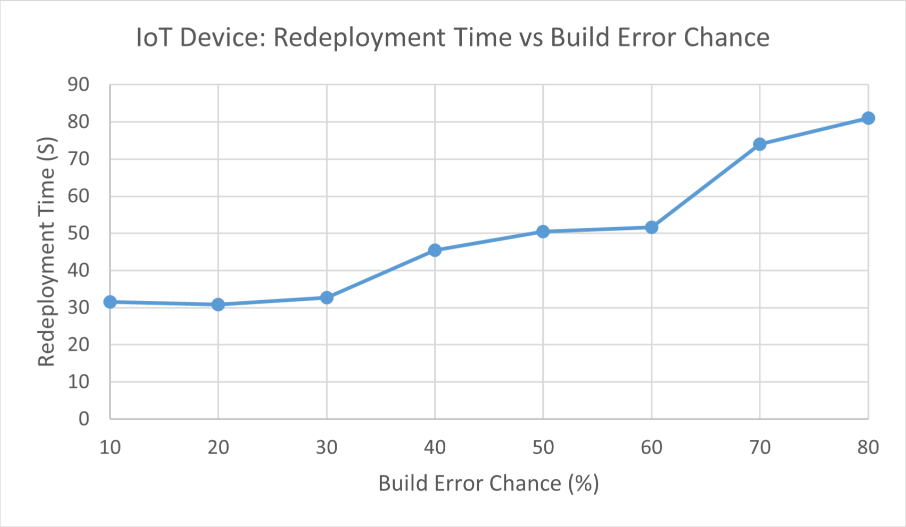


Figure 12. Redeployment Time vs. Build Error Chance.

Scenario 4 does not take into account when the IoT device attempting to update is stuck in an infinite loop due to attempting to start the program that has errors. Currently, the experiment will try as long as it needs to until the IoT device has updated which can result in an extremely high number of attempts.

5.5. Scenario 5: IoT Gateway Redeployment Failure Detection

Scenario 5 explores the average time in seconds for a redeployment request to successfully recover after encountering failure on the IoT gateways. The results from this experiment can be used as a guide to determine on average number of attempts until the redeployment is successful and as well as the elapsed time from failure detection to recovery.

Exploration of Scenario 5 lead to varying the chance of an triggering an error during a redeployment request and measuring the elapsed time and attempts that is required for the system to recover and update the IoT device.

Redeployment time is the overall duration for the following actions to occur: deployment controller sends redeployment request to the IoT gateway. IoT gateways verifies if it is required to execute the redeployment request. IoT gateway completes the redeployment request. IoT gateway helper process (See Figure 2) sends success of failure message back to the IoT gateway. Iot gateway checks if the redeployment request was successful and if it has failed it will request for the IoT gateway helper process to perform the redeployment request again. Upon success of the redeployment request the IoT gateway will pass the message back to the deployment controller and the timer is stopped and time taken is displayed.

Figure 13 shows that the redeploy time is not greatly affected when the chance of a build error occurring is lower than 30 percent. The redeployment time begins to have a major increase from 40 percent chance of build error and does not slow down. Results indicate that minimising the build error as much as possible for the gateway will ensure quality of service for end users will not be tarnished.

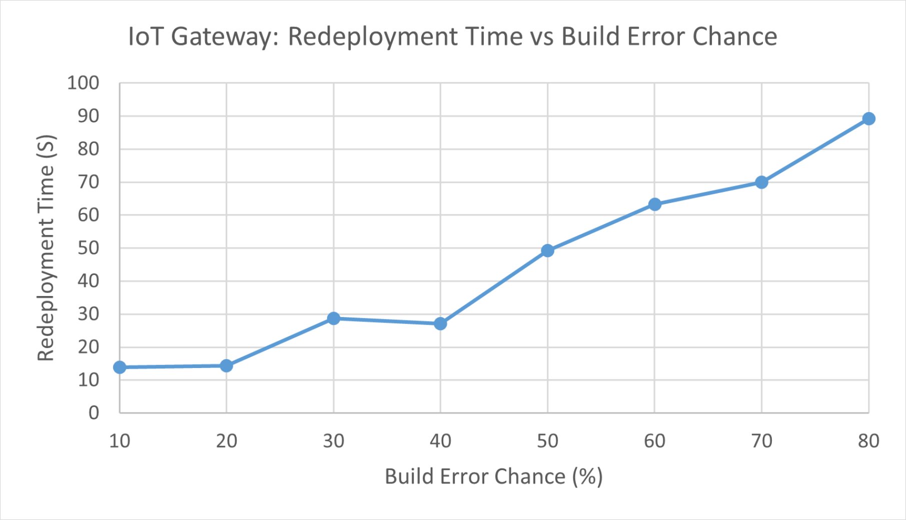


Figure 13. Redeployment Time vs. Build Error Chance.

Scenario 5 does not take into account when an IoT gateway attempting to update is stuck in an infinite loop due to attempting to start the program that has errors. Currently the experiment will try as long as it needs to until the IoT gateway has updated, which can result in an extremely high number of attempts.

6. Conclusions and Future Work

The focus of this paper is to add a layer of failure recovery to deployed container-based IoT edge applications. A framework was proposed the monitors deployed IoT applications and detects if either the IoT end node or IoT gateway has potentially failed. Once potential failures have been detected, the deployment controller will rebuild the application and deploy to containers in the effected node. The aim is to minimise downtime due to potential failures. This paper evaluated an implementation of this approach through a series of experiments testing different configurations for viability.

This paper argues that low latency IoT systems have a significantly higher fault detection and recovery time due to the system being able to verify more frequently if a device has yet to send a message for a specified number of seconds. The paper argues that decreasing the chance of build errors is critical in ensuring a that the quality of service remains consistent even when the devices require new redeployed software. It has demonstrated the design and implementation of a framework that automatically detects faults and attempts to automatically recover and as well as contains functionality to automatically reconfigure and redeploy software to all or targeted IoT devices or gateways. The proposed framework can be used to evaluate the occurrence of errors with an multi tiered system and the average theoretical recovery time for when an IoT device or gateway is down due to faults.

The work in this paper can be expanded on by implementing and refactoring the framework to be suitable to run on low powered devices. This paper had a primary focus on varying many factors to see the significant changes and impacts that could be seen if implemented on a real physical system. However, the paper uses higher tier technologies such as Docker, Raspberry Pis and real-time socket streams with the intended purpose to simulate the behaviour of low powered IoT devices and IoT gateways and the failures that can occur. The paper still presents a general overview of how faults can affect any given multi-tiered IoT application. The proposed framework for automatic failure recovery and the simulation of a multi tiered IoT system can be improved by rate-limiting in the network connection and processing power to closely mimic the behaviour of a lower powered device and what behaviours can occur when faults arise within these systems.

Author Contributions: Conceptualization, K.O., K.L. and J.K.; methodology, K.O., K.L. and J.K.; software, K.O.; validation, K.O., K.L. and J.K.; investigation, K.O.; resources, K.L.; data curation, K.O.; writing—original draft preparation, K.O.; writing—review and editing, K.O., K.L. and J.K.; visualization, K.O.; supervision, K.L. and J.K.; project administration, K.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, S.; Da Xu, L.; Zhao, S. The internet of things: A survey. *Inf. Syst. Front.* **2015**, *17*, 243–259. [CrossRef]
2. Atzori, L.; Iera, A.; Morabito, G. The internet of things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
3. IoT Analytics, State of IoT 2021: Number of Connected IoT Devices Growing 9% to 12.3 Billion Globally, Cellular IoT Now Surpassing 2 Billion. Available online: https://iot-analytics.com/number-connected-iot-devices/ (accessed on 21 November 2021).
4. Wang, W.; Lee, K.; Murray, D. A global generic architecture for the future Internet of Things. *Serv. Oriented Comput. Appl.* **2017**, *11*, 329–344. [CrossRef]
5. Wang, W.; Lee, K.; Murray, D. Building a generic architecture for the Internet of Things. In Proceedings of the 2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, VIC, Australia, 2–5 April 2013; pp. 333–338.
6. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
7. Ai, Y.; Peng, M.; Zhang, K. Edge computing technologies for Internet of Things: A primer. *Digit. Commun. Netw.* **2018**, *4*, 77–86. [CrossRef]
8. Salman, O.; Elhajj, I.; Kayssi, A.; Chehab, A. Edge computing enabling the Internet of Things. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 603–608.
9. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2018**, *6*, 6900–6919. [CrossRef]
10. Kua, J.; Armitage, G.; Branch, P. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1842–1866. [CrossRef]
11. Kua, J.; Nguyen, S.H.; Armitage, G.; Branch, P. Using Active Queue Management to Assist IoT Application Flows in Home Broadband Networks. *IEEE Internet Things J.* **2017**, *4*, 1399–1407. [CrossRef]
12. Kua, J.; Armitage, G.; Branch, P.; But, J. Adaptive Chunklets and AQM for Higher-Performance Content Streaming. *ACM Trans. Multimed. Comput. Commun. Appl.* **2019**, *15*, 115. [CrossRef]
13. Pan, J.; McElhannon, J. Future edge cloud and edge computing for internet of things applications. *IEEE Internet Things J.* **2017**, *5*, 439–449. [CrossRef]
14. Premsankar, G.; Di Francesco, M.; Taleb, T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [CrossRef]
15. Solaiman, E.; Ranjan, R.; Jayaraman, P.P.; Mitra, K. Monitoring internet of things application ecosystems for failure. *IT Prof.* **2016**, *18*, 8–11. [CrossRef]
16. Terry, D. Toward a new approach to IoT fault tolerance. *Computer* **2016**, *49*, 80–83. [CrossRef]
17. Moghaddam, M.T.; Muccini, H. Fault-tolerant iot. In Proceedings of the International Workshop on Software Engineering for Resilient Systems, Naples, Italy, 17 September 2019; pp. 67–84.
18. Nishiguchi, Y.; Yano, A.; Ohtani, T.; Matsukura, R.; Kakuta, J. IoT fault management platform with device virtualization. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 257–262.

19. Kodeswaran, P.A.; Kokku, R.; Sen, S.; Srivatsa, M. Idea: A system for efficient failure management in smart iot environments. In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications and Services, Singapore, 26–30 June 2016; pp. 43–56.
20. Di Modica, G.; Gulino, S.; Tomarchio, O. IoT fault management in cloud/fog environments. In Proceedings of the 9th International Conference on the Internet of Things, Bilbao Spain, 22–25 October 2019; pp. 1–4.
21. Kaiwartya, O.; Abdullah, A.H.; Cao, Y.; Lloret, J.; Kumar, S.; Shah, R.R.; Prasad, M.; Prakash, S. Virtualization in wireless sensor networks: Fault tolerant embedding for internet of things. *IEEE Internet Things J.* **2017**, *5*, 571–580. [CrossRef]
22. Rullo, A.; Serra, E.; Lobo, J. Redundancy as a measure of fault-tolerance for the Internet of Things: A review. In *Policy-Based Autonomic Data Governance*; Springer: Cham, Switzerland, 2019; pp. 202–226.
23. Ratasich, D.; Khalid, F.; Geissler, F.; Grosu, R.; Shafique, M.; Bartocci, E. A roadmap toward the resilient internet of things for cyber-physical systems. *IEEE Access* **2019**, *7*, 13260–13283. [CrossRef]
24. Pahl, C.; Ioini, N.E.; Helmer, S.; Lee, B. An architecture pattern for trusted orchestration in IoT edge clouds. In Proceedings of the 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, Spain, 23–26 April 2018; pp. 63–70. [CrossRef]
25. Ahuja, S.P.; Wheeler, N. Architecture of fog-enabled and cloud-enhanced internet of things applications. *Int. J. Cloud Appl. Comput. (IJCAC)* **2020**, *10*, 1–10. [CrossRef]
26. Hassan, N.; Gillani, S.; Ahmed, E.; Yaqoob, I.; Imran, M. The Role of Edge Computing in Internet of Things. *IEEE Commun. Mag.* **2018**, *56*, 110–115. [CrossRef]
27. Javed, A.; Heljanko, K.; Buda, A.; Främling, K. CEFIoT: A fault-tolerant IoT architecture for edge and cloud. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 813–818. [CrossRef]
28. Silva, J.D.C.; Rodrigues, J.J.P.C.; Saleem, K.; Kozlov, S.A.; Rabêlo, R.A.L. M4DN.IoT-A Networks and Devices Management Platform for Internet of Things. *IEEE Access* **2019**, *7*, 53305–53313. [CrossRef]
29. Dang, H.; Quan, L. SD-IoTR: An SDN-based Internet of Things reprogramming framework. *IET Netw.* **2020**, *9*, 305–314.
30. Bera, S.; Misra, S.; Roy, S.K.; Obaidat, M.S. Soft-WSN: Software-Defined WSN Management System for IoT Applications. *IEEE Syst. J.* **2018**, *12*, 2074–2081. [CrossRef]
31. Wu, D.; Arkhipov, D.I.; Asmare, E.; Qin, Z.; McCann, J.A. UbiFlow: Mobility management in urban-scale software defined IoT. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 26 April–1 May 2015; pp. 208–216.
32. Medjek, F.; Tandjaoui, D.; Djedjig, N.; Romdhani, I. Fault-tolerant AI-driven Intrusion Detection System for the Internet of Things. *Int. J. Crit. Infrastruct. Prot.* **2021**, *34*, 100436. [CrossRef]
33. Power, A.; Kotonya, G. A microservices architecture for reactive and proactive fault tolerance in iot systems. In Proceedings of the 2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Chania, Greece, 12–15 June 2018; pp. 588–599.
34. Grover, J.; Garimella, R.M. Reliable and Fault-Tolerant IoT-Edge Architecture. In Proceedings of the 2018 IEEE SENSORS, New Delhi, India, 28–31 October 2018; pp. 1–4. [CrossRef]
35. Hasan, M.; Al-Turjman, F. Optimizing Multipath Routing with Guaranteed Fault Tolerance in Internet of Things. *IEEE Sens. J.* **2017**, *17*, 6463–6473. [CrossRef]
36. MTBF, MTTR, MTTA and MTTF: Understanding a Few of the Most Common Incident Metrics. Available online: https://www.atlassian.com/incident-management/kpis/common-metrics (accessed on 24 November 2021).
37. Engelhardt, M.; Bain, L.J. On the mean time between failures for repairable systems. *IEEE Trans. Reliab.* **1986**, *35*, 419–422. [CrossRef]
38. Kimura, M.; Yamada, S.; Osaki, S. Statistical software reliability prediction and its applicability based on mean time between failures. *Math. Comput. Model.* **1995**, *22*, 149–155. [CrossRef]
39. Michlin, Y.H.; Grabarnik, G.Y. Sequential testing for comparison of the mean time between failures for two systems. *IEEE Trans. Reliab.* **2007**, *56*, 321–331. [CrossRef]
40. Michlin, Y.H.; Grabarnik, G.Y.; Leshchenko, E. Comparison of the mean time between failures for two systems under short tests. *IEEE Trans. Reliab.* **2009**, *58*, 589–596. [CrossRef]
41. Glynn, P.W.; Heidelberger, P.; Nicola, V.F.; Shahabuddin, P. Efficient estimation of the mean time between failures in non-regenerative dependability models. In Proceedings of the 25th conference on Winter Simulation, Los Angeles, CA, USA, 12–15 December 1993; pp. 311–316.
42. Zagirnyak, M.; Prus, V. Use of neuronets in problems of forecasting the reliability of electric machines with a high degree of mean time between failures. *Prz. Elektrotechniczny (Electr. Rev.)* **2016**, *92*, 132–135. [CrossRef]
43. Suresh, N.; Jayant, N. 'Mean time between failures': A subjectively meaningful video quality metric. In Proceedings of the 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, Toulouse, France, 14–19 May 2006; Volume 2.
44. Duane, J. Learning curve approach to reliability monitoring. *IEEE Trans. Aerosp.* **1964**, *2*, 563–566. [CrossRef]
45. Rushdi, A.M.A.; Hassan, A.K.; Moinuddin, M. System reliability analysis of small-cell deployment in heterogeneous cellular networks. *Telecommun. Syst.* **2020**, *73*, 371–381. [CrossRef]

46. Liu, J.; Zhang, Q. Offloading schemes in mobile edge computing for ultra-reliable low latency communications. *IEEE Access* **2018**, *6*, 12825–12837. [CrossRef]
47. Liu, C.F.; Bennis, M.; Poor, H.V. Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In Proceedings of the 2017 IEEE Globecom Workshops (GC Wkshps), Singapore, 4–8 December 2017; pp. 1–7.
48. Liu, C.F.; Bennis, M.; Debbah, M.; Poor, H.V. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. *IEEE Trans. Commun.* **2019**, *67*, 4132–4150. [CrossRef]
49. Han, B.; Wong, S.; Mannweiler, C.; Crippa, M.R.; Schotten, H.D. Context-awareness enhances 5G multi-access edge computing reliability. *IEEE Access* **2019**, *7*, 21290–21299. [CrossRef]
50. Silva, I.; Leandro, R.; Macedo, D.; Guedes, L.A. A dependability evaluation tool for the Internet of Things. *Comput. Electr. Eng.* **2013**, *39*, 2005–2018. [CrossRef]
51. Benson, K.E.; Wang, G.; Venkatasubramanian, N.; Kim, Y.J. Ride: A resilient IoT data exchange middleware leveraging SDN and edge cloud resources. In Proceedings of the 2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI), Orlando, FL, USA, 17–20 April 2018; pp. 72–83.
52. Qiu, T.; Luo, D.; Xia, F.; Deonauth, N.; Si, W.; Tolba, A. A greedy model with small world for improving the robustness of heterogeneous Internet of Things. *Comput. Netw.* **2016**, *101*, 127–143. [CrossRef]
53. Kwon, J.H.; Kim, E.J. Failure Prediction Model Using Iterative Feature Selection for Industrial Internet of Things. *Symmetry* **2020**, *12*, 454. [CrossRef]
54. Dinh, N.T.; Kim, Y. An efficient availability guaranteed deployment scheme for IoT service chains over fog-core cloud networks. *Sensors* **2018**, *18*, 3970. [CrossRef] [PubMed]
55. Makhshari, A.; Mesbah, A. IoT bugs and development challenges. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 22–30 May 2021; pp. 460–472.

electronics

MDPI

Article

Linked-Object Dynamic Offloading (LODO) for the Cooperation of Data and Tasks on Edge Computing Environment

Svetlana Kim [1], Jieun Kang [2] and YongIk Yoon [2,*]

[1] Research & Business Development Foundation, Sookmyung Women's University, Seoul 04310, Korea; xatyna@sookmyung.ac.kr
[2] Department of IT Engineering, Sookmyung Women's University, Seoul 04310, Korea; kje9209@sookmyung.ac.kr
* Correspondence: yiyoon@sookmyung.ac.kr; Tel.: +82-10-5091-0388

Abstract: With the evolution of the Internet of Things (IoT), edge computing technology is using to process data rapidly increasing from various IoT devices efficiently. Edge computing offloading reduces data processing time and bandwidth usage by processing data in real-time on the device where the data is generating or on a nearby server. Previous studies have proposed offloading between IoT devices through local-edge collaboration from resource-constrained edge servers. However, they did not consider nearby edge servers in the same layer with computing resources. Consequently, quality of service (QoS) degrade due to restricted resources of edge computing and higher execution latency due to congestion. To handle offloaded tasks in a rapidly changing dynamic environment, finding an optimal target server is still challenging. Therefore, a new cooperative offloading method to control edge computing resources is needed to allocate limited resources between distributed edges efficiently. This paper suggests the LODO (linked-object dynamic offloading) algorithm that provides an ideal balance between edges by considering the ready state or running state. LODO algorithm carries out tasks in the list in the order of high correlation between data and tasks through linked objects. Furthermore, dynamic offloading considers the running status of all cooperative terminals and decides to schedule task distribution. That can decrease the average delayed time and average power consumption of terminals. In addition, the resource shortage problem can settle by reducing task processing using its distributions.

Keywords: edge computing; offloading computation; distributed collaboration; data processing; dynamic offloading; IoT; gateways

Citation: Kim, S.; Kang, J.; Yoon, Y. Linked-Object Dynamic Offloading (LODO) for the Cooperation of Data and Tasks on Edge Computing Environment. *Electronics* **2021**, *10*, 2156. https://doi.org/10.3390/electronics10172156

Academic Editors: Ka Lok Man and Kevin Lee

Received: 30 June 2021
Accepted: 30 August 2021
Published: 3 September 2021

1. Introduction

Nowadays, with the rapid evolution of technology and a vast number of Internet of things (IoT) devices, including individual units, have improved processing ability (robust computing environment) with various embedded sensors. Due to this progress, the IoT devices can perform multiple functions (such as receiving/refining data from sensors in real-time, transferring, processing, and storing) independently and without computing resources in the server (external cloud) [1,2]. Edge computing has been proposed to process existing integrated service platforms by moving computing tasks from cloud servers to IoT devices. Edge computing is the workload of devices and offloads computational tasks to nearby computing devices, significantly reducing processing latency [3]. Existing high-latency and low-reliability cloud computing solutions are challenging to support time-critical cloud/IoT services requirements in transmission data and task processing delay. Edge computing is a centralized architecture where all nearby service requests are directed to a 'central' edge server. However, since the computing power of edge servers is not powerful as cloud-based servers, some issues such as resource limitations and computing latency between multiple competing tasks still occur [4,5].

Various approaches such as Mobile Cloud Computing (MCC) and Multiple Access Edge Computing/Mobile Edge Computing (MEC) support complementary cloud computing solutions using services adjacent to the edge network to cope with this disturbing problem [6–8]. MCC offloading, which uses unlimited resources in a cloud for some tasks, is the method to reduce loads of mobile devices. However, MCC has considerable disadvantages, such as low expandability, long propagation distance between the cloud server and devices, excessive consumption of limited bandwidth, personal protection and security problems [9]. Current MCC offloading methods cannot guarantee real-time data transfer and task delay, making them unsuitable for latency-sensitive applications. On the other hand, MEC is the offload concept of tasks collaboratively by leveraging both the MEC server and the end device (such as a mobile phone). Due to the limited battery life of the mobile device and the growing number of latency-sensitive applications, offloading tasks come with additional overhead in terms of latency and power consumption [10,11]. This problem was partially solved by dividing each task into local task and offload task. Local tasks are processing on the end device, and offload tasks are performing on the MEC server. However, computing resources can increase in some areas, which can exacerbate network problems and even cause issues that affect task execution times.

In recent years, much research has proposed addressing the resource limitations and computing latency issues by scheduling strategies of collaboration task offloading in an edge computing environment. The computing resource on edge is mainly composed of intelligent devices (note: intelligent devices (e.g., smart sensors and smartphones, can access a network, resulting in a considerable amount of network data) are called edge devices/end devices) and edge servers. For example, in [12–14], there is a job scheduling algorithm that utilizes the resources of cloud servers to handle highly overload situations. In [15], leverage resources in edge servers by offloading all computing-intensive tasks of the edge device to the edge server. In this case, computing resources and storage of devices are not utilized properly and wasted. In addition, multiple computation tasks or many devices may access the edge server at the same time. As a result, the workload will increase, long queues, and processing delays of tasks accordingly. In limited computational resources, a multi-MEC system has been proposing by joint communication offloading methods at the ends and edges [16–19]. However, due to the dynamic environment of the computing system, it is not easy to achieve task offload performance. In addition, the resource-rich IoT devices do not collaborate and are fully underutilized, resulting in a waste of their computing resources.

Since task offloading plays a critical role in edge-based service, edge must take full advantage of IoT's communication and computational resources. To this end, the edge server needs a balance task scheduler that, according to the characteristics of the computing task and device status, decides which tasks to offload to which IoT devices. Our previous paper [20] suggested distributed collaboration for computing offloading architectures. The architecture consists of a balanced collaboration system by assigning the master node, the second node, and the action node individually to edge nodes which contribute to all connected and communicable areas. It balanced computing resources through edge-to-edge collaboration and minimized latency due to relatively unbalanced overloads. Based on the [20] architecture, this paper proposes the LODO (Linked-object dynamic offloading) algorithm. LODO focuses on reasonable use of the computation resources of the IoT devices to processing computing tasks efficiently. The IoT devices are called edge nodes in the LODO algorithm. The LODO algorithm receives the compute tasks offloaded by the edge nodes and provides the hybrid state with offloading the tasks according to the resources and state of all edge nodes. The offloading location of computing tasks is uncertain and has various types; so are the compute resources status and performance gaps between nodes. Considers both characteristics, the computing resources of edge nodes can be fully balancing utilized. The main feature here is that the computing task can be reasonably offloaded to different edge nodes by collaborative processing.

The main contributions of this article are summarized as follows:

- We propose a Dynamic offloading method (DOM) with hybrid states that contains the resource requirements of offloading tasks and real-time resource availability information of each edge node. According to the current computational task execution state, the hybrid state is formed based on information related to all edge nodes (e.g., compute, storage, and network state). The edge computing reasonably offloaded the task to suitable edge nodes according to the hybrid state.
- We propose a linked-object algorithm that, according to offloading task status, provides two cooperation offloading options. If the computing power of an edge node cannot meet the task requirements, performing the task-linked option. When the edge node's memory/storage computing resource is not more available, executing the data-linked option. Each of these options helps solve the problem of load imbalance among computing nodes.
- We also investigate the application of forest fire that requires real-time sensing data and various time-critical tasks. For example, real-time data (such as temperature, humidity, wind, slope, and others) is necessary to predict the possibility of forest fires moving to other areas and the diffusion speed. If the task processing time is a large percentage of the total time, thus resulting in a processing delay of other tasks. Moreover, the total service time may become unacceptable when performing large tasks at the close but slow (low computing power) edge nodes. In this case, the LODO algorithm can offload computation tasks to suitable edge node according to the real-time computing resources status of the edge nodes.

The rest of this paper is organized as follows. Section 2 explains DOM (Dynamic Offloading Method) by hybrid states is formulating. Section 3, introduces details of the LODO properties approach for collaboration offloading. Scenario and results are presented in Section 4. Section 5 presents the discussion. Finally, Section 6 concludes this.

2. Dynamic Offloading Method (DOM) with Hybrid States

Based on our previous paper [20], this section identifies the reason for loads according to data-linked and task-linked, which are the major processes of an edge node. On edge node have edge gateways, end devices, and schedulers. Edge gateways can provide computing (*CPU*), storage (memory), bandwidth, and other system resources for edge computing operations. It is essential to reasonably use these computing resources depending on the offloading configuration to solve computation offloading efficiently. For example, suppose the goal is to reduce the load for data-linked processing. In this case, memory utilization is more important than *CPU* ratio, so utilizing the edge with more storage resources is necessary. On the other hand, task-linked processing increases *CPU* usage; it should use an edge with free computing resources. Using edge resources according to processing status can increase edge resource utilization and reduce the average execution time of computational tasks.

The resource range that can process in the edge node is definition 80% of *CPU* and 90% of memory. We define this as the reference overload threshold. If one exceeds the existing threshold range using a monitoring process, it determines the overload of data-linked and task-linked. The ratio is classified as a task-linked overload with high *CPU* usage and categorized as a data-linked overload with high memory usage (ref. Figure 1).

Depending on the hybrid state of the edge node, the dynamic offloading method (DOM) defines an overload range that considers the correlation/relationship between data and tasks. The following section describes whole activity states (hybrid states) through discussion and formulation about overload issues according to data and task of an edge node using DOM.

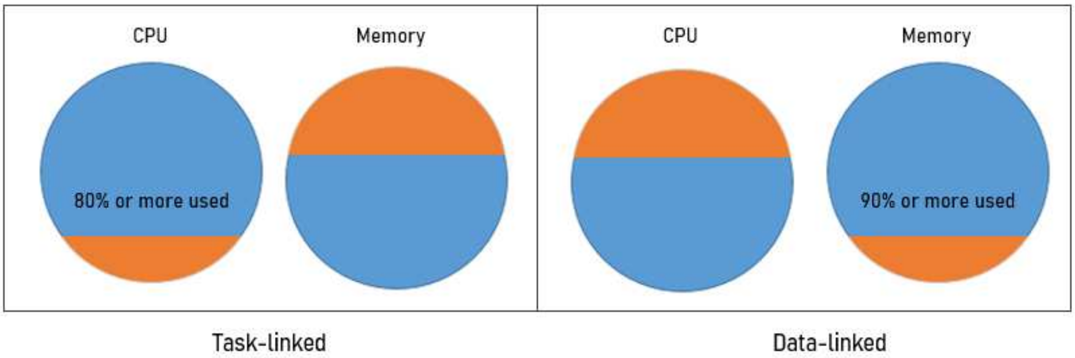


Figure 1. The resource range for data-linked and task-linked.

2.1. Expression of Hybrid States on Edge Node

As shown in Figure 2, edge computing is a process of collaboration offloading by monitoring the state of all edge nodes that change dynamically. Edge node consists of end devices that receive, store, and preprocesses the sensing data, and edge gateways analyze and outputs the results following the purpose of the domain. Since edge nodes have different resources and characteristics, real-time monitoring is a requirement in an ever-changing environment. In addition, there are different data and performance characteristics during the analysis depending on the purpose of the domain, so the offload characteristics may also differ. Representative offloading characteristics can classify into "offloading due to data," "offloading due to task," or "offloading due to Data & Task," and the offloading range need determine according to each characteristic. According to the current offloading execution state, the edge node must also provide suitable edge nodes.

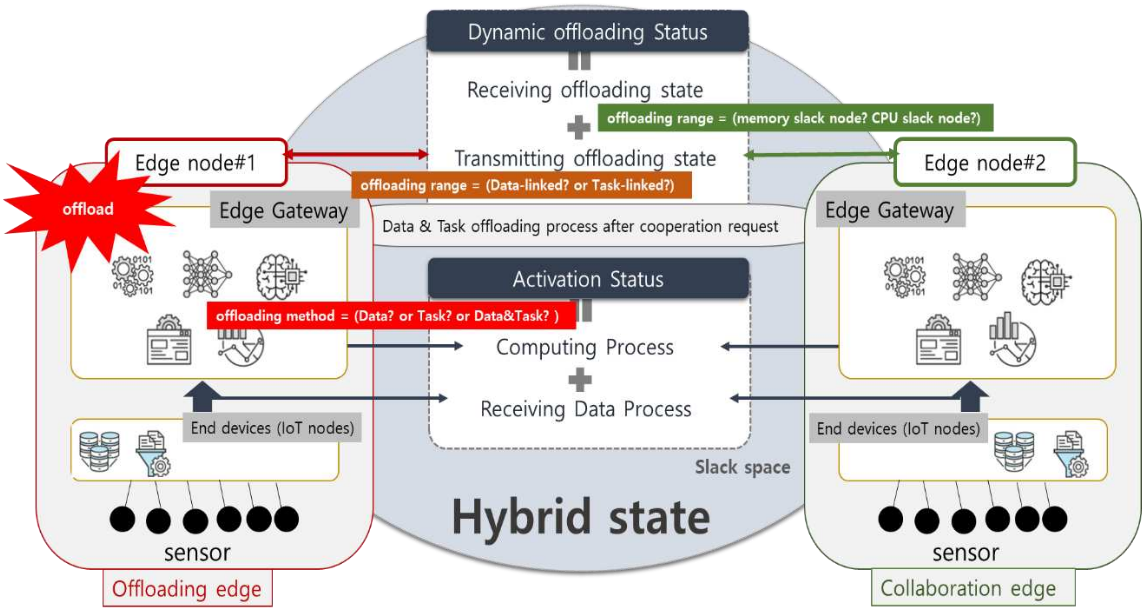

Dynamic offloading Status
Receiving offloading state
offloading range = (memory slack node? CPU slack node?)
Transmitting offloading state
offloading range = (Data-linked? or Task-linked?)
Edge node#1
offload
Edge Gateway
Edge node#2
Edge Gateway
Data & Task offloading process after cooperation request
Activation Status
offloading method = (Data? or Task? or Data&Task?)
Computing Process
Receiving Data Process
End devices (IoT nodes)
End devices (IoT nodes)
Slack space
Hybrid state
sensor
sensor
Offloading edge
Collaboration edge

Figure 2. Monitoring process with hybrid states on edge node.

The hybrid state to reasonably use the computing resource, and computing task is divide into three statuses, the "Activation status", "Dynamic offloading status," and "Slack space." The activation status means accurate assessment and incorporating the edge node state consumption due to the offloaded tasks. Receiving data and computing processes are the functions that derive an offloading task method to the due to data or/and task according to the problem computing resource. Offloading data method is when the edge node uses a memory value is more than 90%, and the task offload method is when the *CPU* value is more than 80%.

The dynamic offloading between edges nodes is one of the essential factors in determining task range based on offloading task method. According to the task-method attributes, the transmitting offloading state extract the range of data-linked or task-linked offloading. Transmitting offloading state extracts the range of data-linked or task-linked offloading according to the offloading method. The Receiving offloading state selects the offloading range that can tolerate in the slack space of the cooperative edge node, excluding 10% memory and 20% *CPU*. Here the slack space means a value obtained by subtracting activation status and dynamic offloading status from the hybrid state. This value can describe the state of the edge node like a hybrid state, should have more than 10% memory or 20% *CPU* basically to prevent overload.

The following Equation (1) computes the hybrid state of the edge node's dynamic active and slack space, depending on the data and task computation function. Equation (1) indicates the total performance state of an edge node. Based on data and task, the hybrid state is the sum of Equation (1a) the current performance (activation) state, Equation (1b) the offloading state, and the slack space.

$$\begin{gathered} Hybrid\ state\{CPU,\ Memory\} \\ = Activation\ Status\{C_i,\ M_i\} + Dynamic\ offloading\ Status\{C_i, M_i\} \\ + Slack\ Space\{C_i, M_i\}, \end{gathered} \tag{1}$$

Equation (1a) shows the current activation state, where C_i is *CPU* and M_i is Memory. This definition is a sum of the data received by the process of edge node and the computing process of data and tasks.

$$\begin{gathered} Activation\ status\{CPU,\ Memory\} \\ = Receiving\ Data\ Process\{C_i,\ M_i\} + Data\&Task\ Computation\ Process\{C_i, M_i\} \end{gathered} \tag{1a}$$

Equation (1b) shows that the dynamic offloading state is defined as the sum of the offloading state by overloads in the edge node and the collaboration offloading state for other nodes.

$$\begin{gathered} Dynamic\ offloading\ Status\{CPU,\ Memory\} \\ = Transmission\ offloading\ Process\{C_i,\ M_i\} + Receiving\ offloading\ Process\{C_i, M_i\}, \end{gathered} \tag{1b}$$

2.2. Definition of Assigning an Offloading Range

To assign the offloading range, the group should divide as being data-linked or task-linked. First, the offloading range based on memory is grouped in data connectivity and minimizes its overlapped offloading. In this computation, let D = $\{d_1,\ d_2,\ \ldots d_n\}$, where D is the set of edge gateway (device), and d_i is the *i*-th edge gateway (device). Equation (2) means the energy E generated by moving data from edge node $d1$ to $d2$. The $S(d1, d2)$ is the total amount of data that must transmit between nodes; hence, the group's energy consumption based on data connectivity is proportional to the amount of data. Therefore, the minimum range of transmitted energy is extracting by the amount of data after selecting collaboration edge nodes.

$$E(d1, d2) = ((w \times S(d1, d2)) \div bandwidth) \times Power_{wifi}, \tag{2}$$

Second, minimizing the optimum edge node's performance time includes offloading by grouping task connectivity based on *CPU* for the offloading range. Because of various *CPU* capacities according to the slack space, each edge node has different processing speeds. Therefore, it is essential to assign the offloading range because the total processing time is dependent on how to use the *CPU*. Equation (3) shows the difference of processing time between the existing node and the collaborating node of the group based on task connectivity for offloading. The $t_{o(i)}$ is the consumed time in an overloaded edge node, $\mathrm{t}_{\mathrm{c}(i)}$ is the consumed time in a collaborated node. The w is a weighted factor that depends on the slack space of the collaboration node. As long as the slack space increases, the weighted factor (w) also increases, so its consumed time becomes rapidly faster than the overload edge. Therefore, the range with a maximum weighted factor must extract as the offloading range by selecting the collaboration edge node.

$$\mathrm{t}_{\mathrm{c}(i)} = t_{o(i)} \div w, \tag{3}$$

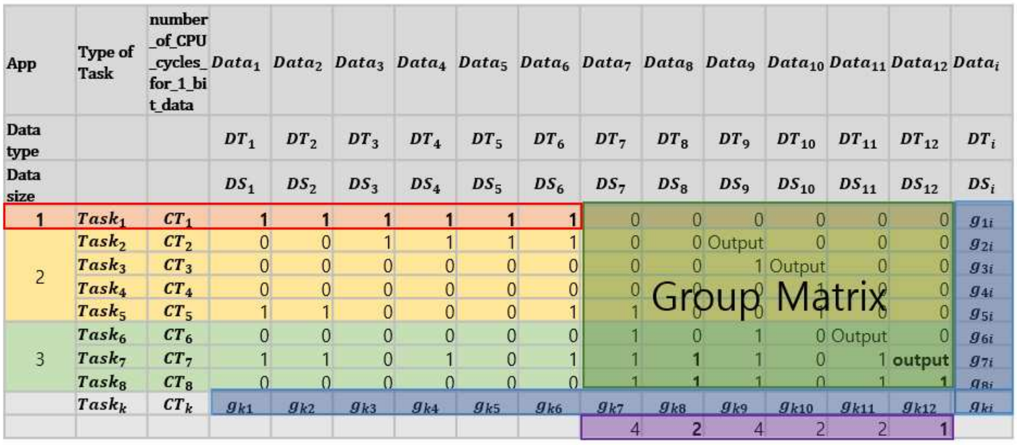
Therefore, the defined Equation needs to compromise (*CPU*, Memory) in the range of data and task. It aims to extract the range of minimizing consumption energy and final time for offloading.

3. LODO (Linked-Object Dynamic Offloading) Algorithm

The LODO algorithm provides two cooperation offloading options depending on the cause of the overload. In Section 2.1, the cause of overload according to the state of an overloaded edge node through monitoring was definition. When a memory overload problem occurs, executing the algorithm using the data-linked offload method described in Section 3.1. If the cause of the overload is in the *CPU*, execute the task-linked offloading option described in Section 3.2.

3.1. Data-Linked Algorithm

The Data-linked offloading (ref. Algorithm 1), which concerns data correlations, improves the energy efficiency by choosing the data redundancy based on memory, the range in minimizing energy during transmitting, and the collaboration node. Create a collaboration data group based on the currently executed data. Figure 3 explains how to create a group when performing offloading based on a scenario that allows Task1 to be performing at the existing edge. Derivation of the task-based offloading range is the least frequent of the remaining data, excluding the data used in Task1. It is possible to set the task priority according to the existing domain and create a list expression task offloading according to the criteria for the task and the data that is reducing due to offloading. For example, offloading the least frequent Task8 reduces the number of data12 on existing edge nodes. Offloading with Task7 to free up additional memory space will free up the amount of data8. A list of expression tasks is an input to the algorithm's data dependency groups. The output categorizes the minor offloading group, the collaboration node, and the total consumed memory.



App	Type of Task	number_of_CPU_cycles_for_1_bit_data	$Data_1$	$Data_2$	$Data_3$	$Data_4$	$Data_5$	$Data_6$	$Data_7$	$Data_8$	$Data_9$	$Data_{10}$	$Data_{11}$	$Data_{12}$	$Data_i$
Data type			DT_1	DT_2	DT_3	DT_4	DT_5	DT_6	DT_7	DT_8	DT_9	DT_{10}	DT_{11}	DT_{12}	DT_i
Data size			DS_1	DS_2	DS_3	DS_4	DS_5	DS_6	DS_7	DS_8	DS_9	DS_{10}	DS_{11}	DS_{12}	DS_i
1	$Task_1$	CT_1	1	1	1	1	1	1	0	0	0	0	0	0	g_{1i}
2	$Task_2$	CT_2	0	0	1	1	1	1	0	0	Output	0	0	0	g_{2i}
	$Task_3$	CT_3	0	0	0	0	0	0	0	0	1	Output	0	0	g_{3i}
	$Task_4$	CT_4	0	0	0	0	0	0	0	0	0	1	0	0	g_{4i}
	$Task_5$	CT_5	1	1	0	0	0	1	1	0	0	1	0	0	g_{5i}
3	$Task_6$	CT_6	0	0	0	0	0	0	1	0	1	0	Output	0	g_{6i}
	$Task_7$	CT_7	1	1	0	1	0	1	1	1	1	0	1	output	g_{7i}
	$Task_8$	CT_8	0	0	0	0	0	0	1	1	1	0	1	1	g_{8i}
	$Task_k$	CT_k	g_{k1}	g_{k2}	g_{k3}	g_{k4}	g_{k5}	g_{k6}	g_{k7}	g_{k8}	g_{k9}	g_{k10}	g_{k11}	g_{k12}	g_{ki}
									4	2	4	2	2	1	

Figure 3. List expression task based on data-centric offloading.

$$\text{Amount of memory in } List_j = \sum_{i=1}^{list[2]} g_{ki} * DT_{ki} * DS_{ki}, \tag{4}$$

$$\text{Amount of } CPU \text{ in } List_j = \sum_{\prime i=1}^{list[2]} g_{ki} * DT_{ki} * DS_{ki} * CT_{ki}, \tag{5}$$

Each group's memory and *CPU* usage can calculate via (Equation (4)) and (Equation (5)). g_{ki} is whether the *i*-th data used in the *k*-th task is used, and DT_{ki} is the data type. DS_{ki} is the amount of sensing data that is instantaneously accumulated as a data size. CT_{ki} is a 'number of *CPU* cycles for 1-bit data' value and is the *CPU* value used in each task. The calculated Equations (4) and (5) values changes depending on the task and data included in the corresponding list. Through the calculated value, the "possible offloading list" is extracting by matching the *CPU* usage rate that is not larger than the idle space of the collaboration edge node. In other words, depending on the cause of the overload, if the memory is insufficient, the data relevance list that can secure the maximum memory is selected in the offload range. The selected data relevance list becomes the "data dependency group," which is the input to the algorithm. The list of feasible groups is sort by examining their executions in collaboration nodes, and the offloading is processed in each node sequentially. If slack energy in the collaboration node is not enough, it could skip out to the next node for offloading. The Data-centric Offloading stops in the case of solving the overload issue. Otherwise, it keeps progressing to offloading.

Algorithm 1 Data-linked offloading

```
Input: Next Execution, Data Dependency Group Output: Offloading Group,
Collaboration Edge Node, TotalMemory
Step 1: Overload detection and determination of cause.
IF Status of EN = Out of memory problem ## result of activation state
    NextExecution = Data-linkedOffloading
    OffloadingApply = True
ELSE
    OffloadingApply = False
Step 2) Calculate group capabilities based on data
FOR I = 1 to length(data dependency group)
        $g_j$ = data dependency group
        GroupList(Data_list, Total_memory) = $g_j$
END FOR
Step 3) Perform offloading after extracting possible groups based on collaboration nodes
WHILE OffloadingApply
        FOR $j$ = 1 to $k$
        IF g_j(Total_memory) $\leq$ Collaboration_EN(Slack Memory) $-$ (100-threshold)
        possiblelist = $g_j$(Data_list, Total_memory
END FOR
DO max(possiblelist) offloading to Colaboration_EN
        IF EN(slack_memory) $\leq$ threshold
        OffloadingApply = False
RETURN {possible list, collaboration edge node, Total_memory, Total_energy}
```

3.2. Task-Linked Algorithm

The task-linked algorithm, which concerns task correlations, reduces the overall time by selecting the most considerable *CPU*, the range in increasing energy efficiency during processing, and the collaboration node (ref. Algorithm 2). Create collaboration groups based on tasks to performing and data to be using. Figure 4 shows a subordinate system consisting of 8 tasks. In this way, the application is complexly configuring. Some tasks use the output value of each task as an input value. A dependency system is a workflow made up of dependent tasks that interact between modules. Task2 creates data9, which is the input value of Task3, Task6, Task7, and Task8. Based on task connectivity, Task3, Task6, Task7, and Task8 cannot start execution until Task2 is executing and the output is displayed.

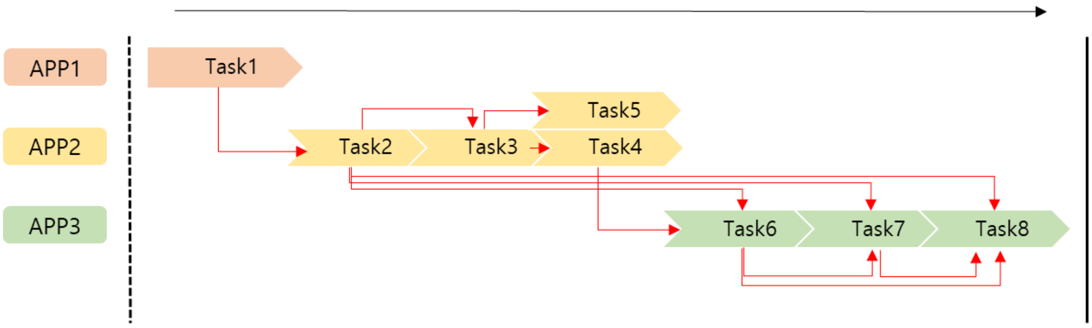


Figure 4. Workflow and Task Connectivity.

When performing tasks, an offloading group is creating in consideration of task relevance. Groups that consider task relevance are composed of one or more tasks, as shown in Figure 5.

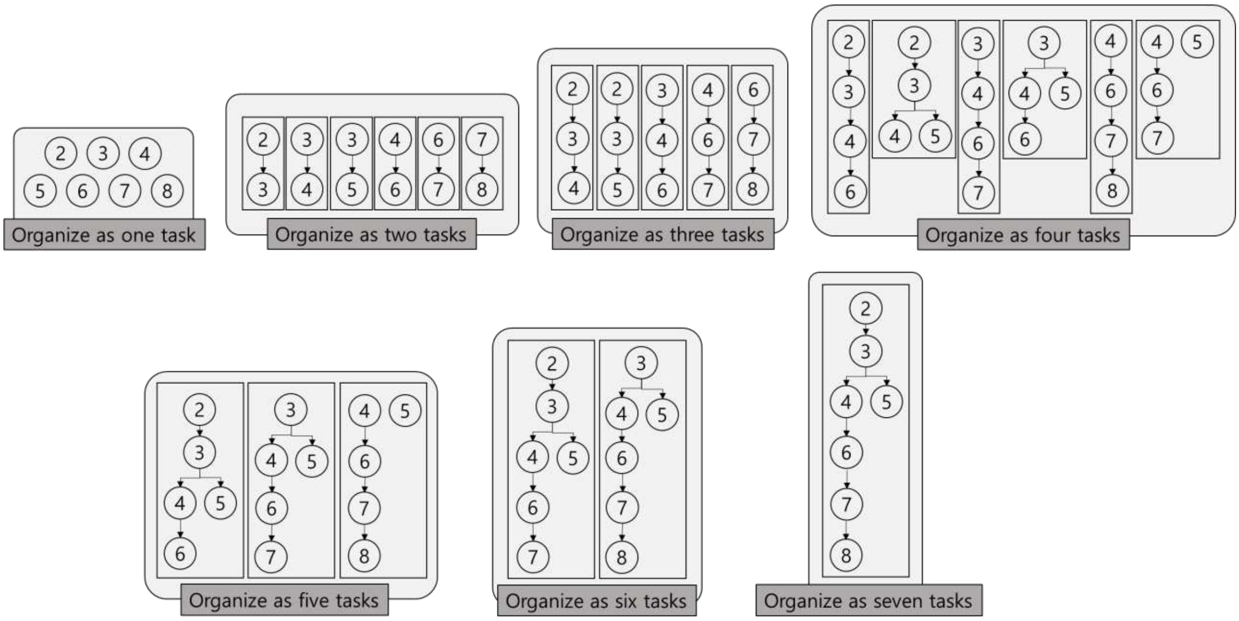


Figure 5. Offloading group of Task relevance.

To select an offloading range is necessary to calculate the mobility of data generated during offloading. Figure 6 illustrates the importance of task connectivity group-based offloading through task mobility. In Case 1, Task3 and Task4 are offloading, and in Case2, Task3 and Task6 are offloading. Task3 and Task4 use the result of Task2 together, and the output of Task3 becomes the input of Task4 and proceeds sequentially. Also, since Task4's output is used by Task5 and Task6 simultaneously, the total number of movements is two times. However, the input value of Task3 and Task6 is common to Task2, but Task6 requires the output value of Task4 as an input value. Each output value is also using as an input value for other tasks. Thus, the total number of moves is four. Even if the same amount of data is using, the delay time increases as the total number of movements increases. Hence, the available memory and *CPU* are calculating through Equations (4) and (5), appropriate lists are extracting, and the group with the least mobility is select as the offloading range.

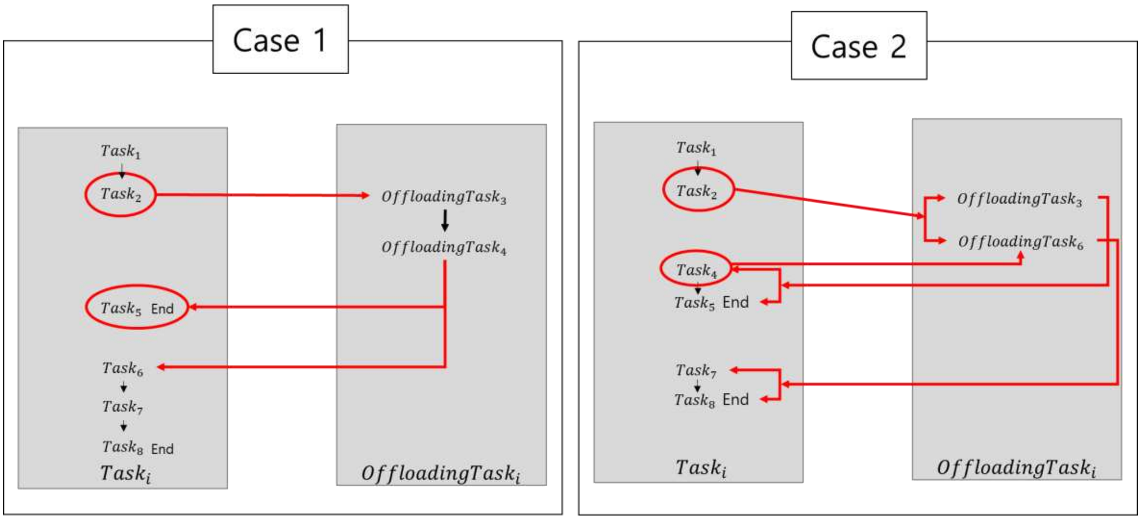


Figure 6. Task connectivity group-based offloading through task mobility.

A grouped list of task dependencies is the input to the algorithm. The task connectivity list described above becomes the "task dependency group," which is the input to the algorithm. The output classifies the least offloading group, the collaboration node, and the overall time. The list of feasible groups is sorting by examining their executions in collaboration nodes, and the offloading is processed in each node sequentially. Offloading continues until the overload problem is resolving, and if slack energy in the collaboration node is not enough, it could skip out to the next node for offloading. Total output time means the time until the end of offloading for a specific application and is calculated from the number of moves and the amount of data.

Algorithm 2 Task-Linked offloading

```
Input: NextExecution, Task Dependency Group
Output: Offloading Group, Total Time(EndPoint)
Step 1) Overload detection and determination of the cause.
IF Status of EN = Out of CPU problem
    NextExecution = Task-linkedOffloading
    OffloadingApply = True
ELSE
OffloadingApply = False
Step 2) Calculate group capabilities based on task
FOR I = 1 to length(task dependency group)
    $g_j$ = task dependency group
    GroupList(Task_list, Total_CPU) = $g_j$
END FOR
Step 3) Perform offloading after extracting possible groups based on collaboration nodes
WHILE OffloadingApply
    FOR $j$ = 1 to k
           IF g_j(Total_Time) $\leq$ Collaboration_EN(Slack space) $-$ (100-threshold)
              possiblelist = $g_j$(Task_list, Total_Time)
END FOR
DO max(possiblelist) offloading to Collaboration_EN
    IF EN(slack_space) $\leq$ threshold
    OffloadingApply = False
RETURN{possiblelist, collaboration edge node, Total_CPU, Total_time}
```

4. Scenario and Results

This section provides an experimental scenario and the LODO algorithm's performance that proposes in data-linked and task-linked algorithms. The performance of the algorithm is tested based on the variables and tasks used in the forest fires scenario and the relationship between them. Based on the algorithm experiment results, the evaluation scenario includes memory usage and execution time. Table 1 lists tasks and variables used in forest fire response scenarios. In practice, the quantity and size of a variable can be the same or different.

Table 1. Tasks for forest fire response and data types used.

Application	Task	Data	Output
APP1. Fire Probability Prediction	Task 1. Fire probability and probability prediction	temperatures (data1), humidity (data2), fuel (data3), mount.terrain (data4), weather (data5), fuel (data3), for geography (data6)	Forest fire Probability (data13)
APP2. Diffusion Range Prediction	Task 2. Diffusion rate	fuel (data3), mount.terrain (data4), weather (data5), for geography (data6)	Diffusion rate (data9)
	Task 3. Forest fire intensity	diffusion rate (data9)	Fire intensity (data10)
	Task 4. Flame height, Fire type prediction	fire intensity (data10)	Fire type (data14)
	Task 5. Fire direction, Diffusion area prediction	temperatures (data1), humidity (data2), for.geography (data6), wind speed (data7), fire intensity (data10)	End
APP3. Diffusion Location Prediction	Task 6. Flame length	fire type (data14), wind speed (data7), diffusion rate (data9)	Flame length (data11)
	Task 7. Non-combustible material lift height calculation	temperatures (data1), humidity (data2), mount.terrain (data4), for.geography (data6), wind speed (data7), wind direction (data8), diffusion rate (data9)	Flame height (data12)
	Task 8. Distance for non-combustible mater. Fireworks Landing Position Prediction	wind speed (data7), wind direction (data8), diffusion rate (data9), flame length (data11), flame height (data12)	End

Tasks are prioritized based on what is essential or should do first. Task 1 is the most basic analysis that starts in the forest fire scenario, so it should be performing on the edge node. In addition, the priority is high because it is performing in real-time. Before the data processing process and algorithm apply, the offloading range list becomes a subset of 127 for all tasks except for Task 1. As the number of tasks and variables increases, the selectable range of offloading becomes more diverse.

In case of memory overload, a list extracted based on data association is using. Table 2 is a data correlation list that derives an offloading range that can quickly acquire memory based on data.

Table 2. Data-linked list.

Task	Acquire Memory
Task8	data12
Task8, Task7	data12, data11, data8
Task8, Task7, Task5 Task6	data12, data11, data8, data7
Task8, Task7, Task3, Task6	data12, data11, data8, data9
Task8, Task7, Task4, Task5	data12, data11, data8, data10

In case of *CPU* overload, a list extracted based on task association is using. As with the data center, except for Task 1, the connectivity between the remaining tasks creates a group that reduces mobility between edges and secure *CPU* space. Table 3 is a task connectivity list that derives an offloading range that reduces mobility while quickly obtaining *CPU* space centered on tasks. Figure 7 shows the total number of moves and the task connectivity list (red dots). In the figure, the ox is each group corresponding to the task connectivity list, and oy is the number of moves that occur when offloading each group.

Table 3. Task-linked list.

	Task	Number of Moves
1	Task2	2
	Task3	2
	Task4	2
	Task5	2
	Task6	3
	Task7	3
	Task8	4
2	Task2-Task3	2
	Task3-Task4	2
	Task3-Task5	2
	Task4-Task6	3
	Task6-Task7	3
	Task7-Task8	3
3	Task2-Task3-Task4	2
	Task2-Task3-Task5	2
	Task3-Task4-Task6	2
	Task4-Task6-Task7	3
	Task6-Task7-Task8	3
4	Task2-Task3-Task4-Task6	2
	Task2-Task3-Task4-Task5	3
	Task3-Task4-Task6-Task7	2
	Task3-Task4-Task5-Task6	3
	Task4-Task6-Task7-Task8	3
	Task4-Task5-Task6-Task7	4
5	Task2-Task3-Task4-Task5-Task6	3
	Task3-Task4-Task5-Task6-Task7	3
	Task4-Task5-Task6-Task7-Task8	4
6	Task2-Task3-Task4-Task5-Task6-Task7	3
	Task3-Task4-Task5-Task6-Task7-Task8	3
7	Task2-Task3-Task4-Task5-Task6-Task7-Task8	3

Table 4 compares the range to which offloading is applying within the range of possible collaboration usage (Memory, *CPU* 70%) and the entire range based on the lists in Tables 2 and 3. "Offloading Transmission" is the memory size calculated based on the data included in the offloading group. "Offloading performance" is a value calculated based on the number of *CPU* cycles for 1-bit data, and the value varies depending on the data used in the task and the function of the task. With "Data-linked algorithm groups," the average data movement is 1,574,156 (Kb), which offloads more data than before applying the algorithm, but the average memory securing is 8.15%, which is faster than before applying the algorithm. Suppose the "task-oriented algorithm group" is used. In that case, the offloading task group sends a smaller amount than before applying the algorithm at an average of 4.19 (GHz). It is possible to secure free space faster, and the average number of moves is an average of 2.63 Cycles enable fast processing.

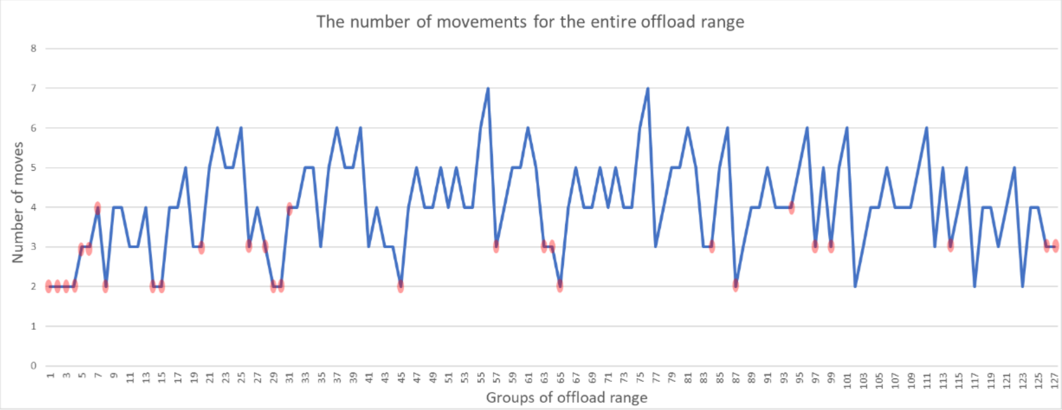


Figure 7. Graph the number of movements for the entire offload range.

Table 4. Data & Task algorithm performance evaluation.

	All Group	Data-Linked Algorithm Group	Task-Linked Algorithm Group
Offloading Transmission (Memory)	1,199,392 (Kb)	1,574,156 (Kb)	921,015.1 (Kb)
Offloading Performance(*CPU*)	5.72 (GHz)	8.67 (GHz)	4.19 (GHz)
Percentage of memory available	45,332.69 (3.78%)	212,775.5 (8.15%)	30,394.8 (3.30%)
The average number of moves	3.72	3.55	2.63

5. Discussion

Spatial dependence has two meanings. The first is that the data generated by one sensor is not used in only one task or app and is can use in multiple spaces. For example, diffusion direction data is generating by combining wind, slope, and temperature data with circumstances such as earthquakes affecting different regions at time intervals. The second is task dependencies. Task dependencies include the order of tasks as well as data correlation between tasks. In refs. [21,22], the data dependency, the output of the task located in Area A, can be entered in Area B.

They should process several tasks, such as knowing the types of forest fires, predicting how the wind changes, and identifying how ignition materials are distributing in fire areas by season, temperature, humidity. Additionally, some cases have data with two or more tasks used simultaneously or including complicatedly mixed order of tasks. For example, to determine the diffusion range with the location of forest fires. However, as the accurate diffusion range and location could identifying many tasks in various areas could be linked and applied to each other. In other words, multiple tasks do not follow in a series of workflows but are spread in many branches or entangled like spider webs.

However, most offloading does consider only a series of processing in [23] and does not include correlations and interactions of data and tasks. One or more outputs can be applying to the next task after starting four different tasks simultaneously, or new input that changes the result executed already. Because output data transferred from a server to a local device is much smaller than input data, the time overhead of a backhaul link could be ignorant [24,25]. This is considered only a series of dynamic applications, not static tasks. If one output data can be the input of many tasks, the increase of task number cause to increase both redundant data and output data; hence the overhead due to another delay and energy consumption from a transmission cannot be disregarded.

Correlations of data and tasks have to consider for dividing tasks of granularity and dependency in various circumstances. Therefore, we proposed a collaboration LODO algorithm that determines idle space and offloads data and tasks based on spatial dependencies.

6. Conclusions

This paper considers collaboration edge computing in offloading with an edge node that can collaborate with tasks. We proposed an energy-efficient LODO algorithm to extract the scope and offload of collaboration nodes to save energy and reduce execution time at the edge nodes. Formulated hybrid states in an edge node could predict overloads through monitoring and applied in the LODO algorithm. Furthermore, in selecting an offload range by considering data correlations and task connectivity, the LODO algorithm reduces data redundancy and delays and minimizes energy consumptions during offloading. Therefore, a collaboration offloading model based on the LODO algorithm minimizes the energy of the entire edge node so that it is more efficient to execute within a short time.

Author Contributions: Conceptualization, S.K., J.K. and Y.Y.; methodology, S.K.; formal analysis, S.K. and J.K.; investigation, J.K.; resources, S.K. and J.K.; data curation, J.K.; writing—original draft preparation, S.K. and J.K.; writing—review and editing, S.K. visualization, J.K.; supervision, Y.Y.; project administration, S.K.; funding acquisition, Y.Y. and S.K. This J.K. author contributed equally to this study as co-first authors. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2019R1I1A1A01064054). This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07047112).

Data Availability Statement: Data is describing within the article. The data that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: The writers would like to thank the editor and anonymous reviewers for their helpful comments for improving the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. El-Sayed, H.; Sankar, S.; Prasad, M.; Puthal, D.; Gupta, A.; Mohanty, M.; Lin, C.-T. Edge of Things: The big picture on the Integration of Edge, IoT and the cloud in a distributed computing environment. *IEEE Access* **2018**, *6*, 1706–1711. [CrossRef]
2. Li, X.; Qin, Y.; Zhou, H.; Cheng, Y.; Zhang, Z.; Ai, Z. Intelligent Rapid adaptive offloading algorithm for computational services in dynamic internet of things system. *Sensors* **2019**, *19*, 3423. [CrossRef] [PubMed]
3. Premsankar, G.; Di Francesco, M.; Taleb, T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [CrossRef]
4. Wang, J.; Pan, J.; Esposito, F.; Calyam, P.; Yang, Z.; Mohapatra, P. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Comput. Surv.* **2019**, *52*, 2. [CrossRef]
5. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
6. Cui, Y.; Ma, X.; Wang, H.; Stojmenovic, I.; Liu, J. A survey of energy efficient wireless transmission and modeling in mobile cloud computing. *Mobile Netw. Appl.* **2013**, *18*, 148–155. [CrossRef]
7. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H.P. Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
8. Kai, C.; Zhou, H.; Yi, Y.; Huang, W. Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks With Limited Communication Capability. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 624–634. [CrossRef]
9. Huang, W.; Huang, Y.; He, S.; Yang, L. Cloud and edge multicast beamforming for cache-enabled ultra-dense networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 3481–3485. [CrossRef]

10. Ren, J.; Yu, G.; Cai, Y.; He, Y. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Trans. Wireless Commun.* **2018**, *17*, 5506–5519. [CrossRef]
11. Kao, Y.; Krishnamachari, B.; Ra, M.; Bai, F. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Trans. Mobile Comput.* **2017**, *16*, 3056–3069. [CrossRef]
12. Auluck, N.; Azim, A.; Fizza, K. Improving the schedulability of real-time tasks using fog computing. *IEEE Trans. Serv. Comput.* **2019**. [CrossRef]
13. Li, H.; Ota, K.; Dong, M. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Netw.* **2018**, *32*, 96–101. [CrossRef]
14. Tran, T.X.; Pompili, D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 856–868. [CrossRef]
15. Zhao, J.; Li, Q.; Gong, Y.; Zhang, K. Computation Offloading and Resource Allocation for Cloud Assisted Mobile Edge Computing in Vehicular Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 7944–7956. [CrossRef]
16. Mao, S.; Leng, S.; Zhang, Y. Joint communication and computation resource optimization for NOMA-assisted mobile edge computing. In Proceedings of the IEEE International Conference Communication (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
17. Haber, E.; Nguyen, T.M.; Assi, C.; Ajib, W. Macro-cell assisted task offloading in MEC-based heterogeneous networks with wireless backhaul. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1754–1767. [CrossRef]
18. Hossain, M.D.; Sultana, T.; Nguyen, V.; Nguyen, T.D.; Huynh, L.N.; Huh, E.N. Fuzzy Based Collaborative Task Offloading Scheme in the Densely Deployed Small-Cell Networks with Multi-Access Edge Computing. *Appl. Sci.* **2020**, *10*, 3115. [CrossRef]
19. Ren, J.; Yu, G.; Cai, Y.; He, Y. Collaborative cloud and edge computing for latency minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
20. Kang, J.; Kim, S.; Kim, J.; Sung, N.; Yoon, Y. Dynamic Offloading Model for Distributed Collaboration in Edge Computing: A Use Case on Forest Fires Management. *Appl. Sci.* **2020**, *10*, 2334. [CrossRef]
21. Zhang, Z.; Wu, J.; Chen, L.; Jiang, G.; Lam, S.K. Collaborative task offloading with computation result reusing for mobile edge computing. *Comput. J.* **2019**, *62*, 1450–1462. [CrossRef]
22. Zhang, D.; Ma, Y.; Zheng, C.; Zhang, Y.; Hu, X.H.; Wang, D. Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 25–27 October 2018; pp. 243–259.
23. Ning, Z.; Dong, P.; Kong, X.; Xia, F. A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things. *IEEE Internet Things J.* **2018**, *6*, 4804–4814. [CrossRef]
24. Yang, L.; Dai, Z.; Li, K. An offloading strategy based on cloud and edge computing for industrial Internet. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, IEEE 17th International Conference on Smart City, IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019.
25. He, B.; Bi, S.; Xing, H.; Lin, X. Collaborative computation offloading in wireless powered mobile-edge computing systems. In Proceedings of the 2019 IEEE Globecom Workshops (GC Wkshps), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–7.

electronics

MDPI

Article

Secure Mobile Edge Server Placement Using Multi-Agent Reinforcement Learning

Mumraiz Khan Kasi [1,*], Sarah Abu Ghazalah [2], Raja Naeem Akram [3] and Damien Sauveron [4]

[1] Department of Computer Science, FICT, BUITEMS, Quetta 87300, Pakistan
[2] Information Security and Cyber Security Unit, King Khaled University, Abha 61421, Saudi Arabia; sabugazalah@kku.edu.sa
[3] Department of Computer Science, University of Aberdeen, Aberdeen AB24 3FX, UK; raja.akram@abdn.ac.uk
[4] Department of Computer Science, University of Limoges, 23204 Limoges, France; damien.sauveron@unilim.fr
* Correspondence: mumraiz.kasi@buitms.edu.pk

Abstract: Mobile edge computing is capable of providing high data processing capabilities while ensuring low latency constraints of low power wireless networks, such as the industrial internet of things. However, optimally placing edge servers (providing storage and computation services to user equipment) is still a challenge. To optimally place mobile edge servers in a wireless network, such that network latency is minimized and load balancing is performed on edge servers, we propose a multi-agent reinforcement learning (RL) solution to solve a formulated mobile edge server placement problem. The RL agents are designed to learn the dynamics of the environment and adapt a joint action policy resulting in the minimization of network latency and balancing the load on edge servers. To ensure that the action policy adapted by RL agents maximized the overall network performance indicators, we propose the sharing of information, such as the latency experienced from each server and the load of each server to other RL agents in the network. Experiment results are obtained to analyze the effectiveness of the proposed solution. Although the sharing of information makes the proposed solution obtain a network-wide maximation of overall network performance at the same time it makes it susceptible to different kinds of security attacks. To further investigate the security issues arising from the proposed solution, we provide a detailed analysis of the types of security attacks possible and their countermeasures.

Keywords: mobile edge computing; mobile edge server placement; multiagent RL; edge security

Citation: Kasi, M.K.; Abu Ghazalah, S.; Akram, R.N.; Sauveron, D. Secure Mobile Edge Server Placement Using Multi-Agent Reinforcement Learning. *Electronics* **2021**, *10*, 2098. https://doi.org/10.3390/electronics10172098

Academic Editor: Kevin Lee

Received: 19 June 2021
Accepted: 24 August 2021
Published: 30 August 2021

1. Introduction

Widespread deployments of robotics, assembly and production, automation, machine intelligence, and virtual reality applications requires high performance computing resources available close to the point-of-service [1]. Integration of smart services, such as predictive analysis, and delay-intolerant applications, such as healthcare applications, in current cellular architecture with limited battery lifetimes and processing power of edge (mobile and IoT) devices have called for the re-imagination of cloud computing architecture.

The traditional cloud-centric architecture provides flexibility and significant computation power. However, the communication and delay sensitive requirements of the IoT environments place constraints on the centralised cloud—making them less preferable for a robust service platform. To circumvent delay in the traditional cloud-centric architecture, several network architectures have been proposed with the idea of bringing the cloud nearer to user devices [2]. One such architecture is edge computing that provides a virtualized application layer between edge devices and cloud engine in an existing network infrastructure. Edge computing introduces distributed control systems replacing the single remote centralized control-centre or cloud allowing the processing of data near the edge of the network with enhanced scalability.

Cloudlets, multi-access or mobile edge computing (MEC), and fog computing are some of the well-known edge computing architectures. In this work, we utilize the MEC architecture that uses the existing network architecture such as cellular base station or Wi-Fi access point to provide computational resources and data storage at the edge network [3]. In MEC, the edge network serves as mid-tier between edge (mobile and IoT) devices and cloud increasing the network's capability to provide high throughput and offer low latency to edge devices. However, the costly hardware components and limited budget of network operators present some practical complications in implementing mobile edge computing solutions. Due to these constraints, only a limited number of mobile edge servers can be located in networks. This further makes the placement of a limited number of mobile edge servers a challenging problem given the performance requirements of a wireless network. Additionally, finding the optimal placement of mobile edge servers given a large set of possible placement options further increases the complexity of finding optimal placement strategy for mobile edge servers.

The large solution space of mobile edge server placement options can be improved by collocating mobile edge servers with existing cellular network base stations or wireless network cluster heads [4]. In this work, we follow a similar strategy where mobile edge servers are placed within an already existing cellular or wireless network infrastructure disbarring the search space for optimal placement strategy from exploding. Further, the optimal placement strategy should take into consideration the individual mobile edge server's workload, access delay and application specific requirements into consideration. Various solutions have been proposed in literature to solve the edge servers placement problem [5–11]. However, most of these solutions apply heuristic algorithms or some sort of linear or quadratic optimization technique. This has motivated us to propose an online learning based solution for the mobile edge server placement problem with special emphasis on possible security threats and its solutions.

In our proposed solution, an RL agent is employed as mobile edge server that learns the dynamics of the environment and chooses the best placement strategy maximizing the reward which is dependent on the utility function. The number of RL agents is equal to the number of edge servers in the network. This demands for information exchange between the RL agents to maximize network-wide utility. In the proposed work, we implement hysteretic Q-learning for coordination which is a decentralized multi-agent RL technique allowing the agents to adopt independent actions by maximizing a common goal. Further, we analyze the security threats and countermeasures that may arise due to the information sharing between the edge servers acting as RL agents.

The primary contributions of this work are:

- The mobile edge server placement is formulated as multi-objective optimization problem which is then solved using a multi-agent RL approach. The objectives of the proposed solution include reducing the access delay and balancing edge servers workload. Further, experimental results are obtained by applying the proposed solution on base station dataset provided by Shanghai Telecom to analyze the performance of the proposed technique;
- We discuss different scenarios in which the proposed architecture's security can be breached if the exchanged data between RL agents is altered. Further, we discuss the counter strategies to tackle with the arising security issues.

The rest of the paper is organized as follows: In Section 2, we discuss the existing theories on the placement of edge servers. We discuss the preliminaries of RL in Section 3. In Section 4, we provide an overview of our proposed solution using RL. In Section 5, we describe network and RL modeling and its implementation. Section 6 explains our findings for various system configurations and parameters. Section 7 provides details on the identification of the security issues involved in the exchange of information between edge servers. Finally, Section 8 concludes our work and suggests future research directions.

2. Literature Review

The literature review related to mobile edge computing can be divided into two parts. The first part inherently deals with efficient resource allocation in a MEC network. Although the second part, which has been trending recently, tackles the mobile edge server placement problem in a MEC network. It is important to note that the solutions provided for resource allocation problems consider an arbitrary placement of edge servers. Although the second part deals with deployment strategies for edge servers such that desired quality of service is met. To that end, this literature review discusses pioneer works in both of these parts followed by the contributions of this work.

In literature belonging to efficient resource allocation, the authors have proposed an online learning based solution for the resource allocation, scheduling or offloading problems in MEC networks [12–15].

The proposed solutions have studied MEC problems in different aspects ranging from computation offloading schemes in MEC to mobile edge application placement. However, they have not explicitly discussed the mobile edge server's positioning as a challenging problem. For example, the authors in [13], have proposed a deep RL solution to allocate computing and network resources adaptively in MEC to reduce the average service time and balance resource usage under varying MEC environment conditions. In [12], the authors propose an RL based management framework to manage the resources at the network edge. They propose a deep Q-learning based algorithm to reduce service migration in MEC aiming at operation cost reduction. In both these proposed approaches and others [14,15], RL has been used to propose a solution to resource allocation challenges in MEC architecture.

On the other hand, the literature belonging to efficient deployment strategies of edge servers includes solving edge server placement problem using genetic, simulated annealing, and hill-climbing algorithms [5,6], k-means clustering with quadratic programming [8], queuing theory and vector quantization technique [16], graph theory [12], cost constrained multi-objective problem [10], and integer programming [11] to optimally place mobile edge servers in a wireless network.

In [5], the authors formulate the problem as a constrained multi-objective problem to balance workloads of mobile edge servers and reduce network access delay. To find the optimal solution, the authors utilize genetic, simulated annealing, and hill-climbing algorithms to show the effectiveness of the proposed solution. The authors in [6] use data mining techniques, such as a non-dominated sorting genetic algorithm to ensure reliability and low latency in social media services using mobile edge computing.

In [7], the authors present an edge provisioning algorithm that can find the ideal edge locations and map them to their physical locations in a MEC network. In [8], the authors make use of k-means algorithms to solve edge placement problems with mixed-integer quadratic programming. The authors in [9] propose a queuing network based solution to find the best position for cloudlets in a cloud computing network. In [10], the authors have proposed an integer programming solution to find the optimal strategy to place mobile edge servers in smart cities.

The authors in [16] propose an optimal edge server deployment strategy using queuing theory and vector quantization technique, with the aim to minimize the service providers cost and service completion time. The authors in [17] uses graph theory to minimize access delay and the number of edge servers in a MEC network. In [18], the authors develop a two-stage solution to optimally place heterogeneous edge servers in MEC using game theory concepts to optimize service response time.

Although the edge server deployment problem has recently received traction from both academia and industry, the proposed solutions assume global information available at a centralized controller which is responsible for the deployment of edge servers. However, in a realistic environment the changing network condition, user mobility, and traffic patterns will make such centralized solution not scalable due to the large amount of processing required at the centralized controller at each transmission time interval. Considering these reported lacking, the proposed solution provides a distributed and learning based solution to mobile edge server placement problem while considering both delay minimization and edge servers load balancing. To the best of our knowledge, the problem of mobile edge server placement has not been solved through RL. This work serves as a primer on distributed placement strategy in MEC in which each edge server on a local set of observations finds its' optimal placement by exchanging limited set of information with other edge servers in the network. The information exchange between the edge servers is an essential part of a distributed edge server placement solution. To that end, this paper investigates the security concerns in implementing a multi-agent RL solution for mobile edge server placement problems and its possible counter-measures.

3. Reinforcement Learning

This section briefly discusses the concepts of reinforcement learning (RL) and its extension to a multi-agent RL.

3.1. One-Agent RL

RL algorithms are built on Markov decision processes (MDP) that allow the agent to receive a reinforcement signal from the environment steering it towards an optimal action policy. MDP is defined as $\langle O, A, P, \rho \rangle$, where O is the set of observations or states perceived from the environment, A is the discrete or finite set of actions, $P : O \times A \times p \rightarrow [0, 1]$ is the probabilistic state transition function, and $\rho : O \times A \times O \rightarrow R$ is the reward function.

At any time step i, the action $a_i \in A$ influences the environment state to change from o_i to o_{i+1} with a transition probability of $P(o_i, a_i, o_{i+1})$. In return of implemented action, the agent receives a scalar reward $r_{i+1} \in R$ according to the expression $\rho : r_{i+1} = \rho(o_i, a_i, o_{i+1})$. The overarching target of an RL agent is to adapt an action policy that maximizes the discounted future expected reward which is given as:

$$Q^{\pi}(o, a) = \mathbb{E}[R_i | o_i = o, a_i = a, \pi], \tag{1}$$

where $R_i = \sum_{j=0}^{\infty} \gamma^j r_{i+j+1}$ is the reward signal, $\gamma \in [0, 1]$ is the discount factor and $Q^{\pi} : O \times A \rightarrow R$ is the Q-function representing the discounted future expected return for a state-action pair.

Mathematically, the maximum value for the discounted expected return is characterized as $Q^*(o, a) = \max_{\pi} Q^{\pi}(o, a)$, which can be learned and estimated using Q-learning in the absence of probabilistic state transition and reward functions [19]. It is theoretically proven that the Q-learning algorithm converges to the optimal solution under certain conditions [19]. The Q-learning algorithms allow a RL agent to iteratively learn the estimates Q^* based on its interactions with the environment using the formula:

$$Q_{i+1}(o_i, a_i) = (1 - \alpha) Q_i(o_i, a_i) + \alpha(r_{i+1} + \gamma \max_{a_{i+1}} Q_i(o_{i+1}, a_{i+1})), \tag{2}$$

where $\alpha \in [0, 1]$ defines the learning rate of the Q-learning algorithm.

3.2. Multi-Agent RL

In a multi-agent RL problem, multiple agents using RL algorithms interact or compete to maximize a well-defined goal. The complexity of multi-agent RL is comparatively more than a one-agent RL solution since the use of multiple RL agents allow the environment to be jointly influenced by the actions of all agents which leads to non-dominance of a specific action policy. Optimal behavior of a multi-agent RL is reached when each agent operates in the Nash equilibrium which is difficult to visualize in a practical applications [20]. In the Nash equilibrium, each RL agent assumes the unvarying behavior from other agents and maximizes its own reward. Due to the complexities involved in implementing Nash equilibrium in practical applications, we discuss a couple of algorithms that can deal with multi-agent RL problems.

3.2.1. Independent Agents

In this method, the RL agent follows a coordination-free strategy with other agents by assuming each agent's independence. This is equivalent to implementing one-agent RL for each agent in the problem without initiating any coordination. The formulation of multi-agent RL problem as independent agents will simplify the solution but it will also make the convergence difficult due to the non-stationarity of independent agents [21].

3.2.2. Indirect-Coordinating Agents

In this method, the RL agent follows a coordination strategy with other agents. The action selection strategy comprises of the joint action of all agents which is made on the reward feedback received from the environment. Although, the learning is still independent but a common objective exists between the RL agents which it tries to maximize. We discuss one such method of indirect-coordination multi-agent RL method which is called hysteretic RL. In hysteretic RL, the agents take actions independently but the reward function is shared between all agents given as [22]:

$$\delta \leftarrow r - Q_k(o, a_k) \tag{3}$$

$$Q_k(o, a_k) \leftarrow \begin{cases} Q_k(o, a_k) + \mu\delta, & \text{if } \delta \geq 0 \\ Q_k(o, a_k) + \sigma\delta, & \text{else} \end{cases} \tag{4}$$

where learning rates μ and σ are between 0 and 1, r is the reward based on the feedback returned by the environment and $Q_k(o, a_k)$ is the Q-value of kth agent. The core idea behind hysteretic RL is to penalize agents for taking a bad action.

4. Multi-Objective Problem Formulation

A mobile edge server positioning problem can be written as an undirected graph, such that the location of base stations in existing cellular architecture makes the vertices of the graph and the base station's distance to mobile edge servers is represented as edge weights. A finite set of mobile edge servers S can be collocated with a set of base station B, such that the number of mobile edge servers will always be less than the number of base stations, as shown in Figure 1. As discussed in Section 1, the constraint of collocating mobile edge servers with the existing base station's location is to reduce the virtually infinite solution space of optimal mobile edge server positioning. Assuming a straightforward communication channel between base stations and mobile edge servers, the access delay at edge devices can be defined as the Euclidean distance (d_b) between a mobile edge server and base station where $b \in |B|$. The workload of a base station (t_b) is defined as the processing of incoming call and flow requests from edge devices.

The desired key performance indicators in MEC are reduced network access delay and balanced load on mobile edge servers which is why the mobile edge server positioning problem in this work is devised to improve these key performance indicators while finding an optimal placement of S mobile edge servers. The goals of the formulated problem are to (i) reduce the access delay or latency between mobile edge servers and base stations, and (ii) balance the workload of mobile edge servers. The key assumptions in formulating the mobile edge server positioning problem considered in this work are:

- A mobile edge server can offload processing and storage requests from more than one base station;
- A base station can offload processing and storage requests to one or more mobile edge servers. If a base station is offloading processing and storage requests to more than one mobile edge server then the workload of incoming mobile call and flow requests at a base station from edge devices will be shared among connected mobile edge servers;
- A mobile edge server is hosted and collocated at a location where a base station in an already existing network infrastructure is present.

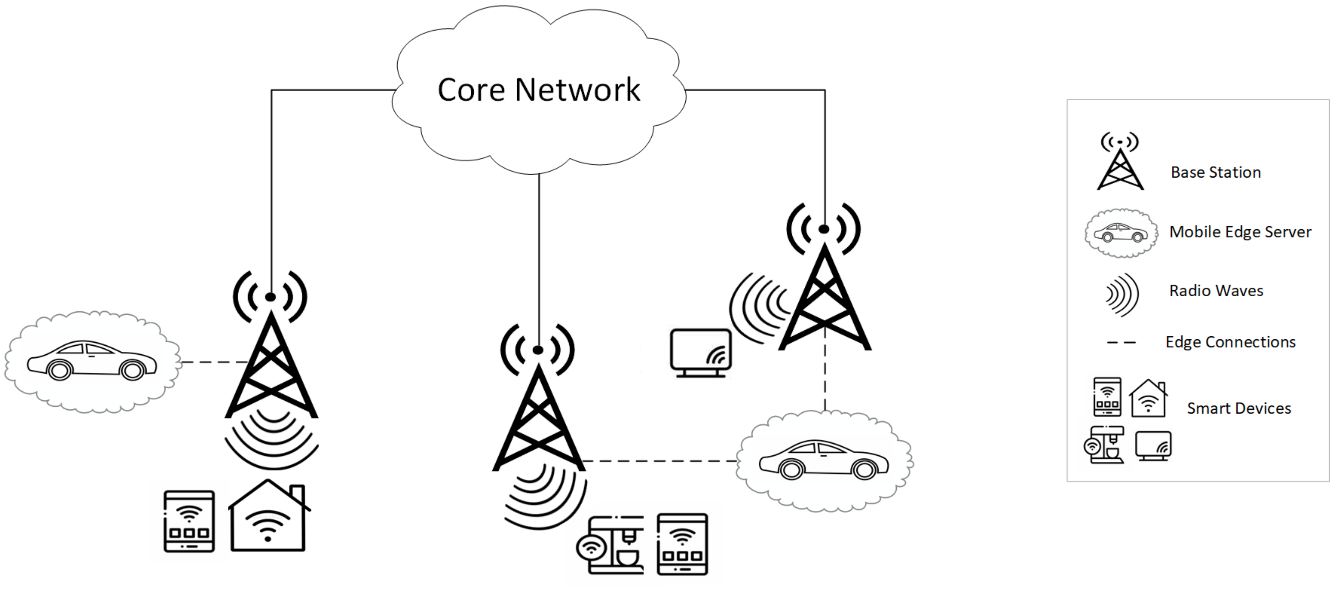


Figure 1. Edge servers placement in mobile edge computing.

The workload of a sth ($s \in |S|$) mobile edge server ($T_s(\ell)$) is made dependent on the processing and storage requests offloaded from connected base stations, such that $T_s(\ell) = \sum_{b \in |B|} t_b$, ℓ is the positioning arrangement of mobile edge servers, t_b is the incoming call and data requests from edge devices to bth base station. It is important to note that in MEC, the base station acts as a relay node transferring the incoming call and data requests from edge devices to mobile edge servers. Similarly, access delay is devised as the sum of Euclidean distances from a sth mobile edge server to one or more base stations which are offloading incoming requests processing and storage to sth mobile edge server, such that, $D(\ell) = \sum_{b \in |B|} \mathbf{d_b}$. Balancing the workload of mobile edge servers ensures that no edge servers is overloaded with offloading requests while some mobile edge server's processing capacity is underutilized. Mathematically, the standard deviation of each mobile edge server's workload is used to devise workload balancing metric ($W(\ell)$) in a MEC, such that,

$$W(\ell) = std(T_j, T_k) \quad \forall j, k \in |S|. \tag{5}$$

Finally, the cost function of multi-objective constrained optimization problem can be defined as:

$$C(\ell) = \beta W^{'}(\ell) + (1 - \beta) D^{'}(\ell), \tag{6}$$

where superscript $z^{'}$ denotes a normalized value of variable z, and $\beta \in [0, 1]$ is the weightage parameter.

Therefore, the formulated mobile edge server positioning problem can be defined as:

1. Find mobile edge server positions such that network access delay is minimized; and,
2. Find the edge connections ($x_{bs}, \ \forall \, b \in |B|, s \in |S|$) for which the mobile edge server's workload is balanced where x_{bs} is an indicator function whose value is 1 if a base station is connected to sth mobile edge server otherwise 0 if its not connected to sth mobile edge server.

Mathematically,

$$\min_{\ell \, \in |B|} \quad C(\ell)$$

such that,

$$\sum_{s=1}^{|S|} x_{bs} \leq |S| \tag{7}$$

where constraint (7) ensures that a base station offloads processing/storage requests to no more than S mobile edge servers. The above formulation of mobile edge servers positioning is a mixed-integer linear program problem that is NP-hard in nature due to the non-linearity of constraint given in (7) [10]. Therefore, this work proposes to solve the mobile edge servers positioning problem using multi-agent RL technique.

5. Proposed Solution

Mobile edge computing architecture is beneficial in providing services to a densely deployed network with low-latency and high-throughput requirements [3]. However, there are certain limitations attached to the MEC architecture. First, as explained above, the cost of infrastructure deployment and maintenance is high, therefore, dense deployment of edge servers is not a cost-effective solution. Second, the service requirement of users changes with respect to time, therefore, a certain strategy of mobile edge server's deployment may be optimal for a specific time while it would be sub-optimal for other times. The varying requirements of mobile users due to mobility require that the proposed solution should be able to adapt to the changing scenarios.

One option could be to manually configure the network at different times of the day to make sure that the edge servers deployment is optimal, however, the associated operator expenditure costs may not be feasible for an operator. To circumvent that, an online learning paradigm, such as RL can be effective in dealing with the changing environment conditions. In RL, the environment is modeled as MDP which allows a RL agent to learn the optimal action policy by interacting with the environment. In our proposed approach, each mobile edge server will be working as a RL agent and the environment will be modeled as the mobile edge computing network with base stations, and user devices. Each RL agent will be taking actions independently based on the perceived notion of state from observations and measurements of the environment, however, reward will be computed based on the network-wide delay and workload observed which would require information exchange, as shown in Figure 2. The network-wide utility is defined as the average communication delay and edge server's workload for all edge servers in the network. The objective of the proposed work is to find a mobile edge servers positioning strategy, such that it caters to the needs of data rate requirements of users, as well as it should be able to minimize the delay and maintain workload balancing between edge servers. In this section, we discuss the methodologies adopted for environment and RL agent design.

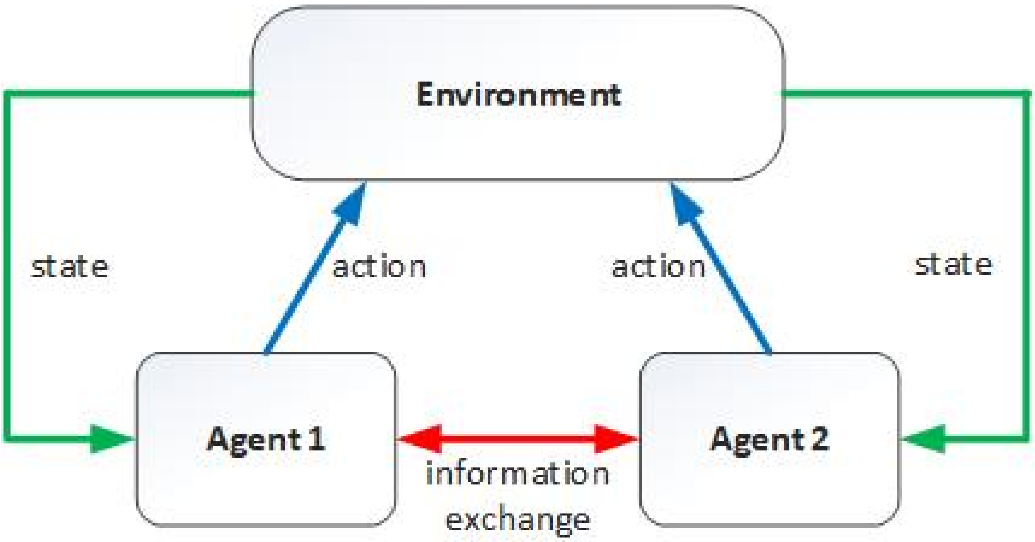


Figure 2. Multi-agent RL assisted mobile edge computing.

5.1. Environment Design

To make the proposed environment design realistic, we make use of base station locations and call and data requests dataset from Shanghai Telecom that include record of approximately 7 million call and data requests made through 2766 base stations from 9481 edge devices [8,23,24]. Each call and data record is a tuple of request access time by an device from a base station. Shanghai being a heavily populated city makes it a suitable dataset to implement a mobile edge server placement solution in an ultra-dense MEC network. Figure 3 shows the base station distribution in Shanghai, China where each dot is a location of base station and the color of a dot represent the intensity of incoming call and data requests from edge devices.

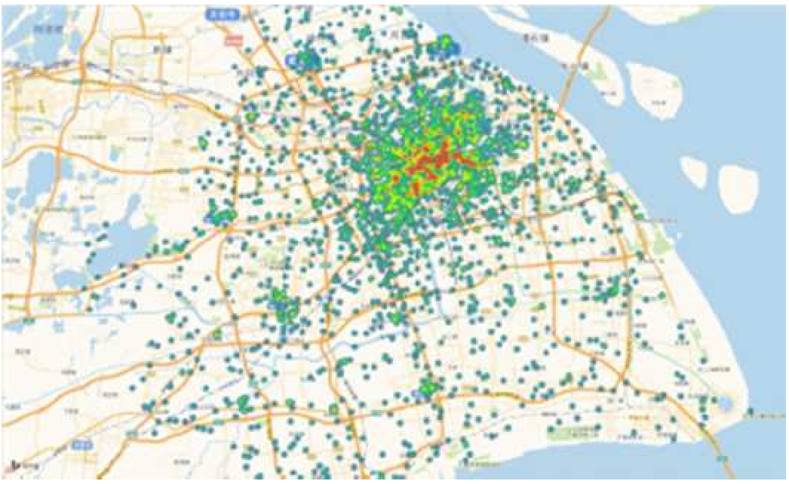


Figure 3. Graphical depiction of base station locations in Shanghai Telecom dataset [8,23,24].

The graphical depiction of base station locations is important to realize that a mobile edge server placement solution would require the edge servers to move to any of other base station locations. This means if there are 2766 base stations in the network then a mobile edge server can move to any of these locations. However, there are two problems associated with this assumption. First, the number of locations a mobile edge server can move at each transmission time interval would be dependent on the number of base stations in the network which would make it not scalable if the number of base stations is too high. Second, in a realistic world a mobile edge server would be deployed in a movable object, such as a vehicle, such that limiting the movement of edge servers to only nearby base station locations.

Considering the above two problems discussed, we transform the distribution of base stations given in Shaghai Telecom dataset to a contour line. A contour line links the base station locations with a line joining the two nearest base stations. This transformation of actual locations of base stations to contour line has following benefits:

- Base stations nearest to each other connected with a line allowing the mobile edge servers to move between nearest base station locations in the search of optimal placement strategy;
- The search space of mobile edge server placement becomes scalable. Even if the number of base stations are increased in the MEC network, the solution space will not explode.

In Figure 4a, we show the simulation depiction of base station locations available in Shanghai Telecom dataset. Note that the dataset assumes base station locations in two-dimensional space. Figure 4b shows the contour representation of base station locations, such that each base station is represented a point on a two-dimensional space which is connected to two nearest base stations. The contour line enables the transformation of actual dataset values to so that a mobile edge server can only move between two adjacent locations.

To quantify workload of a mobile edge server and delay experienced by a user equipment, we make use of records available in Shanghai Telecom dataset. The workload of a mobile edge server is quantified by summing up the requested call and data rates from the connected base stations. As a base station offloads its requested computational processing to the connected mobile edge server, therefore, summing up these requested rate is a reasonable assumption [10]. The delay experiences by a user will be proportional to the distance between base station and mobile edge server locations assuming that user equipments are present in close vicinity to base stations [10]. Therefore, the edge device access delay is defined in term of the sum of Euclidean distances from a base station to mobile edge servers to which incoming call and data requests are offloaded for processing or storage.

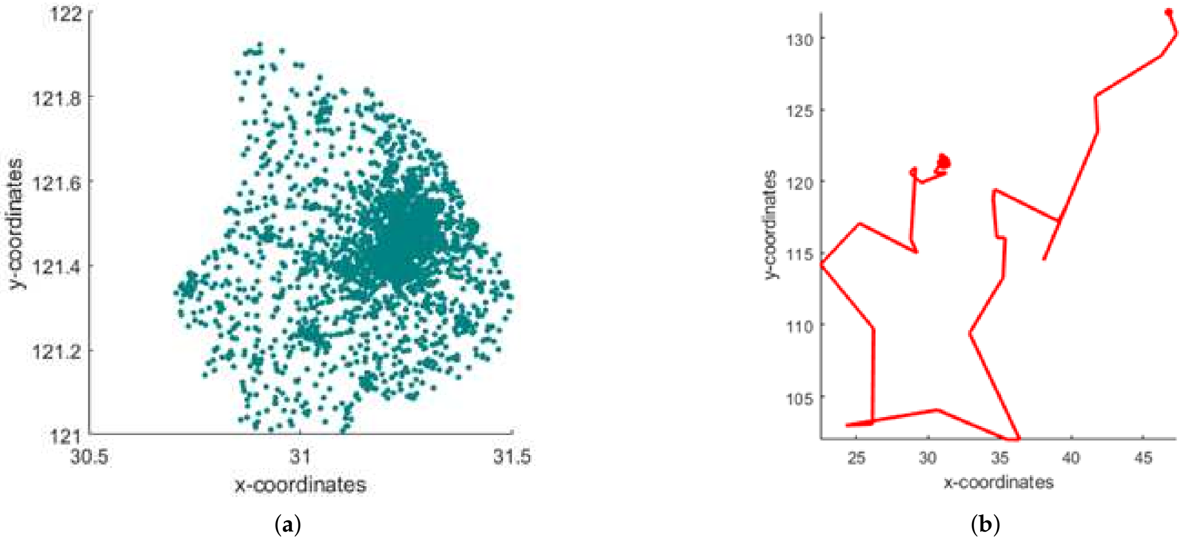


Figure 4. Transformation of base station locations in Shanghai Telecom dataset to contour line. (**a**) Simulated depiction of base station locations. (**b**) Contour line joining nearest base stations.

5.2. RL Agent Design

In this part, we aim to solve the optimization problem formulated in Section 4 for each mobile edge server using RL (see Algorithm 1). The proposed approach considers a scenario where each mobile edge server is placed on a movable vehicle that has the ability to move within the network, as shown in Figure 1. The movement of a moving vehicle is controlled by the actions of a RL agent that aims to learn the optimal placement strategy by maximizing the reward. There are three main components involved in the design of RL agent: action space, state space, and reward.

5.2.1. Action Space

Action space in the proposed work is a set of actions by which the mobile edge server change its locations. These actions are updated at the end of an epoch which is dependent on the change in network traffic. In the proposed work, actions are formed to move the mobile edge server between adjacent locations. Since we have formed a contour line from actual base station locations restricting a mobile edge server to move to only two possible neighbor locations. The action space is comprised of a set of three distinct actions by which a mobile edge server can either move to adjacent location on the right or it can move to adjacent location on the left or it stays at the same location. These set of actions will be available for each mobile edge server with the assumption that multiple mobile edge servers can be positioned in the same mobile edge server location.

Algorithm 1: Multi-agent RL assisted mobile edge computing.

```
Input: $\alpha, \gamma, \mu, \sigma, \epsilon$
$A = \{$left,right,no change$\}$.
$O = \{D(\ell)\}$.
Initialize $|S|$ Q-tables with random values and set $\epsilon = 1$.
Initialize the locations of edge servers randomly.
while converged or aborted do
    For each edge server observe the state (communication delay) of the environment $o$.
    For each server choose one of the action from its action space according to $arg\ \max_a Q(o, a)$ or randomly with probability $\epsilon$.
    calculate network-wide utility according to Equation (8).
    update Q-tables according to Equation (4).
    linearly decrease $\alpha$, $\mu$, $\sigma$, and $\epsilon$.
end
```

5.2.2. State Space

A state in the proposed work is defined by the communication delay between a mobile edge server and base stations that are offloading call and data requests to a mobile edge server. The communication delay as discussed in Section 4 is proportional to the Euclidean distance between a mobile edge server and connected base stations. Delay, as a stand-alone, will be used to infer the state of the environment. Note that other network features, such as location information, data request rate, etc., can also be used to infer on the state of the environment, however, we have made use of a simplified state space model to (i) show the efficacy of proposed solution and (ii) focus on the security aspects that may arise due to the proposed solution.

Even with a simplified state space containing only delay metric as state variable, the number of possible state values can be infinite. For this reason, we quantize the values of delay between maximum and minimum delay which will vary for different MEC networks.

5.2.3. Reward

A RL agent learns from the feedback returned by the environment in the form of rewards. In this problem, a reward is a function of cost values, such that:

$$R_t = C(\ell)_{t-1} - C(\ell)_t \tag{8}$$

The expression in Equation (8) drives the RL agent to take actions, such that the cost is minimized from the previous epoch t. Note that the cost function is dependent on the global observations. For example, a mobile edge server must be aware of the workload and delay of other edge servers in order to compute the cost function. This transforms the problem into a coordinated multi-agent RL problem in which the reward function is a function of network-wide metrics. This makes it equivalent to hysteretic RL algorithm discussed in Section 3 which is used by each mobile edge server to implement RL in this work.

The state and action spaces are still dependent on the local observations and an agent take actions independent. The sharing of information between edge servers controls the behavior of mobile edge server's placement which, if changed for some reason, would affect the performance of overall implementation. We discuss further on the type of security breaches and its counter solutions in Section 7.

6. Results

In this section, we evaluate the performance of multi-agent RL algorithms for mobile edge server positioning problem by experimenting on the Shanghai Telecom's base stations and incoming call and data requests dataset [8,23,24]. The proposed solution is implemented in MATLAB. Multiple RL agents take actions independently to find the best placement strategy, such that the reward returned by the environment is maximized. The proposed solution performance is measured via the cost function value which drives the reward function value. The list of simulation and RL hyperparameters used during the experiments shown in this work are summarized in Table 1. It is important to note that the optimal number of edge servers in a network depends on a number of factors including the network operators budget, and user traffic demand. The proposed model selects an arbitrary number of edge servers (that should be less than the number of base stations), and finds optimal placement for these edge servers, However, the number of edge servers can be found by probability theory and control system rules [25].

Table 1. Simulation parameters.

Parameter	Description	Value
BS	number of BSs sampled from the Shanghai Telecom's dataset	120, 240, 360
ES	number of mobile edge servers controlled by RL agents	20, 30, 40
α	learning rate	0.35
ϕ	learning rate for hysteretic	0.30
β	weightage of delay over workload in utility function	0.5
γ	discount factor	0.9
ϵ	random exploration	0.15

The experimentation presented in this work aim to answer the following questions:

A. Does the proposed solution generalize across different random initialized values used in the experimentation?

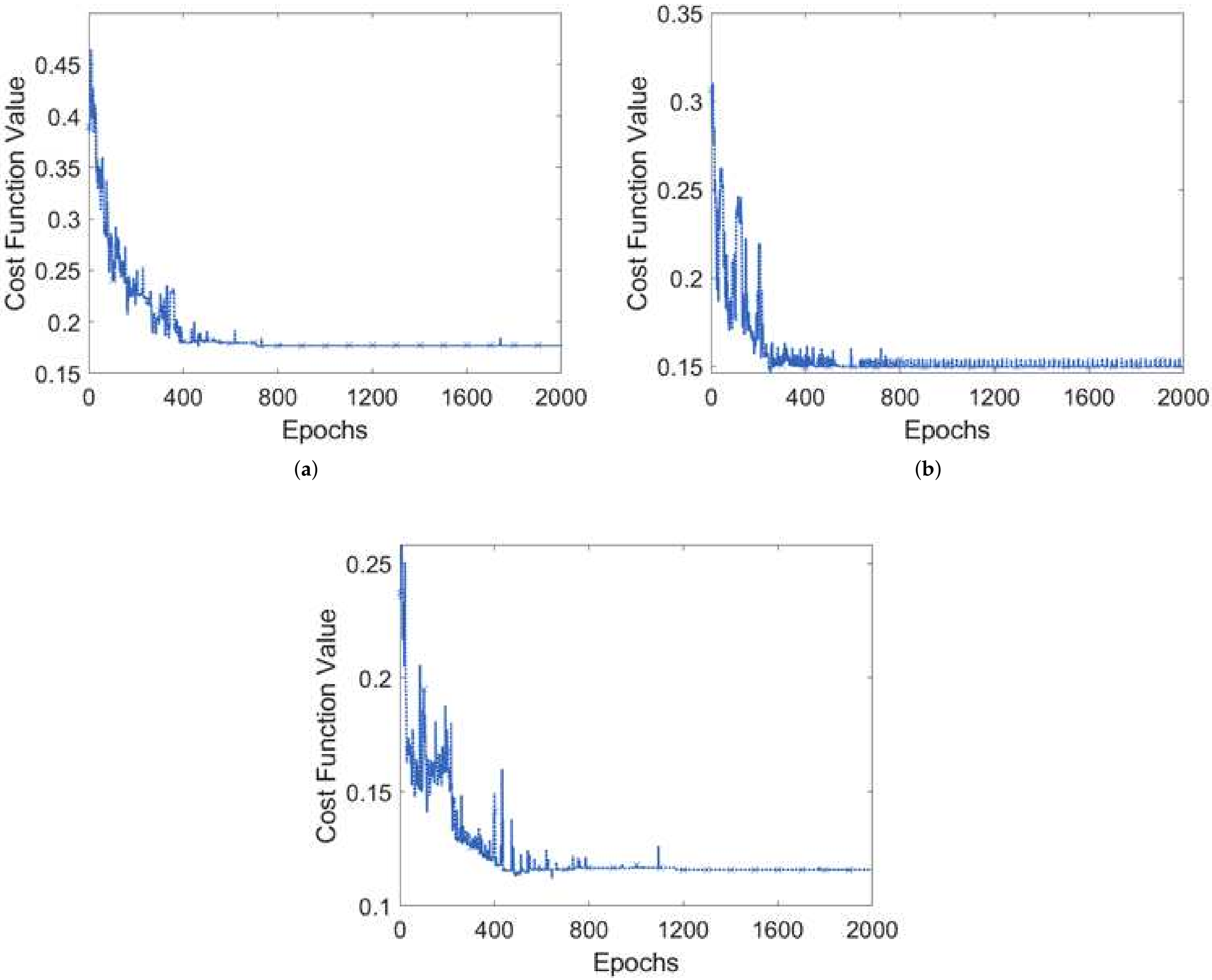
B. Is the proposed solution effective in finding the best placement for mobile edge servers when different number of base stations are present in the network?

C. Is the proposed solution effective in finding the best placement for mobile edge servers when the number of mobile edge servers present in the network is varied?

A. Does the proposed solution generalize across different random initialized values used in the experimentation?

The objective of this experiment is to show the generalizability of the proposed solution across different initial values of parameters used in the experimentation. The RL agent's initial location and other experimentation parameters, such as α, β, and ϵ require assignment of an initial value which is set to random values. Therefore, using different seeds for random values will change the initial state of each RL agent. Ideally, the average final cost for all these experiments should be same. However, the use of distinct random seeds makes the initial state set for RL exploration strikingly different presenting an entirely different search space for the RL agents to explore and exploit.

In Figure 5, the cost function values across number of epochs are shown where each epoch represents the instant at which all RL agents take actions. The plotted cost function value is the averaged cost function value of each mobile edge server used in the experimentation. We can observe that for different seed values impacting the initial state of each agent, the proposed solution is able to minimize the average cost function values. Another significant observation is the fast convergence of the cost function values for each seed, shown in Figure 5. These results enable us to claim that even with simplified state space, the proposed solution without the use of any complex deep learning models is generalizable for different initial states.



(**a**)

(**b**)

(**c**)

Figure 5. Number of BS = 120 and Number of ES = 20. (**a**) Random Seed = 5. (**b**) Random Seed = 8. (**c**) Random Seed = 23.

B. Is the proposed solution effective in finding the best placement for mobile edge servers when different number of base stations are present in the network?

In Figure 6, the performance of the proposed solution is shown for varying number of base stations available in the MEC network. Ideally, the change in the number of base stations in the network should not affect the convergence of the proposed multi-agent RL assisted edge servers placement. The results in Figure 6 show that for 120 and 240 base stations available in the network, the cost values converge after a number of epochs. Another significant observation is the increase in cost function value for initial few epochs when number of base stations are 240. This is mainly because RL agents explore the environment by choosing random actions dependent on ϵ which is reduced at each epoch.

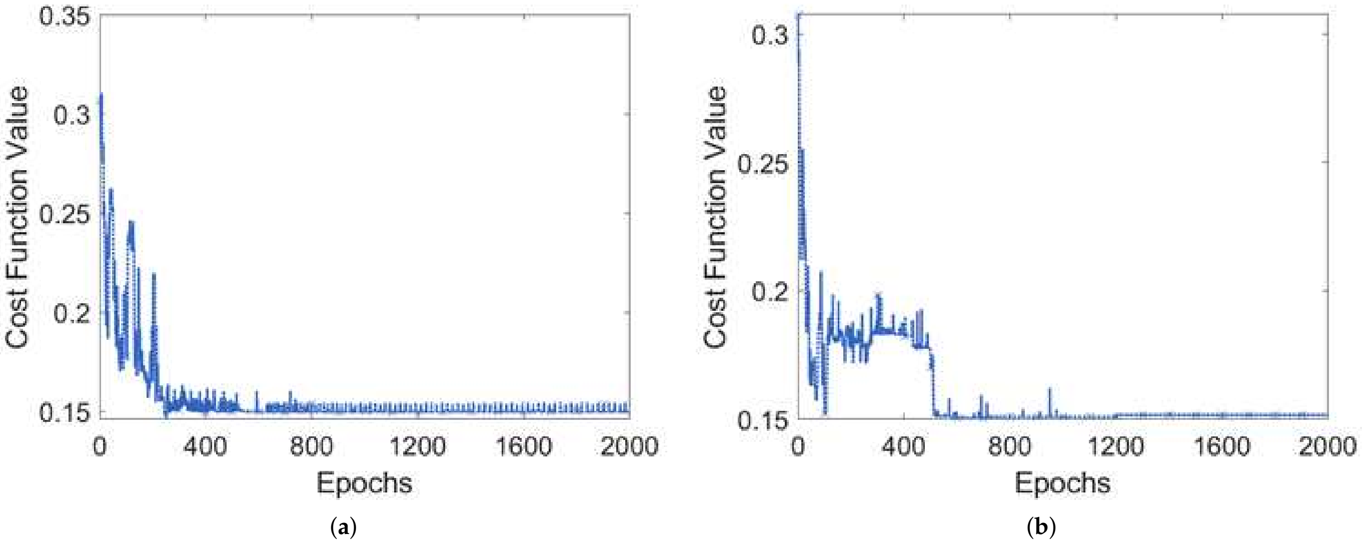


Figure 6. Number of ES = 20 and Seed = 8. (**a**) BS = 120. (**b**) BS = 240.

C. Is the proposed solution effective in finding the best placement for mobile edge servers when the number of mobile edge servers present in the network is varied?

In Figure 7, the performance of the proposed solution is shown for varying number of mobile edge servers. Ideally, varying the number of mobile edge servers in the MEC network should not affect the convergence of the proposed multi-agent RL assisted edge servers placement. The results in Figure 7 show that for 20 and 30 mobile edge servers to be placed in the network, the cost values converge after a number of epochs.

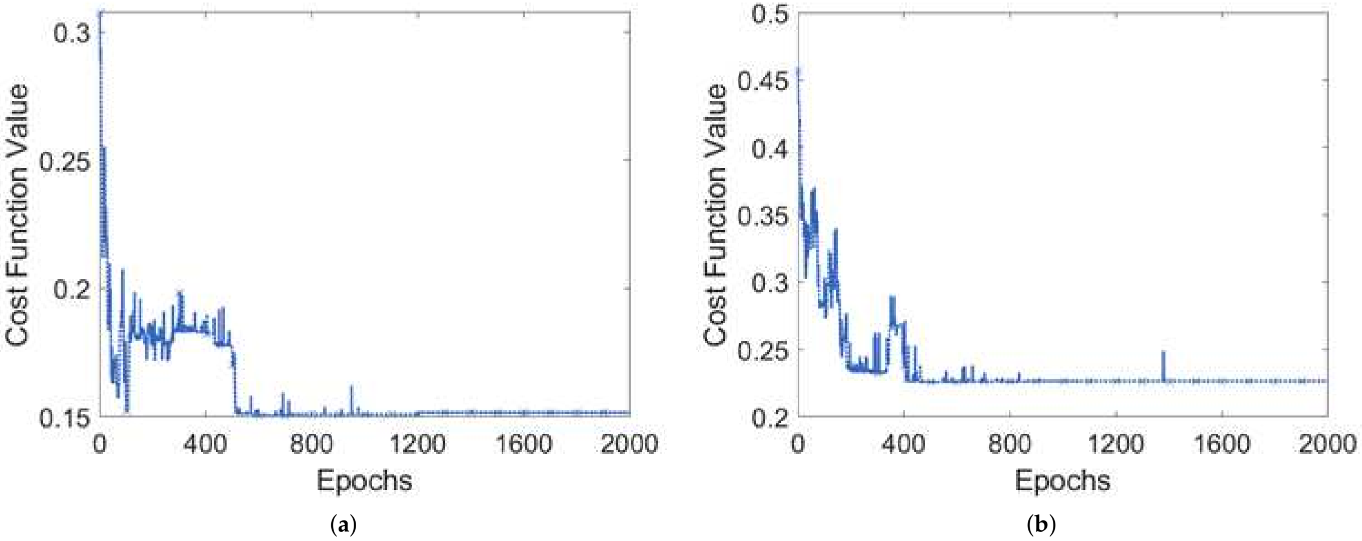


Figure 7. Number of BS = 240 and Random Seed = 8. (**a**) ES = 20. (**b**) ES = 30.

In Figure 8, we present the results of a toy example in which 03 base stations are placed in a network namely A, B, and C. Base stations A and C are placed in the corners and base station B in the middle. Considering the toy example allows us to evaluate the performance of proposed solution against a numerical solution. Through numerical solution, the optimal locations for edge servers are 'A' and 'C' which can be observed that after a certain number of epochs, both the edge servers converge to its optimal locations.

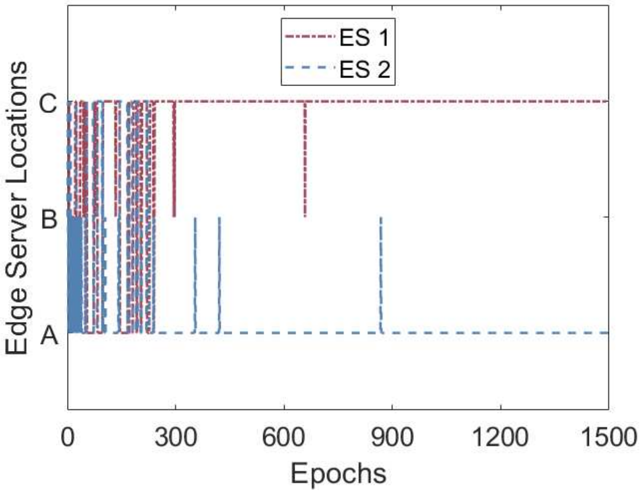


Figure 8. Proposed solution comparison with ground truth for a toy example; where three base stations namely A, B, and C, are placed in a network.

7. Security Perspective

We demonstrate how the reinforcement learning assisted mobile edge server placement can be performed with multi-agent reinforcement learning coordination techniques. The multi-agent coordination problem may give birth to different security related issues since the working of a reinforcement learning agent is based on the observations from other agents, therefore, if the shared information is modified it will affect the working of entire solution. In the following sections, we present a possible scenario by which security can be breached and present the proposed countermeasures.

7.1. Scenarios

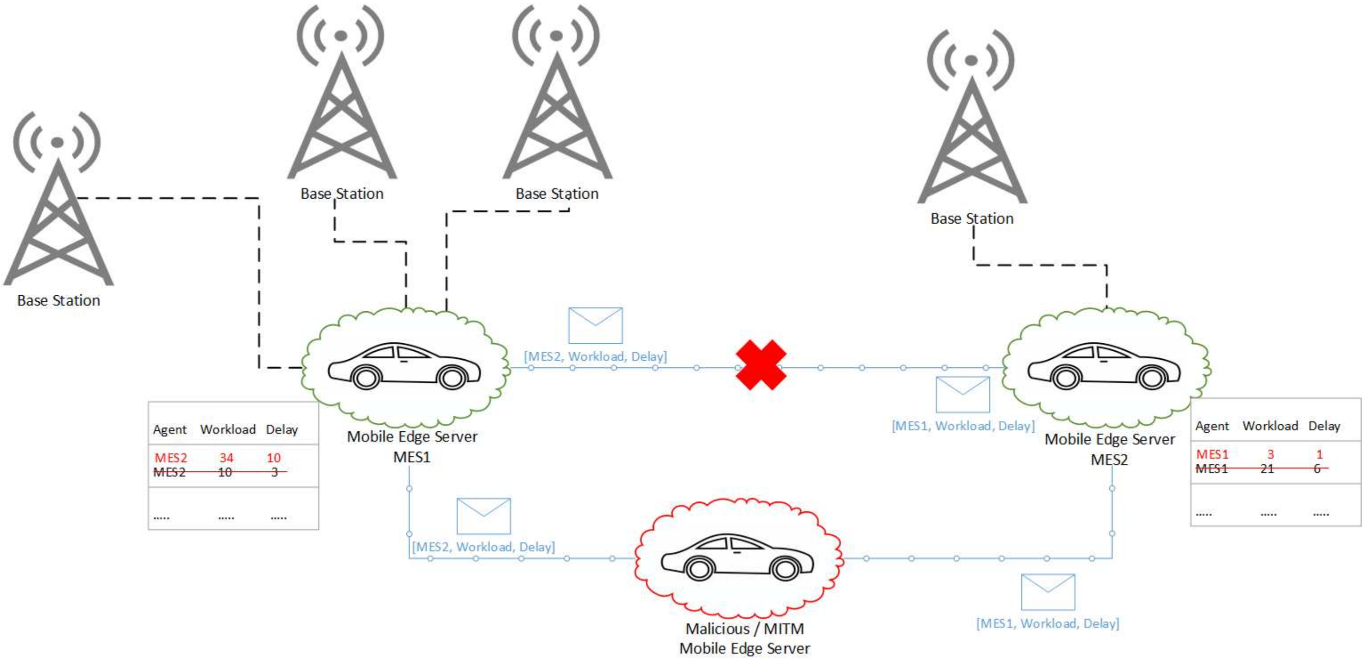
As discussed in earlier sections, workload is the sum of all the data offloaded from connected base stations and delay is the sum of the distance from connected base stations. The agent performs its actions based on the reward function and reward is based on workload balancing and delay.

From a security perspective, the first scenario case we can consider is the when an agent itself or man-in-the-middle (MITM) can alter the information contained in the packets passed between agents. This is done in order to force the agent to change its location to another base station or stay with the same base station despite a need to workload balancing and delay minimization.

Figure 9 presents a scenario of security issues present in the work. Let us assume that the values of workload and delay are increased. This will force the mobile edge server MES1 to move from it current location to a particular location where workload balancing is needed since the agents will assume that it needs to balance the workload and delay by migrating to other base stations. In contrast, if an agent itself or MITM alters the information by decreasing the values of workload and delay, the mobile edge server, MES2, will assume that everything in the network is fine and it does not need to change its location to other base stations.

Since the reward function at the agents makes decision based on the information (workload and delay) received, thus, it will act accordingly. Therefore, in the first scenario, after the information is altered by the malicious node, the mobile edge server MES1 will assume that it needs to move to the MES2 location in order to balance the workload.

The second potential security scenario can be a malicious entity compromised an agent in the network. The attack vector might be different form altering the packet en-route, but the malicious entity can achieve the same impact as the first scenario. Furthermore, a malicious entity compromising an agent in the network may go for eavesdropping with the aim to (a) try to construct a traffic map of the network, (b) build communication patterns between agents, and (c) read communication packets. In the above listed aims, 'a' and 'b' can help the malicious user to understand the network design and communication patterns between agents. This can assist the malicious entity to mount a network wide attack, for example DDoS. The option 'c' allows the malicious entity to read the information communicated between the agents. This might reveal some sensitive information about the agents or applications being executed on these agents.



Agent	Workload	Delay
MES2	34	10
~~MES2~~	~~10~~	~~3~~
.....	.....	.....

Agent	Workload	Delay
MES1	3	1
~~MES1~~	~~21~~	~~6~~
.....	.....	.....

Figure 9. Malicious/MITM Agent scenario at a Mobile Edge Server.

The third potential security scenario can be Trojan horse attacks, whether a Trojan horse is embedded in hardware or software. The objectives of such an attacker can be similar to the malicious entity in "second potential security scenario". The attack objective and impact can also be similar.

The fourth potential security scenario can be insider threat. In this attack, an insider compromises a single node, a collection of the node or the whole network—dependent upon the access of the insider and how senior their role is. The attack objective and impact can be similar to the three scenarios listed before but depending upon the access privilege the impact on the network can be significant.

We are not considering the lack of knowledge or expertise in an organization or a genuine human error as a security threat. As this in most cases leads to the vulnerability that the malicious actors in the above scenarios exploit or the respective impacts.

7.2. Countermeasures

In this section, we explore potential countermeasures to the each of the security scenarios discussed above.

7.2.1. Countermeasure to First Security Scenario

As discussed earlier, an agent running on the mobile edge server has global information of workload and delay of all the agents in the network, whereas, the decision is made locally based on the information received and used in the reward function by an agent.

These security issues require verifying the identity of an agent before allowing access to resources in a system. Therefore, an authentication mechanism needs to enable the identity of an agent to be verified and, thus, to prevent it from faking or masquerading.

Additionally, data integrity needs to be ensured to prevent data from being altered or destroyed while being exchanged amongst the agents in an unauthorized manner to maintain consistency. Hence, a secure protocol should withstand such attacks and offer authentication and integrity of the exchanged data.

The cryptosystem we aim to achieve is one where the entities communicate over an insecure network, resulting in both parties needing to provide identity authentication first, and this then proves to the receiver the integrity of the messages. Peer authentication and secure data transmission are vital in our system.

Regarding authentication, public key infrastructure (PKI) provides the means of digital certificate for providing authentication. In our study we assume that all entities have digital certificates generated by the CA.

To achieve integrity between base station and mobile edge server, one may employ integrity encryption techniques, such as HMAC. However, before doing this, both entities should agree first on a secret key. Due to the key distribution problem, key agreement protocols have emerged where the actual key is not transferred on an untrusted channel.

The proposed protocol is divided into four stages:

1. Stage 1 Mutual Authentication Phase:
   We assume that both parties already registered with CA, trust the same CA, and possess their own public key, own private key, own implicit certificate, and CA's public key. Both entities the base station and mobile edge server perform a handshake where both parties exchange their digital certificate to verify the authenticity of each party.
2. Stage 2 Key Agreement:
   After authentication is done in both parties, they should agree on a shared master key. In our protocol, we will use the elliptic curve Diffie Hellman (ECDH) protocol that is most suitable for constrained environments. The elliptic curve cryptosystems are used for implementing protocols such as the Diffie–Hellman key exchange scheme [26] as follows:

   A. A particular rational base point P is published in a public domain.
   B. The base station and mobile edge server choose random integers k_A and k_B respectively, which they use as private keys.
   C. The base station computes:

   $$A = k_A * P = (x_A, y_B) \tag{9}$$

   D. The mobile edge server computes:

   $$B = k_B * P = (x_B, y_B) \tag{10}$$

   and then both entities exchange these values over an insecure network.
   E. Using the information, they received from each other and their private keys, both entities compute:

   $$Q = k_A * B = k_A * (k_B * P) \tag{11}$$

   and

   $$Q = k_B * A = k_B * (k_A * P) \tag{12}$$

   respectively. This is simply equal to,

   $$Q = (k_A * k_B) * P \tag{13}$$

   which serves as the shared master key that only base station and mobile edge server possess.
3. Stage 3 Key Derivation:

To reduce the computational complexity on both parties, we assume that the mutual authentication phase is done periodically, only the session secret key is generated from the shared master key for achieving integrity protection algorithm, such as message authentication code (MAC) on each session.

The proposed protocol should use the best option for key derivation function (KDF) that ensures randomness, and we advocate the KDF recommendations in [27], which takes into consideration randomness through the use of random numbers (Nonce) and key expansion. Each peer computes the actual session key PK via the chosen key KDF χ, as:

$$PK = \chi(Q, Nonce) \tag{14}$$

4. Stage 4 Message Exchange:
   The exchanged data, such as workload and delay, need to be protected against unauthorized modification, hence HMAC is used to ensure the integrity.
   The base station calculates

$$D = HMAC_{PK}((D(\ell), W(\ell)) \tag{15}$$

   The base station sends W(ℓ), D(ℓ) and D to the mobile edge server.
   The mobile edge server uses the agreed derived session key to calculate HMAC of W(ℓ) and D(ℓ) and verifies its integrity with D.

The performance of the above listed protocol depends on multitude of factors including: the processor speed, availability of specialized cryto-hardware, and communication (network speed). However, to provide a reference performance, we setup a test-bed with each agent node is a Raspberry Pi model B supplied with a Wi-Fi USB dongle TL-WN722N by TP-LINK.

In all the measurements we made, the nodes were configured in ad-hoc mode. Each agent is then connected to a server through an Ethernet connection. The server manages individual agents so as to prepare them for the target scenario and is also in charge of collecting the measurements. In our reference performance measurement, we only consider the scenario of two agents setting up a security communication link directly with each other.

Nevertheless, effective measurement can be done internally on the node initiating the secure channel, called a client, and it can be done at the level of the network data exchanged between the agents of the network and captured with a Wi-Fi card set in monitor mode on the server.

Based on this setup, the stated protocol in this section took 4282 milliseconds (on average) over 100 executions.

7.2.2. Countermeasure to Second Security Scenario

Compromising an agent is basically system security problem. Potential countermeasures to this can be hardening the agent environment, pen-testing it before deployment, updating it regularly when new vulnerabilities come alive, having strong access control policies related to the agent configuration, etc.

Another potential solution to this problem can be to have a secure execution environment in individual agents. The secure execution environment can help protect sensitive code during its execution and avoid any malicious entities from interfering with it. Even then, the above listed precautions should be taken.

7.2.3. Countermeasure to Third Security Scenario

Protection against Trojan horse attacks, especially related to Trojan horse in the hardware is dependent on secure and reliable supply chains. An organization can test their agents to detect whether they have some non-characteristic behavior. Similar actions can also be taken for the software bases Trojan horse. An effective mechanism can be continuous monitoring of the agent (hardware and software) behavior to detect any stealth Trojan that evades the detection pre-deployment.

7.2.4. Countermeasure to Fourth Security Scenario

Insider threat is a significant challenge to overcome in any large network or organization. A potential countermeasure to such a threat can include limited privilege that only requires single user approval. All sensitive actions should require multiple users to approve and deploy the changes. Employee management and making sure that HR revokes credentials of any employee that is leaving the company. Finally, user network activities and behavior monitor can help minimize any impact from disgruntle employees.

8. Conclusions and Future Work

Mobile edge computing facilitates in providing data storage and computational resources to mobile and low-power wireless sensor devices. In this work, we have shown a multi-agent reinforcement learning based solution for the placement of edge services in a mobile network, such that the network latency is minimized and load on edge servers is balanced. The experimental evaluation using Shanghai's Telecom dataset proves that the proposed solution quickly converges. Further, we provided a detailed analysis of the type of security attacks possible in the proposed solution concept. We also listed some of the countermeasures that can be used to deal with the security risks. The effectiveness of the proposed method even with a simple state-space provides a promise to the proposed solution. Much future work remains before the proposed solution can be implemented in the real world, but our findings suggest that this approach has considerable potential. This work serves as the proof of concept for a secure multi-agent RL implementation for edge server placement problem. However, to further validate the results of proposed model, we intend to implement the proposed model in a full-stack emulator such as SIMENA NE5000.

Author Contributions: Conceptualization, M.K.K., S.A.G., R.N.A.; investigation, M.K.K., S.A.G., R.N.A.; methodology, M.K.K., S.A.G., R.N.A.; resources, S.A.G.; software, M.K.K., S.A.G., R.N.A.; supervision, S.A.G.; visualization, M.K.K., R.N.A.; writing—original draft, M.K.K., S.A.G., R.N.A.; writing—review and editing, R.N.A., D.S.; funding acquisition, S.A.G. All authors contributed to the final version. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by King Khaled University under Grant Agreement No. 6204.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lee, J.; Kim, D.; Lee, J. Zone-based multi-access edge computing scheme for user device mobility management. *Appl. Sci.* **2019**, *9*, 2308. [CrossRef]
2. Bilal, K.; Khalid, O.; Erbad, A.; Khan, S.U. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Comput. Netw.* **2018**, *130*, 94–120. [CrossRef]
3. Porambage, P.; Okwuibe, J.; Liyanage, M.; Ylianttila, M.; Taleb, T. Survey on multi-access edge computing for internet of things realization. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2961–2991. [CrossRef]
4. Lähderanta, T.; Leppänen, T.; Ruha, L.; Lovén, L.; Harjula, E.; Ylianttila, M.; Riekki, J.; Sillanpää, M.J. Edge server placement with capacitated location allocation. *arXiv* **2019**, arXiv:1907.07349.
5. Kasi, S.K.; Kasi, M.K.; Ali, K.; Raza, M.; Afzal, H.; Lasebae, A.; Naeem, B.; ul Islam, S.; Rodrigues, J.J. Heuristic edge server placement in Industrial Internet of Things and cellular networks. *IEEE Internet Things J.* **2020**, *8*, 10308–10317. [CrossRef]
6. Xu, X.; Shen, B.; Yin, X.; Khosravi, M.R.; Wu, H.; Qi, L.; Wan, S. Edge Server Quantification and Placement for Offloading Social Media Services in Industrial Cognitive IoV. *IEEE Trans. Ind. Inform.* **2020**, *17*, 2910–2918 [CrossRef]
7. Yin, H.; Zhang, X.; Liu, H.H.; Luo, Y.; Tian, C.; Zhao, S.; Li, F. Edge provisioning with flexible server placement. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *28*, 1031–1045. [CrossRef]
8. Guo, Y.; Wang, S.; Zhou, A.; Xu, J.; Yuan, J.; Hsu, C.H. User allocation-aware edge cloud placement in mobile edge computing. *Softw. Pract. Exp.* **2019**, *50*, 489–502. [CrossRef]
9. Jia, M.; Cao, J.; Liang, W. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Trans. Cloud Comput.* **2015**, *5*, 725–737. [CrossRef]
10. Wang, S.; Zhao, Y.; Xu, J.; Yuan, J.; Hsu, C.H. Edge server placement in mobile edge computing. *J. Parallel Distrib. Comput.* **2019**, *127*, 160–168. [CrossRef]

11. Bouet, M.; Conan, V. Mobile edge computing resources optimization: A geo-clustering approach. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 787–796. [CrossRef]
12. Zeng, D.; Gu, L.; Pan, S.; Cai, J.; Guo, S. Resource Management at the Network Edge: A Deep Reinforcement Learning Approach. *IEEE Netw.* **2019**, *33*, 26–33. [CrossRef]
13. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Emerg. Top. Comput.* **2019**, *6750*, 1. [CrossRef]
14. Huang, L.; Bi, S.; Zhang, Y.J. Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2019**, *1233*, 1. [CrossRef]
15. Zhai, Y.; Bao, T.; Zhu, L.; Shen, M.; Du, X.; Guizani, M. Toward Reinforcement-Learning-Based Service Deployment of 5G Mobile Edge Computing with Request-Aware Scheduling. *IEEE Wirel. Commun.* **2020**, *27*, 84–91. [CrossRef]
16. Li, B.; Hou, P.; Wu, H.; Hou, F. Optimal edge server deployment and allocation strategy in 5G ultra-dense networking environments. *Pervasive Mob. Comput.* **2021**, *72*, 101312. [CrossRef]
17. Zeng, F.; Ren, Y.; Deng, X.; Li, W. Cost-effective edge server placement in wireless metropolitan area networks. *Sensors* **2019**, *19*, 32. [CrossRef]
18. Cao, K.; Li, L.; Cui, Y.; Wei, T.; Hu, S. Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing. *IEEE Trans. Ind. Inform.* **2020**, *17*, 494–503. [CrossRef]
19. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
20. Littman, M.L. Value-function reinforcement learning in Markov games. *Cogn. Syst. Res.* **2001**, *2*, 55–66. [CrossRef]
21. Busoniu, L.; Babuska, R.; De Schutter, B. Multi-agent reinforcement learning: A survey. In Proceedings of the 2006 9th International Conference on Control, Automation, Robotics and Vision, Singapore, 5–8 December 200 ; pp. 1–6.
22. Matignon, L.; Laurent, G.J.; Le Fort-Piat, N. Hysteretic q-learning: An algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, USA, 29 October–2 November 2007; pp. 64–69.
23. Wang, S.; Guo, Y.; Zhang, N.; Yang, P.; Zhou, A.; Shen, X.S. Delay-aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach. *IEEE Trans. Mob. Comput.* **2019**, *20*, 939–951. [CrossRef]
24. Xu, J.; Wang, S.; Bhargava, B.K.; Yang, F. A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing. *IEEE Trans. Ind. Inform.* **2019**, *15*, 3538–3547. [CrossRef]
25. Hajiyev, A. Optimal Choice of Server's Number and the Various Control Rules for Systems with Moving Servers. In *International Conference on Management Science and Engineering Management*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 379–399.
26. Haakegaard, R.; Lang, J. The Elliptic Curve Diffie-Hellman (Ecdh). 2015. Available online: https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf (accessed on 10 June 2020)
27. Charan, K.S.; Nakkina, H.V.; Chandavarkar, B.R. Generation of Symmetric Key Using Randomness of Hash Function, In Proceedings of the 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020; pp. 1–7.

electronics

MDPI

Article

A Task Execution Scheme for Dew Computing with State-of-the-Art Smartphones †

Matías Hirsch [1,*], Cristian Mateos [1], Alejandro Zunino [1], Tim A. Majchrzak [2], Tor-Morten Grønli [3] and Hermann Kaindl [4]

[1] ISISTAN–UNCPBA–CONICET, Tandil, 7000 Buenos Aires, Argentina; cristian.mateos@isistan.unicen.edu.ar (C.M.); alejandro.zunino@isistan.unicen.edu.ar (A.Z.)
[2] Department of Information Systems, University of Agder, 4630 Kristiansand, Norway; timam@uia.no
[3] Mobile Technology Lab, Department of Technology, Kristiania University College, 0176 Oslo, Norway; tor-morten.gronli@kristiania.no
[4] Institute of Computer Technology, TU Wien, 1040 Vienna, Austria; hermann.kaindl@tuwien.ac.at
* Correspondence: matias.hirsch@isistan.unicen.edu.ar

† This paper is an extended version of our paper published in HICSS-54.

Abstract: The computing resources of today's smartphones are underutilized most of the time. Using these resources could be highly beneficial in edge computing and fog computing contexts, for example, to support urban services for citizens. However, new challenges, especially regarding job scheduling, arise. Smartphones may form ad hoc networks, but individual devices highly differ in computational capabilities and (tolerable) energy usage. We take into account these particularities to validate a task execution scheme that relies on the computing power that clusters of mobile devices could provide. In this paper, we expand the study of several practical heuristics for job scheduling including execution scenarios with state-of-the-art smartphones. With the results of new simulated scenarios, we confirm previous findings and better comprehend the baseline approaches already proposed for the problem. This study also sheds some light on the capabilities of small-sized clusters comprising mid-range and low-end smartphones when the objective is to achieve real-time stream processing using Tensorflow object recognition models as edge jobs. Ultimately, we strive for industry applications to improve task scheduling for dew computing contexts. Heuristics such as ours plus supporting dew middleware could improve citizen participation by allowing a much wider use of dew computing resources, especially in urban contexts in order to help build smart cities.

Keywords: dew computing; edge computing; smartphone; job scheduling; scheduling heuristics

Citation: Hirsch, M.; Mateos, C.; Zunino, A.; Majchrzak, T.A.; Grønli, T.-M.; Kaindl, H. A Task Execution Scheme for Dew Computing with State-of-the-Art Smartphones. *Electronics* **2021**, *10*, 2006. https://doi.org/10.3390/electronics10162006

Academic Editors: Ka Lok Man and Kevin Lee

Received: 29 June 2021
Accepted: 16 August 2021
Published: 19 August 2021

1. Introduction

Smartphones have increasing capabilities of processing information, which typically are underutilized [1,2]. Cities (and citizens) could benefit from such a plethora of underutilized resources if these were properly orchestrated. Any person carrying a smartphone could contribute with valuable resources to help cities grow and to manage them in a more sustainable way. For instance, anyone may help to improve urban road maintenance by collecting pavement data [3]. Participatory platforms have been proposed to enable people to voluntarily contribute data sensed with their personal mobile devices [4,5].

Cities generate vast amounts of data for different smart city applications through Internet of Things (IoT) sensors and surveillance cameras [6,7]. Processing locally sensed data can be done in different but not necessarily mutually exclusive ways, for instance, using distant cloud resources, offloaded to proximate fog servers, or with the help of devices with computing capabilities within the data collection context, e.g., with smartphones. This latter architectural option has been considered as an attractive self-supported sensing and computing scheme [8]. In addition, hybrid and volunteer-supported processing architectures were proposed to avoid overloading resource-constrained devices [9]. Depending on the adopted approach—hybrid or self-supported—effectively managing smartphones' limited energy and heterogeneous computing capabilities requires more research [10].

In a previous work [1], we proposed baseline heuristics to perform such resource management for a variant of self-supported sensing and a computing scheme where data collected in a local context is processed by a group of smartphones within the same context. We call that a "dew context" to continue with the cloud/fog metaphor used to describe a layered computing infrastructure.

Complementary, in this work, we subject these baseline heuristics to new execution scenarios, now considering simulated dew contexts with modern smartphone profiles—battery traces and benchmark results from recent smartphone models. Additionally, we go beyond simulating synthetic jobs load by benchmarking CPU usage and inference time of a Tensorflow lite state-of-the-art model trained for recognizing a set of generic objects. Throughout this new set of experiments we shed some light on the viability of distributing jobs at the edge using smartphones for AI-based data stream processing applications in smart cities, which is another major difference with respect to [1].

Specifically, the contributions introduced by this paper are:

1. We evaluate state-of-the-art practical load balancing heuristics for dew computing [1] but using performance parameters and battery traces from recent smartphone models. This helps to ensure the practical significance of such heuristics.
2. We compare results of current and previous experiments to refine our findings on the heuristics' performance;
3. We illustrate the practical benefits of adopting our multi-smartphone computing approach versus single-smartphone computing through a realistic scenario, centered around a real Android application for AI-based object detection that might serve as a kernel for implementing many edge-powered smart city applications.

This paper is organized as follows. Section 2 discusses related work. Then, Section 3 provides a motivating example. Section 4 presents the studied scheduling heuristics. Section 5 describes the evaluation methodology and experimental design, while Section 6 gives an overview of metrics and results. In Section 7, we present an experiment simulating AI-based video processing. A summary of results and practical challenges are discussed in Section 8, while Section 9 concludes and points to future work.

2. Related Work

The exploitation of computing resources provided by smart devices in dew computing contexts—i.e., where both data collection and processing happen at the edge of the network—introduces new challenges in terms of job scheduling algorithms [10]. Since smart devices rely on batteries as their main power source, one of the challenges is to manage the remaining energy of resource provider nodes in the network. Hence, the impact of a given job execution schedule on a device's future battery level must be considered. This involves targeting the maximization of completed jobs without exceeding the node's energy availability.

There are at least two approaches for pursuing this objective. One models job scheduling as an optimization problem. Chen et al. [11], Ghasemi-Falavarjani et al. [12], Wei et al. [13] suggested to include a device's remaining energy as a constraintin the problem formulation, i.e., while exploring feasible solutions, the energy employed in executing jobs must not exceed the available energy on the devices' batteries. To tailor input variables of algorithms following this approach, it is necessary to have accurate job energy consumption data, which are rarely available. To obtain such data in the general case, a detailed quantification of resource demands are needed, which, in turn, vary according to device characteristics. Given the wide variability of device models on the market (cf., e.g., work by Rieger and Majchrzak [14]), it is unrealistic to assume homogeneous device clusters. If not precomputed, scheduling input should be obtained while converting a data stream into jobs to be processed.

The other approach does not require energy-related job details. It performs load balancing based solely on node characteristics. Hirsch et al. [15] combined the last known battery level with a function including different performance scores, which rates the capability of a device to successfully complete the last arrived job. Jobs are scheduled by following an *online* approach, i.e., upon each job arrival, the scheduling logic creates a ranking by evaluating the function for all candidate devices, and the job is assigned to the one ranked best.

Resource heterogeneity imposes further challenges that scheduling algorithms in dew computing contexts must deal with. The co-existence of smart devices that belong to the same or different generations, equipped with hardware able to render dissimilar computing and data transfer throughput, should not be ignored when including them as first-class resource providers. The authors of Yaqoob et al. [16] considered the number of cores, speed, and CPU workload, which are evaluated by the proposed heuristics when allocating computing-intensive tasks to mobile devices. The authors of Hirsch et al. [15] considered heterogeneity by differentiating the nodes' computing capabilities via their MFLOPS indicator, which is a common metric in scientific computing to rate processor speed when executing floating-point operations. All in all, the heuristics by Yaqoob et al. [16] recognize resource heterogeneity related to computing capability only. For stream processing applications, where data transfer under varying delay and energy consumption of wireless communication is present, new practical online heuristics are necessary to deal with both node computing and communication heterogeneity.

3. Motivating Example

Smart cities integrate multiple sources of information and process massive volumes of data to achieve efficient solutions and monitor the state of a wide range of common issues, including maintenance of public spaces and infrastructure or the security of citizens, just to mention two of them. Ultimately, they contribute to societal security [17]. Participatory sensing platforms encourage citizens to contribute incidents data, such as geolocalized photos, videos, and descriptions that have to be analyzed, filtered, and prioritized for proper treatment. This requires a data processing infrastructure and depends on the citizens' willingness to manually enter or record data.

A proactive way to gather relevant data could be installing a dedicated sensor and processing infrastructure. However, to reduce fixed costs and to avoid the congestion of communication networks with a high volume of raw data captured [18], a hybrid approach that exploits near-the-field, ubiquitous computing power of smart mobile devices is feasible. By analyzing a city's dynamics, it is not hard to identify places where citizens are regularly connected to the same local area network with their smartphones, e.g., small parks or public transport. Suppose that citizens in such a context agree to contribute processing power, even though they may not like to provide data sensed with their devices. However, these may be used to filter and identify relevant information from data streams captured by sensors cleverly positioned within the context, and connected to the same network as nearby mobile users.

Consider, for instance, passengers riding a bus, where smartphones receive data via their WiFi connections. These may be samples of environmental sounds or images captured with devices that have been specifically installed in the bus for, e.g., real-time sensing of noise pollution, detecting pavement potholes, counting trees or animals, or whatever information may be useful for a smart city to forecast events, schedule repairs, or public space maintenance duties. The smartphones could be used, on a voluntary basis, for pre-processing such data before it is transferred to the distant cloud in a curated and reduced form. Pre-processed data, in turn, could be used to feed Internet-accessible, real-time heatmaps summarizing such information so that decision-makers can act promptly and accordingly. In terms of the hardware resources to be exploited, the computations required to do so might range from medium-sized, CPU-intensive ones, such as finding patterns in digitized sound streams, to complex CPU/GPU-intensive tasks such as detecting or tracking objects from image streams using deep neural networks. Nowadays, it is not surprising to find affordable smartphones on the market with eight-core processors and GPUs capable of running deep learning frameworks such as Tensorflow (https://www.tensorflow.org/ accessed on 18 August 2021).

How to efficiently and fairly balance data processing among available smartphones in a dew context is challenging, though. This essentially stems from the singularities that characterize smartphones [19], namely user mobility, lack of ownership, and exhaustible resources. Smartphones are inherently mobile devices and their users may enter/leave the dew context in unpredictable ways, constraining the time window within which computations can be submitted and successfully executed on a device. Failure to obtain task results from a leaving device intuitively forces the scheduler to either discard these results, which harms effectiveness from the application standpoint, or re-submitting the associated tasks to alternative devices, which harms application throughput. Moreover, lack of ownership means that from the perspective of a data processing application, smartphones are non-dedicated computing resources, of course. Resources such as CPU/GPU time, memory, storage, and network bandwidth are shared with user processes and mobile applications. Hence, any dew scheduler must ensure that data processing tasks do not significantly degrade the performance of native applications and user experience, otherwise users might become reluctant to contribute or keep contributing their computing resources in dew contexts. Lastly, the serving time of a mobile device is limited both by the energy level of its battery at the time it enters the dew context, and the rate at which energy is consumed by the mobile device, which depends on several factors including screen brightness level, user applications and system services being executed, battery inherent efficiency, and so on. Not only computing capabilities (e.g., CPU/GPU speed) are important for distributing dew tasks, but also energy availability/battery efficiency. Of course, a dew scheduler should not exhaust a mobile device's energy since this would also discourage users to process dew tasks.

4. Load Balancing Heuristics

Figure 1 depicts an overview of a dew context, a distributed computing mobile cluster (in this case, operating inside a bus) for processing jobs generated locally. When close to the dew context local area network, mobile devices are enlisted to contribute with computing resources by registering themselves with the proxy [20]. In the example, the proxy is an on-chip-pc integrated circuit. The proxy balances the jobs processing load with a heuristic that sorts the devices' appropriateness using some given criterion. The best ranked node is assigned with the incoming job and the ranking is re-generated upon each job arrival.

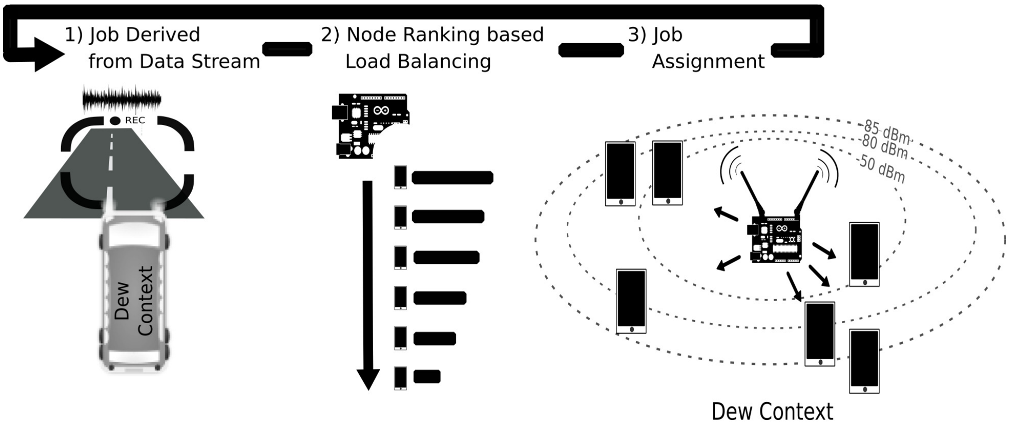


Figure 1. The dew context.

We propose and evaluate practical heuristics to sort devices, which combine easy-to-obtain device and system performance information. One of these is AhESEAS, an improvement to the ESEAS (Enhanced Simple Energy-Aware Scheduler) [20]. Another heuristic is ComTECAC, which was inspired by criteria targeting nodes' fair energy spending [21]. In the following, we provide details of the formulation of these novel criteria to rank resource provider devices.

AhESEAS: the *Ahead Enhanced Simple Energy Aware Scheduler* is a criterion that combines a device's MFLOPS, its last reported battery level (SOC) and a counter of assigned jobs, in the same way as the ESEAS [20] formula, with the exception of a change in the semantics of the last-mentioned counter. While in ESEAS the counter of assigned jobs is updated after the job input has been completely received by the node, in AhESEAS such an update occurs before, i.e., just after a node is selected for executing a job. By issuing an immediate counter update, i.e., without waiting for job input transferring time, gives AhESEAS rapid reaction to fast and continuous job generation, typical in stream processing applications. To differentiate the semantic change and to avoid confusion with the ESEAS formula, we have renamed $assignedJobs$ of ESEAS by $queuedJobs$ in the AhESEAS formula (and adding 1 to avoid that the denominator may become 0):

$$AhESEAS = \frac{MFlops * SOC}{queuedJobs + 1} \tag{1}$$

ComTECAC: the *Computation-Communication Throughput and Energy Contribution Aware Criterion* utilizes indicators of a node's computing and communication capabilities, as well as its energy spent for executing dew jobs, which is implemented with battery level updates reported by nodes. Ranking heuristics using ComTECAC determine the best-ranked node not only using a queued jobs component, but also with an energy contribution component. Thus, the load is evenly distributed among nodes, avoiding that strong nodes drain their batteries too much and earlier than weak nodes. This heuristic's formula is:

$$ComTECAC = \frac{MFlops * netPerf}{queuedJobs + 1} * (SOC - eContrib) \tag{2}$$

where:

- $MFlops$ and $queuedJobs$ have the same semantics as in AhESEAS.
- $netPerf$ is calculated as $\frac{1}{linkEfficiency}$, where $linkEfficiency$ relates node RSSI (received signal strength indicator) with Joules spent per KB of data transferred. $linkEfficiency$ is pre-computed and retrieved from a look-up table for a given RSSI value (see Table 2).
- SOC is a $[0,1]$ normalized value of the last reported battery level.
- $eContrib$ is a $[0,1]$ normalized value that accounts for the energy contributed by a device since it joined the mobile cluster. This value is updated upon each battery level drop reported by a node. The sub-formula to calculate it is $joinBattLevel - SOC$, where $joinBattLevel$ is the node battery level when it joined the cluster.

The rationale that led us to propose node ranking formulas based mainly on node information is that such information is easy to systematize providing that node resources/-capabilities can be determined through the specifications of the constituent hardware parts, either via manufacturer data (e.g., battery capacity) or comparisons with other nodes through benchmark suites [22–25]. For scheduling purposes, the update frequency of such information in our approach would depend on the existence of new device models or benchmarks on the market. Node information can be collected once, systematized, and reused many times. In contrast, except for special-purpose systems running jobs from very specific applications, scheduling logic that uses job information as input—e.g., job execution time—is difficult to generalize, in principle, due to the impossibility to accurately estimate the job execution time of any possibly given job [26]. In the dew contexts studied in this paper, where nodes are battery-driven, balancing load to maintain as many nodes ready for service for as long as possible, is a strategy that maximizes the parallel execution of independent jobs, which in turn aims at reducing makespan. The more cores are available, the more jobs execution can be expected to progress in parallel. For the proposed node ranking formulas, we adopt a fractional shape with a numerator reflecting a node's potential computing capability and a denominator expressing the way such capability diminishes or is decomposed with subsequent job assignments.

In the case of the AhESEAS ranking formula (Equation (1)), the node's computing capabilities are quantified by combining mega float point operations per second (MFLOPS) and the state of charge (SOC). The first factor is a widely used metric in scientific computing as an indicator of a node's computing power and can be obtained with the Linpack for Android multi-thread benchmark. MFLOPS serves the scheduler as a theoretical maximum computing throughput a node is able to deliver to the system. SOC provides information on how long a node could maintain this theoretical throughput. According to this criterion, for instance, nodes able to deliver the same MFLOPS but having different SOC would be assigned different numbers of jobs. This behavior relates to the fact that batteries are the primary energy source for job completion.

In the ComTECAC ranking formula (Equation (2)), a node's potential computing capability emerges from combining more factors than $MFLOPS$ and SOC. The numerator also includes wireless medium performance parameters, which is relevant to account for jobs input/output data transfers. Furthermore, SOC is not included as an isolated factor but as a component of a function that reflects a short-term memory of the energy contributed by nodes in executing jobs.

Note, that all node ranking formula parameters but $queuedJobs$, which simply counts jobs, represent resource capabilities and not job features. Moreover, except for SOC, which needs to be updated periodically, such parameters are constants and can be stored in a lookup table. All this makes the calculation of the formulas for the selection of the most appropriate node to execute the next incoming job, to have complexity $O_{(n)}$, where n is the number of currently connected nodes in the Dew context.

5. Evaluation: Methodology and Experiment Design

These novel load balancing heuristics have been evaluated using DewSim [27], a simulation toolkit for studying scheduling algorithms' performance in clusters of mobile devices. Real data of mobile devices are exploited by DewSim via a trace-based approach that represents performance and energy-related properties of mobile device clusters. This approach makes DewSim the best simulation tool so far to realistically simulate mobile device battery depletion, since existing alternatives use more simplistic approaches, where battery depletion is modeled via linear functions. Moreover, the toolkit allows researchers to configure and compare scheduling performance under complex scenarios driven by nodes' heterogeneity. In such a distributed computing system, cluster performance emerges, on one side, from nodes' aggregation operating as resource providers. On the the other side, performance depends on how the job scheduling logic manages such resources. A node's individual performance responds to node-level features including computing capability, battery capacity, and throughput of the wireless link established with a centralized job handler (proxy). Tables 1 and 2 outline node-level features considered in the experimental scenarios.

The computing-related node-level features presented in Table 1 refer to the performance parameters of real devices, whose brand/model is given in the first column. The performance parameters include the MFLOPS score, which is used by the simulator to represent the speed at which jobs assigned by the scheduler are finalized. The MFLOPS of a device are calculated from averaging 20 runs of the multi-thread benchmark of the Linpack for Android app. The multi-thread version of the test uses all the mobile device processor cores. The columns Node-type, OS version Processor, Chipset, and Released are informative, as these features are not directly configured in simulation scenarios but indirectly considered in the device profiling procedure. This procedure produces battery traces as a result, used to represent different devices' energy depletion curves.

Table 1. Computing-related node-level features.

Device Brand/Model	Node Type	Android Version	Mega FLOPS	Batt. Capacity (Joules)	Processor	Chipset	Released
LG/Optimus L9	smartph.	4.0	56	29,520	Dual-core 1.4 GHz Krait	Qualcomm MSM8230 Snapdragon 400	2013
Samsung/Galaxy S3	smartph.	4.3	179	28,728	Quad-core 1.4 GHz Cortex-A9	Exynos 4412 Quad	2012
Acer/Iconia A100	tablet	4.1.1	61	40,680	Dual-core 1.0 GHz Cortex-A9	Nvidia Tegra 2 T20	2011
Phillips/TLE732	tablet	7.1	104	33,300	Quad-core 1.2 GHz Cortex-A7	RK3126C	2017
Motorola/MotoG6	smartph.	9.0	323	41,040	Octa-core 1.8 GHz Cortex-A53	Qualcomm SDM450 Snapdragon 450	2018
Samsung/A30	smartph.	9.0	674	54,072	Octa-core (2 × 1.8 GHz Cortex-A73 & 6 × 1.6 GHz Cortex-A53)	Exynos 7904	2019
Xiaomi/A2 lite	smartph.	9.0	642	55,440	Octa-core 2.0 GHz Cortex-A53	Qualcomm MSM8953 Snapdragon 625	2018
Xiaomi/Redmi Note 7	smartph.	9.0	914	55,440	Octa-core (4 × 2.2 GHz Kryo 260 Gold & 4 × 1.8 GHz Kryo 260 Silver)	Qualcomm SDM660 Snapdragon 660	2019

Table 2. Communication-related node-level features.

		Send/Receive			
WiFi RSSI	dBm	up to 10 KB Data		more than 10 KB Data	
		Through-Put (KB/ms)	Energy (Joules/KB)	Through-Put (KB/ms)	Energy (Joules/KB)
Excell.	−50	0.09	0.0099	0.4	0.00186
Good	−80	0.08	0.0106	0.25	0.00226
Mean	−85	0.04	0.0133	0.16	0.00333
Poor	−90	0.01	0.0346	0.025	0.01265

Communication-related node-level features, i.e., time and energy consumed in data transferring events, such as job data input/output size and nodes status updates are shown in Table 2. Reference values correspond to a third-party study [28], which performed detailed measurements to characterize data transfer through WiFi interfaces, particularly the impact of received signal strength (RSSI) and data chunks size on time and energy consumption.

Nodes ready to participate in a local, clustered computation form a mobile cluster at the edge, whose computing capabilities derive from the number of aggregated nodes and their features. Cluster-level features considered in experimental scenarios are described in Tables 3 and 4. Specifically, Table 3 shows criteria to derive different types of heterogeneity levels w.r.t. where the instantaneous computing throughput comes from. In short, targeting a defined quality of service by relying on few nodes with high throughput differs in terms of potential points of failures and energy efficiency w.r.t. achieving this with many nodes having lower throughput each. Table 4 outlines criteria to describe communication-related properties of clusters where, for instance, an overall good communication quality—GoodComm—means that a cluster has at least 80% of resource provider nodes with good or excellent RSSI (good_prop + mean_prop + poor_prop = 100% of nodes). In contrast, mean communication quality—MeanComm—suggests that a cluster has at least 60% of resource provider nodes with RSSI of −85 dBm. Finally, Table 5 shows the criteria used to conform cluster instances by combining the computation- and communication-related properties mentioned above. For instance, clusters of type Good2High are instances where nodes providing the fastest instantaneous computing capability relative to other nodes in the cluster also have the best performance in terms of communication throughput. In contrast, the Good2Low category describes cluster instances where the best communication performance is associated with nodes able to provide the slowest instantaneous computing capability. Finally, the Balanced cluster category means that best communication performance is equally associated with nodes with the fastest and the slowest instantaneous computing capabilities.

Table 3. Computing-related cluster-level features.

Heterogeneity	**Rules**
HtL0	100% of instantaneous computing capability provided by nodes of the same model
HtL1	74–76% of instantaneous computing capability provided by strong node models (relative to intracluster nodes)
HtL2	74–76% of instantaneous computing capability provided by weak node models (relative to intracluster nodes)

Table 4. Communication cluster-level features.

Communication Performance	**Rules**
Good Comm	good_prop: above 80% of nodes with Excellent/Good RSSI mean_prop: between 0% and (100% − good_prop) of nodes with Mean RSSI poor_prop: between 0% and 100% − (good_prop + mean_prop) of nodes with Poor RSSI
Mean Comm	good_prop: between 0% and 20% of nodes with Excellent/Good RSSI mean_prop: 100% − (good_prop + bad_prop) of nodes with Mean RSSI poor_prop: between 0% and 20% of nodes with Poor RSSI

Table 5. Mapping of communication and computation cluster-level features.

@Cluster type	Communication/computation nodes assignment criteria
@Good2High	Good RSSI values are assigned firstly, among strong nodes. Remaining RSSI values among remaining nodes
@Good2Low	Good RSSI values are assigned firstly, among weak nodes. Remaining RSSI values among remaining nodes
@Balanced	25% of Mean RSSI values assigned to strong nodes, 25% of Mean RSSI values assigned to weak nodes and remaining RSSI values—good, mean, poor—are randomly assigned among remaining nodes

Job sets were created using the *siminput* package utility of the DewSim toolkit [27]. We defined job bursts that arrive at varying intervals during a thirty minutes time window. Such a window represents a time extension where a vehicle can travel and scan a considerable part of its trajectory. Moreover, within this window the mobile devices of a group of passengers in a transport vehicle (e.g., a bus) can reasonably stay connected to the same shared access point. Intervals represent video or audio recording, i.e., in-bus data capturing periods. It is assumed that the recording system has a limited buffer, which is emptied at a point in time defined by a normal distribution with mean of 12 s and deviation of 500 ms. With every buffer-emptying action, a new jobs burst is created and all captured data, which serves as input for a CPU-intensive program, is transferred to mobile devices that participate in the distributed processing of such data. Jobs are of fixed input size. We created job sets where each job input has 1 MB and 500 KB, while output size varies between 1 and 100 KB. A single job takes 0.45–1.85 s of computing time when it executes on the fastest (Samsung Galaxy SIII) and the slowest (LG L9) device model of an older cluster, respectively. Moreover, this time is considerably less when a job is executed in a cluster with nodes that belong to the group of recently launched, i.e., 90–320 milliseconds when executing on a Xiaomi Redmi Note 7 and a Motorola Moto G6, respectively. For defining time ranges, a pavement crack and pothole detection application implemented for devices with similar performance to those in the experiments of Tedeschi and Benedetto [3] was the reference. Bursts are composed of varying numbers of jobs, depending on the interval's extension. Job requirements in terms of floating-point operations fall within 80.69 to 104.69 MFLOP.

Figure 2 depicts a graphical representation of the computing and data transferring load characteristics of the job sets simulated in dew context scenarios. Frequency bar subplots at the bottom show the data volumes and arrival times of job bursts transferred during a 30 min time window. Subplots at the middle and top show, for the same time window, the MFLOP and job counts of each job burst. For example, when jobs input data was set to 1 MB, approximately 52.78 GB of data were transferred, and derived jobs required 4 775 GigaFLOP to be executed within such a time window.

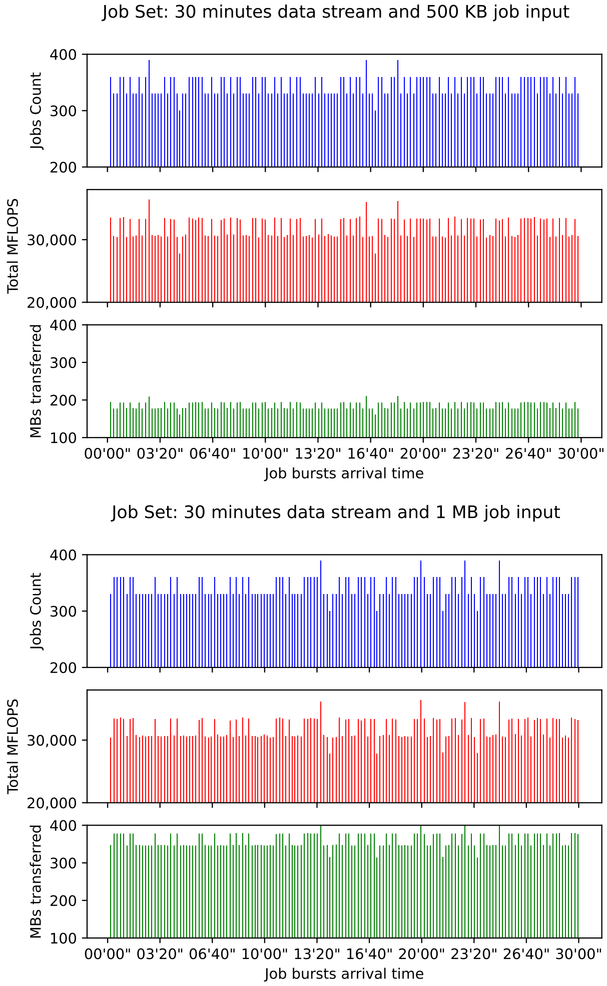


Figure 2. Job set characteristics.

6. Metrics and Experimental Simulation Results

The scheduling heuristics' performance is measured in terms of completed jobs, makespan, and fairness, which are metrics reported in similar distributed computing systems studies [15,16,29,30].

Completed jobs: Providing that mobile device clusters rely on the energy stored in the mobile devices' batteries to execute jobs, scheduling technique A is considered to be more energy efficient than scheduling technique B, if the former completes more jobs than the latter with the same amount of energy. The job completion count finishes when all nodes leave the cluster, in this case, due to running out of energy.

Makespan: Measures the time the distributed system needs to complete the execution of a job set. We normalized these times duration into a 0–1 scale, where the value 1 refers to the heuristic that requires the longest makespan. To calculate the makespan, we compute the difference between the time when the first job arrives and the time when the last job is completed. To calculate the latter when all compared heuristics achieved different numbers of completed jobs, we first compute the maximum number of jobs that all heuristics completed, and use this value as a pivot to obtain the time when the last job is completed.

Fairness: The difference in energy contributed by provider nodes from the time each one joins the cluster to the time each one completes its last assigned job, is quantified via the Jain's fairness index [31]. This index was originally used to measure the bandwidth received by clients of a networking provider but, in our case, much as by Ghasemi-Falavarjani et al. [12], Viswanathan et al. [30], it is used to measure the disparity of energy pulled by the system from provider nodes. The metric complements the performance information given by completed jobs and makespan metrics.

We ran all heuristics on 2304 scenarios with varying nodes, cluster, and job characteristics. We distinguished between *older* and *recent* clusters. Older clusters are conformed by devices with instantaneous computing capability of 300 MegaFLOPS and below, which includes the LG L9, Samsumg Galaxy S3, Acer A100, and Phillips TLE732 device models of Table 1. This is the cluster used in our previous work, see Hirsch et al. [1]. Conversely, recent clusters are conformed by the Motorola MotoG6, Samsumg A30, Xiaomi A2 lite, and Xiaomi RN7 device models with floating-point computing capability of 300 MegaFLOPS and above. Figure 3 depicts the position that each group of scenarios occupies in the heatmap representation used to display the performance values obtained for each heuristic.

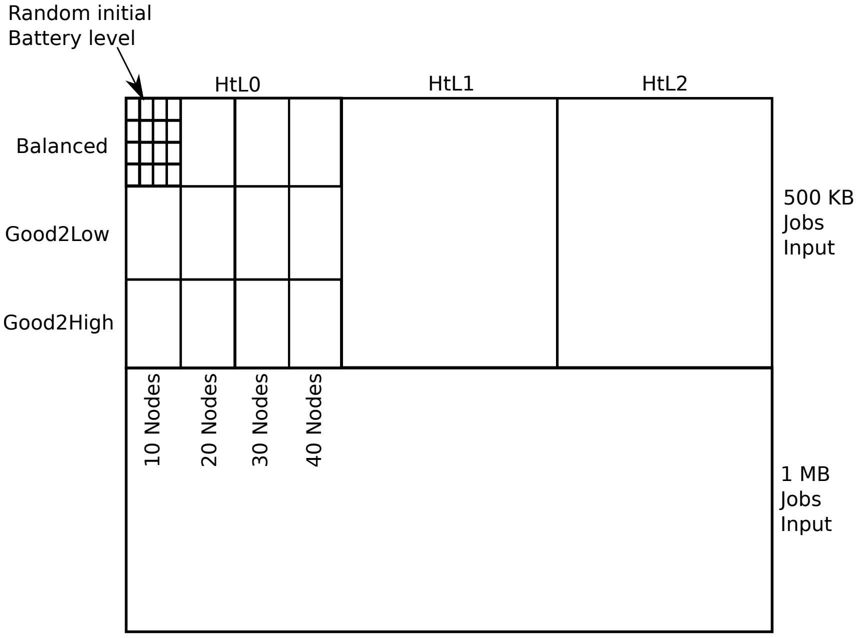


Figure 3. Simulated scenarios: heatmap pixels.

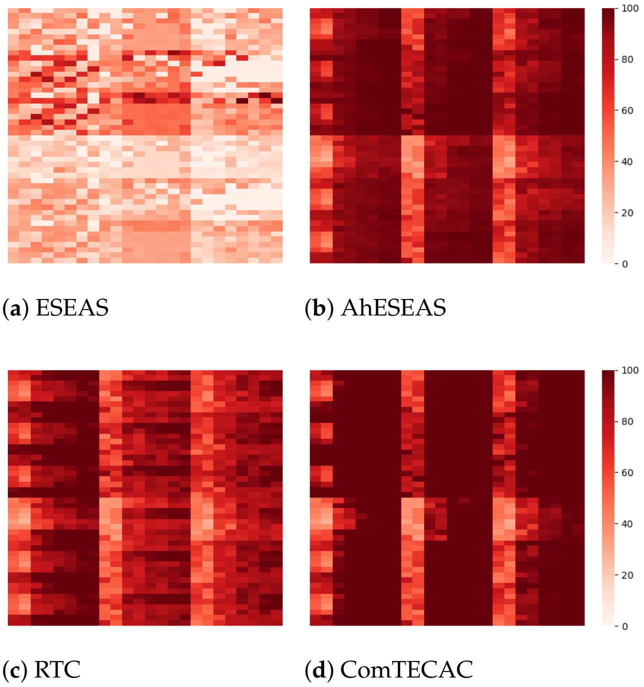
Figure 4 shows the results of each heuristic's completed jobs for older cluster scenarios. The darker the pixel intensity, the better is the performance achieved. Several effects of simulated variables on completed jobs are observed. First, by comparing Figure 4a,b, which show the numbers of completed jobs for AhESEAS and ESEAS, respectively, we see the magnitude of improvement introduced by the AhESEAS denominator component update policy (see Section 4). In the presence of job input above 500 KB and approximately 360 jobs generated every 12 s, which is the injected load in the scenarios, load balancing is better managed by the denominator update policy of the AhESEAS formula than that of ESEAS. AhESEAS exceeds ESEAS's completed jobs in all scenarios. On average, the former was better than the latter by 58.6% with a standard deviation of 18.5%. Such an advantage is maintained in recent clusters scenarios, where AhESEAS (Figure 5b) is better than ESEAS (Figure 5a) by 55.2% completed jobs on average with a standard deviation of 19.1%.

By comparing scenario results presented at the top half and bottom half heat maps, we see the effect of job input size, i.e., completed jobs decreases as job input size increases. Such an effect is more noticeable in load balancing performed with ESEAS and AhESEAS than with the other heuristics. This indicates that job completion is not exclusively determined by how a heuristic weighs node computing capability, but also how job data and nodes' communication-related features are included in the node ranking formula. RTC and ComTECAC include communication-related parameters in their respective node ranking formulas. RTC uses job data input/output size and node RSSI, while ComTECAC employs a function of node RSSI that relates this parameter with network efficiency. For this reason, job input size does not affect RTC and ComTECAC but certainly affects the ESEAS and AhESEAS heuristics.

Particularly, the remaining transfer capacity (RTC) heuristic [32] is instead inspired by the online MCT (minimum completion time) heuristic, which has been extensively studied in traditional computational environments such as grids and computer clusters. RTC immediately assigns the next incoming job to the node whose remaining transfer capacity is the least affected, interpreting it as the estimated capability of a battery-driven device to transfer a volume of data, considering its remaining battery level, energy cost in transferring a fixed data unit, and all jobs data scheduled in the past that waits to be transferred. At the time the remaining transfer capacity of a node is estimated, all future job output data transfers from previous job assignments are also considered.

Other variables with orthogonal effect in the number of completed jobs, i.e., observed for all heuristics, are cluster size and cluster heterogeneity. By scanning heat map results from left to right, it can seen that, for instance, in 10 nodes cluster size scenarios there are fewer completed jobs than in 20, 30, and 40 nodes cluster size scenarios. Moreover, cluster heterogeneity degrades the performance of ESEAS, AhESEAS, and RTC more than the one of the ComTECAC heuristics.

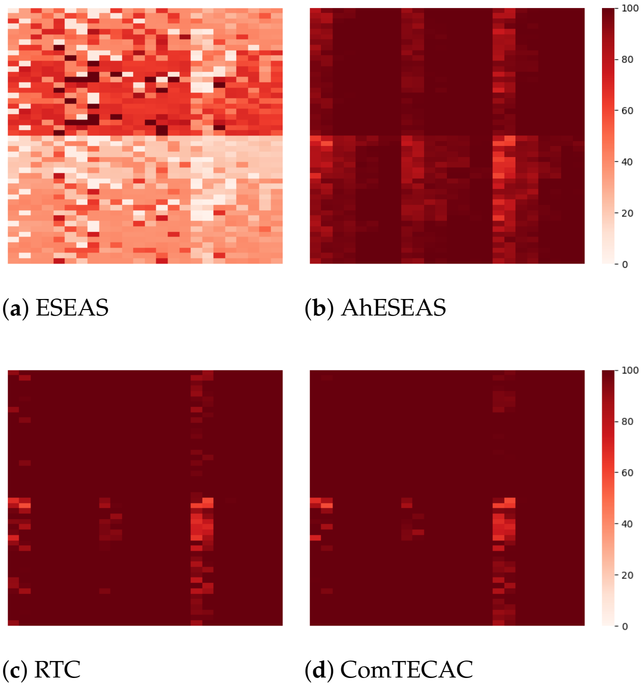
Now, focusing on overall scenario performance, when comparing the relative performance of AhESEAS and RTC, a noticeable effect emerges. In older clusters scenarios (see Figure 4), AhESEAS achieves on average higher job completion rates than RTC. RTC's weakness in assessing the nodes' computing capability seems to cause an unbalanced load with a clear decrease in job completion. However, this weakness seems not to be present in recent clusters scenarios (see Figure 5), where the relative performance between AhESEAS and RTC is inverted. With recent clusters, RTC's unbalanced load is impeded by higher computing capability nodes of all recent clusters.



(**a**) ESEAS (**b**) AhESEAS

(**c**) RTC (**d**) ComTECAC

Figure 4. Older clusters completed jobs (dark is better).



(**a**) ESEAS (**b**) AhESEAS

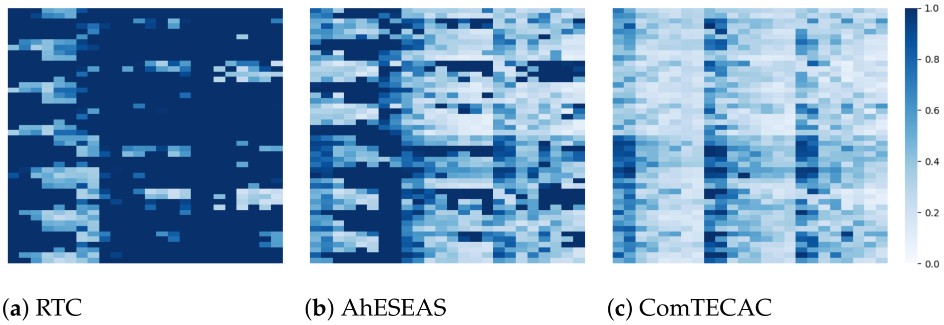
(**c**) RTC (**d**) ComTECAC

Figure 5. Recent clusters completed jobs (dark is better).

To finish this analysis, we can say that by taking this metric as an energy-harvesting indicator, we confirm that ComTECAC is the most energy-efficient scheduling heuristic on average. Having the same amount of energy consumption as all other heuristics, it completes more jobs in older and recent clusters simulated scenarios.

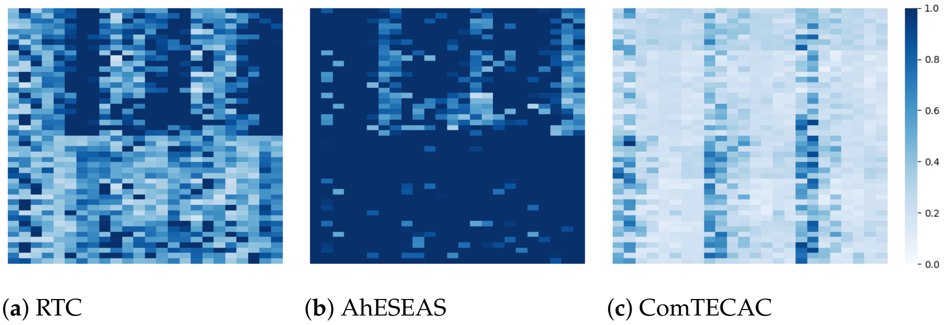
At this point, before starting to discuss other performance metrics, it is worth mentioning that due to the poor performance that ESEAS showed in terms of completed jobs, we leave it aside and focus on AhESEAS, RTC, and ComTECAC.

Figures 6 and 7 depict the makespan of AhESEAS, RTC, and ComTECAC for older clusters and recent clusters scenarios. In this case, the pixels' color intensity inversely relates to high performance, i.e., the lighter the pixel the better the performance. As explained above, where the metrics are presented, to report a comparable makespan measurement between all heuristics, we computed makespan, considering a subset of completed jobs, instead of all jobs, in the following manner. For each scenario, we determined the maximum of completed jobs for all heuristics and used this value as a reference to compute the makespan value of each heuristic. The makespan values reported in Figures 6 and 7 were calculated using completed jobs of the RTC, AhESEAS, and ComTECAC heuristics. The dark blue pixels' predominance in Figure 6a indicates that RTC was the heuristic with the overall longest makespan on average, compared to that obtained by the AhESEAS and ComTECAC heuristics. Moreover, when comparing RTC and AhESEAS, we see that one's makespan mirrors the other, i.e., in scenarios where RTC achieves good makespan, AhESEAS obtained bad makespan values and vice versa.



(**a**) RTC (**b**) AhESEAS (**c**) ComTECAC

Figure 6. Older clusters' makespan (light is better).



(**a**) RTC (**b**) AhESEAS (**c**) ComTECAC

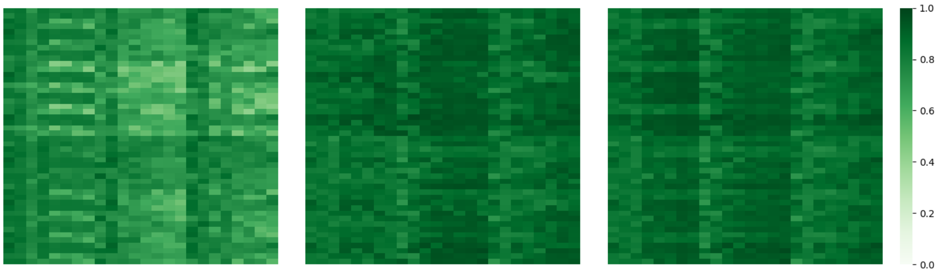
Figure 7. Recent clusters' makespan (light is better).

When analyzing recent clusters scenarios, shown in Figure 7a, we see the effect of device models with more computing capability on the relative performance of RTC and AhESEAS. In these scenarios, RTC achieves, on average, better makespan than AhESEAS. These results are in line with the behavior observed via the completed jobs metric. These results provide evidence on an AhESEAS node ranking formula's weakness, which underestimates the jobs data transferring requirement in the nodes ranking formula. In summary, ranking formulas of both RTC and AhESEAS heuristics have weaknesses. By increasing clusters' instantaneous computing capability with new device models, we see that the RTC weakness can be compensated, while the AhESEAS weakness is increasingly visible. Consistent with this affirmation, when comparing 1 MB input with 500 KB input scenarios presented in Figure 7a,b, we see that, when considering those scenarios taking the lowest time to transfer data, i.e., where jobs have 500 KB input, RTC performed worse than AhESEAS.

Job input size, cluster size, and cluster heterogeneity effects described for the completed jobs metric still apply. To finish this analysis, we may say that the ComTECAC ranking formula combines the strengths from RTC and AhESEAS, and its advantage over the other heuristics in the majority of scenarios is remarkable.

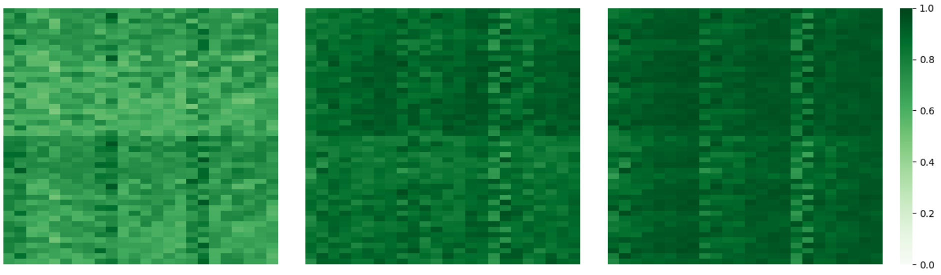
Finally, we report the performance of the heuristics using the fairness metric. Provided that the heuristics target different numbers of completed jobs for the same scenario, to calculate fairness, we followed similar initial steps as with makespan. For each scenario, we first determined the maximum number of completed jobs by all heuristics (except ESEAS), and used it as reference value to compute the fairness score. Once obtained, we searched for each heuristic for the associated time stamp when this number of completed jobs has been reached. The time stamp is another reference, in this case, to get the last battery level reported by each participating node. Then, with such data, and the initial battery level reported by each node, we computed an energy delta, i.e., the node energy contribution, which is interpreted as a sample in the fairness score calculation formula.

According to Figures 8 and 9, RTC's fairness is clearly lower than that of AhESEAS and ComTECAC, on average. In contrast, since the fairness scores of the last two heuristics are quite similar, we formulated the null hypothesis H_0 that the fairness achieved is the same. We tested H_0 with the Wilcoxon test, pairing the fairness values of AhESEAS and ComTECAC for 2304 scenarios. This resulted in a p-value of $p = 1.7 \times 10^{-67}$, which lead us to reject H_0. To conclude this analysis and figure out which of the last two heuristics performed better, we re-computed the fairness metric, this time considering completed jobs only by the AhESEAS and ComTECAC heuristics. The ComTECAC fairness values shown in Figures 10b and 11b are seemingly better than the ones of AhESEAS shown in Figures 10a and 11a. We confirm this by complementing heat maps with a cumulative scenarios density function for older cluster scenarios in Figure 10c and recent cluster scenarios in Figure 11c. In older and recent clusters, the ComTECAC CDF increase is more pronounced than that of AhESEAS as the fairness score increases, i.e., for many scenarios, ComTECAC achieves higher fairness than AhESEAS.



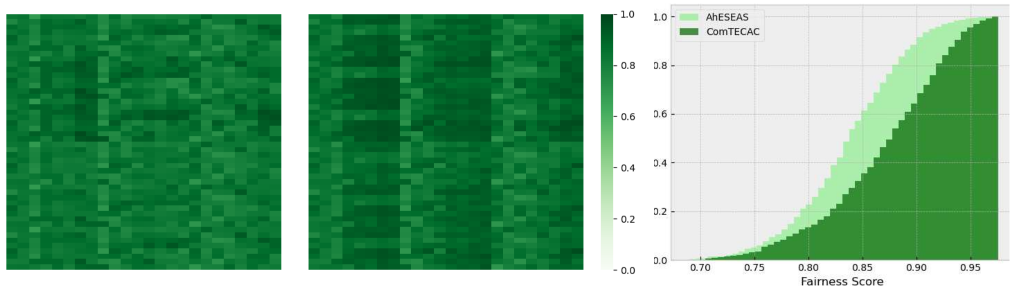
(**a**) RTC (**b**) AhESEAS (**c**) ComTECAC

Figure 8. Older clusters' fairness (dark is better).



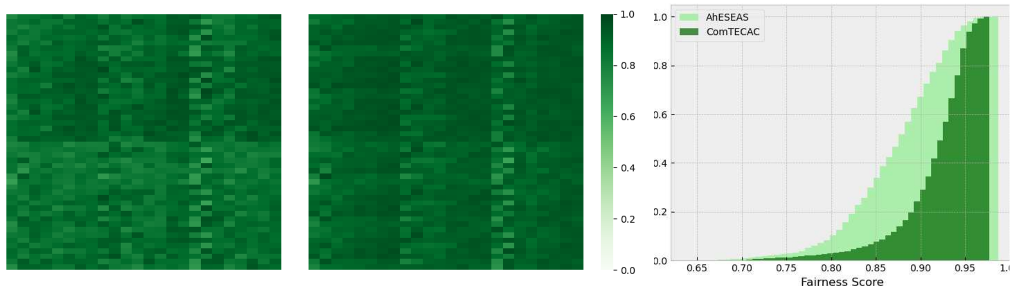
(**a**) RTC (**b**) AhESEAS (**c**) ComTECAC

Figure 9. Recent clusters' fairness (dark is better).



(**a**) AhESEAS (**b**) ComTECAC (**c**) CDF

Figure 10. Recomputed fairness considering AhESEAS and ComTECAC heuristics only (dark is better).



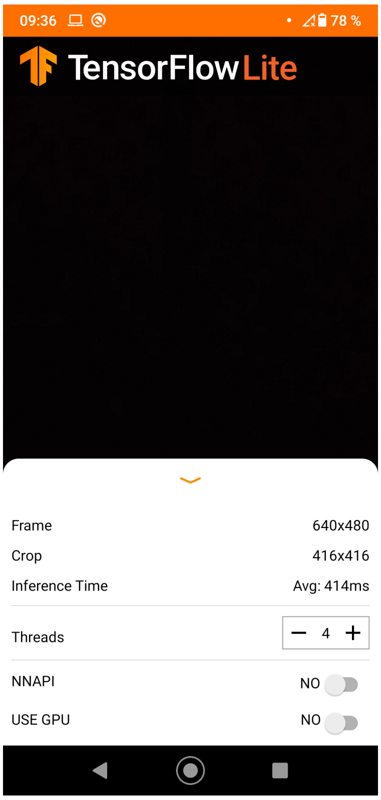
(**a**) AhESEAS (**b**) ComTECAC (**c**) CDF

Figure 11. Recomputed fairness for recent clusters scenarios considering AhESEAS and ComTECAC heuristics only (dark is better).

7. An Experiment of Simulating Video Processing

In spite of the experiments presented above, a question that remained unanswered was whether distributing and executing tasks that process a given stream of edge data using nearby mobile devices is actually beneficial as compared to gather and process the stream individually by smartphones. Assuming that finding a generic answer is difficult, we focus our analysis on a generic class of stream processing mobile application that might be commonplace in smart city dew contexts: per-frame object detection using widespread deep learning architectures over video streams. The goal of the experiment in this section is to simulate a realistic video processing scenario at the edge using a cluster of smartphones. We base this on real benchmark data from mobile devices performing deep learning-based object detection.

As a starting point, we took the Object Detection Android application from the Tensorflow Lite framework (https://www.tensorflow.org/lite/examples/object_detection/overview accessed on 18 August 2021). It includes a YOLO v3 (You Only Look Once) neural network able to detect among 80 different classes. The application operates by reading frames from the smartphone camera and producing a new video with annotated objects, indicating detected class (e.g., "dog") and confidence value. We modified the application to include the newer YOLO v4 for Tensorflow Lite, which improves detection accuracy and speed, see Bochkovskiy et al. [33], and the average frame processing time since application start. A screenshot of the application is shown in Figure 12. The APK is available at https://drive.google.com/file/d/18Q5SLrKtvgsyAb_wA7QZ0TMjIM_jK9hz/view accessed on 18 August 2021.



Frame 640x480

Crop 416x416

Inference Time Avg: 414ms

Threads − 4 +

NNAPI NO

USE GPU NO

Figure 12. Screenshot of the Tensorflow Lite modified object detection application. The dark area displays the camera input and detections..

After that, we took each smartphone belonging to the *recent* cluster used in the previous section, and performed the following procedure:

1. Install the APK.
2. Run a shell script that monitors CPU usage every second, and stores this measure in a log file. This script simply parses the information in the `/proc/stats` system file to produce CPU usage values between 0% and 100%.
3. Run the app by configuring 1 (one) thread.
4. Wait until the inference time (in milliseconds) converges.
5. Close the app, save the log file from step (2) and the inference time from step (4).

Then, we repeated steps (2)–(5) by incrementally using more threads in the device, until finding the lowest average inference time. Lastly, we processed the log files by removing the initial entries representing small values that were due to the app not yet processing camera video.

Table 6 shows the results from the procedure when applied to the *recent* cluster smartphones. To instantiate the DewSim simulator with Tensorflow Lite benchmark information, there was the need to convert job inference time into job operations count. This is because DewSim uses job operations count, node flops, and node CPU usage to simulate jobs completion time. To approximate the job operations count, we applied the following formula, which combines inference time, MaxFlops, and CPU usage percentage:

Table 6. Tensorflow Lite app on recent cluster smartphones: benchmark results.

Smartphone Model	Avg. Inference Time (ms)	Avg. CPU Usage	Threads
Motorola Moto G6	556	50	4
Samsumg A30	375	75	4
Xiaomi A2 lite	420	50	4
Xiaomi RN7	297	75	4

$$jobOps = IT * MF * CPU \tag{3}$$

where IT is the average inference time (in seconds), MF is the MaxDeviceFlops (in multi-thread mode), and CPU is the CPU usage percentage, which is known data derived from the benchmarking tasks described above.

However, jobOps varies when instantiating this formula for different device models. To feed DewSim with a unified jobOps value—in practice the same job whose completion time varies with device computing capabilities—we adjusted the DewSim internal logic to express nodes' computing capability as a linear combination of the fastest node which was used as pivot. In this way, devices computing capability is expressed by MaxFlops multiplied by a coefficient obtained from the following linear equation system:

$$jobOps = IT_{pivot} * MF_{pivot} * CPU_{pivot} \tag{4}$$

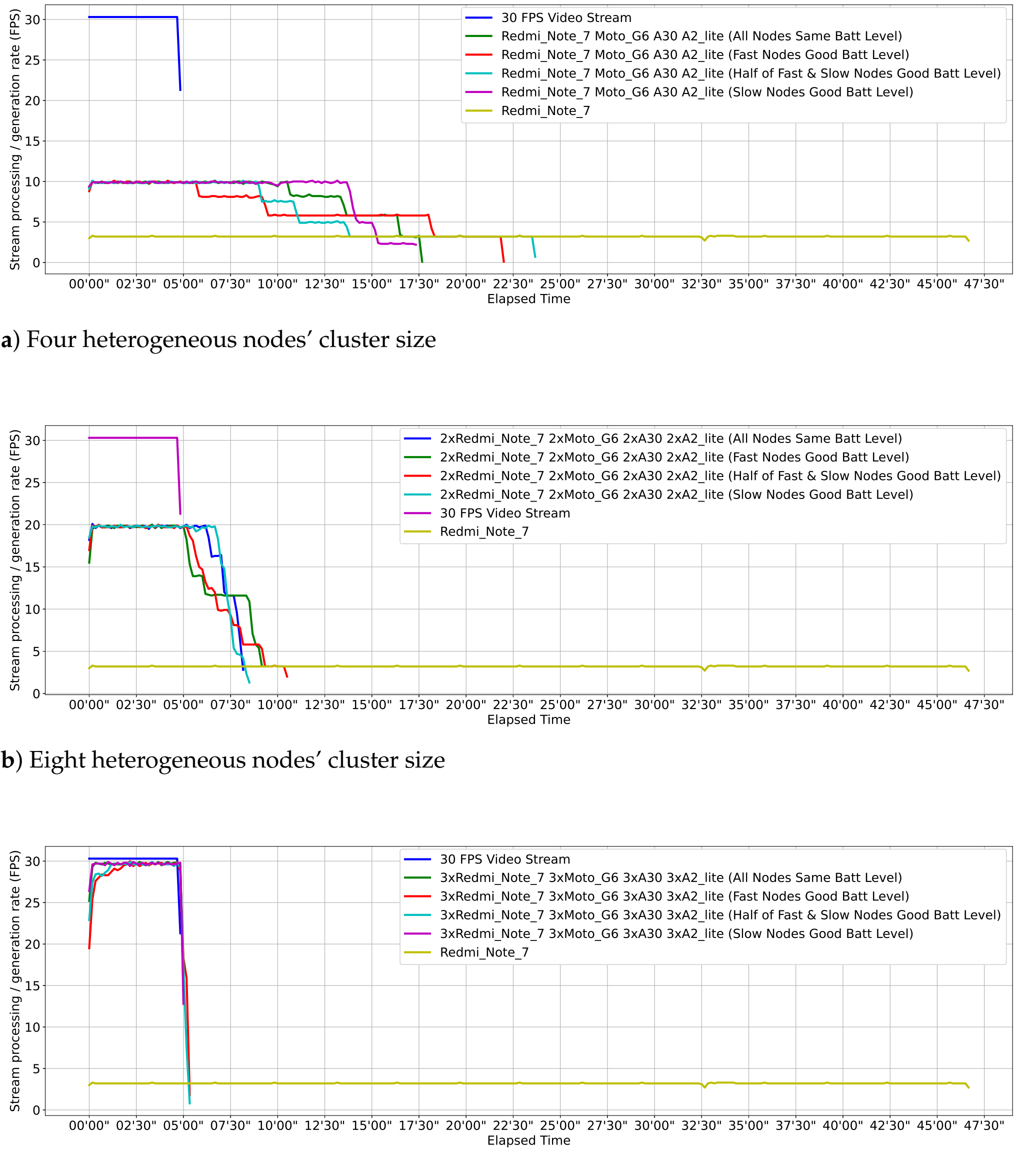
$$jobOps = IT_{dev1} * MF_{dev1} * CPU_{dev1} * Co_{dev1} \tag{5}$$

$$jobOps = IT_{dev2} * MF_{dev2} * CPU_{dev2} * Co_{dev2} \tag{6}$$

$$jobOps = IT_{dev3} * MF_{dev3} * CPU_{dev3} * Co_{dev3} \tag{7}$$

This adjustment allowed DewSim to realistically mimic the inference time and CPU usage as benchmarked for each device model. In addition, the energy consumption trace for the observed CPU usage in devices while processing frames in a certain device model was also configured to DewSim.

Figure 13 shows the processing power and time employed by different smartphone clusters (dew contexts) in processing 9000 jobs, which directly map to individual frames contained in a 5 min 30 FPS video stream. Each job has an input/output size of 24 KB/1 KB, respectively. The input size relates to 414 × 414 images size that the Tensorflow Lite model accepts as input. Besides, 1 KB is the average size of a plain text file that the model can produce with information of objects identified in a frame. Job distribution in all dew contexts was done via ComTECAC since, as previous experiments showed, this scheduling heuristic achieved the best performance in terms of completed jobs, makespan, and fairness metrics as compared to other state-of-the-art heuristics. Heterogeneity in all configured dew contexts is given not only by the combination of nodes with different computing capabilities but also by different initial battery levels because ComTECAC also uses this parameter to rank nodes. We configured scenarios where good battery levels (above 50%) are assigned either to fast—FastNodesWGoodBattLevel—or to slow—SlowNodesWGoodBattLevel—nodes, or they are equally distributed between fast and slow nodes—HalfFastAndSlowNodesWGoodBattLevel. Another scenario considered all nodes having the same battery level. Figure 13a reveals that dew contexts with four nodes can reduce makespan by more than half as compared to what a single instance of our fastest benchmarked smartphone is able to achieve, which means collaborative smartphone-powered edge computing for these kinds of applications is very useful. Figure 13b,c show that makespan can be considerably reduced as more nodes are present in the dew context. Particularly in a dew context with 12 nodes, see Figure 13c, near real-time stream processing is observed, where the job processing rate is scarcely a few seconds behind the job generation rate.



(**a**) Four heterogeneous nodes' cluster size

(**b**) Eight heterogeneous nodes' cluster size

(**c**) Twelve heterogeneous nodes' cluster size

Figure 13. Object detection using the ComTECAC scheduling heuristic over a 5-min, 30 FPS video stream.

8. Discussion

8.1. Summary of Results

From the extensive experiments performed using battery traces and performance parameters from real devices, we arrived at several conclusions about all presented heuristics. First of all, by comparing ESEAS and AhESEAS completed jobs, the performance improvement achieved when changing the denominator update policy is remarkable, i.e., the logic followed by AhESEAS of updating the queued jobs component as soon as a node is selected for executing the next job. This holds when the job input size is at least 500 KB w.r.t. updating the denominator component when the node completely receives the whole job input which is the logic to update the denominator of ESEAS. With this change, between 55.2 and 58.6% more jobs are completed on average.

In older clusters, AhESEAS completed 8.2% and 4.3% more jobs on average than the RTC heuristic in 500 KB and 1 MB data input scenarios, respectively. In contrast, for recent cluster scenarios, RTC outperformed AhESEAS by 1.46% and 3.4% of completed jobs on average in 500 KB and 1 MB data input scenarios, respectively. Besides, we see a direct relationship between completed jobs and makespan, meaning that the heuristic which beats the other, either AhESEAS or RTC, in jobs completion also achieved the relatively least makespan. Moreover, by comparing the fairness of these two heuristics in older and recent clusters scenarios, it yields that AhESEAS always behaves better than RTC.

In summary, AhESEAS and RTC show complementary behavior. Simulated scenarios where one of these achieves high performance the other targets low performance and vice-versa. Both heuristics would be needed to achieve high performance in a wide variety of scenarios.

On the contrary, ComTECAC performance is stable and high in all scenarios. For ranking nodes, ComTECAC considers communication and computing capabilities, as well as energy contribution parameters. The combination of all these allows ComTECAC to complete slightly more jobs than the second-best heuristic, between 0.2% and 3%. At the same time, ComTECAC targets considerably less makespan than its competitors. It achieves a speedup around 1.69 and 2.74 w.r.t. the second-fastest heuristic. ComTECAC is also the fairest heuristic for load balancing. In older clusters, the first 50% (median), 80%, and 90% distribution samples, present fairness values of 0.88, 0.92, and 0.94, respectively, while the second fairest heuristic for these cutting points results in fairness values of 0.84, 0.86, and 0.87, respectively. In recent clusters, the same analysis is even more in favor to ComTECAC, whose fairness values are 0.92, 0.94, and 0.95 vs. 0.88, 0.92, and 0.94 achieved by AhESEAS.

Finally, we instantiated our simulations with a practical case in which mobile devices recognize objects from video streams using deep learning models. For running these simulations, we explained the adaptations that were performed to DewSim, which represent a starting point for incorporating other benchmarked models. We conclude that close to real object detection is viable with a reasonable amount of dedicated mobile devices.

8.2. Practical Challenges

In the course of doing this work, we have identified some challenges towards exploiting the proposed scheduling heuristics in particular, and the collaborative computing scheme as a whole in general, in order to build smart cities. First, our collaborative computing scheme lacks mechanisms for promoting citizens' participation, accounting for computing contribution, and preventing fraud in reporting results. Incentive mechanisms proposed for collaborative sensing are not applicable for resource-intensive tasks and, in fact, some research has been done on this topic [19]. Some of the questions that remain unanswered regarding these challenges are: Is the job completion event a good checkpoint for giving credits to resource provider nodes? What are the consequences of giving a fixed amount of credits upon a job completion irrespective of the time and energy employed by a device? How many results of the same job would be necessary to collect in order to prevent fraud in reporting job results? In short, apart from luring citizens into using our scheme, it is necessary to reward good users and to identify malicious ones.

Another evident challenge is that a middleware implementing basic software services for supporting the above collaborative scheme is necessary. Besides, wide mobile OS and hardware support in the associated client-side app(s) must be ensured, which is known to be a difficult problem from a software engineering standpoint. To bridge this gap, we are working on a middleware-prototype for validating our findings. We already integrated libraries that use traditional machine vision and deep learning object recognition and tracking algorithms into our device profiling platform, which is a satellite project of the DewSim toolkit. This is necessary to validate our load balancing heuristics with real object recognition algorithms, which in turn complement our battery-trace capturing method that currently exercises CPU floating-point capabilities through a generic yet synthetic algorithm. This integration also allows for deriving new heuristics to refine the exploitation of mobile devices by profiling specialized accelerator hardware such as GPUs and NPUs, which are suited for running complex AI models [22]. The first steps have been already taken using our Tensorflow-based Android benchmarking application, but we need to study how to generate GPU-aware energy traces, how to properly profile GPU hardware capabilities into indicators, and how to exploit these traces and indicators through specialized AI job schedulers.

Lastly, another crucial challenge is how to ensure proper QoS (Quality of Service) in our computing scheme considering that, in smart city applications, there is high uncertainty regarding the computing power—in terms of the number of nodes and their capabilities—available at any given moment to process jobs. The reason is that devices might join and leave a dew context dynamically. In this line, several questions arise: is it possible to predict within acceptable error margins how much time an individual device will stay connected in a dew context? For example, in certain smart city dew contexts (e.g., public transport) connectivity profiles for each contributing user could be derived by exploiting users' travel/mobility patterns. Then, long-lasting devices could be given more jobs to execute. Another question is how to regulate job creation in a dew context, so that no useless computations are performed and, hence, higher QoS (in terms of, e.g., energy spent and response time) is delivered to smart city applications? As an extreme example, the number of jobs created from a continuous video stream in applications involving public transport should not be constant, but depend on the current speed of the bus.

9. Conclusions and Future Work

In this paper, we present a performance evaluation of practical job scheduling heuristics for stream processing in dew computing contexts. ESEAS and RTC are heuristics from previous work, while AhESEAS and ComTECAC are new. We measured the performance using the completed jobs metric, which quantifies how efficiently the available energy in the system is utilized: the makespan metric, which indicates how fast the system completes job arrivals, and the fairness metric, which measures the energy contribution differences among participating devices. The new heuristics, specially ComTECAC, had superior performance. These results present a step towards materializing the concept of dew computing using mobile devices from regular users. It will be applied to real-world situations where online data gathering and processing at the edge are important, such as smart city applications.

Despite our focus on heuristics to orchestrate a self-supported distributed computing architecture that leverages idle resources from clusters of battery-driven nodes, to extend the architecture's applicability, new efforts will follow. We will study complementing battery-driven resource provider nodes with non-battery-driven fog nodes, e.g., single-board computers, in a similar way to other work that studied the synergy among different distributed computing layers, e.g., fog nodes and cloud providers [34].

With the goal of making our experimental methodology easier to adopt and use, another aspect involves the development of adequate soft/hard support to simplify the process of battery trace creation to feed DewSim. For instance, Hirsch et al. [35] have recently proposed a prototype platform based on commodity IoT hardware such as smart Wifi switches and Arduino boards to automate this process as much as possible. The prototype, called Motrol, supports batch benchmarking of up to four smartphones simultaneously, and provides a simple, JSON-based configuration language to specify various benchmark conditions such as CPU level, required battery state/levels, etc. A more recent prototype called Motrol 2.0, see Mateos et al. [36], extends the previous support with extra charging sources for attached smartphones (fast, AC charging and slow, USB charging) and a web-based GUI written in Angular to launch and monitor benchmarks. Further work along these lines is already on its way.

Author Contributions: Conceptualization, M.H., C.M., A.Z. T.A.M., T.-M.G., and H.K.; methodology, M.H. and C.M.; software, M.H. and C.M.; validation, M.H.; writing—original draft preparation, M.H.; writing—review and editing, C.M., A.Z., T.A.M., T.-M.G., and H.K.; visualization, M.H.; supervision, C.M. and A.Z.; funding acquisition, C.M., A.Z., and T.A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by CONICET grant number 11220170100490CO and ANPCyT grant number PICT-2018-03323.

Data Availability Statement: To obtain data and software supporting the reported results feel free to email the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hirsch, M.; Mateos, C.; Zunino, A.; Majchrzak, T.A.; Grønli, T.M.; Kaindl, H. A Simulation-based Performance Evaluation of Heuristics for Dew Computing. In Proceedings of the 54th Hawaii International Conference on System Sciences, Hawaii, 5–8 January 2021; p. 7207.
2. Hintze, D.; Findling, R.D.; Scholz, S.; Mayrhofer, R. Mobile device usage characteristics: The effect of context and form factor on locked and unlocked usage. In Proceedings of the 12th International Conference on Advances in Mobile Computing and Multimedia (MoMM), Kaohsiung, Taiwan, 8–10 December 2014; pp. 105–114.
3. Tedeschi, A.; Benedetto, F. A real-time automatic pavement crack and pothole recognition system for mobile Android-based devices. *Adv. Eng. Inform.* **2017**, *32*, 11–25. [CrossRef]
4. Restuccia, F.; Das, S.K.; Payton, J. Incentive mechanisms for participatory sensing: Survey and research challenges. *ACM TOSN* **2016**, *12*, 1–40. [CrossRef]
5. Gao, H.; Liu, C.H.; Wang, W.; Zhao, J.; Song, Z.; Su, X.; Crowcroft, J.; Leung, K.K. A survey of incentive mechanisms for participatory sensing. *IEEE Commun. Surv* **2015**, *17*, 918–943. [CrossRef]
6. Moustaka, V.; Vakali, A.; Anthopoulos, L.G. A systematic review for smart city data analytics. *ACM CSUR* **2018**, *51*, 1–41. [CrossRef]
7. Dobre, C.; Xhafa, F. Intelligent services for big data science. *Future Gener. Comput. Syst.* **2014**, *37*, 267–281. [CrossRef]
8. Guo, B.; Wang, Z.; Yu, Z.; Wang, Y.; Yen, N.Y.; Huang, R.; Zhou, X. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM CSUR* **2015**, *48*, 1–31. [CrossRef]
9. Ali, B.; Pasha, M.A.; ul Islam, S.; Song, H.; Buyya, R. A Volunteer Supported Fog Computing Environment for Delay-Sensitive IoT Applications. *IEEE IoT J.* **2020**, *8*, 3822–3830. [CrossRef]
10. Olaniyan, R.; Fadahunsi, O.; Maheswaran, M.; Zhani, M.F. Opportunistic Edge Computing: Concepts, opportunities and research challenges. *Future Gener. Comp. Syst.* **2018**, *89*, 633–645. [CrossRef]
11. Chen, X.; Pu, L.; Gao, L.; Wu, W.; Wu, D. Exploiting massive D2D collaboration for energy-efficient mobile edge computing. *IEEE Wirel. Commun.* **2017**, *24*, 64–71. [CrossRef]
12. Ghasemi-Falavarjani, S.; Nematbakhsh, M.; Ghahfarokhi, B.S. Context-aware multi-objective resource allocation in mobile cloud. *Comput. Electr. Eng.* **2015**, *44*, 218–240. [CrossRef]
13. Wei, X.; Fan, J.; Lu, Z.; Ding, K. Application scheduling in mobile cloud computing with load balancing. *J. Appl. Math.* **2013**, *2013*, 409539. [CrossRef]
14. Rieger, C.; Majchrzak, T.A. A Taxonomy for App-Enabled Devices: Mastering the Mobile Device Jungle. In *LNBIP*; Revised Selected Papers WEBIST 2017; Springer: Cham, Switzerland, 2018; Volume 322, pp. 202–220.
15. Hirsch, M.; Rodríguez, J.M.; Mateos, C.; Zunino, A. A Two-Phase Energy-Aware Scheduling Approach for CPU-Intensive Jobs in Mobile Grids. *J. Grid Comput.* **2017**, *15*, 55–80. [CrossRef]
16. Yaqoob, I.; Ahmed, E.; Gani, A.; Mokhtar, S.; Imran, M. Heterogeneity-aware task allocation in mobile ad hoc cloud. *IEEE Access* **2017**, *5*, 1779–1795. [CrossRef]
17. Marana, P.; Eden, C.; Eriksson, H.; Grimes, C.; Hernantes, J.; Howick, S.; Labaka, L.; Latinos, V.; Lindner, R.; Majchrzak, T.A.; et al. Towards a resilience management guideline—Cities as a starting point for societal resilience. *Sustain. Cities Soc.* **2019**, *48*, 101531. [CrossRef]
18. Mahmud, R.; Ramamohanarao, K.; Buyya, R. Application Management in Fog Computing Environments: A Taxonomy, Review and Future Directions. *arXiv* **2020**, arXiv:2005.10460.
19. Hirsch, M.; Mateos, C.; Zunino, A. Augmenting computing capabilities at the edge by jointly exploiting mobile devices: A survey. *Future Gener. Comput. Syst.* **2018**, *88*, 644–662. [CrossRef]
20. Hirsch, M.; Rodríguez, J.M.; Zunino, A.; Mateos, C. Battery-aware centralized schedulers for CPU-bound jobs in mobile Grids. *Pervasive Mob. Comput.* **2016**, *29*, 73–94. [CrossRef]
21. Furthmüller, J.; Waldhorst, O.P. Energy-aware resource sharing with mobile devices. *Comput. Netw.* **2012**, *56*, 1920–1934. [CrossRef]
22. Ignatov, A.; Timofte, R.; Kulik, A.; Yang, S.; Wang, K.; Baum, F.; Wu, M.; Xu, L.; Van Gool, L. Ai benchmark: All about deep learning on smartphones in 2019. In Proceedings of the 2019 IEEE/CVF ICCVW, Seoul, Korea, 27–28 October 2019; pp. 3617–3635.
23. Silva, F.A.; Zaicaner, G.; Quesado, E.; Dornelas, M.; Silva, B.; Maciel, P. Benchmark applications used in mobile cloud computing research: A systematic mapping study. *J. Supercomput.* **2016**, *72*, 1431–1452. [CrossRef]
24. Nah, J.H.; Suh, Y.; Lim, Y. L-Bench: An Android benchmark set for low-power mobile GPUs. *Comput. Graph.* **2016**, *61*, 40–49. [CrossRef]

25. Luo, C.; Zhang, F.; Huang, C.; Xiong, X.; Chen, J.; Wang, L.; Gao, W.; Ye, H.; Wu, T.; Zhou, R.; et al. AIoT Bench: Towards comprehensive Benchmarking Mobile and Embedded Device Intelligence. In *Benchmarking, Measuring, and Optimizing 2019*; Springer International Publishing: Cham, Switzerland, 201 ; pp. 31–35.
26. Wilhelm, R.; Engblom, J.; Ermedahl, A.; Holsti, N.; Thesing, S.; Whalley, D.; Bernat, G.; Ferdinand, C.; Heckmann, R.; Mitra, T.; et al. The worst-case execution-time problem—Overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst. (TECS)* **2008**, *7*, 1–53. [CrossRef]
27. Hirsch, M.; Mateos, C.; Rodriguez, J.M.; Alejandro, Z. DewSim: A trace-driven toolkit for simulating mobile device clusters in Dew computing environments. *Softw. Pract. Exp.* **202** , *50*, 688–718. [CrossRef]
28. Ding, N.; Wagner, D.; Chen, X.; Pathak, A.; Hu, Y.C.; Rice, A. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. *ACM SIGMETRICS Perform. Eval. Rev.* **2013**, *41*, 29–40. [CrossRef]
29. Pandey, P.; Pompili, D. Exploiting the untapped potential of mobile distributed computing via approximation. *Pervasive Mob. Comput.* **2017**, *38*, 381–395. [CrossRef]
30. Viswanathan, H.; Lee, E.K.; Rodero, I.; Pompili, D. Uncertainty-aware autonomic resource provisioning for mobile cloud computing. *IEEE TPDS* **2014**, *26*, 2363–2372. [CrossRef]
31. Jain, R.K.; Chiu, D.M.W.; Hawe, W.R. *A Quantitative Measure of Fairness and Discrimination*; Eastern Research Laboratory, Digital Equipment Corporation: Hudson, MA, USA, 198 .
32. Hirsch, M.; Mateos, C.; Rodriguez, J.M.; Zunino, A.; Garı, Y.; Monge, D.A. A performance comparison of data-aware heuristics for scheduling jobs in mobile Grids. In Proceedings of the 2017 XLIII CLEI, Cordoba, Argentina, 4–8 September 2017; pp. 1–8. [CrossRef]
33. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
34. Elgamal, T.; Sandur, A.; Nguyen, P.; Nahrstedt, K.; Agha, G. Droplet: Distributed operator placement for iot applications spanning edge and cloud resources. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 1–8.
35. Hirsch, M.; Mateos, C.; Zunino, A.; Toloza, J. A platform for automating battery-driven batch benchmarking and profiling of Android-based mobile devices. *Simul. Model. Pract. Theory* **2021**, *109*, 102266. [CrossRef]
36. Mateos, C.; Hirsch, M.; Toloza, J.; Zunino, A. Motrol 2.0: A Dew-oriented hardware/software platform for batch-benchmarking smartphones. In Proceedings of the IEEE 6th IEEE International Workshop on Dew Computing (DewCom 2021)—COMPSAC, Huizhou, China, 12–16 July 202 ; pp. 1–6.

electronics

MDPI

Article

Spectral Classification Based on Deep Learning Algorithms

Laixiang Xu [1], Jun Xie [2], Fuhong Cai [1] and Jingjin Wu [3,*]

[1] School of Information and Communication Engineering and School of Biomedical Engineering, Hainan University, Haikou 570228, China; xulaixiang@hainanu.edu.cn (L.X.); caifuhong@hainanu.edu.cn (F.C.)
[2] School of Information Engineering, Guangzhou Panyu Polytechnic, Guangzhou 511483, China; xiejun@gzpyp.edu.cn
[3] Mechanical and Electrical Engineering College, Hainan University, Haikou 570228, China
* Correspondence: jingjin.wu@hainanu.edu.cn

Abstract: Convolutional neural networks (CNN) can achieve accurate image classification, indicating the current best performance of deep learning algorithms. However, the complexity of spectral data limits the performance of many CNN models. Due to the potential redundancy and noise of the spectral data, the standard CNN model is usually unable to perform correct spectral classification. Furthermore, deeper CNN architectures also face some difficulties when other network layers are added, which hinders the network convergence and produces low classification accuracy. To alleviate these problems, we proposed a new CNN architecture specially designed for 2D spectral data. Firstly, we collected the reflectance spectra of five samples using a portable optical fiber spectrometer and converted them into 2D matrix data to adapt to the deep learning algorithms' feature extraction. Secondly, the number of convolutional layers and pooling layers were adjusted according to the characteristics of the spectral data to enhance the feature extraction ability. Finally, the discard rate selection principle of the dropout layer was determined by visual analysis to improve the classification accuracy. Experimental results demonstrate our CNN system, which has advantages over the traditional AlexNet, Unet, and support vector machine (SVM)-based approaches in many aspects, such as easy implementation, short time, higher accuracy, and strong robustness.

Keywords: spectral classification; convolutional neural network; portable optical fiber spectrometers

Citation: Xu, L.; Xie, J.; Cai, F.; Wu, J. Spectral Classification Based on Deep Learning Algorithms. *Electronics* **2021**, *10*, 1892. https://doi.org/10.3390/electronics10161892

Academic Editor: Nurul I. Sarkar

Received: 26 June 2021
Accepted: 27 July 2021
Published: 6 August 2021

1. Introduction

The emergence of the Internet of Things (IoT) has promoted the rise of edge computing. In IoT applications, data processing, analysis, and storage are increasingly occurring at the edge of the network, close to where users and devices need to access information, which makes edge computing an important development direction.

There were already applications of deep learning in IoT, for example, deep learning predicted household electricity consumption based on data collected by smart meters [1]; and a load balancing scheme based on the deep learning of the IoT was introduced [2]. Through the analysis of a large amount of user data, the network load and processing configuration are measured, and the deep belief network method is adopted to achieve efficient load balancing in the IoT. In [3], an IoT data analysis method based on deep learning algorithms and Apache Spark was proposed. The inference phase was executed on mobile devices, while Apache Spark was deployed in the cloud server to support data training. This two-tier design was very similar to edge computing, which showed that processing tasks can be offloaded from the cloud. In [4], it is proven that due to the limited network performance of data transmission, the centralized cloud computing structure can no longer process and analyze the large amount of data collected from IoT devices. In [5], the authors indicated that edge computing can offload computing tasks from the centralized cloud to the edge near the IoT devices, and the data transmitted during the preprocessing process will be greatly reduced. This operation made edge computing another key technology for IoT services.

The data generated by IoT sensor terminal devices need to use deep learning for real-time analysis or for training deep learning models. However, deep learning [6] inference and training require a lot of computing resources to run quickly. Edge computing is a viable method, as it stores a large number of computing nodes at the terminal location to meet the requirements of high computation and low latency of edge devices. It shows good performance in privacy, bandwidth efficiency, and scalability. Edge computing has been applied to deep learning with different aims: fabric defect detection [7], falling detection in smart cities, street garbage detection and classification [8], multi-task partial computation offloading and network flow scheduling [9], road accidents detection [10], and real-time video optimization [11].

Red, green, and blue (RGB) cameras mainly use red, green, and blue light to classify objects. From the point of view of the spectrum, there are three bands that are only in the visible band. The number of spectral bands we use has 1024, including some near-infrared light bands, which is more helpful for accurate classifications. For instance, the red-edge effect of the infrared band inside can distinguish real leaves from plastic leaves in vegetation detection. Therefore, we believe that increasing the number of spectral channels is more conducive to the application expansion of the system in the future.

The optical fiber spectrometer has been reported for applications in photo-luminescence properties detection [12], the smartphone spectral self-calibration [13], and phosphor thermometry [14]. At present, some imaging spectrometers can obtain spatial images, depth information, and spectral data of objects simultaneously [15]. However, most of the data processed by deep learning algorithms are image data information obtained by these imaging spectrometers. Deep learning algorithms are rarely used to process the reflection spectrum data obtained by the optical fiber spectrometer.

In hyperspectral remote sensing, deep learning algorithms have been widely applied to hyperspectral imaging classification processing tasks. For example, in [16], a spatial-spectral feature extraction framework for robust hyperspectral images classification was proposed to combine a 3D convolutional neural network. Testing overall classification accuracies was 4.23% higher than SVM on Pavia data sets and Pines data sets. In [17], a new recurrent neural network architecture was designed and the testing accuracy was 11.52% higher than that of a long short-term memory network, which is on the HSI data sets Pavia and Salinas. A new recursive neural network structure was designed in [18], and an approach based on a deep belief network was introduced for hyperspectral images classification. Compared with SVM, overall classification accuracies of Salinas, Pines, and Pavia data sets increased by 3.17%. Currently, hyperspectral imagers are mainly used to detect objects [19]. Although the optical fiber spectrometer is easy to carry and collect the spectra of objects, it cannot realize the imaging detection research of objects. However, deep learning algorithms are data-driven and can realize end-to-end feature processing. If we process spectral data by combining deep learning algorithms with fiber optic spectrometers, it can further perform the detection and research of objects.

However, most spectrometers need to be connected to the host computer via USB, which cannot be carried easily. In this work, we designed and manufactured a portable optical fiber spectrometer. After testing the stability of the system, we collected the reflectance spectra of five fruit samples and proposed a depth called the convolutional neural network learning method, which performs spectral classification. The accuracy of this method is 94.78%. We boldly combined the deep learning algorithm and the system to complete the accurate classification of spectral data. Using this portable spectrometer, we use edge computing technology to increase the speed of deep learning while processing spectral data.

We have designed a portable spectrometer with a screen; the system can get rid of the heavy host computer and realize real-time detection of fruit quality.

Our portable spectrometer is shown in Figure 1a. The spectrometer has a 5-inch touch screen, and users can view the visualized sample spectrum information on the spectrometer in real-time. As shown in Figure 1b, the system is equipped with an Ocean Optics-USB2000+ spectrometer (Ocean Optics, Delray Beach, FL, USA), to ensure that the system has a spectral resolution of not less than 5nm. As shown in Figure 1d, the system configuration of our spectrometer is a GOLE1 microcomputer, Ocean Optics USB2000+ spectrometer, and a high-precision packaging fixture. The optical fiber can be connected with the spectrometer through a fine-pitch mechanical thread. The overall structure is treated with electroplating black paint, to effectively avoid external light interference and greatly improve the signal-to-noise ratio of spectral information. When using our spectrometer to detect samples, we connect one end of the optical fiber to the spectrometer through a mechanical thread and hold the other end close to the test sample. The reflected light from the sample surface enters the spectrometer through the optical fiber, and the spectrometer converts the collected optical information into electrical signals and transmits it to the microcomputer through the USB cable. The microcomputer visualizes the signal on the screen. Users can view, store, and transmit spectral information through the system's touch screen, innovating the functions of traditional spectrometers that need to be operated by the keyboard and mouse of the host computer.

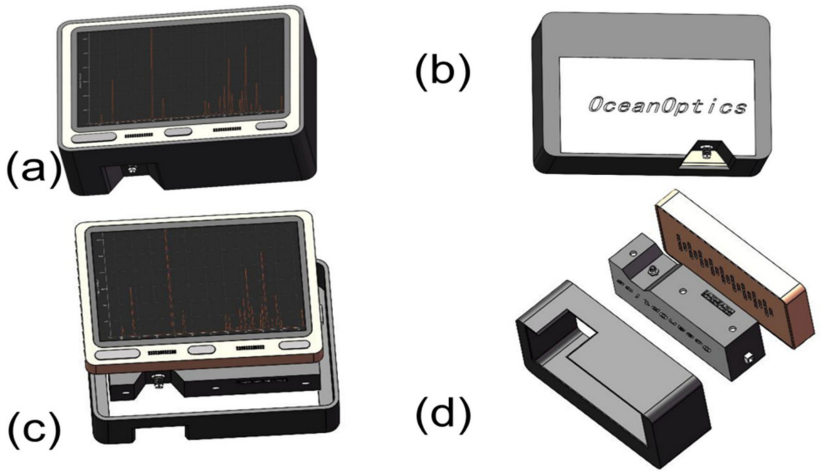


Figure 1. (**a**) The GOLE1 mini-computer used in the experiment. (**b**) The Ocean Optics USB2000+ spectrometer that was used in the experiment. (**c**) Front view of the assembly of the mini-computer and the spectrometer; (**d**) Integration of the spectrometer and the mini-computer through a self-designed housing.

To ensure the accuracy of the system data acquisition and to demonstrate the system's ability to detect various data, the team tested the stability of the entire hyperspectral imaging system. First, we adjust the imaging to the same state as the data acquisition, then we collect 10 sets of solar light spectral data by the spectrograph at 10 s intervals; then, we adjust the display of the system to red, green, and blue colors in turn, and repeat the above steps to obtain the corresponding data. Finally, we input 40 groups of data collected by the above methods into Matlab and then use two different processing methods to demonstrate the stability of the whole hyperspectral imaging system.

The first method is to extract one data point of the same wavelength from all 10 groups of data of the same kind, and arrange the data points in order and draw them into the pictures as shown below.

As shown in Figure 2, we can see clearly that the intensity fluctuation of the same 10 groups of data at the same wavelength is very small. This shows that the error of data acquisition of the same object is very small in a short time, which proves that the system has high accuracy.

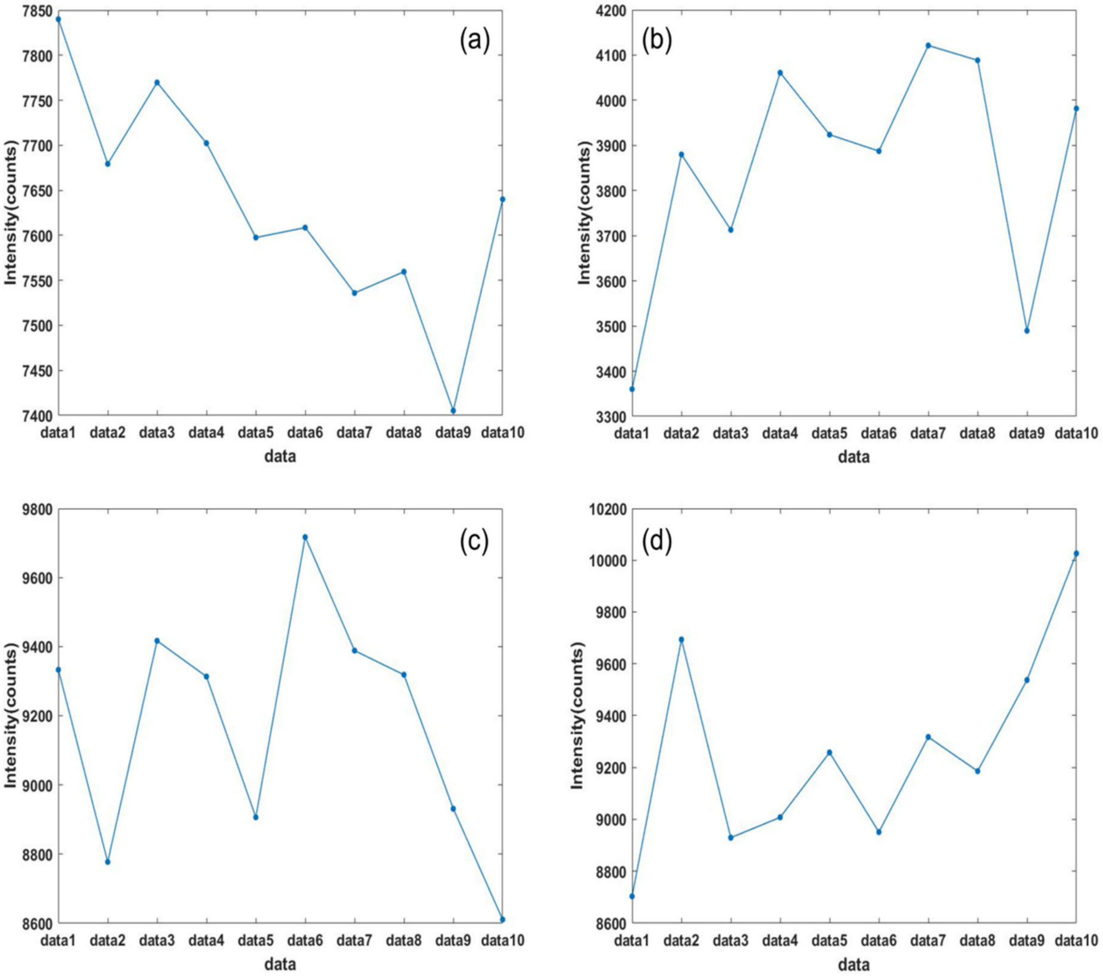


Figure 2. (**a**) Point data of the same wavelength of 10 sets of sunlight spectrum data collected every 10 s. (**b**) Point data of the same wavelength of 10 sets of screen green light spectrum data collected every 10 s. (**c**) Point data of the same wavelength of 10 sets of screen blue light spectrum data collected every 10 s. (**d**) Point data of the same wavelength of 10 sets of screen red light spectrum data collected every 10 s.

In the second method, we plot the whole spectrum of the same 10 groups of data on one graph and distinguish them by different colors. As shown in Figure 3, we can clearly see two points: one is that the spectral images of 10 groups of similar data almost coincide; the other is that the spectra of sunlight, blue light, green light, and red light shown in the figure are very classic and do not violate the laws of nature. The above phenomenon shows that the measurement accuracy of the hyperspectral imaging system is high, and the detection ability of each band light is excellent, and the whole system has good stability.

We discussed the edge computing technology under IoT combined with deep learning algorithms to realize street garbage classification, fabric defect detection, et al. We wanted to use edge computing technology combined with deep learning algorithms, to classify more spectral data. The current mainstream spectral data processing algorithm is still for one-dimensional spectral data analysis. The machine learning image processing methods widely used in these processing methods are incompatible. As mentioned previously, the current deep learning algorithms are very in-depth in image processing research, these algorithms have relatively high processing efficiency and classification accuracy. If we can preprocess the spectral data, then we use deep learning algorithms for classification, which will greatly improve the efficiency and accuracy of spectral classification.

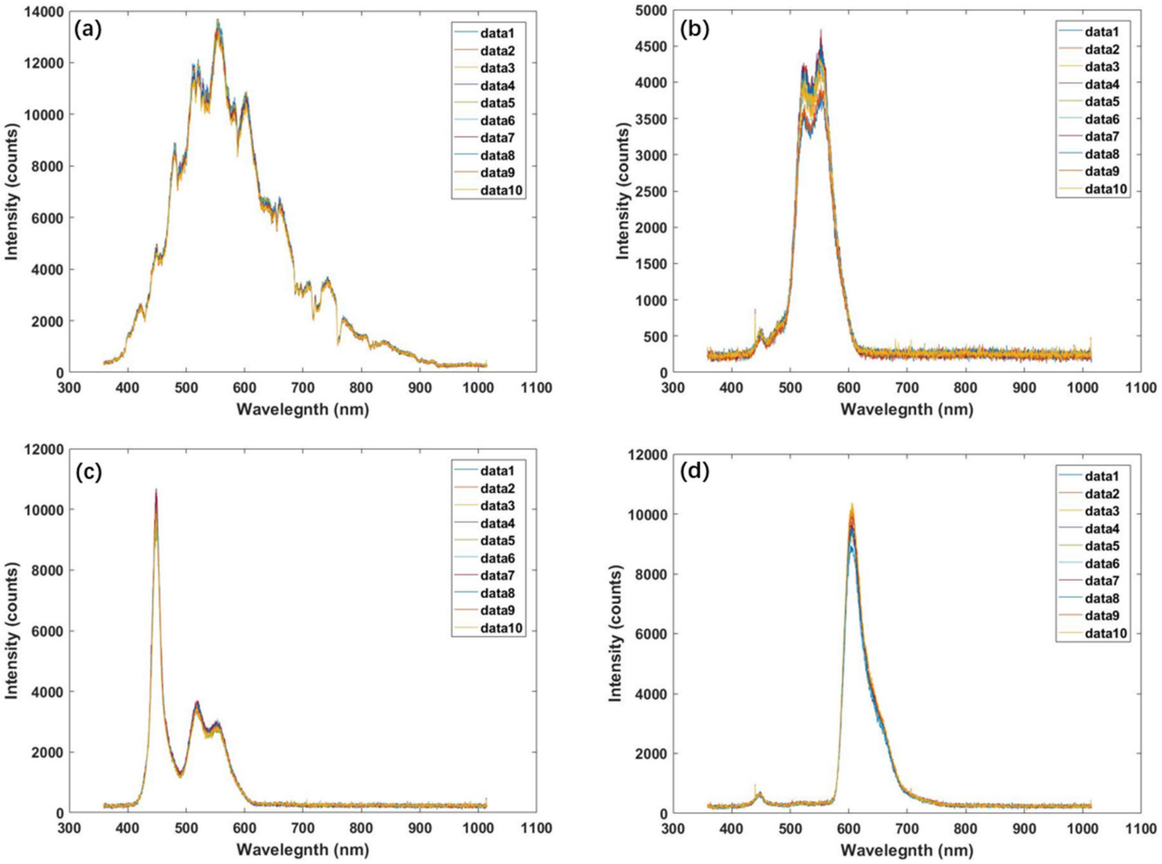


Figure 3. (**a**) 10 sets of sunlight spectrum data collected every 10 s. (**b**) 10 sets of screen green light spectrum data collected every 10 s; (**c**) 10 sets of screen blue light spectrum data collected every 10 s; (**d**) 10 sets of screen red light spectrum data collected every 10 s.

In our work, we randomly selected five kinds of fruit for testing and achieved accurate classification results through the algorithms. Generally, as long as we obtain enough spectral data and design effective algorithms, we can achieve accurate classification. A large number of literature results have verified the effectiveness of classification based on spectral data. For instance, in [20], the classification based on spectral data was also realized for different algae.

In this paper, we designed a portable optical fiber spectrometer with a screen and verified the stability, accuracy, and detection ability of the system through two different experimental processing methods shown in Figures 2 and 3. We used the spectrometer to collect one-dimensional reflectance spectrum data from five fruit samples, then we reshaped the spectral data structure and transformed it into 2D spectral data. We used our proposed CNN algorithm to extract and classify the 2D spectral image data of five samples. Its maximum classification accuracy rate was 94.78%, and the average accuracy rate was 92.94%, which is better than the traditional AlexNet, Unet, and SVM. Our method makes the spectral data analysis compatible with the deep learning algorithm and implements the deep learning algorithm to process the reflection spectral data from the optical fiber spectrometer.

The remaining paper is organized as follows: Section 2 introduces the optical detection experiment in brief. Section 3 provides the details of the proposed spectral classification method. Section 4 reports our experiments and discusses our results. Finally, Section 5 concludes the work and presents some insights for further research.

2. Optical Detection Experiment

We collected one-dimensional data of grapes, jujubes, kumquats, pears, and tomatoes through a portable optical fiber spectrometer. The pictures of the five samples are presented in Figure 4.

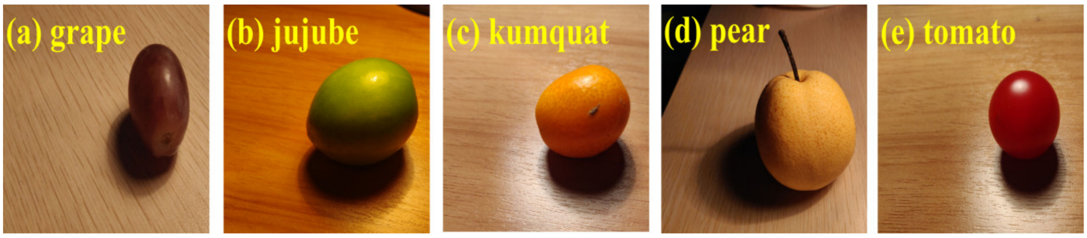


Figure 4. Experimental samples.

Some reasons will affect remote sensing spectral detection in the real world. For instance, the incident angle and reflection angle of light is not stable in the real optical platform detection environment. Therefore, to adapt to the change of angle, we adjusted the alignment direction of the optical fiber port of the equipment to better achieve the good effect of the optical detection experiment.

Most two-dimensional images are processed and classified by convolution neural network models. However, the spectral data we obtained through the spectrometer is in a one-dimensional format, which is incompatible with the method of deep learning algorithms to process the 2D spectral data. To transform one-dimensional data into two-dimensional data, to realize the classification of deep learning algorithms, we finished the transformation of the one-dimensional data through the "Reshape" function in Matlab. After processing, we obtained five kinds of two-dimensional spectral data (32 × 32 pixels). These images are presented in Figure 5. In Section 3, we chose a method for deep learning called a convolutional neural network, which classifies these 2D spectral data.

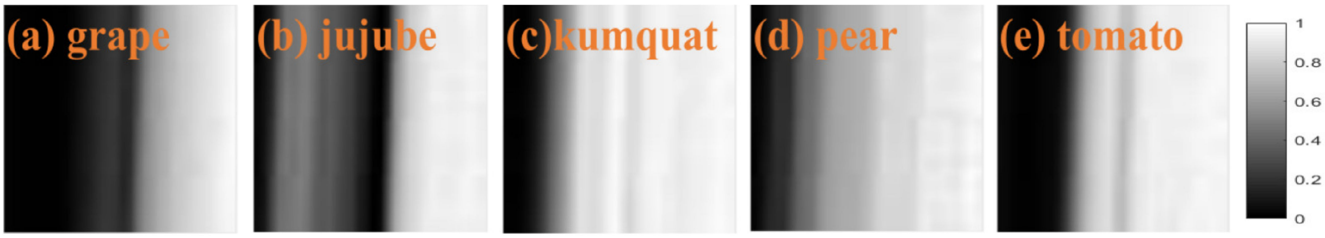


Figure 5. 2D spectral data samples.

3. Proposed Method

3.1. Model Description

Using a deep learning convolutional neural network model to identify spectral data can be divided into two steps. First, perform feature extraction on the images, and then use the classifier to classify the images. The specific recognition process is depicted in Figure 6.

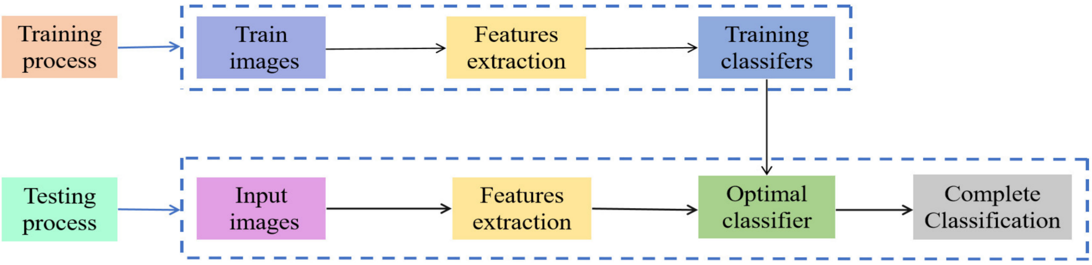


Figure 6. Convolutional neural networks (CNN) recognition process.

In general, there are convolutional layers, pooling layers, and fully connected layers in a convolutional neural network architecture. Compared with other deep learning models, CNNs show better classification performance.

When CNNs perform convolution operations, the image feature size of the upper layer is calculated and processed through convolution kernels, strides, filling, and activation functions. The output and the input of the previous layer establish a convolution relationship with each other. The convolution operation of feature maps uses the following formula.

$$x_j^l = f(\sum_{i=1}^{n} w_{ij}^l \times x_i^{l-1} + b_j^l) \tag{1}$$

where $f(\cdot)$ is the activation function, x_i^{l-1} is the output value of the i-th neuron in the $(l-1)$-th layer, w_{ij}^l represents the weight value of the i-th neuron of the l-th convolutional layer connected to the j-th neuron of the output layer, b_j^l represents the bias value of the j-th neuron of the l-th convolutional layer.

$$x_j^l = f(\rho_j^l down(x_j^{l-1} + b_j^l)) \tag{2}$$

where $f(\cdot)$ is the activation function, $down\ (\cdot)$ represents the downsampling function, ρ is the constants used when the feature map performs the sampling operation, b_j^l represents the bias value of the j-th neuron of the l-th convolutional layer.

The convolutional neural network is usually equipped with a fully connected layer in the last few layers. The fully connected layer normalizes the features after multiple convolutions and pooling. It outputs a probability for various classification situations. In other words, the fully connected layer acts as a classifier.

The Dropout [21] technology is used in CNN to randomly hide some units so that they do not participate in the CNN training process to prevent overfitting. The convolutional layer without the Dropout layer can be calculated using the following formula.

$$z_j^{l+1} = w_j^{l+1} y_j^l + b_j^{l+1} \tag{3}$$

$$y_j^{l+1} = f(z_j^{l+1}) \tag{4}$$

The mean of w, b, and $f(\cdot)$ is the same as that of Equation (1).

The discard rate with the Dropout layer can be described as (5):

$$r_j^l \sim Bernoulli(p) \tag{5}$$

In fact, the Bernoulli function conforms to the distribution trend of Bernoulli. Through the action of the Bernoulli distribution, the Bernoulli function is randomly decomposed into a matrix vector of 0 or 1 according to a certain probability. Where r is the probability matrix vector obtained by the action of the Bernoulli function. In the training process of models, it is temporarily discarded from the network according to a certain probability, that is, the activation site of a neuron no longer acts with probability p (p is 0).

We multiply the input of neurons by Equation (5) and define the result as the input of neurons with the discard rate. It can be described as.

$$\widetilde{y}_j^l = r_j^l * y_j^l \tag{6}$$

Therefore, the output was determined using the following formula.

$$\widetilde{z}_j^{l+1} = w_j^{l+1} \widetilde{y}_j^l + b_j^{l+1} \tag{7}$$

$$\widetilde{y}_j^{l+1} = f(\sum_{j=1}^{k} \widetilde{z}_j^{l+1}) \tag{8}$$

Here, k represents the number of the output neurons.

In this work, we classified 2D spectral data using AlexNet. However, the recognition rate was not high. Mainly, the reasons were analyzed as follows:

(1) Due to the small amount of spectral data sample set collected in this experiment, the training data sets are difficult to meet the needs of deeper AlexNet for feature extraction, learning, and processing. Therefore, the traditional network architecture needed to be streamlined.

(2) In the convolutional process, the more times of convolution, the more spectral features can be fully extracted. The process also uses a large number of convolution kernels, which will bring difficulties to the calculation. The long stride affected the classification accuracy, it was necessary to decrease the traditional parameters of the network convolutional layer.

(3) If a wrong pooling method was used, it would decrease the efficiency of the network learning features and the accuracy of targets classification. The traditional network pooling layer needed to be reduced.

Therefore, we simplified the traditional AlexNet network architecture, decreased the parameters of the convolutional layers, reduced the number of pooling layers, and proposed a new CNN spectral classification model. Figure 7 reveals a specific deep learning spectral classification model framework. Additionally, we added a Dropout layer after each convolutional layer, k represents the size of convolution kernels or pooling kernels, s is the step size moved during convolution or pooling in the CNN operation, and p represents the value of filling the edge after the convolutional layer operation, and generally, the filling value is 0, 1, and 2.

Since the CNN model requires images of uniform size as input, all spectral data images are normalized to a size of 32 × 32 as input images. We divided the spectral data into n categories, so in the seventh layer, after the Dropout layer and the activation function *softmax* were calculated, $n \times 1 \times 1$ neurons were output, that is, the probability of the category where the n nodes were located.

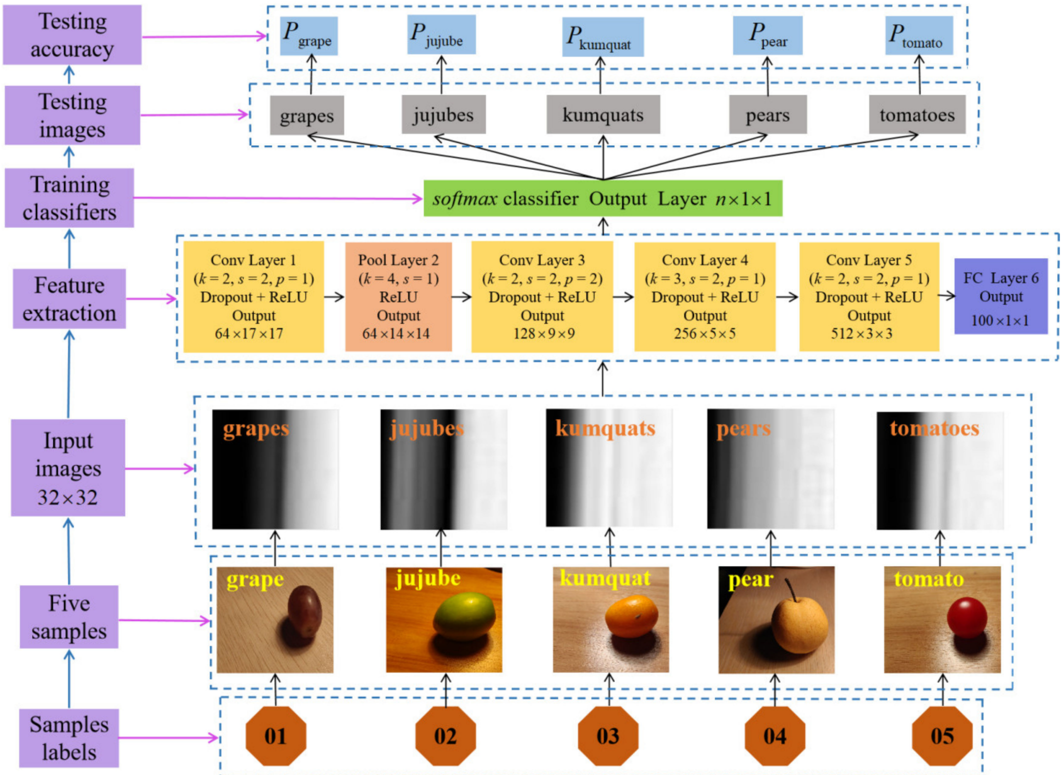

Testing accuracy
Testing images
Training classifiers
Feature extraction
Input images 32×32
Five samples
Samples labels
P_{grape}
P_{jujube}
$P_{kumquat}$
P_{pear}
P_{tomato}
grapes
jujubes
kumquats
pears
tomatoes
softmax classifier Output Layer n×1×1
Conv Layer 1 (k = 2, s = 2, p = 1) Dropout + ReLU Output 64×17×17
Pool Layer 2 (k = 4, s = 1) ReLU Output 64×14×14
Conv Layer 3 (k = 2, s = 2, p = 2) Dropout + ReLU Output 128×9×9
Conv Layer 4 (k = 3, s = 2, p = 1) Dropout + ReLU Output 256×5×5
Conv Layer 5 (k = 2, s = 2, p = 1) Dropout + ReLU Output 512×3×3
FC Layer 6 Output 100×1×1
grapes
jujubes
kumquats
pears
tomatoes
grape
jujube
kumquat
pear
tomato
01
02
03
04
05

Figure 7. Deep learning spectral classification model framework diagram.

3.2. Dropout Selection Principle

Dropout can be used as a kind of trick for training convolutional neural networks. In each training batch, it reduces overfitting by ignoring half of the feature detectors. This method can reduce the interaction in feature hidden layer nodes. In brief, Dropout makes the activation value of a certain neuron stop working with a certain probability p when it propagates forward.

In a deep learning model, if the model has too many parameters and too few training samples, the trained model is prone to overfitting. When we train neural networks, we often encounter overfitting problems. The model has a small loss function on the training data and a high prediction accuracy. The loss function on the test data is relatively large, and the prediction accuracy rate is low. If the model is overfitted, then the resulting model is almost unusable. Dropout can effectively alleviate the occurrence of over-fitting and achieve the effect of regularization to a certain extent. The value of the discard rate plays an important role in the deep learning model. An appropriate Dropout value can reduce the complex co-adaptation relationship between neurons and makes the model converge quickly.

In the training process of CNNs, when the steps of the convolution operation are different, the number of output neurons is different, which will reduce their dependence and correlation. If we quantify the correlation, it will increase the dependence. Therefore, we set the discard rate to narrow the range of correlation. After we successively take values in the narrow range, we train and predict the network model again. It will make any two neurons in different states have a higher correlation and improves the recognition accuracy of the model.

When we trained our proposed CNN model, we visualized the movable trend in dropout layers. Figure 8 presents the movable trend. Figure 8 demonstrates that it is very unstable between 0.5 and 1, which is prone to over-fitting. In (0, 0.1) and (0.2, 0.5), when increasing the epoch, the discard rate drops rapidly, and it is prone to under-fitting. In (0.1, 0.2), the discard rate gradually tends to a stable and convergent state, it is indicated that the value is more appropriate in the interval.

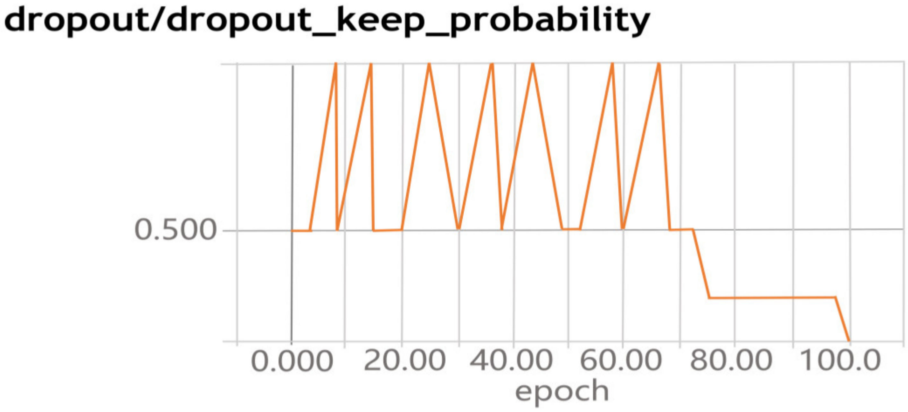


Figure 8. Dropout change graph.

4. Experimental Results and Discussion

In the algorithms' experiments, our hardware platform was: CPU frequency 3.00 GHz, the 32 GB memory, a GTX 1080ti GPU graphics card, and the Cuda 9.2 (Cudnn 7.0) accelerator. Our software platform was Keras 2.2.4, TensorFlow 1.14.0, Anaconda 3 5.2.0, Spyder, and Python 3.7 under win10 and a 64-bit operating system.

4.1. Data Distribution

In the experiments of the algorithm classification, the 2D spectral data of five fruit samples were obtained from the optical detection experiment. We divided the 2D spectral data into the training set, the verification set, and the testing set. The number of the training sets is about three times that of the testing set. Results are shown in Table 1.

Table 1. The data set.

Samples	Grapes	Jujubes	Kumquats	Pears	Tomatoes	Total
train	28	28	28	28	27	139
test	9	9	9	9	9	45
validate	13	13	13	13	13	65

4.2. Train Results

We trained our proposed CNN model, and the results are presented in Figure 9. From Figure 9 we can see that the loss of the training set and the validation set is always between 0.1 and 0.5, and there are no irregular up-and-down violent fluctuations. Both the training accuracy and the validating accuracy are rising and eventually reach a stable value; there is no longer a trend of large value changes. It can find out that if we increase the epoch, the training loss and the validating loss gradually become smaller, and eventually stabilizes. To sum up, our proposed CNN can overcome vanishing gradient in the process of training and validating, and can fully extract features of spectral data from end to end, which is conducive to the correct classification of spectra

Through the model's training time, accuracy, and loss curve, we can comprehensively judge the performance of the model. If the model consumes less training time, the accuracy and loss curves also tend to be stable and fast, and it is illustrated that the model has good convergence performance in a short time. If the model consumes for a long time, the accuracy and loss curve also tends to be steady and slow, and this indicates that the model has poor performance. Through the length of time consumed and the change in accuracy loss, some parameters of the model such as learning rate, batch processing times, etc, can be fine-tuned to improve the performance of the model. Therefore, we not only consider the model's accuracy and loss changes to the training data, but also consider the time consumption.

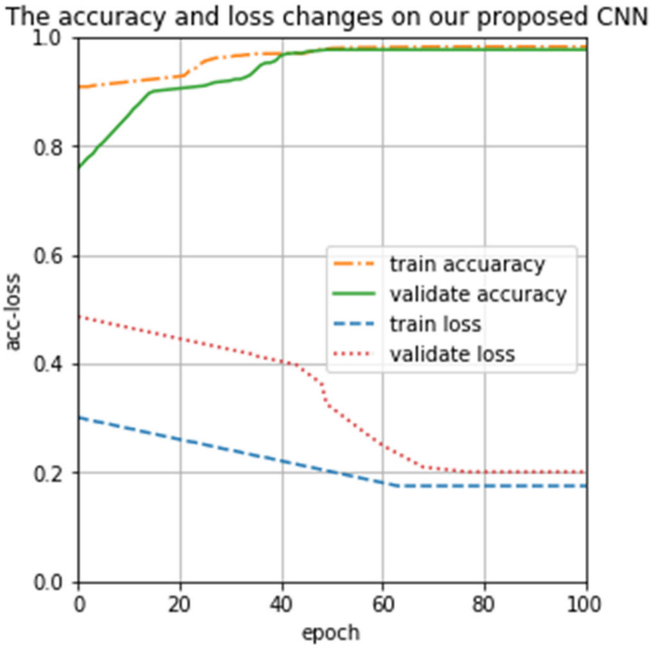


Figure 9. Proposed CNN iteration training change graph.

We recorded the time of training 100 times under four algorithms, Table 2 reveals the time of the four algorithms.

Table 2. Training time.

Algorithms Names	Proposed CNN	AlexNet	Unet	SVM
Time (s)	30.1	56.5	70.3	89.6

As shown in Table 2, our proposed method consumes the least time. It is proved that our proposed method can adapt to the feature extraction of spectral data, and does not bring too much parameters calculation to occupy memory.

4.3. Test Results

The SOTA image recognition model ViT-G/14 uses the JFT-3B data set containing 3 billion images, and the amount of parameters is up to 2 billion. On the ImageNet image data set, it achieved a Top-1 accuracy of 90.45%, it has surpassed the Meta Pseudo Labels model. Although ViT-G/14 performs well in addition to better performance, our data volume and categories are limited. Our data cannot adapt to the parameter training and testing of the SOTA image recognition model ViT-G/14 with more categories and large amounts of data. Therefore, we chose AlexNet, Unet, SVM, and our proposed CNN for comparison.

When epochs are set as 100, the testing accuracy of four algorithms under different parameters is presented in Table 3. Table 3 reports that different parameters correspond to different testing accuracy. For instance, the inputting shape is a batch size of 32, the learning rate is 0.001, the optimizer is SGD, our testing accuracy is 92.57%. It is superior to other parameters. Furthermore, the testing accuracy obtained by the values of different parameters also shows that our proposed CNN achieves an improvement in the classification accuracy of 22.86% when compared to AlexNet.

Table 3. Test accuracy under different parameters.

Methods	Input Shape	Batch Size	Learning Rate	Optimizer	Test Loss	Test Accuracy
Proposed CNN	32 × 32	32	0.0001	SGD	0.0815	0.9275
	64 × 64	64	0.0005	RMS	0.1014	0.8627
	128 × 128	128	0.0050	Adam	0.2571	0.8835
AlexNet	32 × 32	32	0.0001	SGD	1.3359	0.8264
	64 × 64	64	0.0005	RMS	0.9954	0.7549
	128 × 128	128	0.0050	Adam	1.3587	0.7918
Unet	32 × 32	32	0.0001	SGD	1.7529	0.8031
	64 × 64	64	0.0005	RMS	1.0349	0.7429
	128 × 128	128	0.0050	Adam	1.4681	0.7069
SVM	32 × 32	32	0.0001	SGD	3.5248	0.5317
	64 × 64	64	0.0005	RMS	1.2481	0.6728
	128 × 128	128	0.0050	Adam	1.5643	0.6109

Figure 8 illustrates that the discard rate is the most appropriate value in (0.1, 0.2). We divided it into four sub-intervals to test the precision of our proposed CNN. Testing results are revealed in Figure 10. The results in Figure 10 demonstrate that the accuracy in (0.175, 0.200) is higher than in (0.100, 0.125), (0.125, 0.150), and (0.150, 0.175). Evidently, our proposed CNN model has the best performance in (0.175, 0.200), it verifies the correctness of the dropout discard rate analysis and selection principle simultaneously in Section 3.2.

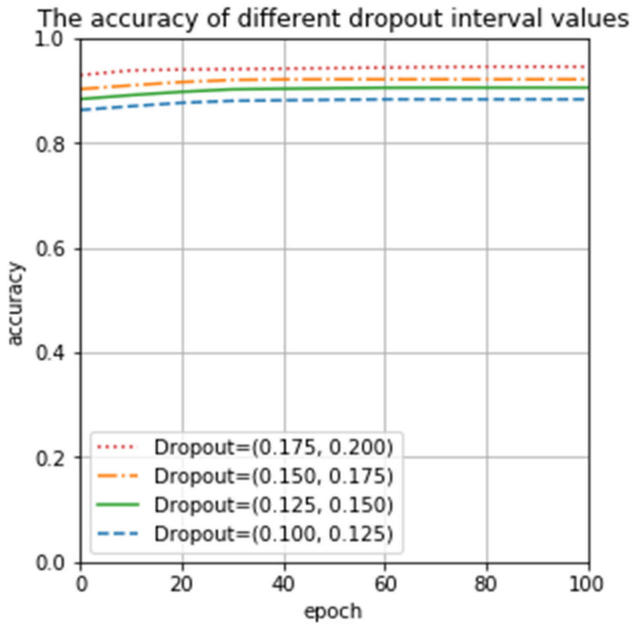


Figure 10. Accuracy in different intervals.

To verify the feasibility of ReLU and the discard rate in this work, we again used ReLU with the Dropout layer (dropout = 0.2) and without the Dropout layer (dropout = 0) for testing. Testing results are shown in Figure 11. The experimental results confirm that the recognition rate is as high as 94.57% when ReLU is used and the discard rate is 0.2, which is significantly higher than the recognition result without the dropout layer (dropout = 0). In summary, our proposed CNN model outperforms AlexNet and SVM tested in terms of classification accuracy, and it can perform accurate spectral classification.

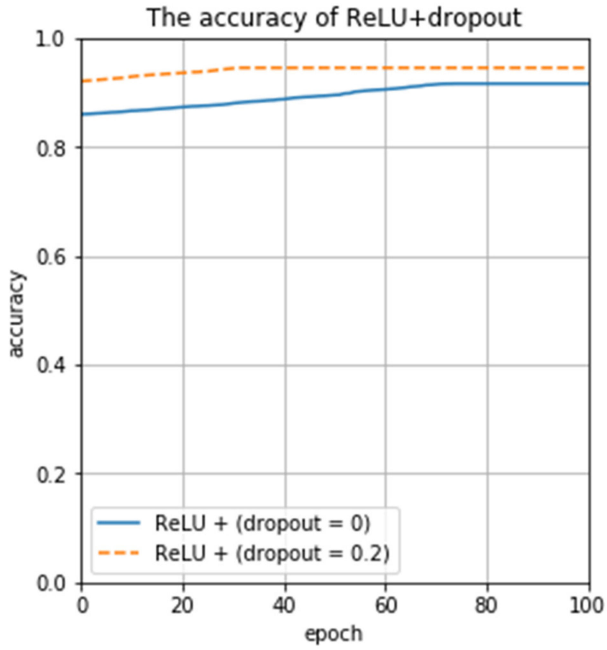


Figure 11. The impact of the dropout value on the recognition rate.

As shown in Table 4, the testing time of our proposed CNN is lower than AlexNet, Unet, and SVM. Evidently, our proposed model can quickly extract two-dimensional spectral features and gives the prediction result in the testing process.

Table 4. Testing times using four different methods.

Samples	Grapes	Jujubes	Kumquats	Pears	Tomatoes	Total Times
Proposed CNN	6.7 s	6.5 s	7.4 s	7.2 s	7.1 s	34.9 s
AlexNet	8.9 s	8.4 s	8.8 s	8.7 s	8.6 s	43.4 s
Unet	8.7 s	9.0 s	9.1 s	9.2 s	8.9 s	44.9 s
SVM	11.1 s	9.3 s	8.9 s	9.4 s	9.5 s	48.2 s

To compare the performance of four different classification methods, we tested five samples one by one. Tables 5–8 show testing results. Tables 5–8 report that the testing precision of our proposed CNN is superior to AlexNet, Unet, and SVM. Therefore, our proposed CNN model has strong robustness to 2D spectral data.

Table 5. Testing five samples with our proposed CNN.

Samples	Grapes	Jujubes	Kumquats	Pears	Tomatoes
Num-1	0.9341	0.9346	0.9125	0.9354	0.9431
Num-2	0.9364	0.9038	0.9227	0.9209	0.9359
Num-3	0.9380	0.9265	0.9104	0.9469	0.9449
Num-4	0.9440	0.9268	0.9255	0.9307	0.9237
Num-5	0.9451	0.9110	0.9214	0.9258	0.9326
Num-6	0.9437	0.9190	0.9264	0.9185	0.9394
Num-7	0.9461	0.9283	0.9109	0.9257	0.9478
Num-8	0.9439	0.9192	0.9217	0.9426	0.9232
Num-9	0.9729	0.9173	0.9164	0.9346	0.9359
Average precision	0.9399	0.9207	0.9187	0.9312	0.9363

Table 6. Testing five samples with AlexNet.

Samples	Grapes	Jujubes	Kumquats	Pears	Tomatoes
Num-1	0.8775	0.7109	0.7516	0.8228	0.7598
Num-2	0.8806	0.7099	0.7722	0.8061	0.7567
Num-3	0.8861	0.7202	0.7417	0.7679	0.7860
Num-4	0.8879	0.7084	0.7233	0.7918	0.8048
Num-5	0.8855	0.7096	0.7213	0.8254	0.7361
Num-6	0.8821	0.7153	0.7706	0.8366	0.7559
Num-7	0.8857	0.7054	0.7512	0.7953	0.7770
Num-8	0.8827	0.7159	0.7717	0.8024	0.8041
Num-9	0.8947	0.7018	0.7669	0.8127	0.7436
Average precision	0.8859	0.7108	0.7523	0.8068	0.7693

Table 7. Testing five samples with Unet.

Samples	Grapes	Jujubes	Kumquats	Pears	Tomatoes
Num-1	0.8437	0.6859	0.7149	0.7986	0.7689
Num-2	0.8371	0.6971	0.7029	0.7961	0.7586
Num-3	0.8257	0.6782	0.7116	0.7881	0.7828
Num-4	0.8159	0.7015	0.7036	0.7989	0.7659
Num-5	0.8041	0.6985	0.7219	0.8007	0.7458
Num-6	0.8358	0.7031	0.7108	0.7896	0.7675
Num-7	0.8539	0.7007	0.7019	0.7968	0.7752
Num-8	0.8489	0.6995	0.7125	0.8027	0.7871
Num-9	0.8148	0.7037	0.7034	0.8007	0.7923
Average precision	0.8311	0.6965	0.7093	0.7969	0.7716

In Section 4.2, we considered the model training time, we also consider the speed of the model during testing images, simultaneously. If we proposed model is slower on testing image speed, it is revealed that the performance of the model does not take into account. The quality of a model not only depends on its training time, training accuracy, and verification accuracy, etc, but also its testing accuracy and testing time.

Table 8. Testing five samples using the support vector machine (SVM).

Samples	Grapes	Jujubes	Kumquats	Pears	Tomatoes
Num-1	0.5330	0.6215	0.5309	0.6143	0.5415
Num-2	0.5371	0.5956	0.5524	0.5681	0.5319
Num-3	0.5386	0.6293	0.5300	0.6211	0.5475
Num-4	0.5356	0.6220	0.5614	0.6148	0.5606
Num-5	0.5297	0.6045	0.5315	0.6301	0.5226
Num-6	0.5268	0.6124	0.5524	0.5761	0.5270
Num-7	0.5327	0.6224	0.5772	0.6161	0.5409
Num-8	0.5295	0.6095	0.5296	0.6184	0.5578
Num-9	0.5406	0.6135	0.5650	0.6631	0.5232
Average precision	0.5337	0.6145	0.5478	0.6136	0.5392

Figure 12 shows the maximum testing precision of each sample under four different algorithms. It can figure out if the testing effect of our proposed CNN is significantly greater than the other three methods in Figure 12. Obviously, our proposed CNN has high classification precision and generalization ability to 2D spectral data.

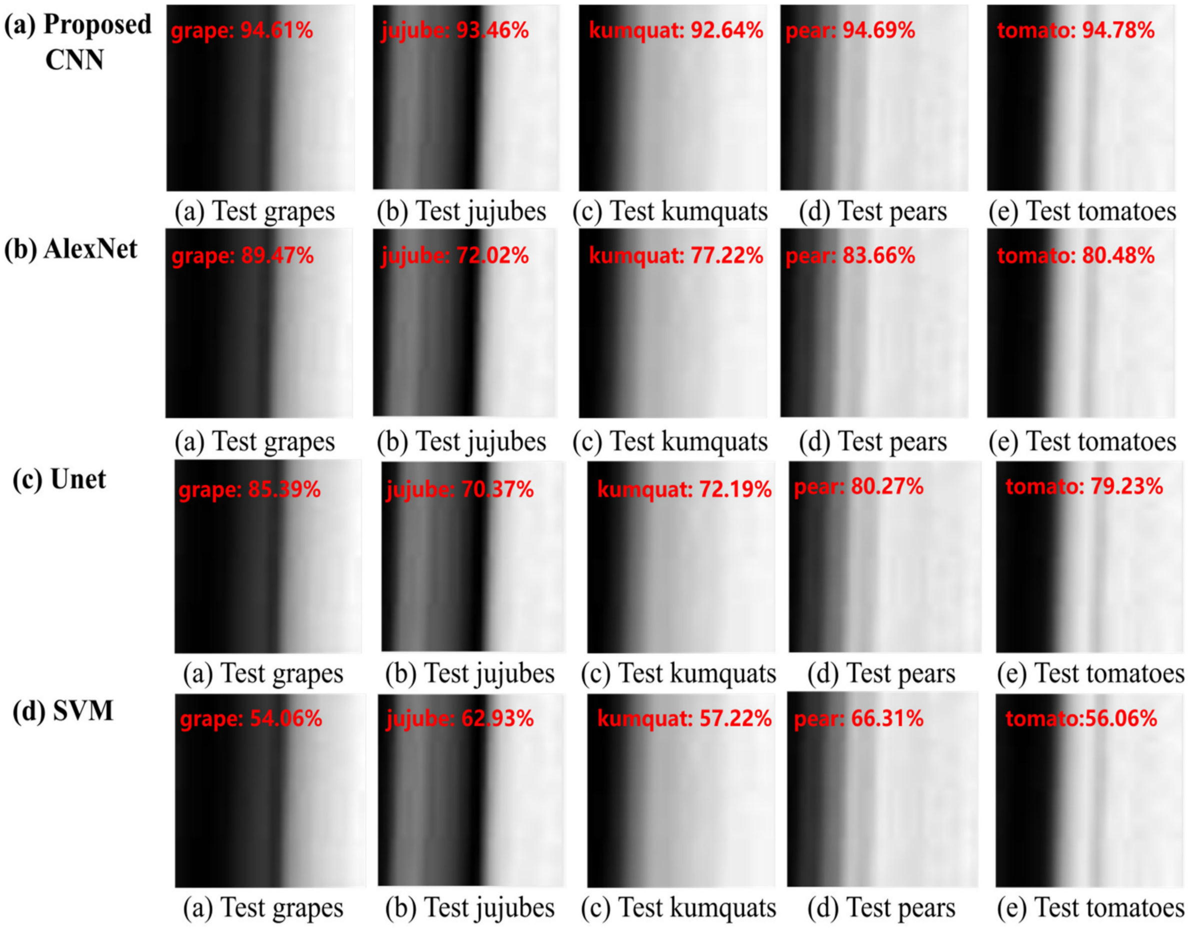


Figure 12. (**a**) The maximum testing results of the proposed CNN are between 90% and 95%. (**b**,**c**) The maximum testing results of AlexNet and Unet are between 70% and 90%. (**d**) The maximum testing results of the SVM are between 50% and 70%.

5. Conclusions

In this work, a new CNN architecture was designed to effectively classify 2D spectral data of five samples. Specifically, we added a Dropout layer behind each convolutional layer of the network to randomly discard some useless neurons and effectively enhance the feature extraction ability. In this way, the features uncovered by the network became stronger, which eventually lead to a reduction of the network architecture parameters calculation complexity and, therefore, to a more accurate spectral classification. The experimental comparisons conducted in this work shows that our proposed approach exhibits competitive advantages with respect to AlexNet, Unet, and SVM classification methods. Although fiber optic spectrometers cannot directly perform spectral imaging classification research, our work has confirmed that deep learning algorithms can be combined with the spectral data obtained by the optical fiber spectrometer for classification research. We will use fiber optic spectrometers to obtain more samples of spectral data and combine edge computing technology to send to the deep learning model for data processing and classification research in the future.

Author Contributions: Conceptualization, L.X. and F.C.; methodology, L.X., F.C. and J.W.; software, L.X. and F.C.; validation, L.X., J.X. and J.W.; formal analysis, L.X., F.C. and J.W.; data curation, L.X. and J.X.; writing—original draft preparation, L.X., F.C. and J.W.; writing—review and editing, L.X., F.C. and J.W.; supervision, J.W.; funding acquisition, F.C. and J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by National Key Research and Development Program of China (No. 2018YFC1407505); National Natural Science Foundation of China (No. 81971692); the Natural Science Foundation of Hainan Province (No. 119MS001) and the scientific research fund of Hainan University (No. kyqd1653).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, L.; Ota, K.; Dong, M. When weather matters: IoT based electrical load forecasting for smart grid. *IEEE Commun. Mag.* **2017**, *55*, 46–51. [CrossRef]
2. Kim, H.Y.; Kim, J.M. A load balancing scheme based on deep learning in IoT. *Clust. Comput.* **2017**, *20*, 873–878. [CrossRef]
3. Alsheikh, M.A.; Niyato, D.; Lin, S.; Tan, H.-P.; Han, Z. Mobile big data analytics using deep learning and apache spark. *IEEE Netw.* **2016**, *30*, 22–29. [CrossRef]
4. Liu, J.; Guo, H.; Nishiyama, H.; Ujikawa, H.; Suzuki, K.; Kato, N. New perspectives on future smart FiWi networks: Scalability, reliability, and energy efficiency. *IEEE Commun. Surv. Tutor.* **2017**, *18*, 1045–1072. [CrossRef]
5. Sun, X.; Ansari, N. EdgeIoT: Mobile edge computing for the Intemet of Things. *IEEE Commun. Mag.* **2016**, *54*, 22–29. [CrossRef]
6. Zhu, Z.; Han, G.; Jia, G.; Shu, L. Modified DenseNet for Automatic Fabric Defect Detection with Edge Computing for Minimizing Latency. *IEEE Internet Things J.* **2020**, *7*, 9623–9636. [CrossRef]
7. Pan, D.; Liu, H.; Qu, D.; Zhang, Z. CNN-Based Fall Detection Strategy with Edge Computing Scheduling in Smart Cities. *Electronics* **2020**, *9*, 1780. [CrossRef]
8. Ping, P.; Xu, G.; Kumala, E.; Xu, G. Smart Street Litter Detection and Classification Based on Faster R-CNN and Edge Computing. *Int. J. Softw. Eng. Knowl. Eng.* **2020**, *30*, 537–553. [CrossRef]
9. Sahni, Y.; Cao, J.; Yang, L.; Ji, Y. Multi-Hop Multi-Task Partial Computation Offloading in Collaborative Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1133–1145. [CrossRef]
10. Khaliq, K.A.; Chughtai, O.; Shahwani, A.; Qayyum, A.; Pannek, J. Road Accidents Detection, Data Collection and Data Analysis Using V2X Communication and Edge/Cloud Computing. *Electronics* **2019**, *8*, 896. [CrossRef]
11. Dou, W.; Zhao, X.; Yin, X.; Wang, H.; Luo, Y.; Qi, L. Edge Computing-Enabled Deep Learning for Real-time Video Optimization in IIoT. *IEEE Trans. Ind. Inform.* **2020**, *17*, 2842–2851. [CrossRef]
12. Zhou, Z.; Liu, Q.; Fu, Y.; Xu, X.; Wang, C.; Deng, M. Multi-channel fiber optical spectrometer for high-throughput characterization of photoluminescence properties. *Rev. Sci. Instrum.* **2020**, *91*, 123113. [CrossRef]
13. Markvart, A.; Liokumovich, L.; Medvedev, I.; Ushakov, N. Continuous Hue-Based Self-Calibration of a Smartphone Spectrometer Applied to Optical Fiber Fabry-Perot Sensor Interrogation. *Sensors* **2020**, *20*, 6304. [CrossRef] [PubMed]
14. Yan, L.; Song, Y.; Liu, W.; Lv, Z.; Yang, Y. Phosphor thermometry at 5 kHz rate using a high-speed fiber-optic spectrometer. *J. Appl. Phys.* **2020**, *127*, 124501. [CrossRef]
15. Cai, F.; Wang, T.; Wu, J.; Zhang, X. Handheld four-dimensional optical sensor. *Optik* **2020**, *203*, 164001. [CrossRef]

16. Praveen, B.; Menon, V. Study of Spatial–Spectral Feature Extraction Frameworks with 3-D Convolutional Neural Network for Robust Hyperspectral Imagery Classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2020**, *14*, 1717–1727. [CrossRef]
17. Anthony, M.; Wang, Z. Hyperspectral Image Classification Using Similarity Measurements-Based Deep Recurrent Neural Networks. *Remote Sens.* **2019**, *11*, 194. [CrossRef]
18. Li, J.; Xi, B.; Li, Y.; Du, Q.; Wang, K. Hyperspectral Classification Based on Texture Feature Enhancement and Deep Belief Networks. *Remote Sens.* **2018**, *10*, 396. [CrossRef]
19. Liu, X.; Jiang, Z.; Wang, T.; Cai, F.; Wang, D. Fast hyperspectral imager driven by a low-cost and compact galvo-mirror. *Optik* **2020**, *224*, 165716. [CrossRef]
20. Xu, Z.; Jiang, Y.; Ji, J.; Forsberg, E.; Li, Y.; He, S. Classification, identification, and growth stage estimation of microalgae based on transmission hyperspectral microscopic imaging and machine learning. *Opt. Express* **2020**, *28*, 30686–30700. [CrossRef]
21. Hahn, S.; Choi, H. Understanding dropout as an optimization trick. *Neurocomputing* **2020**, *398*, 64–70. [CrossRef]

electronics MDPI

Article

Method for Dynamic Service Orchestration in Fog Computing

Nerijus Morkevicius *, **Algimantas Venčkauskas**, **Nerijus Šatkauskas** and **Jevgenijus Toldinas**

Department of Computers Science, Kaunas University of Technology, Studentų st. 50, LT-51368 Kaunas, Lithuania; algimantas.venckauskas@ktu.lt (A.V.); nerijus.satkauskas@ktu.lt (N.Š.); eugenijus.toldinas@ktu.lt (J.T.)
* Correspondence: nerijus.morkevicius@ktu.lt

Abstract: Fog computing is meant to deal with the problems which cloud computing cannot solve alone. As the fog is closer to a user, it can improve some very important QoS characteristics, such as a latency and availability. One of the challenges in the fog architecture is heterogeneous constrained devices and the dynamic nature of the end devices, which requires a dynamic service orchestration to provide an efficient service placement inside the fog nodes. An optimization method is needed to ensure the required level of QoS while requiring minimal resources from fog and end devices, thus ensuring the longest lifecycle of the whole IoT system. A two-stage multi-objective optimization method to find the best placement of services among available fog nodes is presented in this paper. A Pareto set of non-dominated possible service distributions is found using the integer multi-objective particle swarm optimization method. Then, the analytical hierarchy process is used to choose the best service distribution according to the application-specific judgment matrix. An illustrative scenario with experimental results is presented to demonstrate characteristics of the proposed method.

Keywords: fog computing; Internet of Things; service placement; fog service orchestration

Citation: Morkevicius, N.; Venčkauskas, A.; Šatkauskas, N.; Toldinas, J. Method for Dynamic Service Orchestration in Fog Computing. *Electronics* **2021**, *10*, 1796. https://doi.org/10.3390/electronics10151796

Academic Editors: Kevin Lee and Ka Lok Man

Received: 29 June 2021
Accepted: 26 July 2021
Published: 27 July 2021

1. Introduction

Fog computing acts as a missing link in the cloud-to-thing continuum. Services are provided closer to the edge of the network to enhance frequent services, latency, availability, and analysis. Fog computing places some computation resources in close proximity to a user where numerous heterogeneous end devices have to work in harmony. Control functions must work autonomously in such a heterogeneous and complex environment. Therefore, an orchestration is a centralized executable process to coordinate any interaction among any application or service [1]. Figure 1 shows the fog computing architecture.

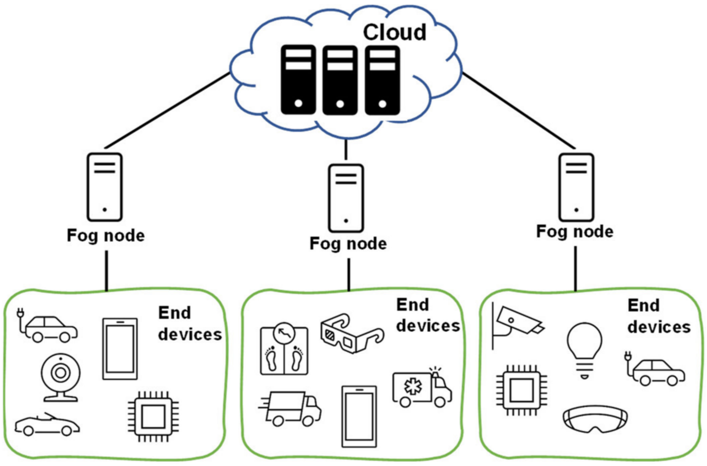


Figure 1. Fog computing architecture.

There is a wide variety of the application areas. As the review paper [2] classifies it in their fog computing application taxonomy, it is an application area which is made of municipal services, smart citizens, smart education, smart healthcare, smart buildings, smart energy, smart governance, etc. The main concerns which were identified in the reviewed papers were the following: bandwidth management, power management, security, mobility, resource management, and latency.

In order to ensure any heterogeneous service provision infrastructure with a scalability, interoperability, and adaptability in mind, fog nodes have to be dynamically deployable [3]. These fog nodes also use constrained resources [1] if they are compared to a cloud infrastructure. Additionally, in order to give access to relevant services as well as to prevent any unauthorized access, access control or privacy control is required [4]. Having that in mind, security and privacy are a big concern [5,6]. Fog computing solutions frequently have insufficient security due to the fact that they rely on intensive communications with constrained devices [7] and constrained resources [8] located at the end device layer. If one of many end devices does not support a communication protocol of a sufficient strength, the security of the whole solution may be compromised. Moreover, roaming services are supported in some fog solutions, when the service follows a human, vehicle, etc., and travels from one fog node to another. In such cases, when any lower security service is brought to a secure fog node, the security of this fog node may be compromised. Similar problems may also occur with other QoS parameters such as a latency, bandwidth, range, etc. However, service orchestrators which are placed in the fog nodes may be used to monitor the whole situation in the fog node (including any communications with neighboring fog nodes) to take any required measures in the case of any potential violations of security and other QoS parameters.

After a dynamic service orchestrator deploys any relevant services within specific fog nodes, there is another hurdle to overcome, the optimization [9]. Fog computing keeps computing resources close to users and to the end nodes to reduce any delay for IoT services. It can also deal with the privacy, data locality, and bandwidth consumption. There are several objectives that can be enhanced by an optimization, such as a latency, cost, or energy management. It is a part of the quality of service (QoS) but it may come with a trade-off.

Any fog service orchestration can be challenging in such conditions. Cloud service orchestration may already be reliable enough at the moment [10], but the situation is different with fog computing. The complexity builds up due to the diversity of different services and resources. There are also concerns about interoperability, performance and service assurance, lifecycle management, scalability, security, and resilience, as identified in the review [11]. The paper [12] suggests that scalability, dynamics, and security are among the most common orchestration challenges which are specified in research papers.

Our goal in this research is to offer an effective orchestrator working in the fog layer of the fog computing architecture by providing effective means to solve the QoS- and security-related problems of the orchestration in heterogeneous fog layer devices and services. The idea is to check the placement of fog devices and services for any potential QoS and security issues in order to find any non-optimal distribution of services among fog nodes.

This paper includes three main contributions aimed at the fog service orchestration problem of an optimal placement of services inside the available fog nodes. First, it presents a detailed review of the fog service orchestration challenges and solutions proposed by other authors. The review clearly demonstrates the most promising mechanisms to be used for a fog service orchestration and it defines the problem more formally, which is addressed in this paper. Second, a two-stage optimal service placement algorithm based on integer multi-objective particle swarm optimization (IMOPSO) and the analytical hierarchy process (AHP) is proposed and formally described. The first stage of the proposed method finds a Pareto set of non-dominated potential placements of services, then the AHP is used to choose one best solution according to the application-specific judgment matrix provided by a user. Third, the proposed method is experimentally evaluated using an illustrative scenario, showing the performance of the algorithm in some likely situations.

The rest of this paper is structured in the following way: related publications are presented in Section 2, following by Section 3, which presents a more formal definition of the service orchestration problem. We describe our proposed method in Section 4. Evaluation and experimental results are summarized in Section 5, and, finally, Section 6 is dedicated to conclusions and a discussion.

2. Related Work Review

Fog orchestrator components, as concluded in the paper [5], can generally be divided into three main groups: fog orchestrator, fog node which can function as a fog orchestrator agent (FOA), and end devices. A fog orchestrator needs to consult its catalogs and certain monitoring data to make an orchestration plan. A fog orchestrator can start its orchestration manually or after reaching a benchmark, such as the availability of other nodes. FOA, in turn, can handle only local resources which are within that particular node.

The main research challenges in fog orchestration identified by Velasquez et al. [13] are the following: resource management, performance, and security management. Resource and service allocation optimization techniques are used, among others, to address these challenges. The problem is that an allocation procedure is a non-trivial problem, because essentially it is a multi-objective optimization problem.

In order to address the issues in the perspective of the fog computing, the authors of the paper [14] suggest using four of their proposed algorithms for their identified construction phase and maintenance phase. The construction phase aims to find some probable candidate locations to place the gateways while using the candidate location identification (CLI) algorithm. A Hungarian method-based topology construction (HTC) algorithm is used to select the optimal gateway locations. Meanwhile, the maintenance phase increases the processing resources in the gateways by intelligent sleep scheduling with the help of the vacation-based resource allocation (VRA) algorithm. Their processing and storage resources in the gateways are further improved based on the tracked data arrival rates with the help of the dynamic resource allocation (DRA) algorithm. Another option which can be beneficial to improving the performance of a network in terms of reliability and response is caching, as was noted in the publication [15]. The caching at the fog nodes can reduce computational complexity and network load. Even though the computing power is the most critical aspect in the fog node to complete specific tasks as the paper [16] suggests, an effective allocation of resources can vary due to limitations. These limitations may include the hierarchy of the fog network, network communication resources, and storage resources.

Yang et al. [17,18] confirm that the orchestration has to deal with a number of factors such as resource filtering and assignment, component selection and placement, dynamics with the runtime QoS, systematic data-driven optimization, or machine learning for orchestration. They implemented a novel parallel genetic algorithm-based framework (GA-Par) on Spark. They normalized the utility of security and network QoS into an objective fitness function within GA-Par. It reduces any security risks and performance deterioration. As their experiments later demonstrated, GA-Par outperforms a standalone genetic algorithm (SGA). Skarlat et al. [19] proposed to solve the fog service placement problem (FSPP) by using orchestration control nodes which place each service either in the fog cells or in the fog orchestration control nodes. The goal of optimization is to maximize the number of service placements in the fog nodes (rather than in cloud ones), while satisfying the requirements of each application. The authors used a genetic algorithm to find the optimal FSPP.

The authors of another paper identified resource allocation and provisioning as a challenging task considering dynamic changes of user requirements and limited resources [20]. They proposed their resource allocation and provisioning algorithms based on the resource ranking. They evaluated their algorithms in a simulation environment after extending their CloudSim toolkit. There are mainly two steps which are used to solve a deadline-based user dynamic behavior problem. First, they ranked resources based on processing power, bandwidth, and response time. Later, they provided resources by prioritizing processing application requests.

As the deployment infrastructure has to adapt itself to extremely dynamic requirements, the fog layer may not provide enough resources and, meanwhile, the cloud layer can fail due to latency requirements [21]. The paper presents a rewriting-based approach to design and verify a self-adaptation and orchestration process in order to achieve a low latency and the right quantity of resources. An executable solution is provided based on Maude, the formal specification language. Properties are expressed using linear temporal logic (LTL). Their proposed cloud–fog orchestrator works as a self-adaptation controller. It is deployed in the fog layer as a fog node master for low latency requirements. The orchestrator triggers the right actions after a decision is made.

Smart service placement and management of services in big fog platforms can be challenging due to a dynamic nature of the workload applications and user requirements for low energy consumption and good response time. Container orchestration platforms are to help with this issue [22]. These solutions either use heuristics for their timely decisions or AI methods such as reinforcement learning and evolutionary approaches for dynamic scenarios. Heuristics cannot quickly adapt to extremely dynamic environments, while the second option can negatively impact response time. The authors also noted that they need scheduling policies which are efficient in volatile environments. They offer a gradient based optimization strategy using back-propagation of gradients with respect to the input (GOBI). They also developed a coupled simulation and container orchestration framework (COSCO) that enabled the creation of a hybrid simulation decision approach (GOBI*) which they used to optimize their quality of service (QoS) parameters.

As the service offloading is relevant enough in the perspective of time and energy, selection of the best fog node can be a serious challenge [23]. The researchers presented in their paper a module placement method by classification and regression tree algorithm (MPCA). Decision parameters select the best fog node, including authentication, confidentiality, integrity, availability, capacity, speed, and cost. They later analyzed and applied the probability of network resource utilization in the module offloading to optimize the MPCA.

Linear programming is another very popular optimization method used for resource allocation and service placement in fog nodes. Arkian et al. [24] linearized a mixed-integer non-linear program (MINLP) into the mixed-integer linear program (MILP) for optimal task distribution and virtual machine placement by using the minimization of cost. Velasquez et al. [25] proposed the service orchestrator which tries to minimize the latency of services using integer linear programming (ILP) to minimize the hop count between communicating nodes.

The authors of [26] present a method used to help deployments of composite applications in fog infrastructures, which have to satisfy software, hardware, and QoS requirements. The developed prototype (FogTorch) uses the Monte Carlo method to find the best deployment which ensures the lowest fog resource consumption—the aggregated averaged percentage of consumed RAM and storage in all the fog nodes.

A sequential decision-making Markov decision problem (MDP) enhanced by the technique of Lyapunov optimization is used by the authors of [27] to minimize operational costs of an IoT system while providing rigorous performance guarantees. The proposed method is intended to be used for a general problem of resource allocation and workload scheduling in cloud computing, but it may also be applied to a service placement problem in fog nodes.

As fog computing has a number of challenges to deal with, optimization is vital, and the classification of optimization problems can play an important role [28]. A service placement problem, in general, has been shown to be NP-complete by the authors of [29]. An optimization is typically made up of [30] (a) a set of variables to encode decisions, (b) a set of possible values for each variable, (c) a set of constraints which the variables are to satisfy, and (d) an objective function. Optimization solutions involving end devices and fog nodes differ based on their application area.

Our analyses of the methods used by other authors for service placement problem optimization, as well as findings of other researchers [31], show that various well-established optimization methods are used for this task, including integer linear programming, genetic algorithms, the Markov decision process, gradient based optimization, the Monte Carlo method, reinforcement learning, etc. The objective functions used by the authors of these methods vary from an overall cost minimization [24,27], to network latency [25], hop and service migration count [25], and response time and latency of the IoT system [26]. The literature review allows us to conclude that the most optimization methods tend to seek for an optimal placement of the services based on the most important parameter of the IoT system, which is represented by the objective function used in an optimization process. Other important parameters of IoT systems in such cases are used as restrictions, and usually include latency, power, bandwidth and QoS [24,26], CPU, RAM, and storage demands [19]. This kind of optimization problem formulation allows one to avoid the challenges of multi-objective optimization, but it may not be used in situations where more than one objective function is required. Some other approaches tend to evaluate several characteristics by combining them into one composite criterion, such as cost [24,27] or fog resource consumption [26] composed from an average RAM and storage usage in the fog nodes. The composite criteria calculation equations usually are provided by the authors of the proposed algorithms, and they use some predefined coefficients which are difficult to justify and validate. One very important challenge remains in this area in that case—how to find the best placement of the services according to several different heterogeneous criteria, with different origins and different units of a measurement, when they often contradict each other. The usage of composite criteria is not always the best answer to this.

The service placement optimization method proposed in this paper tries to address these challenges by using a multi-objective optimization method to find all non-dominated placements of the services and then to select one best placement using an analytical hierarchy process which simplifies the process of the criterion comparison performed by the experts of the application area. In this way, any number of objective functions (optimization criteria) may be used in the optimization process as long as experts are able to provide a consistent pair-to-pair comparison of their priority in the context of a concrete area of application.

3. Orchestrator Components and Architecture

In this paper, we consider the fog orchestration architecture and components presented in Figure 2. We have a service orchestrator in the cloud layer which is used to optimally distribute the services between several orchestrated fog nodes. The orchestrated fog nodes host some services which communicate with end devices, collect and process data, and make some local decisions on the control of actuators located in the end device layer. Special services (orchestrator agents) are physically located in each fog node and they communicate with the orchestrator to provide it with all the necessary information needed to make any decisions on service placement.

Orchestrator agents locally monitor the hardware and software environment of the fog nodes. They are aware of the current CPU and RAM usage, power requirement and energy levels, available communication protocols and bandwidth, security capabilities, state of the hosted services, etc. They summarize all the collected information to provide it to the orchestrator in the cloud layer. The orchestrator is aware of the current situation in all the fog nodes and, additionally, it has security and QoS requirements imposed by the application area of the IoT solution, and it makes decisions on starting, stopping, or moving particular services among the orchestrated fog nodes. The decisions made by the orchestrator are communicated down to the orchestrator agents inside the fog nodes, then the orchestrator agents initialize the required actions on the services. A control cycle performed inside the orchestrator is illustrated in Figure 3.

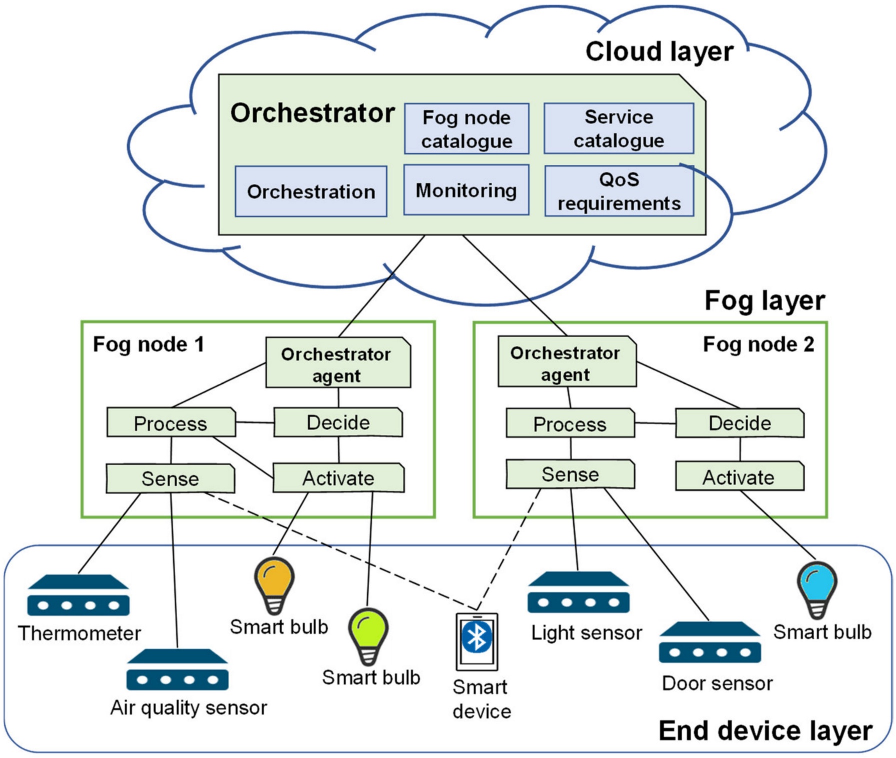


Figure 2. Fog orchestrator architecture and components.

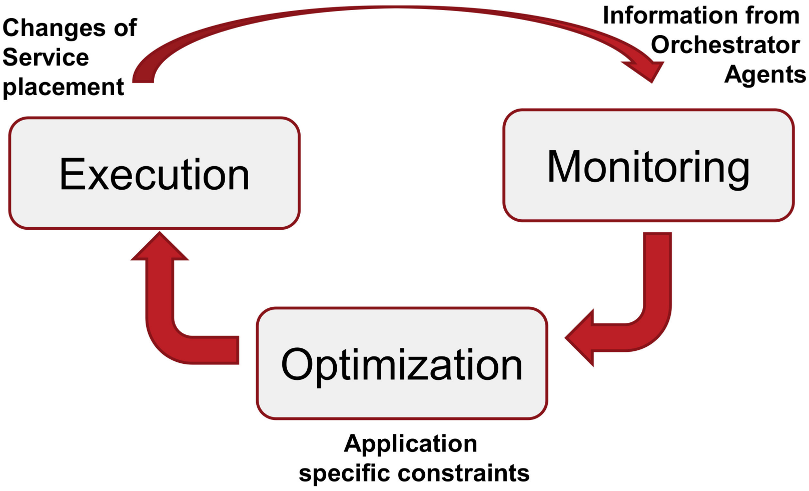

Changes of Service placement
Information from Orchestrator Agents
Execution
Monitoring
Optimization
Application specific constraints

Figure 3. Control cycle inside the orchestrator.

The method of the fog service orchestration presented in this paper is intended to be used inside the orchestrator. The main task of the proposed method is to optimally distribute *n* services among *k* fog nodes according to the information collected from the corresponding fog nodes and the requirements imposed by the area of an application of the IoT system. This task of a service distribution is not trivial since several different optimization criteria which contradict each other must usually be considered (i.e., security level, energy consumption, bandwidth capabilities, latency, etc.). The number of possible different distributions of services among fog nodes increases rapidly with the increase in the number of available fog nodes and services. Any evaluation of all the possible placements of the services is infeasible, therefore, more sophisticated methods are needed. Moreover, the situation and the evaluation criteria can change dynamically due to the dynamic environment of the fog architecture. Some currently available fog nodes as well as end devices may change their location or new fog nodes may even emerge while, on the other hand, some currently running services may become unused and some new services may occur.

4. Method for Fog Service Orchestration

We propose to use multi-objective optimization to decide which placement of n available services in k fog nodes is the best according to given constraints and conditions. The overall flow chart of the proposed two-stage optimization method is presented in Figure 4.

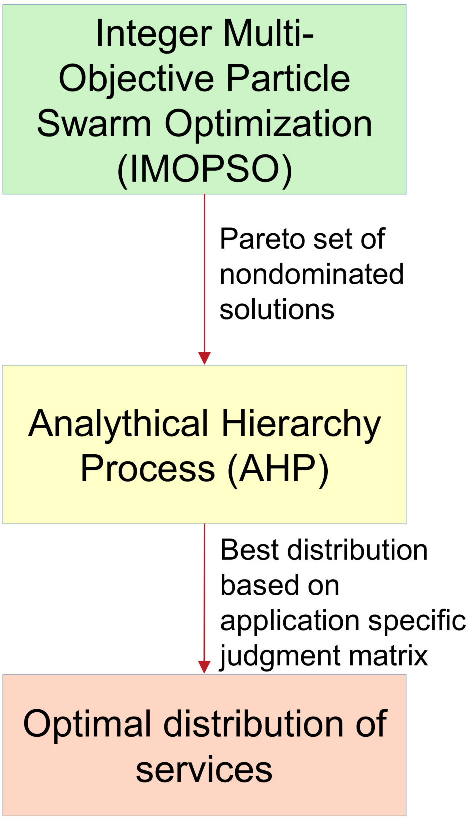

Integer Multi-Objective Particle Swarm Optimization (IMOPSO)
Pareto set of nondominated solutions
Analythical Hierarchy Process (AHP)
Best distribution based on application specific judgment matrix
Optimal distribution of services

Figure 4. Flow chart of the proposed service distribution optimization process.

The optimization process has two main steps—multi-objective optimization and a multi-objective decision, but the problem must be expressed as a formal mathematical model before using any formal optimization methods. The following subsections describe the optimization process in detail. We summarize the key notations used in this paper in Table 1 in order to give a description of the optimization process.

Table 1. Key notations used in this paper.

Notation	Description
n	Total number of services
k	Total number of available fog nodes for hosting the services
$X_i = (x_1, x_2, \ldots, x_n)^T$, $x_j \in \{1, 2, \ldots, k\}, j = 1, 2, \ldots, n$	i-th possible distribution of services among the fog nodes, also the position of the i-th particle in n-dimensional definition area
m	Total number of evaluation criteria, also the number of objective functions
$f_j(x), j = 1, 2, \ldots, m$	j-th evaluation criterion, objective function
$F_i = (f_1(X_i), f_2(X_i), \ \ldots, f_m(X_i))^T$	Score vector of the i-th particle
V_i, $V_i \in R^n$	Velocity of the i-th particle
$pBest_i$	The best score of the i-th particle
$pBPos_i$	The best position of the i-th particle
$gBest$	The best global score of all the particles
$gBPos$	Position of the particle with the best global score
S	Set of particles, swarm
R	External repository of particles, a set of Pareto optimal solutions
X_{opt}	The best service distribution among all the available fog nodes, particle with the best score

4.1. The Optimization Model of a Service Distribution Problem

The main task of this optimization procedure is to find an optimal distribution of n services among possible k fog nodes. Each fog node may have slightly different characteristics, but we assume that all the nodes are capable of running all the services. The goal of optimization is to distribute all the services in such a way that a set of important characteristics is optimal. Characteristics of the i-th possible service distribution X_i are expressed by the values of the objective functions $f_j(X_i)$, $j = 1, 2, \ldots, m$ and constraint conditions. The objective of the optimization process is to find the best service distribution X_{opt} which minimizes all the objective functions f_j, i.e., we have a multi-objective optimization problem:

$$X_{opt} = \arg\underset{i}{min} F(X_i) \tag{1}$$

where $F(x) = \{f_1(x),\ f_2(x),\ \ldots,\ f_m(x)\}$ is a set of the objective functions, and $x \in \{X_i\}$ is a member of the set with all the possible service distributions.

Constraint conditions are expressed by the following equations:

$$\begin{cases} g_j(X_i) \geq 0, j = 1, 2, \ldots, n_g \\ h_K(X_i) = 0, k = 1, 2, \ldots, n_h \end{cases} \tag{2}$$

4.2. Objective Functions

Different fog nodes have different performance, network bandwidth, and security characteristics. Different distributions of services among the fog nodes may produce a working system with slightly different characteristics. For example, if one fog node supports a lower level of a security (due to limited hardware capabilities), and an important service is placed in this node, then the overall security of the whole system is reduced to the security of the least secure fog node. We consider multiple objective functions ($f_j(.)$, $j = 1, 2, \ldots, m$) to evaluate all such situations, which include: overall security of the system, CPU utilization, RAM utilization, power utilization, range, etc. Some objective functions which were used in our experiments are provided in the following paragraphs.

A security of the whole system while using the i-th service distribution $f_{sec}(X_i)$ is defined by the lowest security of all the services. We assign security levels (expressed in security bits, according to the NIST publication [32]) to fog nodes based on their capabilities to support corresponding security protocols. We assume that services are capable working on all the fog nodes, then a value of the security criteria function $f_{sec}(X_i)$ for the service distribution X_i is the lowest security level of all the fog nodes in which at least one service is hosted. For example, if we have a situation where three fog nodes are able to provide 128 bits of security and one fog node is constrained to support only 86 bits of security, and if at least one service is hosted by it, then the overall security of the service distribution is equal to 86 bits, i.e., the value of the objective function $f_{sec}(X_i) = 86$. If we use our services in the application area which requires a specific level of security, then such a requirement is expressed as a constraint condition, i.e., if some application area requires at least 128 bits of security, then we have a corresponding constraint $g_{sec}(X_i) \geq 128$.

The criterion of CPU usage $f_{CPU}(.)$ evaluates how evenly, CPU utilization-wise, the services are distributed among the fog nodes. The main idea here is to try to decrease the overall CPU utilization of the system to allow hosting of additional services more easily in the case they occur during the runtime of the system. Each fog node has its CPU capabilities expressed in MIPS, which depend on HW capabilities of the corresponding fog node. All services are also evaluated for required CPU resources. To calculate the value of CPU usage of the whole system while using the i-th service distribution $f_{CPU}(X_i)$, we first calculate a relative CPU usage for each fog node (dividing the sum of CPU resources required by all the services hosted in each fog node by the capabilities of the corresponding fog nodes) and we find afterwards the maximum CPU utilization among all the fog nodes. The lower the maximum CPU utilization is, the better service distribution we have. We can obtain this situation while using this method of calculation, when some service distributions make up a CPU allocation greater than 100% in some fog nodes and, therefore, corresponding constraints are added to the optimization problem. The usage of this criterion automatically solves some frequent restrictions and incompatibilities, i.e., situations when some services require CPU resources which may not be provided by some fog nodes.

The criterion of RAM usage $f_{RAM}(.)$ which evaluates how evenly RAM utilization is distributed among any fog nodes hosting the services is very similar to CPU usage. The calculation of this criterion is the same as the calculation of CPU usage. A constraint which does not allow exceeding 100% of the RAM utilization in each fog node is also added.

A criterion of the power usage $f_{pw}(X_i)$ of possible service distribution X_i is evaluated using the average power requirements of each service (expressed in mW) and the available power of fog nodes (expressed in mW). The main objective of this evaluation is to maximize the overall runtime of the system. A calculation is performed by dividing the sum of power requirements of all the services hosted in each fog node by the available power of a corresponding fog node to find the maximum among all the fog nodes. A distribution of services is better in such a case when all the fog nodes are evenly loaded power-wise, i.e., the maximum power utilization is minimized.

The communication of fog devices with sensors and actuators is affected by the physical range between devices in some cases. Some communications protocols add strict requirements for the range as some of them may be less efficient if the communication range is increased. A criterion of the maximum range $f_{rng}(.)$ may be used to assess these properties. In this study, a criterion of the range is calculated by averaging the range of each fog node location with respect to all the devices the particular fog node is communicating with to find the maximum of these ranges among all the fog nodes hosting at least one service which requires communication with end devices. The main idea of this criterion is to prefer a shorter communication path as it ensures better performance in most cases. Any corresponding constraints on the range may be also added if a communication protocol induces such restrictions.

Other application-specific criteria such as local storage capabilities, communication latency, bandwidth, etc. may also be evaluated, defining corresponding objective functions representing system characteristics which are important in a particular application scenario. All specific implementations of the criteria evaluation functions $f_i(.)$ are implementation specific and are out of the scope of this paper. The proposed optimization procedure is not limited to any specific amount or nature of the objective functions as long as they follow a few common criteria:

- A return value of the objective function must be a positive real number.
- Better values of the criteria must be expressed by smaller numbers (this is because the particle swarm optimization method searches for a minimum of the function).

Generally, one common feature of all these objective functions is that they are mutually exclusive. Any optimization of one objective will often be at the expense of affecting the other one. For example, we may consider moving all the services to more secure fog nodes to increase security, but such a service distribution will likely cause reduced power efficiency, excessive load on some of the nodes, and a lower overall runtime of the system. Moreover, different objectives have different measurement units, e.g., security may be evaluated in bits while the power requirement of the services is measured in Watts, any available network bandwidth is measured in kbps, etc. Even if all the measurements are converted to real positive numbers, it is still very difficult to objectively compare them. There is no single solution to a multi-objective optimization problem that optimizes all the objectives at the same time. The objective functions are contradictory in this situation, therefore a set of non-dominated (Pareto optimal) solutions can be found. We propose to use a two-stage optimization procedure (see Figure 4) in order to deal with this situation, where the first step will use a multi-objective optimization to find a set of solutions (possible distributions of services), the elements of which are a Pareto optimal. We propose to use for this the integer multi-objective particle swarm optimization (IMOPSO) method described in the next paragraph. A choice of the particle swarm optimization method is based on the research of other authors [33–35] which shows that this method is suitable for a similar class of problems, and it demonstrates good results. We will use the analytical hierarchy process (AHP) in the second step to choose the best solution from a Pareto optimal set.

4.3. IMOPSO for Finding a Pareto Set of Possible Service Distributions

The original particle swarm optimization (PSO) algorithm is best suited for an optimization of continuous problems, but several modifications [36,37] exist, which enable it to be used for discrete problems. In the case of multiple objectives which contradict each other, the PSO algorithm may be adapted to find a Pareto optimal set of solutions [38,39]. We used the Multi-objective particle swarm optimization (MOPSO) method proposed by Coello et. all in [39] to find a Pareto set of the possible service distributions among fog nodes. In order to use this method, we had to slightly adapt it for it to work in the constrained integer n-dimensional space of possible distributions of services represented as the particles of a swarm (the original method uses a continuous real number space).

We used the vector $X_i = (x_1, x_2, \ldots, x_n)^T$, $x_j \in \{1, 2, \ldots, k\}$, $j = 1, 2, \ldots, n$ to encode the i-th distribution of services, where n is the number of services which have to be distributed among k fog nodes. The meaning of the vector element $x_j = l$ is that the j-th service must be placed in the l-th fog node.

A flow diagram of the integer multi-objective particle swarm optimization (IMOPSO) algorithm is shown in Figure 5.

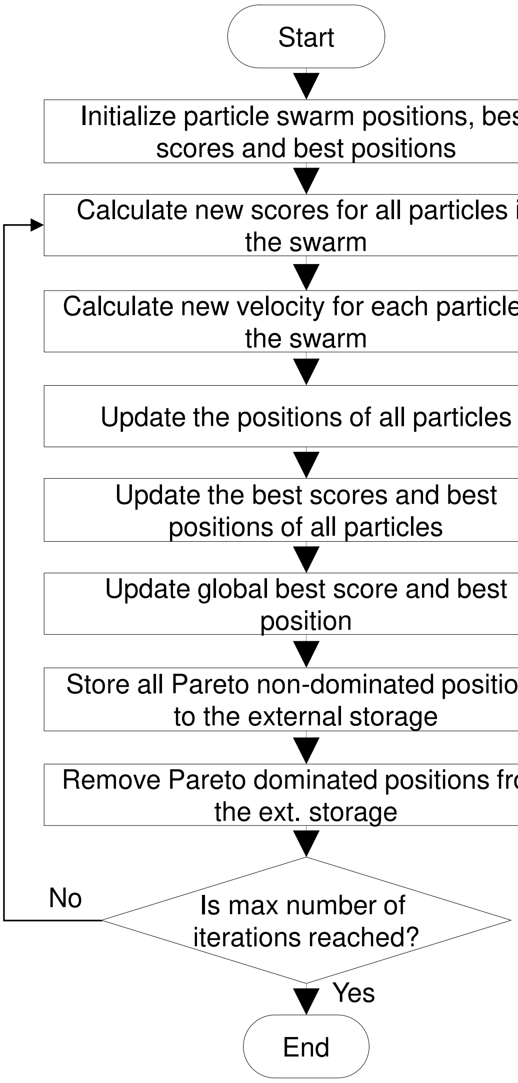

Start
Initialize particle swarm positions, best scores and best positions
Calculate new scores for all particles in the swarm
Calculate new velocity for each particle in the swarm
Update the positions of all particles
Update the best scores and best positions of all particles
Update global best score and best position
Store all Pareto non-dominated positions to the external storage
Remove Pareto dominated positions from the ext. storage
Is max number of iterations reached?
No
Yes
End

Figure 5. Flow chart of IMOPSO algorithm.

The main steps of the IMOPSO algorithm are the following:

1. Initialize the particle swarm S by randomly generating an initial set of positions of the particles (possible service distributions) X_i, $i = 1, 2, \ldots, |S|$, where $|S|$ is the initial size of a particle swarm.
2. Initialize the velocities $V_i = \overrightarrow{0}$, the best scores $pBest_i = \overrightarrow{\imath nf}$, and the best positions $pBPos_i = X_i$ of all the particles in the swarm S. Initialize the global best position $gBPos = \overrightarrow{0}$ and the global best score $gBPos = \overrightarrow{\imath nf}$.
3. Repeat it until a maximum number of iterations is reached:
   - Evaluate the new scores F_i of all the particles in the swarm S using all the objective functions: $F_i = (f_1(X_i), f_2(X_i), \ldots, f_m(X_i))^T$, $i = 1, 2, \ldots, |S|$.
   - Calculate new velocities of each particle using the expression $V_i = wV_i + r_1(pBPos_i - X_i) + r_2(gBPos - X_i)$, where w is an inertia weight (initially a real value around 0.4); r_1 and r_2 are random numbers in the range of $[0..1]$; V_i is the velocity of the i-th particle; $pBPos_i$ is the position of the i-th particle with the best score; X_i is the current position of the i-th particle; and $gBPos$ is the position of the particle with the best global score.
   - Update positions of all the particles in the swarm: $X_i = round(X_i + V_i)$, $i = 1, 2, \ldots, |S|$. The position is approximated to the nearest integer value.

If the particle is out of the range, give it the opposite direction of the speed ($V_i = -V_i$), and set the position X_i to the edge of the range of its definition.

- Update all the best scores $pBest_i$ and the best positions $pBPos_i$ of all the particles in the swarm $i = 1, 2, \ldots, |S|$. If the new score F_i dominates the current best score $pBest_i$, then update the best position and the best score of the i-th particle. If the new score neither dominates nor is dominated by the current best score of the i-th particle, then set the best position (and the best score) of the particle to a new position with a probability of 0.5.
- Update the global best score $gBest$ and the global best position $gBPos$ of the particles using the same algorithm used for updating the best scores of the individual particles.
- Store the positions and scores of the particles that are non-dominated in the external repository (set R). Use all the available particles in the sets S and R during any dominance comparison.
- Analyze the repository R and remove all the duplicated and dominated scores.

4. The external repository R is a set of Pareto optimal solutions.

4.4. Finding an Optimal Service Distribution Using the AHP

We used the analytical hierarchy process (AHP) [40,41] to choose the best solution from a Pareto optimal set by using pairwise comparisons of all non-dominated distributions of services using all the available objective functions. The AHP is usually used in situations where a decision must be made using a small amount of quantitative data, using a deep analysis performed by several decision-making parties, by applying a pair-to-pair comparison of possible solutions. The AHP may be adapted to be used by machine-based decision making in the scenarios where complex multiple criteria problems are evaluated [42–44]. The choice of the AHP instead of other more formal multi-criteria decision-making algorithms is based on the following reasons [40].

The AHP allows one to automatically check the consistency of the evaluations provided by decision makers. The AHP uses normalized values of criteria, so it allows one to use heterogeneous measurement scales for different criteria. For example, one can use a purely qualitative scale for the security (high, low, medium) and use inconsistent numeric scales for any power and CPU requirements at the same moment. The AHP uses pairwise comparisons of the alternatives only, which eases multi-objective decision making to obtain improved reliability of the results. The importance of the criteria used in the AHP is also evaluated using the same methodology, which allows one to skip the most controversial step of a manual weight assignment to different criteria.

A three-level hierarchical structure of the AHP is generalized in Figure 6. Level one is an objective of the process which in our case is to choose an optimal distribution of all the available services among fog nodes. The second level is the criteria, which are the same as the objective functions used in the IMOPSO part of the optimization process. An important step in this level is to use the same AHP to find the weight of all the criteria by using a pairwise comparison of the criteria. This step should be done manually before putting an automatic service allocation algorithm into production. Moreover, a step of the evaluation of criterion importance should be different based on the application area in which a service orchestrator is applied. For example, security may be evaluated as more important than power efficiency in a healthcare application compared to a home automation application. We assume in our algorithm that the step of the evaluation of criterion importance is already performed, and the decision-making system already has its judgment matrix with all the required weights of all the criteria in level 2.

The third level is alternatives. These are filled with all the Pareto optimal solutions from a previous step of the optimization process using the IMOPSO method. Then, the AHP is started to choose the best alternative. The whole process is summarized in Figure 7.

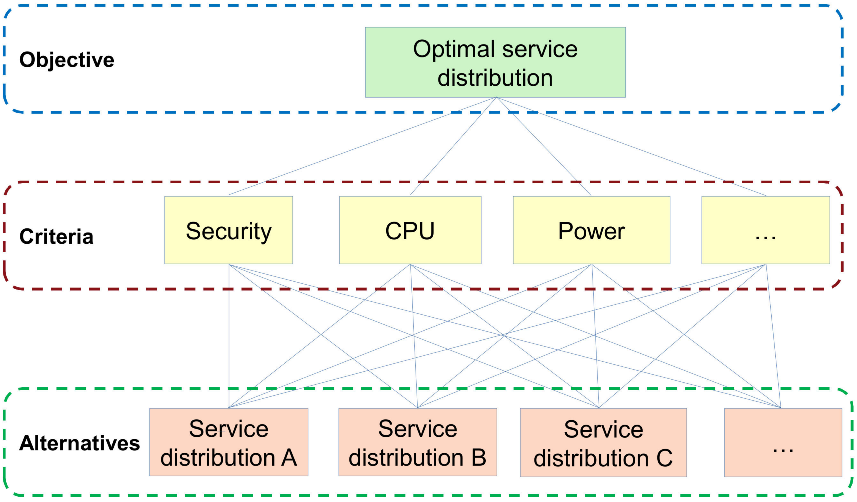


Figure 6. Hierarchical framework for AHP.

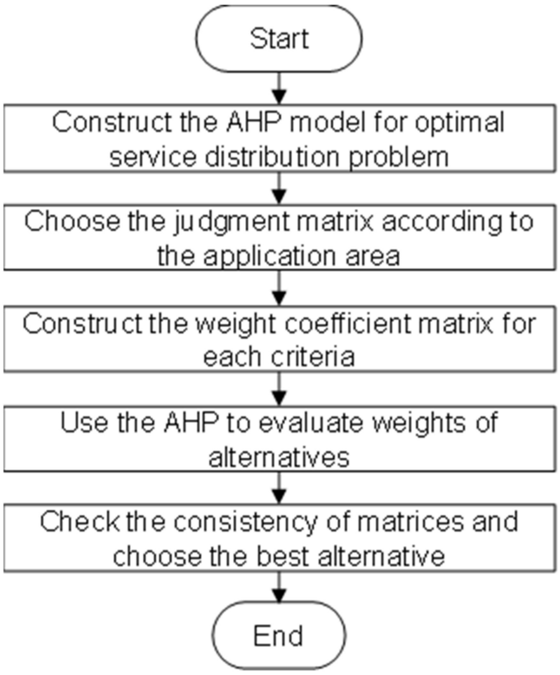

Start
Construct the AHP model for optimal service distribution problem
Choose the judgment matrix according to the application area
Construct the weight coefficient matrix for each criteria
Use the AHP to evaluate weights of alternatives
Check the consistency of matrices and choose the best alternative
End

Figure 7. Process of AHP decision making.

The main steps of the AHP are the following:

1. Construct a corresponding AHP framework using a Pareto optimal solution set to prepare all the data structures for a comparison of alternatives using all the available criteria.
2. Load a judgment matrix with the results of pairwise comparisons of criteria prepared to be used in the current application area.
3. Repeat the following step for each criterion (objective function) $f_k(.)$, $k = 1, 2, \ldots, m$:
   - Construct the weight coefficient matrix $M_k = (m_{i,j})$ using all the alternatives in the Pareto optimal solution set R. The size of the matrix M_k is $s \times s$, where $s = |R|$; $m_{i,j} \in (0, 9]$; $m_{i,j} = \frac{1}{m_{j,i}}$; $m_{i,i} = 1$; $i, j = 1,\ 2,\ \ldots,\ s$. The matrix M_k elements are calculated using special comparison functions $m_{i,j} = comp_k(X_i, X_j)$ which use a corresponding objective function $f_k(.)$, which calculates two objective function values $f_k(X_i)$ and $f_k(X_j)$, and compares them with each other to transform the result into the required real number from the interval $(0, 9]$. A comparison

function heavily depends on the meaning of the criteria and the corresponding real number represents a preference of one alternative over another [35].

4. Provide all the created matrices to the standard AHP decision-making method to obtain any estimated weights of all the alternatives.
5. Check the consistency of the provided matrices using consistency indicators provided by the AHP. Choose the best alternative as the final solution of an optimization process.

5. Implementation and Evaluation

Implementation results of our method are summarized in this section with a discussion on each result. The implementation of a real fog computing environment with a measurement of all the parameters used in the service placement decisions is out of the scope of this paper, and it also makes it difficult to scale the solution and reproduce the results, therefore, we used a simulation. The main objective is to show how the proposed method performs in different situations as well as to test the feasibility of the proposed service placement method.

We implemented the proposed optimal service placement-finding method using Matlab. The implementation uses as an input some basic performance information on the fog nodes and services, a set of the objective functions, and an application area-specific judgment matrix J. The method performs integer multi-objective particle swarm optimization, finds a Pareto optimal set of solutions, automatically performs an AHP using a provided judgment matrix, and finds the best placement of the services in the fog nodes.

5.1. Illustrative Scenario

We used an illustrative scenario to evaluate the characteristics of the proposed method. We have 4 fog nodes and 13 services in this scenario, and they must be optimally placed in those fog nodes. Capabilities of the fog nodes and requirements of the services are chosen to show how the proposed method performs in different situations. We used several papers [19,45,46] analyzing various requirements of real hardware and software IoT systems to provide realistic numbers. A summary of the fog node parameters and requirements of the services are presented in Tables 2 and 3.

A security level of any fog device is determined by the hardware and software capabilities as well as by the availability of corresponding libraries, and it is expressed in bits according to the NIST guidelines [32].

All services are divided into three main groups. Sense1, Sense2, and Sense3 services are primarily used to communicate with any corresponding sensor devices, collect the measurement data, and provide it to the other services for processing. On the other hand, services Actuate1, Actuate2, and Actuate3 are mainly used to communicate with the actuator devices. The rest of the services are primarily used to collect data, perform calculations, and make decisions. Resource requirements of the services from different classes are very different.

Table 2. Resources available in the fog nodes.

	Power (mW)	**CPU (MIPS)**	**RAM (MB)**	**Security (bits)**
Fog1	1000	2000	512	256
Fog2	2000	1000	256	112
Fog3	1000	1000	256	128
Fog4	2000	500	512	86

Table 3. Resources required by the services.

Service	Processing Power (mW)	Transfer Power (mW)	CPU (MIPS)	RAM (MB)
Sense1	5	20	50	10
Sense2	5	25	60	15
Sense3	5	20	50	20
Process1	100	0	200	60
Process2	150	0	250	75
Process3	130	0	230	70
Process4	120	0	300	50
Process5	120	0	240	80
Process6	140	0	250	55
Process7	200	0	200	70
Actuate1	4	21	50	10
Actuate2	5	20	60	15
Actuate3	4	19	50	10

We used a “dynamic” objective function representing power requirements of the service to better illustrate the capabilities of our method. The power requirements of the service depend on which fog node is used to host this service. This is achieved by dividing the power requirements into two parts: processing and transfer power. The processing power is constant, and it is always required to perform an operation (the values of power requirements were taken from the publication [19]). On the other hand, the transfer power presented in Table 3 is required if no security is used to transfer the data (i.e., a plain http protocol is used). The information on required power levels for a data transfer without any security is based on the experimental results presented in the paper [47]. When the service is placed in a fog node providing more security, then the corresponding requirement for a transfer power is increased. For example, if a service is placed in a fog node providing 86 bits of security (e.g., this node is using 1024-bit RSA for a key agreement), then the corresponding transfer power is multiplied by a coefficient of 1.5. The transfer power increase coefficients were based on the results presented in [45] and [48]. We decided after an analysis of the provided data to use these multipliers for modeling the increase in power due to increased security: 1.5 for 86 bits of security, 2.25 for 112 bits, 4 for 128 bits, and 7.5 for 256 bits of security.

5.2. Evaluation Results

We use a simplified scenario where only two objective functions are used to show how the IMOPSO algorithm works and how the Pareto set of solutions looks. A Pareto set may be displayed in this case using a two-dimensional chart. A judgment matrix used in this case consists only of 4 elements:

$$J = \begin{pmatrix} 1 & 3 \\ 1/3 & 1 \end{pmatrix} \tag{3}$$

If two objective functions, RAM and CPU, are used, then this judgment matrix means that an even RAM usage distribution among all the fog nodes is more important than an even CPU usage. A Pareto set produced by the IMOPSO algorithm is presented in Figure 8. Then, a Pareto solution set is used in the second stage, employing an AHP, to find the best placement of services. The best placement is summarized in Table 4.

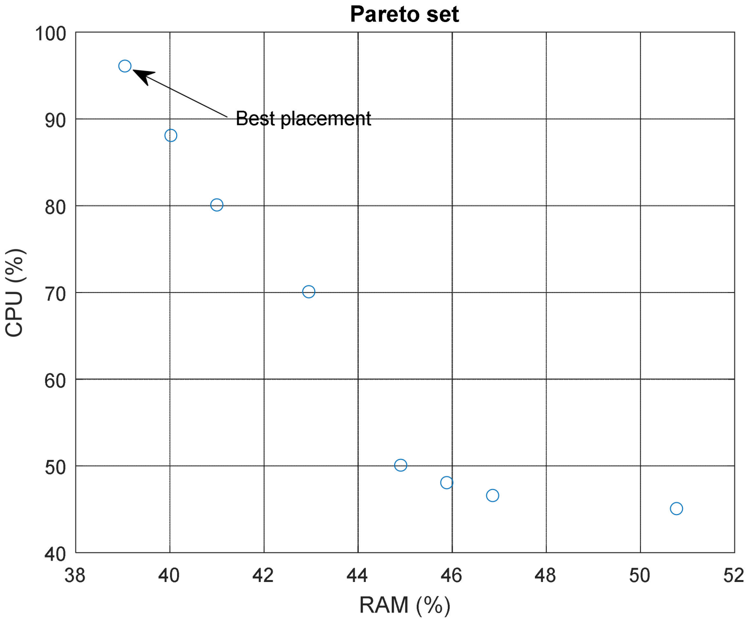


Figure 8. Pareto set of a simplified scenario.

Table 4. The best service placement in a simplified scenario.

	Fog1	Fog2	Fog3	Fog4
Services	Actuate3 Process4 Process5 Process6	Sense1 Sense2 Actuate2 Process1	Sense3 Process2	Actuate1 Process3 Process7

The best score (the best values of the objective functions) in this case is $(39, 96)^T$, meaning that this service placement ensures a maximal RAM usage of 39% among all four fog nodes. The maximal usage of CPU is 96% in this case.

The second scenario shows an influence of the judgment matrix on the optimal placement of services. Four objective functions are used in this case: power, CPU, security, and RAM. The first judgment matrix prioritizes security and energy over the CPU and RAM:

$$J_1 = \begin{pmatrix} 1 & 3 & 1/6 & 3 \\ 1/3 & 1 & 1/6 & 1 \\ 6 & 6 & 1 & 6 \\ 1/3 & 1 & 1/6 & 1 \end{pmatrix} \tag{4}$$

The second judgment matrix prioritizes an even power consumption:

$$J_2 = \begin{pmatrix} 1 & 7 & 3 & 6 \\ 1/7 & 1 & 1/2 & 1 \\ 1/3 & 2 & 1 & 2 \\ 1/6 & 1 & 1/2 & 1 \end{pmatrix} \tag{5}$$

A Pareto set of solutions using the judgment matrix J_1 is shown in Figure 9. Only some projections of the set are shown as the set members are four-dimensional vectors and they cannot be fully rendered in charts.

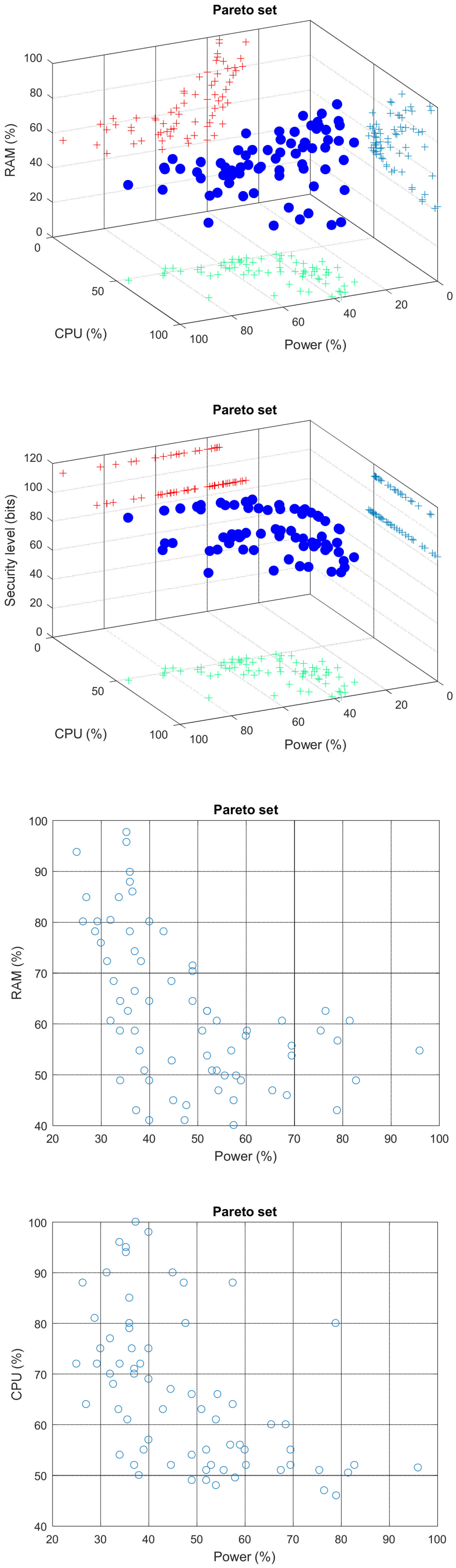


Figure 9. Pareto set of the second scenario, security and energy are prioritized.

The best placement of services is presented in Table 5. The best score in this case is $(35,\ 94,\ 112,\ 98)^T$. The overall security (determined by a security level of the least security-capable fog node hosting at least one service) is 112 bits in this case, and the fog node Fog4 is not hosting any services, as its security is only 86 bits.

Table 5. Best service placement in the second scenario, security is prioritized over other criteria.

	Fog1	Fog2	Fog3	Fog4
Services	Process2 Process7	Sense1, Sense2, Sense3, Process1, Process3, Process6 Activate1, Activate3	Process4 Process5 Activate2	-

If the judgment matrix J_2, which prioritizes even power consumption, is used in the same situation, then the best placement is different (see Table 6), and the best score is $(26,\ 88,\ 86,\ 84)^T$.

Table 6. Best service placement in the second scenario, power is prioritized over other criteria.

	Fog1	Fog2	Fog3	Fog4
Services	Process5 Process6	Process2, Process3, Process4, Activate1, Activate3	Process1	Sense1, Sense2, Sense3, Process7, Activate2

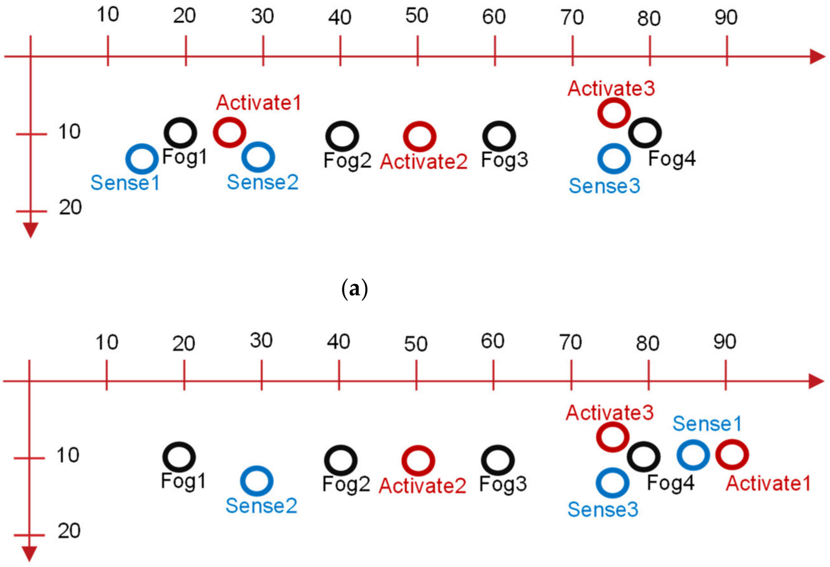
The maximal power consumption among all the fog nodes is 26% in this case, and it is significantly better than in the first variant (35%), but the overall security of the solution is degraded to 86 bits, as several services are placed in the fog node Fog4.

The third illustrative scenario is meant to illustrate how the proposed service placement method works in cases when some devices change their positions, and corresponding services must be reallocated. We use an objective function considering the range from a physical sensor device to the service monitoring device which is physically placed in one of the fog nodes to demonstrate this scenario. The range in this case is only important for services which are communicating with sensors or actuators. The range is considered 0 independently of the fog nodes they are hosted in with the services which are processing data. A judgment matrix prioritizing the range is used in this scenario, while the objective functions in this case are: range, CPU, security, RAM.

$$J_3 = \begin{pmatrix} 1 & 5 & 3 & 5 \\ 1/5 & 1 & 1/2 & 1 \\ 1/3 & 2 & 1 & 2 \\ 1/5 & 1 & 1/2 & 1 \end{pmatrix} \tag{6}$$

We used the data presented in the diagram (see Figure 10) to model the placement of the services. All coordinates here are presented in meters.

The best service placements in each case are summarized in Tables 7 and 8, and the corresponding scores are $(13,\ 67,\ 128,\ 73)^T$ and $(9,\ 54,\ 86,\ 55)^T$.



(**a**)

(**b**)

Figure 10. Service placement diagram. (**a**) Initial placement, (**b**) modified placement.

Table 7. Best service placement in the third scenario, the initial placement of sensors and actuators.

	Fog1	Fog2	Fog3	Fog4
Services	Sense1, Sense2, Activate1, Process1, Process2, Process3, Process5, Process6.	-	Sense3, Activate2, Activate3, Process4, Process7.	-

Table 8. Best service placement in the third scenario, the placement of sensors and actuators after changes in their location.

	Fog1	Fog2	Fog3	Fog4
Services	Process1, Process2, Process4, Process7	Sense2, Activate1, Process5	Process3, Process6	Sense1, Sense3, Activate1, Activate3

The evaluation results clearly show that if the range is the most important objective function, then the services are more likely to be placed in the adjacent fog nodes. On the other hand, if more sensors are located near a less secure fog node, then the overall security of the solutions may decrease (128 bits vs. 86 bits in the second scenario).

6. Discussion, Conclusions, and Future Work

An increase in IoT-based services has led to a need for more efficient means of handling resources in systems comprising heterogeneous devices. A fog computing paradigm brings computational resources closer to the edge of the cloud, but energy-, communication-, and computation resource-constrained devices dominate near the edge. Different application areas (healthcare, multimedia, home automation, etc.) require different characteristics of the IoT system. The usage of various heterogeneous devices leads to difficulties in predicting how much of the resources would be required within the fog nodes when all the services are going to allocate all the resources they need. Moreover, the need for roaming services which follow the actors (i.e., a person is moving inside a building, cars, etc.) arises due to the limitations of some hardware devices (i.e., a limited range of communication protocols), and therefore the resources in the fog nodes need to be reallocated in this case every time the situation changes. The best way to deal with these dynamic service reallocations is to use service orchestrators, which decide the best way to allocate and move, start, and stop any corresponding services as needed. One of the main challenges while designing an effective service orchestrator is the need for a specialized method to obtain an optimized service placement inside the available fog nodes.

A new optimization method for an optimal distribution of services among available fog nodes was proposed in this paper. The two-stage method uses integer multi-objective particle swarm optimization to find a Pareto optimal set of solutions and the analytical hierarchy process using an application-specific judgment matrix for a decision on any optimal distribution of services. Such a processing distribution allows one to assess different heterogeneous criteria with different units of a measurement and different natures (qualitative or quantitative). The method, apart from providing one best solution, also ranks all the Pareto optimal solutions, enabling one to compare them with each other (answering the question "how much better is one solution than the other?") and, if needed, to choose the second best, the third best, etc. solution.

The proposed method effectively works with the whole range of objective functions (evaluation criteria), which could be easily expanded by new objective functions representing different criteria. Moreover, the objective functions may be dynamic, meaning that not only the value but also the algorithm of an objective function calculation may be different based on the service placement in particular fog nodes with particular software and hardware capabilities.

If the same end device, service, and fog device set is used in a different application area (i.e., healthcare vs. home automation) which requires different prioritization of criteria (i.e., security is more important in healthcare compared to home automation) then only the AHP judgment matrix must be changed. The method adapts to the situation and provides appropriate results.

A number of interesting aspects of the proposed method could be explored in the future. It would be interesting to use it in a real orchestrator of IoT infrastructure to practically evaluate how different placements of services inside fog nodes influence the performance of the whole IoT system. Another very interesting aspect to investigate is objective function construction according to the experimentally obtained real-life results involving all the interrelations among different criteria. An experiment using real hardware and software would help to estimate some additional aspects of the proposed algorithm, including the performance under different configurations of the infrastructure (number of fog nodes, number of end devices, etc.) and different architectures of the corresponding devices (supporting parallel processing, optimization using CPU or GPU, offloading an optimization task to the cloud services, etc.).

We believe that the results of this work will be useful in further research in the area of IoT fog computing service orchestration, and it will allow researchers to develop more efficient IoT systems.

Author Contributions: Conceptualization, A.V., N.M., N.Š.; investigation, N.M., A.V., J.T.; methodology, A.V., N.M., N.Š.; resources, A.V.; software, N.M., J.T.; supervision, A.V.; visualization, N.M., N.Š.; writing—original draft, A.V., N.M., N.Š.; writing—review and editing, A.V., N.M., N.Š., J.T.; funding acquisition, A.V. All authors contributed to the final version. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported in part by the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 830892, project "Strategic programs for advanced re-search and technology in Europe" (SPARTA).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Velasquez, K.; Abreu, D.P.; Goncalves, D.; Bittencourt, L.; Curado, M.; Monteiro, E.; Madeira, E. Service Orchestration in Fog Environments. In Proceedings of the 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), Prague, Czech Republic, 21–23 August 2017; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2017; pp. 329–336.
2. Javadzadeh, G.; Rahmani, A.M. Fog Computing Applications in Smart Cities: A Systematic Survey. *Wirel. Netw.* **2020**, *26*, 1433–1457. [CrossRef]
3. Imrith, V.N.; Ranaweera, P.; Jugurnauth, R.A.; Liyanage, M. Dynamic Orchestration of Security Services at Fog Nodes for 5G IoT. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2020; pp. 1–6.
4. Kayes, A.; Rahayu, W.; Watters, P.; Alazab, M.; Dillon, T.; Chang, E. Achieving security scalability and flexibility using Fog-Based Context-Aware Access Control. *Futur. Gener. Comput. Syst.* **2020**, *107*, 307–323. [CrossRef]
5. De Brito, M.S.; Hoque, S.; Magedanz, T.; Steinke, R.; Willner, A.; Nehls, D.; Keils, O.; Schreiner, F. A service orchestration architecture for Fog-enabled infrastructures. In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, Spain, 8–11 May 2017; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2017; pp. 127–132.
6. Dsouza, C.; Ahn, G.-J.; Taguinod, M. Policy-driven security management for fog computing: Preliminary framework and a case study. In Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014), Redwood City, CA, USA, 13–15 August 2014; IEEE: New York, NY, USA, 2014; pp. 16–23.
7. Capossele, A.; Cervo, V.; De Cicco, G.; Petrioli, C. Security as a CoAP resource: An optimized DTLS implementation for the IoT. In Proceedings of the 2015 IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; IEEE: New York, NY, USA, 2015; pp. 549–554.
8. Jamil, B.; Shojafar, M.; Ahmed, I.; Ullah, A.; Munir, K.; Ijaz, H. A job scheduling algorithm for delay and performance optimization in fog computing. *Concurr. Comput. Pr. Exp.* **2019**, *32*, 5581. [CrossRef]
9. Aslanpour, M.S.; Gill, S.S.; Toosi, A.N. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet Things* **2020**, *12*, 100273. [CrossRef]
10. Tsai, J.-S.; Chuang, I.-H.; Liu, J.-J.; Kuo, Y.-H.; Liao, W. QoS-Aware Fog Service Orchestration for Industrial Internet of Things. *IEEE Trans. Serv. Comput.* **2020**, 1. [CrossRef]
11. De Sousa, N.F.S.; Perez, D.A.L.; Rosa, R.V.; Santos, M.; Rothenberg, C.E. Network Service Orchestration: A survey. *Comput. Commun.* **2019**, *142–143*, 69–94. [CrossRef]
12. Šatkauskas, N.; Venčkauskas, A.; Morkevičius, N.; Liutkevičius, A. Orchestration Security Challenges in the Fog Compu-ting. In *Communications in Computer and Information Science, Proceedings of the International Conference on Information and Software Technologies, Kaunas, Lithuania, 15–17 October 2020*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 196–207.
13. Velasquez, K.; Abreu, D.P.; Assis, M.R.M.; Senna, C.; Aranha, D.; Bittencourt, L.F.; Laranjeiro, N.; Curado, M.; Vieira, M.; Monteiro, E.; et al. Fog orchestration for the Internet of Everything: State-of-the-art and research challenges. *J. Internet Serv. Appl.* **2018**, *9*, 14. [CrossRef]
14. Desikan, K.S.; Kotagi, V.J.; Murthy, C.S.R. Topology Control in Fog Computing Enabled IoT Networks for Smart Cities. *Comput. Netw.* **2020**, *176*, 107270. [CrossRef]
15. Zahmatkesh, H.; Al-Turjman, F. Fog computing for sustainable smart cities in the IoT era: Caching techniques and enabling technologies—An overview. *Sustain. Cities Soc.* **2020**, *59*, 102139. [CrossRef]
16. Zhang, C. Design and application of fog computing and Internet of Things service platform for smart city. *Futur. Gener. Comput. Syst.* **2020**, *112*, 630–640. [CrossRef]
17. Yang, R.; Wen, Z.; McKee, D.; Lin, T.; Xu, J.; Garraghan, P. Fog Orchestration and Simulation for IoT Services. In *Fog and Fogonomics*; Yang, Y., Huang, J., Zhang, T., Weinman, J., Eds.; Wiley: Hoboken, NJ, USA, 2020; pp. 179–212. ISBN 978-1-119-50109-1.
18. Wen, Z.; Yang, R.; Garraghan, P.; Lin, T.; Xu, J.; Rovatsos, M. Fog Orchestration for Internet of Things Services. *IEEE Internet Comput.* **2017**, *21*, 16–24. [CrossRef]
19. Skarlat, O.; Nardelli, M.; Schulte, S.; Borkowski, M.; Leitner, P. Optimized IoT service placement in the fog. *Serv. Oriented Comput. Appl.* **2017**, *11*, 427–443. [CrossRef]
20. Naha, R.K.; Garg, S.; Chan, A.; Battula, S.K. Deadline-based dynamic resource allocation and provisioning algorithms in Fog-Cloud environment. *Futur. Gener. Comput. Syst.* **2020**, *104*, 131–141. [CrossRef]
21. Khebbeb, K.; Hameurlain, N.; Belala, F. A Maude-Based rewriting approach to model and verify Cloud/Fog self-adaptation and orchestration. *J. Syst. Arch.* **2020**, *110*, 101821. [CrossRef]
22. Tuli, S.; Poojara, S.R.; Srirama, S.N.; Casale, G.; Jennings, N.R. COSCO: Container Orchestration using Co-Simulation and Gradient Based Optimization for Fog Computing Environments. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 1. [CrossRef]
23. Rahbari, D.; Nickray, M. Task offloading in mobile fog computing by classification and regression tree. *Peer Peer Netw. Appl.* **2019**, *13*, 104–122. [CrossRef]
24. Arkian, H.R.; Diyanat, A.; Pourkhalili, A. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *J. Netw. Comput. Appl.* **2017**, *82*, 152–165. [CrossRef]

25. Velasquez, K.; Abreu, D.P.; Curado, M.; Monteiro, E. Service placement for latency reduction in the internet of things. *Ann. Telecommun.* **2016**, *72*, 105–115. [CrossRef]
26. Brogi, A.; Forti, S.; Ibrahim, A. How to Best Deploy Your Fog Applications, Probably. In Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, Spain, 14–15 May 2017; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2017; pp. 105–114.
27. Urgaonkar, R.; Wang, S.; He, T.; Zafer, M.; Chan, K.; Leung, K.K. Dynamic service migration and workload scheduling in edge-clouds. *Perform. Eval.* **2015**, *91*, 205–228. [CrossRef]
28. Bellendorf, J.; Mann, Z. Ádám Classification of optimization problems in fog computing. *Futur. Gener. Comput. Syst.* **2020**, *107*, 158–176. [CrossRef]
29. Huang, X.; Ganapathy, S.; Wolf, T. Evaluating Algorithms for Composable Service Placement in Computer Networks. In Proceedings of the 2009 IEEE International Conference on Communications, Dresden, Germany, 14–18 June 2009; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2009; pp. 1–6.
30. Mann, Z.Á. *Optimization in Computer Engineering–Theory and Applications*; Scientific Research Publishing: Wuhan, China, 2011; ISBN 978-1-935068-58-7.
31. Guerrero, C.; Lera, I.; Juiz, C. A lightweight decentralized service placement policy for performance optimization in fog computing. *J. Ambient. Intell. Humaniz. Comput.* **2018**, *10*, 2435–2452. [CrossRef]
32. Barker, E. *Recommendation for Key Management Part 1: General*; NIST Special Publication 800-57 Part 1 Revision 4; NIST: Gaithersburg, MD, USA, 2016.
33. Yu, B.; Wu, S.; Jiao, Z.; Shang, Y. Multi-Objective Optimization Design of an Electrohydrostatic Actuator Based on a Particle Swarm Optimization Algorithm and an Analytic Hierarchy Process. *Energies* **2018**, *11*, 2426. [CrossRef]
34. Yang, T.; Huang, Z.; Pen, H.; Zhang, Y. Optimal Planning of Communication System of CPS for Distribution Network. *J. Sens.* **2017**, *2017*. [CrossRef]
35. Pan, Q.-K.; Tasgetiren, M.F.; Liang, Y.-C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Comput. Oper. Res.* **2008**, *35*, 2807–2839. [CrossRef]
36. Wang, B.Z.; Deng, X.; Ye, W.C.; Wei, H.F. Study on Discrete Particle Swarm Optimization Algorithm. *Appl. Mech. Mater.* **2012**, *220–223*, 1787–1794. [CrossRef]
37. Strasser, S.; Goodman, R.; Sheppard, J.; Butcher, S. A New Discrete Particle Swarm Optimization Algorithm. In Proceedings of the 2016 on SIGMOD'16 PhD Symposium, San Francisco, CA, USA, 26 June–1 July 2016; ACM: New York, NY, USA, 2016; pp. 53–60.
38. Wang, L.; Ye, W.; Fu, X.; Menhas, M.I. A Modified Multi-Objective Binary Particle Swarm Optimization Algorithm. In *Lecture Notes in Computer Science, Proceedings of the Advances in Swarm Intelligence, Chongqing, China, 12–15 June 2011*; Tan, Y., Shi, Y., Chai, Y., Wang, G., Eds.; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 2011; pp. 41–48.
39. Coello, C.C.; Toscano-Pulido, G.; Lechuga, M. Handling multiple objectives with particle swarm optimization. *IEEE Trans. Evol. Comput.* **2004**, *8*, 256–279. [CrossRef]
40. Khaira, A.; Dwivedi, R.K. A State of the Art Review of Analytical Hierarchy Process. *Mater. Today Proc.* **2018**, *5*, 4029–4035. [CrossRef]
41. Saaty, T.L.; Vargas, L.G. The Seven Pillars of the Analytic Hierarchy Process. In *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*; Springer: Boston, MA, USA, 2001; pp. 27–46. ISBN 978-1-4615-1665-1.
42. Higgins, M.; Benaroya, H. Utilizing the Analytical Hierarchy Process to determine the optimal lunar habitat configuration. *Acta Astronaut.* **2020**, *173*, 145–154. [CrossRef]
43. Cheikhrouhou, O.; Koubaa, A.; Zaard, A. Analytical Hierarchy Process based Multi-objective Multiple Traveling Salesman Problem. In Proceedings of the 2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC), Bragança, Portugal, 4–6 May 2016; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2016; pp. 130–136.
44. Chang, S.-J.; Li, T.-H.S. Design and Implementation of Fuzzy Parallel-Parking Control for a Car-Type Mobile Robot. *J. Intell. Robot. Syst.* **2002**, *34*, 175–194. [CrossRef]
45. Suárez-Albela, M.; Fernández-Caramés, T.M.; Fraga-Lamas, P.; Castedo, L. A Practical Evaluation of a High-Security Energy-Efficient Gateway for IoT Fog Computing Applications. *Sensors* **2017**, *17*, 1978. [CrossRef]
46. Aazam, M.; Huh, E.-N. Dynamic resource provisioning through Fog micro datacenter. In Proceedings of the 2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), St. Louis, MO, USA, 23–27 March 2015; IEEE: New York, NY, USA, 2015; pp. 105–110.
47. Venčkauskas, A.; Morkevicius, N.; Jukavičius, V.; Damaševičius, R.; Toldinas, J.; Grigaliūnas, Š. An Edge-Fog Secure Self-Authenticable Data Transfer Protocol. *Sensors* **2019**, *19*, 3612. [CrossRef] [PubMed]
48. Suárez-Albela, M.; Fraga-Lamas, P.; Fernández-Caramés, T.M. A Practical Evaluation on RSA and ECC-Based Cipher Suites for IoT High-Security Energy-Efficient Fog and Mist Computing Devices. *Sensors* **2018**, *18*, 3868. [CrossRef] [PubMed]

electronics

MDPI

Article

Deep Learning-Based Content Caching in the Fog Access Points

Sovit Bhandari [1], Navin Ranjan [1], Pervez Khan [1], Hoon Kim [1,*] and Youn-Sik Hong [2]

[1] IoT and Big-Data Research Center, Department of Electronics Engineering, Incheon National University, Yeonsu-gu, Incheon 22012, Korea; sovit198@gmail.com (S.B.); ranjannavin07@gmail.com (N.R.); pervaizkanju@hotmail.com (P.K.)

[2] Department of Computer Science and Engineering, Incheon National University, Yeonsu-gu, Incheon 22012, Korea; yshong@inu.ac.kr

* Correspondence: hoon@inu.ac.kr

Abstract: Proactive caching of the most popular contents in the cache memory of fog-access points (F-APs) is regarded as a promising solution for the 5G and beyond cellular communication to address latency-related issues caused by the unprecedented demand of multimedia data traffic. However, it is still challenging to correctly predict the user's content and store it in the cache memory of the F-APs efficiently as the user preference is dynamic. In this article, to solve this issue to some extent, the deep learning-based content caching (DLCC) method is proposed due to recent advances in deep learning. In DLCC, a 2D CNN-based method is exploited to formulate the caching model. The simulation results in terms of deep learning (DL) accuracy, mean square error (MSE), the cache hit ratio, and the overall system delay is displayed to show that the proposed method outperforms the performance of known DL-based caching strategies, as well as transfer learning-based cooperative caching (LECC) strategy, randomized replacement (RR), and the Zipf's probability distribution.

Keywords: fog access points; cache memory; convolutional neural network; proactive caching

Citation: Bhandari, S.; Ranjan, N.; Khan, P.; Kim, H.; Hong, Y.-S. Deep Learning-Based Content Caching in the Fog Access Points. *Electronics* **2021**, *10*, 512. https://doi.org/10.3390/electronics10040512

Academic Editor: Kevin Lee

Received: 29 January 2021
Accepted: 18 February 2021
Published: 22 February 2021

1. Introduction

With the blooming of IoT devices, it is expected that the demand for mobile data traffic will grow at an unprecedented rate. To solve this issue to some extent, cisco coined fog computing-based network architecture to address latency-related problems [1]. Fog computing is a decentralized version of cloud computing with limited computational and signal processing capability, which brings the benefit of cloud computing nearer to the user side [2]. The remote radio heads with caching and signal processing capabilities in the fog computing architecture are referred to as fog access points (F-APs) [3,4]. F-APs have limited computational capability as compared to the cloud. So, F-APs should store popular cache contents proactively to maintain desirable fronthaul load to provide a better quality of service [5–7].

There has been extensive research related to caching in F-APs. Some of the related works are worth mentioning. In [8], the learning-based optimal solution is provided to place a cache memory content in a small cell base station based on the historical data. In [9], based on the user's mobility, a device-to-device optimal contents placement strategy is introduced. Likewise, in [10], the caching problem in multiple fog-nodes is studied to optimize delay in the large-scale cellular network. Similarly, in [11], the authors formulated; delay minimization and content placement-based joint optimization problems.

In the aforementioned literature [8–11], the research predicted the popularity of the contents based on Zipf's probability distribution method. However, this method cannot accurately predict the user content as user behavior is dynamic.

AI is likely to bring the fourth industrial revolution due to recent advances in its field [12]. There has been increased interest in deep learning (DL) models due to their remarkable impact in a wide variety of fields such as natural language processing, computer vision, and so on [13,14].

Recently, there has been more focused research on the DL-based approach to predict the future popular contents to solve the prevalent cache content placement issue [15]. In this paper, motivated by the DL-based approach, we propose a DL-based content caching (DLCC) to proactively store cache contents in the fog computing environment.

1.1. Related Works

The concept of fog radio access network (F-RAN) architecture had been introduced to bring cloud contents nearer to the end-users side, such that the fronthaul burden prevalent in the cloud radio access networks can be lessened. The cloud contents can be offloaded nearer to the end-user side by deploying the proper content caching technique in the F-APs.

In recent years, there has been enormous investigation to address content caching problems for wireless networks. The work of [16] discussed a joint routing and caching problem to maximize the portion of contents served locally by the small base stations in the cellular networks. In [17], femtocell caching followed by D2D-based content sharing idea is put forward to improve cellular throughput. The work of [18] focused on the latency-centric analysis of the degree of freedom of an F-RAN system for optimal caching and edge transmission policies. The works in [16–18] did not consider taking popular contents into account to address caching problem.

Since the F-APs have limited storage as well as signal processing capabilities, the highly preferred user contents should be placed in the cache memory of the fog nodes. The traditional approach of allocating cache contents in the wireless networks includes; least recently used, least frequently used, first-in-first-out, random replacement (RR), and time-to-live [19]. These methods have become impractical to use in the live network as user requirement changes over time. To mitigate the traditional approach of solving caching problems, some authors recommended placing the cache contents by analyzing the user's social information. In [20], social-aware edge caching techniques have been studied to minimize bandwidth consumption. In [21], social information and edge computing environment have been fully exploited to alleviate the end-to-end latency. However, social ties alone cannot be a sole deterministic factor to determine the dynamic nature of user preference.

Lately, there has been a marked increment in the utilization of big-data analytics, ML, and deep learning (DL) in academia, as well as in industry. They have been used to solve problems related to diverse domains such as autonomous driving, medical diagnosis, road traffic prediction, radio-resource management, caching, and so on, due to their high prediction accuracy [22–24]. Due to promising solutions provided by the artificial intelligence (AI) technology in various domains, a trend to exploit ML-based and DL-based models to the pre-determined future requirement of the user content has been set-up.

For instance, the works which have used an ML-based approach to determine the cache contents proactively are listed in [25,26]. In [25], a collaborative filtering (CF)-based technique is introduced to estimate the file popularity matrix in a small cellular network. However, the CF algorithm provides the sub-optimal solution when the training data are sparse. To solve the caching problem without undergoing any data sparseness problem, the authors in [26] proposed a transfer learning (TL)-based approach. However, in this approach, if similar content is migrated improperly, the prediction accuracy becomes worse.

Likewise, some of the papers, which have used DL-based models to forecast popular contents for caching are listed in [15,27–29]. In [15], an auto-encoder-based model is used to forecast the popularity of the contents. Likewise, in [27], a bidirectional recurrent neural network is used to determine content request distribution. In [28], a convolutional neural network-based caching strategy is presented. Moreover, in [29], the authors used different DL-based models such as a recurrent neural network, convolutional neural network (CNN), and convolutional recurrent neural network (CRNN) to determine the best cache contents and increase the cache hit ratio. However, in the above works, the DL-based model could not achieve validation accuracy greater than 77%. Therefore, accurately predicting F-APs cache contents with DL-models has become a challenging task.

1.2. Contribution and Organization

In this paper, to minimize the delay while accessing users' content, a DLCC strategy is introduced to store the most popular users' contents in the F-APs. In DLCC policy, we introduced a supervised learning-based 2D CNN model to train using 1D real-world datasets. We identified key features and key labels of the datasets by different data pre-processing techniques such as data cleaning, one-hot encoding, principal component analysis (PCA), k-means clustering, correlation analysis, and so on. The goal of our DLCC algorithm is to predict the popularity of contents in terms of different categorical classes. Then, based on the prediction result, the data of the most popular class will be stored in the cache memory of the nearby F-APs. We quantified the performance of the proposed caching policy on F-APs by showing DL accuracy, cache-hit ratio, and overall system delay.

The methodology of this article is shown in Figure 1 and summarized as follows:

1. An optimization problem to minimize content access delay in the future time is introduced.
2. DLCC strategy is proposed.
3. Open access real-life large dataset, such as MovieLens dataset [30] is analyzed and formatted using different data pre-processing techniques for the proper use for supervised DL-based approach.
4. 2D CNN model is trained using 1D dataset to obtain the most popular future data.
5. The most popular data are then stored in the cache memory of the F-APs.
6. The performance is shown in terms of mean square error (MSE), DL-accuracy, cache hit ratio, and overall system delay.

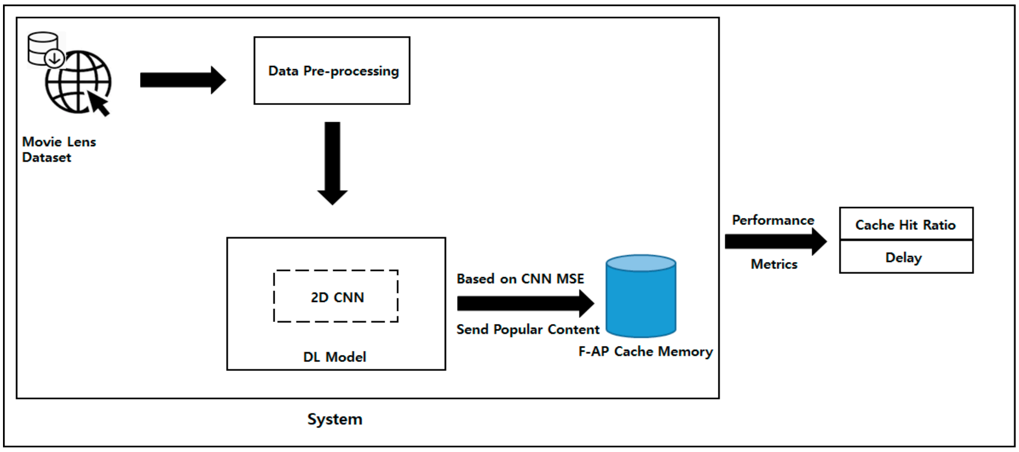


Figure 1. Methodology for deep learning-based content caching (DLCC).

The remainder of this paper is organized as follows; In Section 2, the system model is described. In Section 3, the DLCC policy is presented. Then, in Section 4, the performance of the proposed scheme is evaluated. Finally, in Section 5, conclusions are drawn.

2. System Model

In this section, a caching scenario for $N \times M$ fog radio access network (F-RAN) system having N user equipment (UEs), M F-APs, and one centralized base-band unit (BBU) cloud is modeled, as shown in Figure 2. In the system model diagram, we have $\mathcal{U} = \{1, 2, 3, \ldots, N\}$ as the set of N users requesting data from $\mathcal{F} = \{1, 2, 3, \ldots M\}$ as the set of M F-APs. In the diagram, the solid line connecting the BBU cloud to the access points represents the common public radio interface (CPRI) cable; whereas, the dashed-lines connecting the end-users to the access points denote the air-interface link.

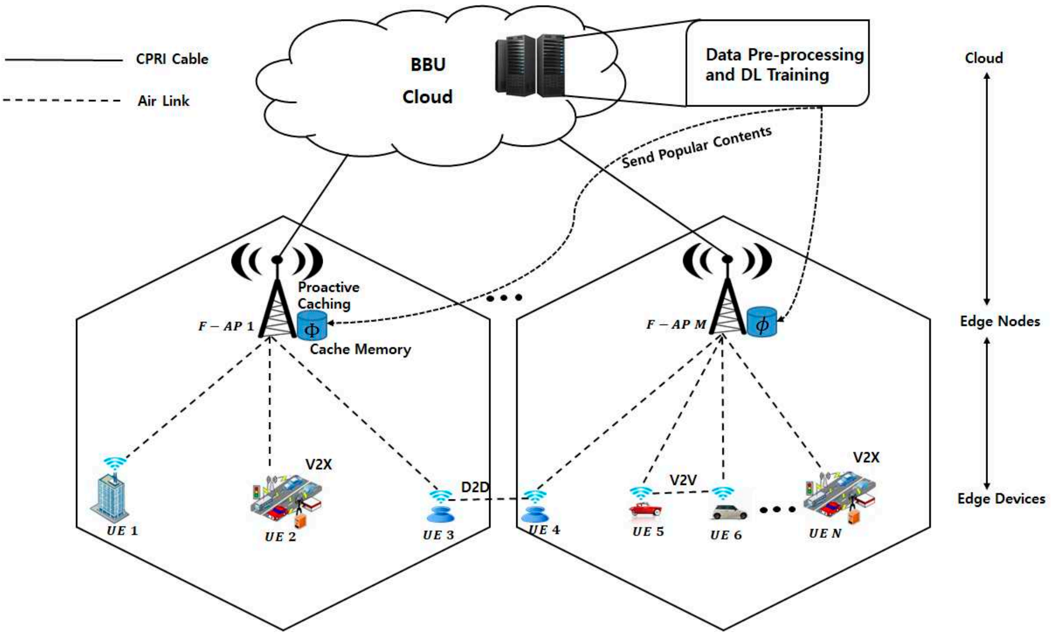


Figure 2. DL-based content caching in the fog-access points (F-APs).

As per the system model diagram, the DL-based training is done on the centralized cloud (CC) considering the proactive as well as the reactive caching case. The testing results provided by the DL-based model on the CC are used to send the most popular contents to the F-APs. Since the computational capabilities of the F-APs are way lesser than that of the cloud, only the top most popular contents are stored in it. The contents stored in the F-APs are based on location-based user preference. Based on the proactive and reactive caching scenario in an F-RAN system, the delay system is formulated and is shown in part 2.1.

2.1. Delay Formulation

In this part, we are interested in formulating overall system delay for the $N \times M$ F-RAN system, for some time instance $t+1$. We assumed that the air-interface link capacity connecting the UE i ($i \in \mathcal{U}$) to F-AP j ($j \in \mathcal{F}$) as $C_{i,j}^{Ai}$, and the fronthaul link capacity connecting F-AP j to the CC as $C_{j,1}^{Fh}$. For simplicity, we considered that the cache memory of all the F-APs has the same capacity to store the files, i.e., $\varnothing$ (GB). We also assumed that the size of each file p cached at the fog nodes is the same. Let $S_{i,j}^{t+1}$ denote the file size requested by the UE i with F-AP j, at any time instance $t+1$. In our problem formulation, for direct transmission, we considered only the delay for transferring the data from the cloud to the F-APs. On the other hand, for cached transmission, the delay for offloading the cached contents from the F-APs to the UEs is neglected. Considering the above scenario, the overall delay for the $N \times M$ F-RAN system for any time instance $t+1$ can be devised as:

$$\text{P}(1)\ \delta_{sys}^{t+1} = \min\left[\sum_{i=1}^{N}\sum_{j=1}^{M}\left(1 - x_{i,j}^{t+1}\right)\frac{S_{i,j}^{t+1}}{C_{j,1}^{Fh}}\right] \tag{1}$$

$$\text{s.t. } S_{i,j}^{t+1}x_{i,j}^{t+1} \le \varnothing\ \forall i,j \tag{2}$$

where δ_{sys}^{t+1} is the overall delay of the F-RAN system at time $t+1$, and $x_{i,j}^{t+1}$ is the decision control variable to show whether the file requested by the UE i with F-AP j at any time

$t+1$ is available in the cache memory of the latched F-AP, or not. The value of $x_{i,j}^{t+1}$ can either be 1 or 0, not otherwise. This can be represented as:

$$x_{i,j}^{t+1} = \begin{cases} 1, & \textit{if the content requested by UE i is available in } F-AP\ j \\ 0, & \textit{otherwise} \end{cases} \tag{3}$$

The constraint in the problem statement (P1) indicates that the size of the file cached in the cache memory of F-AP should be lesser than or equal to the overall memory size of that particular F-AP.

3. DL-based Caching Policy

In this section, DLCC is presented, so that the overall delay of the F-RAN system formulated in (P1) can be minimized in the best possible way. The general overview of the proposed caching policy is shown in Figure 3.

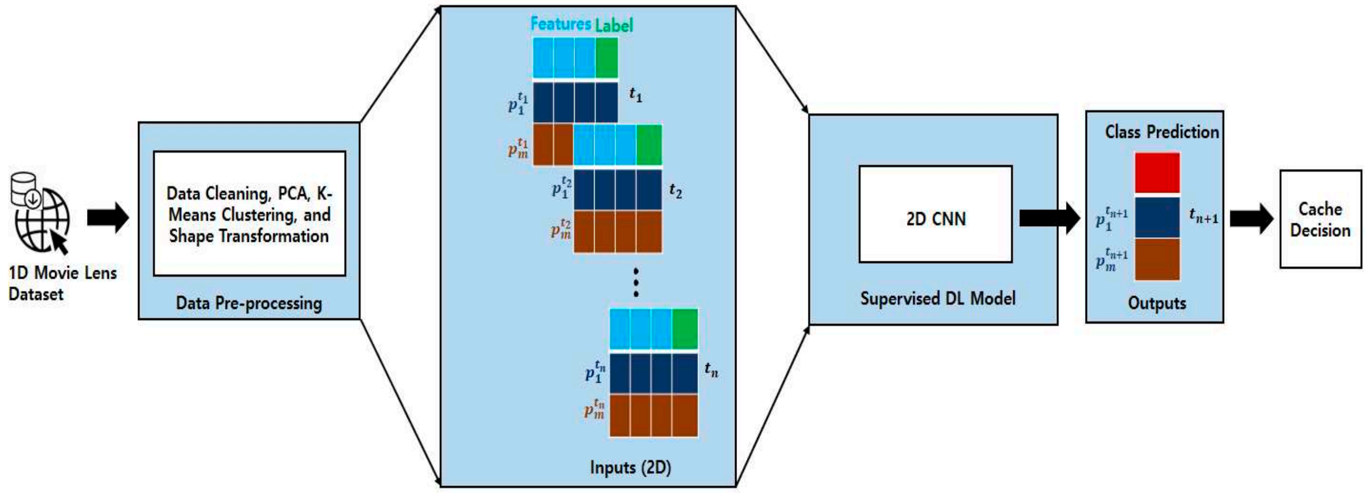


Figure 3. The overview of the DLCC policy to get the popular content in terms of popular class.

Figure 3 contains a series of the task required for predicting the best cache contents for the F-APs. The initial step includes the extraction of the most popular 1D real-life datasets from the cloud. After the initial step, the downloaded data are pre-processed using various techniques to make it a suitable 2D dataset for the preferred supervised DL-based model. Then, the suitable 2D dataset is trained using the 2D CNN model. After that, the trained model is used to predict the contents on the basis of different categorical classes for time $t+1$. Due to the memory constraint of the F-APs, only the contents of the top-most class are selected randomly to be stored in the F-APs for future user requirements. The steps mentioned above are coherently described in the subsections given below.

3.1. Dataset

MovieLens dataset is used for training the DL-based model, as it is a large dataset available in the open-source platform. Moreover, live streaming of the movies utilizes most of the fronthaul capacity. So, to lessen fronthaul load to some extent, proactive storing of the most popular movies in the cache memory of the F-APs is considered the most viable approach.

We downloaded the MovieLens dataset from [30]. It contains around 25 Million ratings and around 1 Million tag applications for 62,423 movies. Moreover, the dataset contains movies from 1 January 1995 to 21 November 2019, rated by 162,541 users. This dataset was generated on 21 November 2019. In this dataset, random users represented by a unique id had rated at least 20 movies. The data contained in this dataset represent genome-scores.csv, genome-tags.csv, links.csv, ratings.csv, and tags.csv.

3.2. Data Pre-Processing

It requires a large computational effort and also not relevant to train the whole MovieLens dataset, so the portion of the dataset is only taken for training and validation purposes. We used data from 1 January 2015 to 21 November 2019 from the dataset. The selected portion of the dataset contains only around 7.5 million ratings for 58,133 movies.

The initial dataset contains the categorical variable such as User-ID, Movie-ID, Rating, Date, Year, Month, Day, and Genres. Based on the dataset key features such as Year, Month, and Day, the daily requested movies are counted and are portrayed in the form of a yearly-based box plot, as shown in Figure 4.

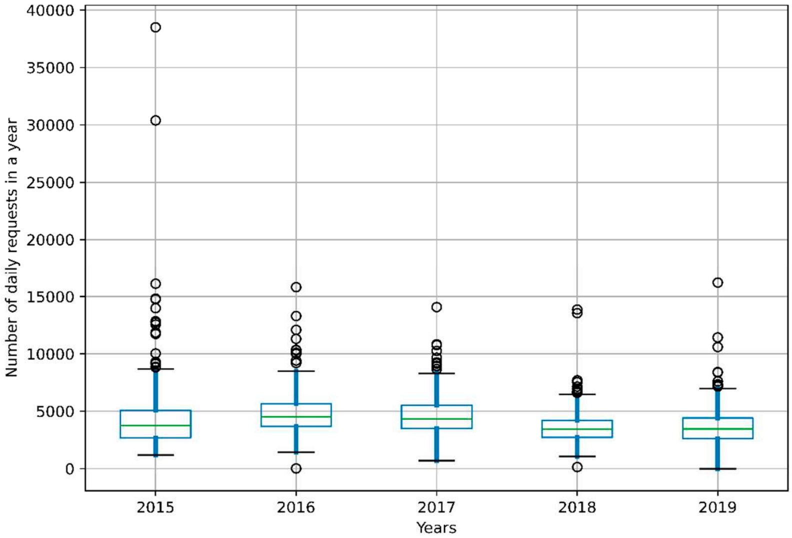


Figure 4. Daily request count of movies on the basis of Movie-ID for each year.

As per Figure 4, we can see that on average, around 800 movies are requested daily. Likewise, the maximum movie request on a particular day is around 38,000, whereas the minimum movie request is 1.

If we go with the average daily request for the movie, it is still beyond the computational capacity of the F-AP to store 800 movies in its cache memory. To solve this, the dataset is further analyzed on the basis of the Movie-ID and daily movie request count, and it is depicted in Figure 5.

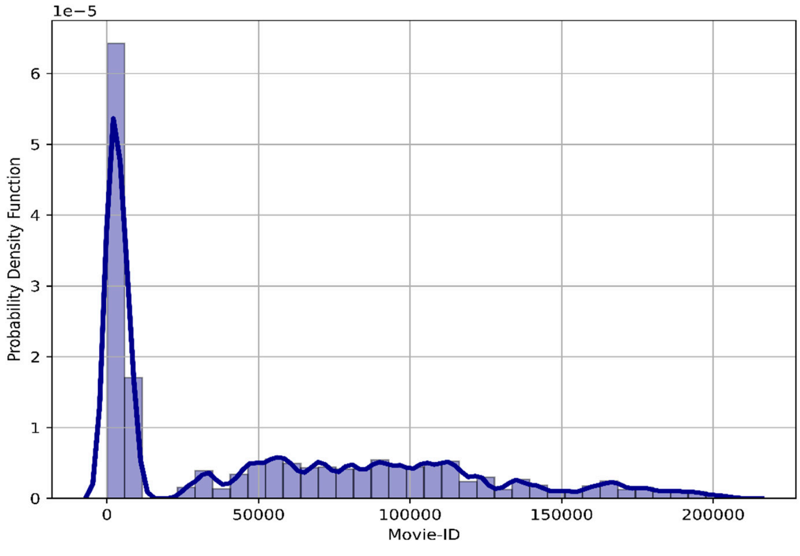


Figure 5. Probability density function (PDF) of the Movie-ID on the basis of movie request.

Figure 5 shows the probability density function (PDF) of the Movie-ID based on the movies requested. As per the figure, we can observe that the PDF is comparatively higher for the movie id number within 0–12,000. The PDF is smaller for the movies having id numbers greater than 12,000. Thus, for our analysis, movies with id numbers 0–12,000 are taken into account. The PDF of the selected movies is shown in Figure 6.

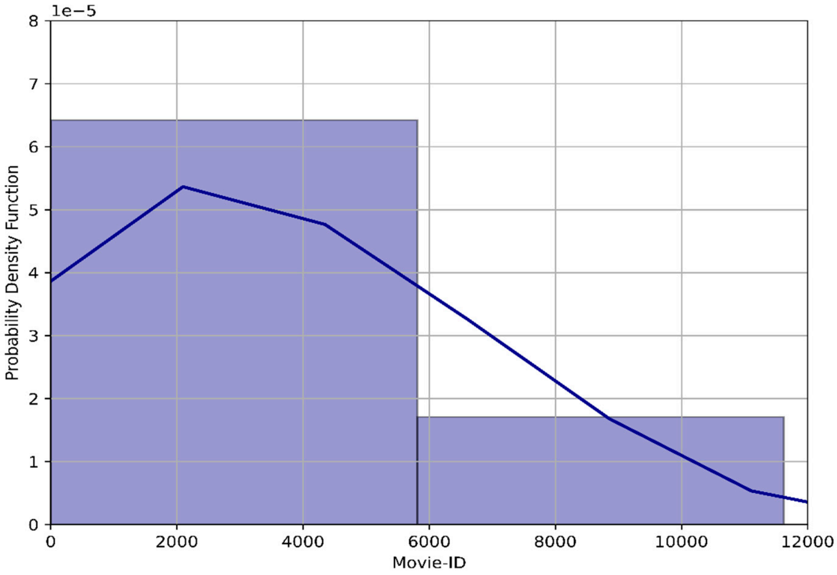


Figure 6. PDF of the selected Movie-ID.

After selection of Movie-ID from 0–12,000, the number of rows in the dataset is reduced to around 3.5 Million from 7.5 Million. This selected portion of the dataset accounts to be around 14% of the total dataset (25 Million). Then, the per-day count of the movie as per the unique Movie-ID is calculated to further reduce the number of rows to around 1.6 Million. After that, the dataset is re-arranged as per Movie-ID with its corresponding attributes such as year, month, day, genre, and movie counter.

In the dataset, the genre feature contains the string values. Since the genre feature in the dataset contains the different categorical string values, it is further processed by the One-Hot Encoding method to convert it into numerical form, as the DL-model employs on the numerical data. One-Hot encoding is one of the natural language processing techniques to convert categorical string variables into a numerical form such that machine learning algorithms can perform better prediction [31].

When the One-Hot Encoding technique is applied to a genre column containing multiple categorical string variables, the single genre column is transformed into 19 columns, as it contained 19 different categories. The increment in the column numbers adds up computational complexity to train the model. Therefore, to reduce the computational complexity, principal component analysis (PCA) of the categorical variable of the genre data is performed. PCA is a robust approach for reducing the dimension of datasets; by preserving most of the useful information. It does so by creating new uncorrelated variables that successively maximize variance [32].

The PCA analysis on 19 categorical columns is done to reduce 19 categorical columns to 3 categorical columns. The newly formed categorical columns are named PCA-1, PCA-2, and PCA-3, respectively. Figure 7 shows the individual and cumulative weightage of variance provided by the formed three principal components. In the figure, the first, the second, and the third principal components are indicated by the x-axis values 0, 1, and 2, respectively. The three principal components contain the Eigen-values of the PCA-1, PCA-2, and PCA-3, respectively. We reduced the 18-columned matrices to 3-columned matrices because the cumulative sum of the variance of three principal components accounted for 45.70% of the total variance. As per the figure, the Eigen-values of PCA-1, PCA-2, and PCA-

3 contributed 19.66%, 15.59%, and 10.45% of the total variance, respectively. The portion of the variance contributed by PCA-1 is greater, so it is referred to as principal component 1. Since the portion of the variance contributed by the Eigen-values of succeeding columns is lesser than the preceding, they are indicated as principal component 2 and principal component 3, correspondingly.

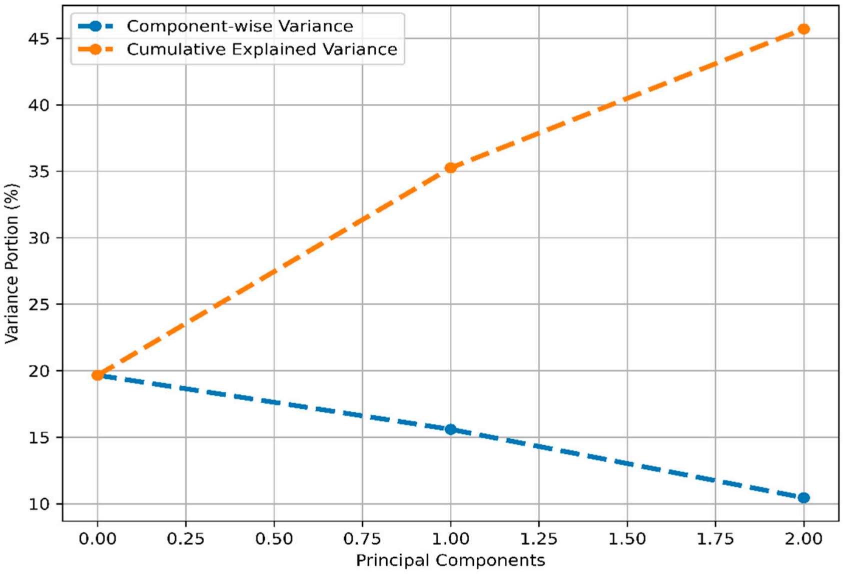


Figure 7. Individual and cumulative variance of the principal components.

After the column reduction technique is applied to the dataset, the 1D dataset is converted to a 2D dataset for the selected 9019 Movie-ID’s. There are around 1787 days from the starting of 2015 to 21 November 2019. When 9019 Movie-ID’s is multiplied to 1787, the resulting 2D dataset will have 16,107,934 rows. This is a 1000% increment in the size of the dataset from the reduced version of the 1D dataset.

The resulting 2D dataset is clustered based on per day’s movie request count to add label to the dataset. The dataset is categorized into four categories, i.e., Class 0, Class 1, Class 2, and Class 3, by using the k-means clustering technique, as shown in Figure 8.

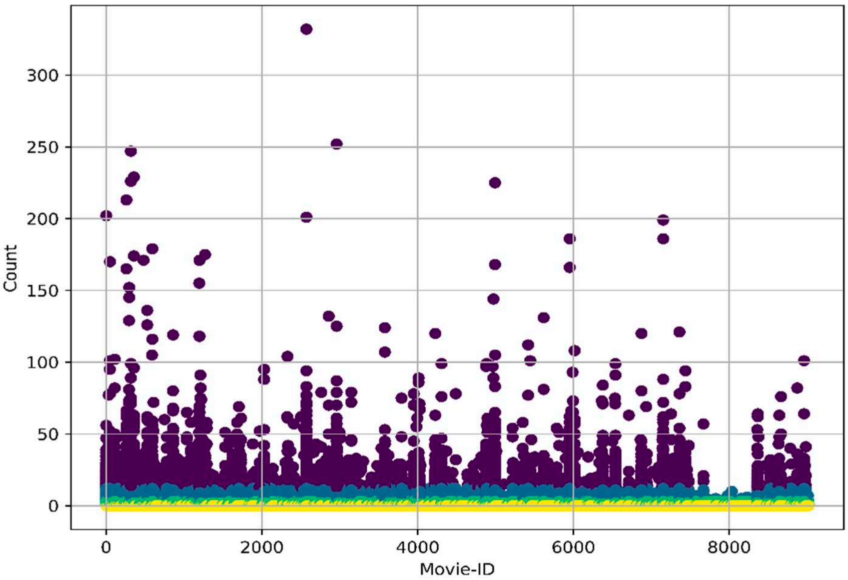


Figure 8. Vertical clustering of the dataset on the basis of per-day count of the Movie-ID.

In Figure 8, Class 0 is represented by the purple color. This class includes Movie-ID having the almost higher count on a particular day. Likewise, Class 1, Class 2, and Class 3 are represented by the blue, green, and yellow colors, having a range of values such as 12–362, 4–11, and 1–3, respectively. In our 2D dataset, Class 0, Class 1, Class 2, and Class 3 contains 22,604, 212,759, 1,406,486, and 14,736,655 rows, respectively. The Class 0 movies are referred to as highly preferred movies, whereas Class 3 movies are regarded as the least requested ones. These categorized values are placed under the column name Class of the dataset.

The resulting dataset contains Year, Month, Day, Movie-ID, PCA-1, PCA-2, and PCA-3 as the key features, and the Class as a key label. The correlation matrix in Figure 9 is shown to depict the usefulness of our dataset for the 2D CNN model.

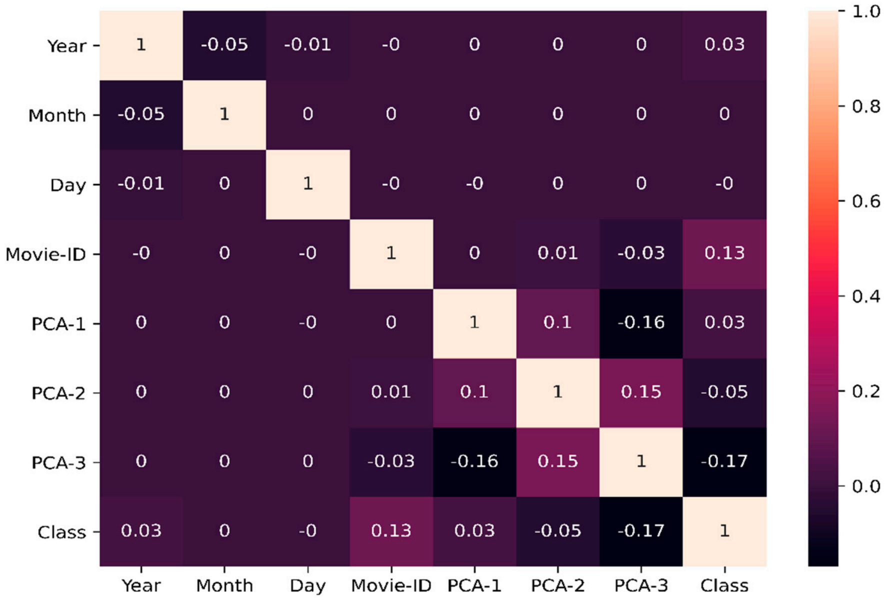


Figure 9. Dataset correlation matrix.

3.3. DLCC Model

In this section, at first, we explain the problem statement and then discuss the architecture for predicting the future popularity of movies listed in the MovieLens dataset by using the time-series sequence of historical data.

3.3.1. Problem Statement

The main objective of the DLCC model is to realize the future likelihood of the data contents being accessed by the connected UEs. In this study, the DLCC model is trained based on the MovieLens dataset d, containing movie lists up to time t. Let $\mathrm{X} = \{X_1,\ X_2,\ X_3,\ \ldots, X_t\}$ be the chronological order of time-variant historical movies list. Its corresponding output label, which is particularly the classification of movies list based on popularity, can be represented as $Y = \{Y_1,\ Y_2, Y_3, \ldots, Y_t\}$. The i^{th} time input of X. can be denoted as:

$$\mathrm{X}_i = \begin{bmatrix} x^i_{11} & x^i_{12} & \cdots & x^i_{1f} \\ x^i_{21} & x^i_{22} & \cdots & x^i_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ x^i_{n1} & x^i_{n2} & \cdots & x^i_{nf} \end{bmatrix} \in \mathbb{R}^{n\times f} \quad (4)$$

ere X_i contains collection of n movie samples, each having f features. Similarly, its corresponding i^{th} time output label can be represented as:

$$Y_i = \begin{bmatrix} y_1^i \\ y_2^i \\ \vdots \\ y_n^i \end{bmatrix} \in \mathbb{R}^{n \times 1} \tag{5}$$

where Y_i contains the category of each movie sample based on popularity. The primary objective of this study is to develop the prediction model $\mathcal{M}$, which uses input features X_i to predict the popular class Y_i, which can be defined as:

$$Y_i = \mathcal{M}(X_i,\ \theta) \tag{6}$$

where θ is the model parameter of DLCC model.

3.3.2. Model Implementation

In this part, we use 2D CNN for feature extraction from the input 2D MovieLens dataset. Moreover, we use a regression-based approach to solve the classification problem. The main goal of the DLCC model is to categorize MovieLens dataset on the basis of popularity. The DLCC model used in this paper is shown in Figure 10.

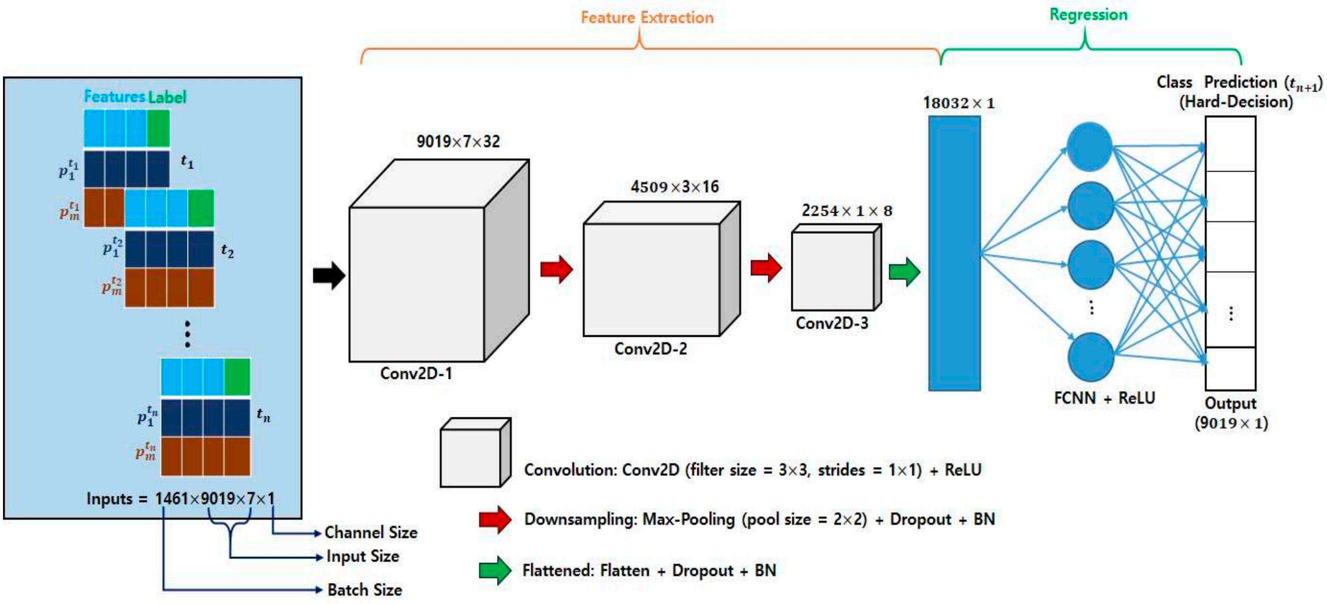


Figure 10. DLCC model to classify MovieLens dataset on the basis of popularity.

In Figure 10, the architecture of the DLCC model is formed by stacking three convolutional layers, two max-poling layers, one flatten layer, and one dense layer. Mathematically, the f^{th} feature map of l^{th} convolutional layer y_f^l can be obtained by first convoluting 2D input or previous layer output with the convolutional filter and then applying bit-wise non-linear activation, which is shown in Equation (7).

$$y_f^l = \sigma\left(\sum_{k=1}^{f_{l-1}} y_k^{l-1} \bigoplus W_{kf}^l + b_f^l\right),\ f \in [1, f_l] \tag{7}$$

where y_k^{l-1} is the k^{th} feature map of $(l-1)^{th}$ layer, W_{kf}^l is the kernel weight at position k connected to the f^{th} feature map of l^{th} layer, b_f^l is the bias of f^{th} filter of l^{th} layer, f_l is the number of the filter in l^{th} layer and $\sigma(.)$ represent element-wise non-linear activation function. Equation (8) shows the output of l^{th} convolutional layer and pooling layer.

$$y_f^l = pool\left(\sigma\left(\sum_{k=1}^{f_{l-1}} y_k^{l-1} \bigoplus W_{kf}^l + b_f^l\right)\right),\ f \in [1, f_l] \tag{8}$$

In the DLCC model, the feature learned from the 2D CNN model is concatenated into a dense vector by flattening operation. The dense layer contains the high feature extraction from the input. Let L be the previous layer before flattening layer, having f_L number of feature maps, then the output of $L+1$ layer, y^{L+1} is given as:

$$y^{L+1} = o_{flatten}^L = flatten\left(\left[y_1^L,\ y_2^L,\ \ldots,\ y_{f_L}^L\right]\right) \tag{9}$$

where $y_1^L, y_2^L, \ldots, y_{f_L}^L$ are the feature maps of L^{th} layers, and $o_{flatten}^L$ is the flatten vector of L layer. Finally, the flattened layers are transformed to model output through a fully connected layer, having W_d and b_d weight and bias of fully connected dense layer. The model output can be written as:

$$\hat{y} = W_d o_{flatten}^L + b_d = W_d\left(flatten\left(pool\left(\sigma\left(\sum_{k=1}^{f_{l-1}} y_k^{l-1} \bigoplus W_{kf}^l + b_f^l\right)\right)\right)\right) + b_d \tag{10}$$

In our model MSE loss function $\mathcal{L}(\theta)$ is used to optimize the target. Minimizing MSE is taken as the training goal of our model. Mathematically, MSE can be written as:

$$\mathcal{L}(\theta) = \|y_t - \hat{y}_t\|_2^2 \tag{11}$$

In Figure 10, at first 2D input of size 9019 × 7 (rows number × column numbers) is employed to the first convolutional 2D layer of the portrayed DLCC model. In the first convolutional layer, the input is scaled up by using 32 filters of size 2 × 2 with a stride of 1 × 1. The convoluted output of the first layer is then fed to the downsampling layer. In the downsampling layer, the max-pooling technique with a pool size of 2 × 2 is used along with the batch normalization (BN) and dropout techniques. In our DLCC model, BN is used to stabilize the learning process and reduce the number of epochs required to train the neural networks [33], whereas dropout is used to prevent the trained model from overfitting [34]. The convoluted downsampled data of size 4509 × 3 × 32 are employed in the second convolutional layer to reduce the filter number from 32 to 16 by using the same filter and stride size used in the first convolutional layer. After that, the convoluted outputs of the second convolutional layer are again employed in the downsampling layer to reduce the size of inputs to 2254 × 1 × 16. Again, the downsampled data of the second downsampling layer are fed to the third convolutional layer to reduce the filter size from 16 to 8. Since the necessary features were extracted after the implementation of the third convolutional layer, the features of size 2254 × 1 × 8 are flattened to employ it to the fully-connected neural network (FCNN) for the regression process. In each convolutional layer, a rectified linear unit (ReLU) activation function is used to increase the non-linearity in our input data, as well as to solve a vanishing gradient problem. In the regression process, the input of size 18,032 is fed to the FCNN layer to get the output of size 9019. The detailed structure of our DLCC model is shown in Table 1.

Table 1. Detailed structure of DLCC with three convolutional layers.

Layer Name	Input Size	Output Size	Filter Size
Conv2D_1	9019 × 7, 1	9019 × 7, 32	3 × 3, 32
Max_Pooling_1	9019 × 7, 32	4509 × 3, 32	——
Dropout_1	4509 × 3, 32	4509 × 3, 32	——
Batch_Normalization_1	4509 × 3, 32	4509 × 3, 32	——
Conv2D_2	4509 × 3, 32	4509 × 3, 16	3 × 3, 16
Max_Pooling_2	4509 × 3, 16	2254 × 1, 16	——
Dropout_2	2254 × 1, 16	2254 × 1, 16	——
Batch_Normalization_2	2254 × 1, 16	2254 × 1, 16	——
Conv2D_3	2254 × 1, 16	2254 × 1, 8	3 × 3, 8
Dropout_3	2254 × 1, 8	2254 × 1, 8	——
Flatten_1	2254 × 1, 8	18,032	——
Batch_Normalization_3	18,032	18,032	——
FCNN_1	18,032	9019	——

In the output of FCNN, the ReLU activation function is used to get output greater than one. After the implementation of the ReLU activation function, the predicted outputs are rounded-off to make hard-decision. The obtained hard-decision is the classification of the MovieLens dataset based on popularity. We trained 1461 data of size 9019 × 7 to predict output for time t_{n+1}.

Algorithm 1: Training process for DLCC model.

```
Input: Training dataset $d$, model $m$
Ouput: Trained model $m^k$
Initialize: $m_{terror}$, $m_{verror}$, $m_\alpha$, $m_b$ = 0
Find the best parameters: To train the model $m$
for $i$ in range(10) do
    $m_b^i$ ← 2 ** $i$
    for $j$ in range(1000) do
        $m_\alpha^{i,j}$ ← rand(0,1)
        Train the model $m$ with dataset $d$ minimizing $\mathcal{L}(\theta)$
        Store all of training information of model $m$ for each training loop $i$, $j$ in array
        $val_m^{i,j}$ ← {$m$: $m_b^i$, $m_\alpha^{i,j}$, $m_{terror}^{i,j}$, $m_{verror}^{i,j}$}
    endfor
endfor
Choose the with best parameters of index $k$ = argmin($m_{terror}^{i,j} m_{verror}^{i,j}$) to train the model $m$
Train: model $m$ with $k$ index parameters to produce trained model $m^k$
```

Since our problem is a multi-variant regression problem, the mean square error method is used in the training process [35]. Moreover, the Adam optimizer is used to update the weight and learning rate values as it is straightforward to implement, computationally efficient, and has low memory requirements [36]. It is very difficult to tune the hyper-parameters required to train the DL model. The detailed procedure to select hyper-parameters such as batch size (b) and learning rate (α) to train the proposed model (m) is shown in Algorithm 1.

In Algorithm 1, the CNN model m is trained for the random values of batch size m_b and learning rate m_α. For each value of m_b, 1000 random learning rates having a value in between (0, 1) is realized to train on dataset d. The value of m_b is selected by increasing the power of base integer two from 0–9. Using every value of m_b and m_α, the model is trained on dataset d to obtain training error (m_{terror}) and validation error (m_{verror}), which is stored in the val array. After the completion of the loop, the k^{th} index on the val array providing the minimum value of m_{terror} and m_{verror} is selected to extract the hyper-parameters value stored in that particular index. Finally, the obtained hyper-parameter values are selected to train the model m to get trained model m^k.

Choosing the depth of convolutional neural network plays a crucial role in determining the performance of the model as each addition of convolutional layer in the model leads to the increment of the feature map, so the learning. However, beyond a certain limit, each addition of a convolutional layer in the model tends to overfit the data. So, based on DLCC model accuracy, we experimented using different depths of the model to find a better one. Table 2 justifies why we only chose three convolutional layers in our model.

Table 2. Comparison of the different model depths of DLCC.

Model	Description	Filter Configuration	Validation Loss (MSE)	Computational Time (min)
DLCC_1_1	1 2D-CNN and 1 FCNN	32	N/A	N/A
DLCC_2_1	2 2D-CNN and 1 FCNN	32_16	0.2785	23.75
DLCC_3_1	3 2D-CNN and 1 FCNN	32_16_8	0.0452	13.35
DLCC_4_1	4 2D-CNN and 1 FCNN	32_16_8_4	0.0596	7.98

As per Table 2, we can see that the DLCC model having the single convolutional layer could not provide any output because of the large number of trainable parameters, i.e., 39B. The MSE is minimum for the DLCC model having three convolutional layers in its architecture with a filter configuration of 64_32_8, while the computational time is lesser for the DLCC model having four convolutional layers. Since we require to select the model which provides minimum MSE in a reasonable time, we selected the DLCC_3_1 model to solve the caching issue. The validation MSE provided by the DLCC_3_1 model at the cost of 13.35 min is 0.0452. The number of the input parameters and hyper-parameters was the same for generating results for all four configurations.

Furthermore, the number of filters in the convolution neural networks also plays an important role in determining the performance of the model. So, to find out the best filter configuration for our DLCC_3_1 model, we tried three different configurations, as shown in Table 3.

Table 3. Comparison of the different filter configuration of DLCC.

Model	Filter Configuration	Validation Loss (MSE)	Computational Time (min)
DLCC_3_1	64_32_16	0.0729	26.3
DLCC_3_1	32_16_8	0.0452	13.35
DLCC_3_1	16_8_4	0.0741	7.28

Table 3 shows that the DLCC_3_1 model having filter configuration 32_16_8 provides a better validation loss as compared to the other two different types of filter configuration. So, we selected the filter configuration of 32_16_8 for our DLCC_3_1 model.

3.4. Cache Decision

In this part, caching decision process is described to allocate the best cache contents to F-APs. The initial step includes the training of the DLCC model on the cloud, based on Algorithm 1. Then the trained model is used to predict cache contents for time $t + 1$. After that, the list of contents categorized based on popular classes is transferred to the F-APs for the selection process. On the priority order, the contents of Class 0 is stored in the available cache memory of the F-APs. If the is still some memory available, the contents of Class 1 followed by Class 2 are recommended to be stored in the available cache memory. The contents of Class 3 are not stored even if there is any memory space available in the F-APs as contents of Class 3 are the least preferred ones. The detailed procedure is summarized in Algorithm 2.

Algorithm 2: Cache content decision process.

```
Input: Requested contents history
Output: Selected content list to be stored in the cache memory of F-APs
Training of the DLCC model in the Cloud based on Algorithm 1
Prediction of content list categorized on the basis of classes for time t+1
Send the predicted information to the F-APs
if the total size of contents of Class 0 $\leq M\emptyset$ then
    Store all the contents of Class 0 in the cache memory of F-APs
    if the total size of Class 1 contents $\leq M\emptyset$—total size of Class 1 then
        Store all the contents of Class 1 in the remaining cache memory of F-APs
    else
        Store the contents of Class 1 randomly until the cache memory of F-APs is full
        if the total size of Class 2 contents $\leq M\emptyset$—total size of Class 1 and Class 2 then
            Store all the contents of Class 2 in the available cache memory of F-APs
        else
            Store the contents of Class 2 randomly until the cache memory of F-APs is full
else
    Store the contents of Class 0 randomly until the cache memory of F-APs is full
```

4. Performance Analysis

In this section, the performance of the proposed CNN-based model is shown in terms of model key performance indicators (KPI): such as MSE and prediction accuracy. Likewise, the performance cache content decision is quantified in terms of cache hit ratio and system delay.

To train the DLCC model, we used the Keras library on top of the TensorFlow framework in Python 3.7 as a programming platform. The training process for our datasets is performed by using a computation server (MiruWare, Seoul, Korea). The specification of the computational server includes; one Intel Core i7 CPU, four Intel Xeon E7-1680 processors, and 128 GB random access memory. The results are obtained by using a computer with 16 GB random access memory and an Intel Core i7-8700 processor.

4.1. Model KPI

In this part, the DLCC model KPI in terms of MSE (regression), and prediction accuracy (classification) is presented. The proposed 2D CNN-based model is trained on each day data of the MovieLens dataset from January 2015–December 2018. Likewise, the validation of the trained model is done on the data of January 2019–October 2019. Finally, the trained model is tested on the data of November 2019. The simulation parameter used while training the model is summarized in Table 4.

Before the application of hard-decision on the obtained results, the MSE obtained while testing the trained model on the November 2019 data of MovieLens dataset is shown in Table 5. The obtained result is compared with the results shown in [29].

The hard-decision rule is implemented for classifying the prediction results of the CNN-based regression model, which is shown in Table 6.

Table 4. Simulation parameters for the training model.

Parameters	Values
Training Size	(1461, 9019, 7, 1)
Validation Size	(304, 9019, 7, 1)
Testing Size	(21, 9019, 7, 1)
Training Period	January 2015– December 2018
Validation Period	January 2019–October 2019
Testing Period	November 2019
Number of 2D CNN Layers	3
Number of FCNN Layer	1
Number of Features	7
Number of Label	1
Output Activation Function	ReLU
Batch Size	8
Learning Rate	0.001
Epoch	1–8

Table 5. Comparison of results obtained from different DL methods.

Model Type	Validation Loss (MSE)
DLCC (Proposed)	0.045
1D CNN [29]	0.066
1D LSTM [29]	0.056
1D CRNN [29]	0.059

Table 6. Mapping table for classifying the results of convolutional neural network (CNN)-based model.

Range (Predicted Values)	Classification (Hard-Decision)
0–0.5	0
0.5–1	1
1–1.5	1
1.5–2	2
2–2.5	2
2.5–3	3

After the implementation of mapping table shown in Table 6, the average value of prediction accuracy and prediction error of November 2018 is shown in Figure 11 for the different values of training epochs.

As per Figure 11, it can be seen that there is an exponential rise in the prediction accuracy of the model and exponential decay in the prediction error of the model till the five training epoch. Beyond the five training epoch, the learning potential of the model enters the saturation phase. The prediction accuracy of the model is 92.81% for the five training epochs, but it is around 1% for the training epoch less than four. Moreover, the error curve shown in Figure 11 is plotted based on the formula; $Classification\ Error\ (\%)\ =\ 100\ (\%) - Classification\ Accuracy\ (\%)$.

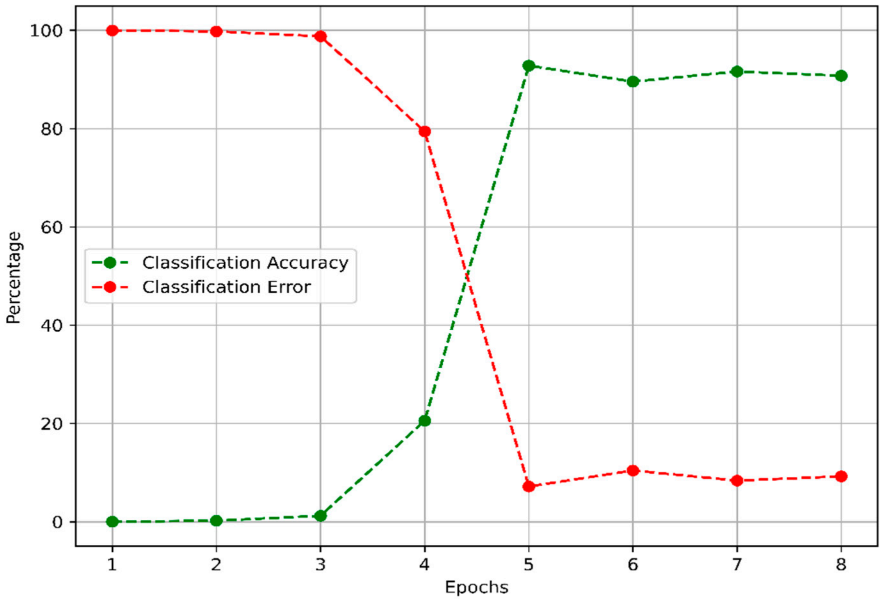


Figure 11. Accuracy of DLCC model to classify MovieLens dataset on the basis of popularity.

To lime-light the prediction accuracy of each class, a multi-class confusion matrix is drawn for the prediction date: 1 November 2019 to 21 November 2019. The confusion matrix value of the whole testing period is averaged and is shown in Table 7. The accuracy of the confusion matrix shown in Table 4 can be calculated using the following formula [37]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

where "TP", "TN", "FP", and "FN" corresponds to "true positive", "true negative", "false positive", and "false negative", respectively. Using the above equation, the value of the accuracy for the confusion matrix shown in Table 7 is calculated to be 92.81%. This accuracy is around 21–54% greater than the prediction accuracy reported in the papers [15,27–29] while solving a similar problem.

Table 7. Multi-class confusion matrix (average value) for the prediction of the popularity of cache contents for 1 November 2019–21 November 2019.

		Predicted Values			
	Class	0	1	2	3
Actual Values	0	5.713	2.433	2.230	0.676
	1	2.676	50.926	49.227	1.984
	2	0.572	22.858	300.249	401.784
	3	0.369	1.781	161.292	8014.230

4.2. System KPI

In this section, the cache hit ratio and overall system delay are shown to portray the usefulness of the DLCC policy in the F-RAN system. The movies of the categories "0", "1" and "2" are proactively stored in the F-APs, whereas the movies under category "3" are not stored as they are the least preferred ones.

Mathematically, the cache hit ratio for any time instance can be calculated as:

$$Cache\ hit\ ratio\ (t) = \frac{Total\ cache\ hits\ (t)}{Total\ cache\ hits\ (t)\ +\ Total\ cache\ misses\ (t)} \tag{13}$$

The system parameters used to calculate the cache hit ratio and the system delay are summarized in Table 8.

Table 8. Fog radio access network (F-RAN) system parameters.

Parameters	Values
Number of F-APs (M)	50
Number of UEs (N)	400
Number of movie files in the pool (p)	9019
Size of each movie (S)	1 (GB)
Fronthaul link capacity ($C_{j,1}^{Fh}$)	10 Gbps @ 10 km+
Total cache memory ($\varnothing$)	0–600 (GB)
Distance between F-AP and central cloud	10 km

+ Greater than.

As shown in Table 8, an F-RAN system consisting of 50 F-APs and 400 UEs is designed to request movie files from the pool of files. It is assumed that each user can request only one movie file from F-APs at time $t + 1$. Likewise, the size of each movie file listed in the MovieLens dataset is considered to be 1 GB, making a total of 9019 GB. Since there are 400 UEs in our system, 400 movie requests at time $t + 1$ make a total demand size of 400 GB. Based on the above simulation parameters, the cache hit ratio is calculated for the different values of total cache memory and is portrayed in Figure 12.

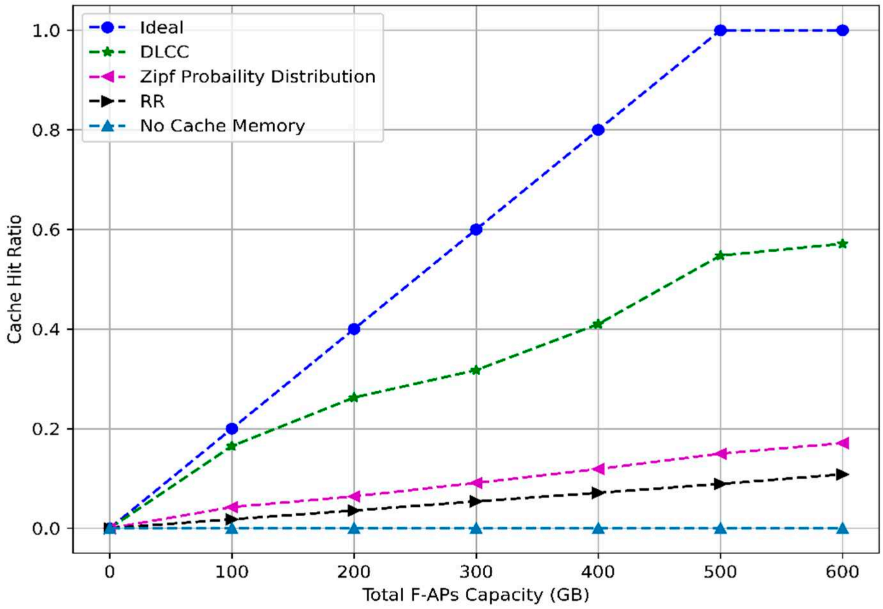


Figure 12. Total F-AP capacity vs. cache hit ratio.

Figure 12 shows that the cache hit ratio for five different caching policies such as ideal, DLCC, Zipf's probability distribution, randomized replacement (RR), and no-cache condition for variant cache memory. The cache hit ratio of DLCC is approximately 527% and 334% greater than RR and Zipf's probability distribution, respectively, for the total storage space of 600 GB. Moreover, the cache hit ratio obtained using the DLCC approach in this paper is compared with the transfer learning-based cooperative caching (LECC) strategy introduced in the paper [26] and is shown in Table 9.

Table 9. Comparison between DLCC and learning-based cooperative caching (LECC) approach on the basis of cache hit ratio.

Parameter	DLCC	LECC [26]
Number of F-APs	50	4
Total content items	9019 GB	500 GB
Total F-APs Capacity	600 GB	400 GB
Total F-APs capacity normalized by the total content items	0.066	0.8
Cache hit ratio	57%	55%

As per Table 9, we can see that the cache hit ratio for the DLCC approach is 57% for the 0.66 F-APs capacity (normalized by the total contents). Likewise, in the LECC-based approach, the cache hit ratio is 55% for the 0.8 F-APs capacity (normalized by the total contents). Based on the above comparative analysis, we can say that the DLCC approach is better than the LECC approach for proactive caching.

Figure 13 shows the overall delay in the F-RAN system for the proposed DNN-based proactive caching policy. The total system delay is calculated by using equations listed in (1), (2) and (3). It is assumed that each CPRI cable connecting CC to F-AP has an average downloading speed of 10 Gbps for a distance range of more than 10 km. The delay added by the DLCC in the F-RAN system is approximately 200% and 193% lesser than RR and Zipf's probability distribution method, respectively, for 600 GB of the storage capacity. As per the figure, the total average delay of 5.33 min is added to the system to download movies of cumulative size 400 GB for the no-cache memory scenario. Whereas, in the case of the DLCC scheme, a minimum value of total delay of 2.28 min can be experienced, provided that the total F-AP capacity is greater than 400 GB.

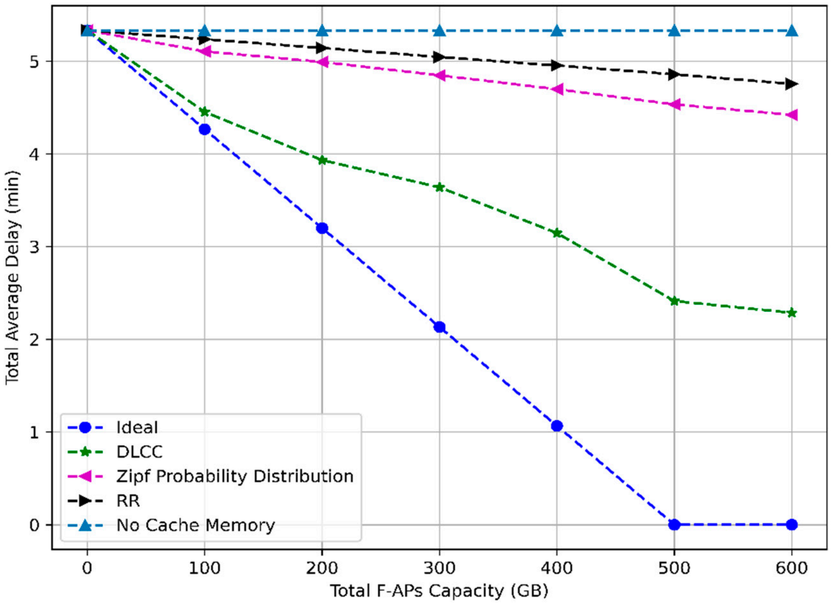


Figure 13. Total F-AP capacity vs. total system delay

5. Conclusions

In this paper, a 2D CNN-based DLCC approach is proposed to proactively store the most popular file contents in the cache memory of F-APs. For training the DLCC model, a publicly available MovieLens dataset containing the movie's historical feedback information is taken into account since movie files are responsible for a major portion of the fronthaul load in the F-RAN system. Simulation results showed that our proposed model acquired an average testing accuracy of 92.81%, which is around 21–54% greater than the prediction accuracy reported in the papers [15,27–29] while solving a similar problem. Likewise, when the trained model is deployed in the F-RAN system, it obtained the maximum cache hit ratio of 0.57 and an overall delay of 2.28 min. In comparison with RR and Zipf's probability distribution methods, the cache hit ratio obtained using DLCC is approximately 527% and 334% greater, and the overall delay is approximately 200% and 193% lesser, respectively. Moreover, the cache hit ratio reported in this paper is better than the cache hit ratio obtained using the LECC strategy.

Author Contributions: Conceptualization, S.B. and H.K.; methodology, S.B.; software, S.B. and N.R.; validation, S.B., N.R., P.K., H.K. and Y.-S.H.; resources, H.K. and Y.H.; data curation, S.B. and N.R.; writing—original draft preparation, S.B.; writing—review and editing, S.B., N.R., P.K., H.K. and Y.-S.H.; visualization, S.B. and P.K.; supervision, H.K.; project administration, H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://grouplens.org/datasets/movielens/.

Acknowledgments: This work was supported by Post-Doctoral Research Program of Incheon National University in 2017.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cisco. White Paper. Internet of Things at a Glance. Available online: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/iot-aag.pdf (accessed on 1 January 2021).
2. Anawar, M.; Wang, S.; Azam Zia, M.; Jadoon, A.; Akram, U.; Raza, S. Fog Computing: An Overview of Big IoT Data Analytics. *Wireless Commun. Mobile Comput.* **2018**, *2018*, 1–22. [CrossRef]
3. Peng, M.; Yan, S.; Zhang, K.; Wang, C. Fog Computing based Radio Access Networks: Issues and Challenges. *IEEE Netw.* **2015**, *30*, 46–53. [CrossRef]
4. Bhandari, S.; Kim, H.; Ranjan, N.; Zhao, H.P.; Khan, P. Optimal Cache Resource Allocation Based on Deep Neural Networks for Fog Radio Access Networks. *J. Internet Technol.* **2020**, *21*, 967–975.
5. Zeydan, E.; Bastug, E.; Bennis, M.; Kader, M.A.; Karatepe, I.A.; Er, A.S.; Debbah, M. Big data caching for networking: Moving from cloud to edge. *IEEE Commun. Mag.* **2016**, *54*, 36–42. [CrossRef]
6. Jiang, Y.; Huang, W.; Bennis, M.; Zheng, F.-C. Decentralized asynchronous coded caching design and performance analysis in fog radio access networks. *IEEE Trans. Mobile Comput.* **2020**, *19*, 540–551. [CrossRef]
7. Jiang, Y.; Hu, Y.; Bennis, M.; Zheng, F.-C.; You, X. A mean field game-based distributed edge caching in fog radio access networks. *IEEE Trans. Commun.* **2020**, *68*, 1567–1580. [CrossRef]
8. Blasco, P.; Gündüz, D. Learning-based optimization of cache content in a small cell base station. In Proceedings of the 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, Australia, 10–14 June 2014; pp. 1897–1903.
9. Yang, P.; Zhang, N.; Zhang, S.; Yu, L.; Zhang, J.; Shen, X. Content popularity prediction towards location-aware mobile edge caching. *IEEE Trans. Multimedia* **2019**, *21*, 915–929. [CrossRef]
10. Zhang, S.; He, P.; Suto, K.; Yang, P.; Zhao, L.; Shen, X. Cooperative edge caching in user-centric clustered mobile networks. *IEEE Trans. Mobile Comput.* **2018**, *17*, 1791–1805. [CrossRef]
11. Deng, T.; You, L.; Fan, P.; Yuan, D. Device caching for network offloading: Delay minimization with presence of user mobility. *IEEE Wireless Commun. Lett.* **2018**, *7*, 558–561. [CrossRef]
12. Xu, M.; David, J.; Kim, S. The Fourth Industrial Revolution: Opportunities and Challenges. *Inter J. Financ. Res.* **2018**, *9*, 1–90. [CrossRef]
13. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *arXiv* **2016**, arXiv:1506.02640.
14. Schmidhuber, J. Deep learning in neural network: An overview. *arXiv* **2014**, arXiv:1404.7828. [CrossRef]
15. Liu, W.; Zhang, J.; Liang, Z.; Peng, L.; Cai, J. Content Popularity Prediction and Caching for ICN: A Deep Learning Approach With SDN. *IEEE Access* **2018**, *6*, 5075–5089. [CrossRef]
16. Poularakis, K.; Iosifidis, G.; Tassiulas, L. Approximation algorithms for mobile data caching in small cell networks. *IEEE Trans. Wireless Commun.* **2014**, *62*, 3665–3677. [CrossRef]
17. Golrezaei, N.; Molisch, A.F.; Dimakis, A.G.; Caire, G. Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution. *IEEE Commun. Mag.* **2013**, *51*, 142–149. [CrossRef]
18. Tandon, R.; Simeone, O. Cloud-aided wireless networks with edge caching: Fundamental latency trade-offs in fog radio access networks. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, 10–15 July 2016; pp. 2029–2033.

19. Din, I.U.; Hassan, S.; Khan, M.K.; Guizani, M.; Ghazali, O.; Habbal, A. Caching in Information-Centric Networking: Strategies, Challenges, and Future Research Directions. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1443–1474. [CrossRef]
20. Wang, X.; Leng, S.; Yang, K. Social-aware edge caching in fog radio access networks. *IEEE Access* **2017**, *5*, 8492–8501. [CrossRef]
21. Hung, S.; Hsu, H.; Lien, S.; Chen, K. Architecture Harmonization between Cloud Radio Access Networks and Fog Networks. *IEEE Access* **2015**, *3*, 3019–3034. [CrossRef]
22. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
23. Sun, Y.; Peng, M.; Zhou, Y.; Huang, Y.; Mao, S. Application of machine learning in wireless networks: Key techniques and open issues. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3072–3108. [CrossRef]
24. Ranjan, N.; Bhandari, S.; Zhao, H.P.; Kim, H.; Khan, P. City-Wide Traffic Congestion Prediction Based on CNN, LSTM and Transpose CNN. *IEEE Access* **2020**, *8*, 81606–81620. [CrossRef]
25. Bastug, E.; Bennis, M.; Debbah, M. Living on the edge: The role of proactive caching in 5g wireless networks. *IEEE Commun. Mag.* **2014**, *52*, 82–89. [CrossRef]
26. Hou, T.; Feng, G.; Qin, S.; Jiang, W. Proactive Content Caching by Exploiting Transfer Learning for Mobile Edge Computing. In Proceedings of the GLOBECOM 2017—2017 IEEE Global Communications Conference, Singapore, Singapore, 4–8 December 2017; pp. 1–6.
27. Ale, L.; Zhang, N.; Wu, H.; Chen, D.; Han, T. Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network. *J. IEEE Internet Things* **2019**, *6*, 5520–5530. [CrossRef]
28. Tsai, K.C.; Wang, L.; Han, Z. Mobile social media networks caching with convolutional neural network. In Proceedings of the 2018 IEEE Wireless Communications Network Conference Workshops (WCNCW), Barcelona, Spain, 15–18 April 2018; pp. 83–88.
29. Thar, K.; Tran, N.H.; Oo, T.Z.; Hong, C.S. DeepMEC: Mobile Edge Caching Using Deep Learning. *IEEE Access* **2018**, *6*, 78260–78275. [CrossRef]
30. GroupLens. Available online: https://grouplens.org/datasets/movielens/ (accessed on 5 January 2021).
31. Hancock, J.; Khoshgoftaar, T. Survey on categorical data for neural networks. *J. Big Data* **2020**, *7*, 28. [CrossRef]
32. Jolliffe, I.T.; Cadima, J. Principal component analysis: A review and recent developments. *Philos. Trans. A Math. Phys. Eng. Sci.* **2016**, *374*. [CrossRef] [PubMed]
33. Loffe, S. Szegedy, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* **2015**, arXiv:1502.03167.
34. Wu, H.; Gu, X. Towards dropout training for convolutional neural networks. *Neural Netw.* **2015**, *71*, 1–10. [CrossRef]
35. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv* **2016**, arXiv:1609.04747.
36. Kingma, D.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
37. Visa, S.; Ramsay, B.; Ralescu, A.; Knaap, E. Confusion Matrix-based Feature Selection. In Proceedings of the 2011 CEUR Workshop, Heraklion, Crete, Greece, 30 May 2011; pp. 120–127.

electronics

Article

Application of Wireless Sensor Network Based on Hierarchical Edge Computing Structure in Rapid Response System

Yuechun Wang [1], Ka Lok Man [1,2,3,*], Kevin Lee [4], Danny Hughes [5], Sheng-Uei Guan [1] and Prudence Wong [6]

[1] Department of Computer Science and Software Engineering (CSSE), Xi'an Jiaotong-Liverpool University (XJTLU), Suzhou 215123, China; yuechun.wang@xjtlu.edu.cn (Y.W.); Steven.Guan@xjtlu.edu.cn (S.-U.G.)
[2] AI University Research Centre (AI-URC), Xi'an Jiaotong-Liverpool University (XJTLU), Suzhou 215123, China
[3] Faculty of Engineering, Computing and Science, Swinburne University of Technology Sarawak, Kuching 93350, Malaysia
[4] School of Information Technology, Deakin University, Geelong, VIC 3216, Australia; kevin.lee@deakin.edu.au
[5] imec-DistriNet, KU Leuven, B-3001 Leuven, Belgium; danny.hughes@cs.kuleuven.be
[6] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK; p.wong@liverpool.ac.uk
* Correspondence: ka.man@xjtlu.edu.cn

Received: 6 June 2020; Accepted: 16 July 2020; Published: 20 July 2020

Abstract: This paper presents a rapid response system architecture for the distributed management of warehouses in logistics by applying the concept of tiered edge computing. A tiered edge node architecture is proposed for the system to process computing tasks of different complexity, and a corresponding rapid response algorithm is introduced. The paper emphasizes the classification of abstracted outlier sensing data which could better match different sensing types and transplant to various application fields. A software-defined simulation is used to evaluate the system performance on response time and response accuracy, from which it can be concluded that common predefined emergency cases can be detected and responded to, rapidly.

Keywords: hierarchical edge computing; WSN; rapid response strategy; edge node

1. Introduction

Wireless Sensor Networks (WSNs) [1] have been widely applied to transport [2], agriculture [3], smart cities [4] and smart homes [5] domains as the critical environmental sensing infrastructure in Internet of Things (IoT) systems. Tens of thousands of ubiquitous sensors enable Wireless Sensor Networks to continuously capture large amounts of sensed data, which will keep on growing in the coming years. Even though WSNs have a strong sense of the environment with increasingly accurate data capture ability, how to give meaning to these huge amounts of data, and how to use these data intelligently and quickly in various mainstream applications are the challenges of current research [6]. The value of WSNs as the tentacles of IoT systems cannot be realized under inaccurate data analytics and delayed system response. Therefore, the real-time screening and efficient processing of these sensor data have become a hot research direction and challenge.

From the perspective of intelligent logistics, the warehouse is one of the data centers that generate massive interlaced and correlative data. For general warehouse management, there are two application uses of sensed data: one is for cargo management, which includes goods identification (using RFID) and goods tracking (location and movement); the other is for safety management, which refers to environmental monitoring and data security. It is practical to filter and process data rapidly for applications requiring a fast response and real-time tracking, such as those involving safety hazards [7]. Some existing warehouse applications enable safety hazards to be detected only by visualized raw data and human response at the proximity of the users and rely upon the higher-level computational ability and strategy-making ability of the cloud for prediction. The resource limitations of WSN infrastructures restrict the state-of-the-art paradigms such as deep learning, which requires large capacities in terms of computing and storage, to be implemented on terminal nodes [8]. Such a WSN-plus-cloud mode causes problems that lead to either high bandwidth usage or high latency in undertaking emergency interventions. To satisfy the requirements on both WSN localization and cloud globalization which cannot be satisfied by a simple WSN-Cloud architecture, a well-designed WSN-Edge-Cloud system architecture [9] would be a suitable solution. A WSN-Edge-Cloud system integrates edge computing features with the WSN-Cloud architecture to solve the problems mentioned above so as to improve the efficiency of business logistics.

Known as a new paradigm for the IoT domain, edge computing [10] moves complex computation from the cloud to the edge of the networks and devices. Edge computing does not aim to replace cloud computing, but rather focuses on balancing latency of the communications and accurate computation for the wider IoT hierarchy [11–13]. In a WSN-Edge-Cloud system architecture, edge computing can be regarded as a relay between the cloud and sensor nodes in the WSN [14]. The edge computing nodes extend the cloud computing paradigm to the edge of the network in a bidirectional way [15]. On the node-to-cloud direction, edge nodes focus on local functionality to support geographically closer sensing with the additional feature of data preprocessing and rapid reaction time. This aggregated, filtered and preprocessed data is sent to the cloud selectively according to application requirements. In the cloud-to-node direction, edge nodes achieve distributed deployment of the broad class of applications under the macrocontrol of the cloud and perform the tasks allocated by the cloud. Since edge computing reduces both the response time in IoT communications and the upload bandwidth to the cloud, the resulting applications will have the features of real-time interaction at the edge and prediction analysis in the cloud [16].

In this paper we propose a hierarchical structure for the edge layer in the WSN-Edge-Cloud architecture to meet the requirements of rapid response in the intelligent warehouse scenario. Rapid response indicates that by taking advantage of the edge computing, the system gives the first time response to exceptions near the data-generating location rather than waiting for the response strategies from cloud computing, so as to improve the response speed of the system. Instead of the standard approach of an integrated single edge layer in the WSN-Edge-Cloud architecture, this paper proposes a novel approach of a hierarchical edge layer with cooperated edge nodes in each of the different tiers.

In this hierarchical structure, the edge functions are modularized and abstracted according to the coupling degree of functions and applications. General functions that loosely coupled with concrete applications such as data formatting are achieved on low-level edge nodes while the application-specific functions are allocated to high-level edge nodes. Such grading strategy keeps the advantages of edge computing with low latency and enhances the reconfiguration of hierarchical modules at the edge of the network.

This paper is organized as follows: Section 2 presents the architecture and methodology of the proposed hierarchical structure in the rapid response system, Section 3 depicts the simulations and analyses the outcomes and finally, Section 4 gives some conclusions and discusses potential future work.

2. Architecture and Methodology

2.1. Hierarchical Edge Computing Structure

Compared with classical WSN system architectures, the proposed edge computing-based graded system architecture consists of three core layers as illustrated in Figure 1. As a widely accepted environmental sensing infrastructure, sensor nodes in the WSN collect sensing data and track changes to the environment continuously. For better identification and management, sensor nodes in the WSN are logically separated into different areas. By moving the computation to the edge of the networks, the edge computing layer reduces the response time during communications as well as the required upload bandwidth between the sensor networks and the cloud. Compared to just using the cloud, the edge layer is physically close to the environment. The functionality of the edge layer in the proposed structure is refined into three grades of edge nodes. Grade one and two edge nodes are focused on general functions including data formatting, preliminary data processing for WSN data collection, as well as the execution of tasks and control commands allocated by the upper layer (higher grade edge nodes or the cloud). Grade three edge nodes contribute to more complex data analysis, which involves data that is potentially useful for prediction and control, as well as generating or relaying control commands from the upper layer down to the lower layer. Cloud computing is docked to the cloud layer, which contributes to the centralized analysis of global data and management of the entire network. In addition, the connection between users and the system via the cloud realizes the remote operation and control of all areas covered by the terminal devices. For application developers, the system can be accessed via the cloud or edge node for application deployment depending on the deployment requirements and the network condition.

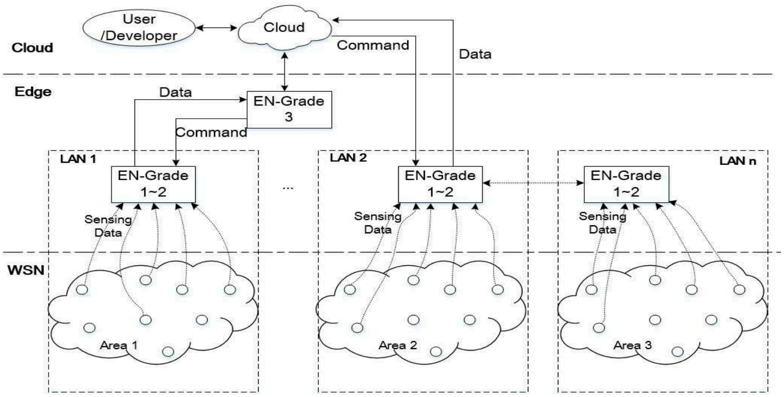


Figure 1. Edge-computing-based tiered system architecture. 'EN' indicates edge node. A smaller EN-Grade number indicates the physically closer location from the edge node to the sensor nodes and sensing devices.

2.2. Distributed Micro-Database

As stated, grade one and two edge nodes in the edge layer are focused on general functions such as data formatting, preliminary data processing for WSN data collection. A lightweight database (SQLite [17]) component is inserted between the grad-1 and grad-2 edge nodes in the system. The purposes of including a light-weight database are (a) capturing the data effectively, (b) supporting selective data retrieval, (c) filtering data without additional programming, and (d) enhancing data readability and translatability. The interaction between grade one and two edge nodes is illustrated in Figure 2.

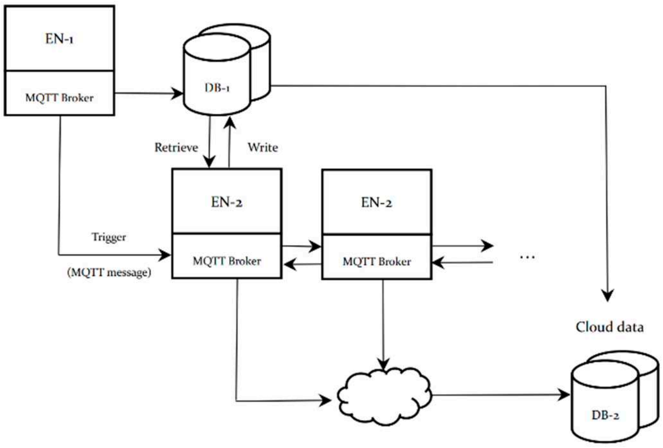


Figure 2. The interaction between grade one and two edge computing nodes in the architecture for WSN.

Compared to some popular databases such as MySQL [18] and PostgreSQL [19], SQLite is a good choice as it has low resource usage suitable for use in embedded products for IoT applications. Due to communication dependencies on the database, the main communication protocols are called via APIs within the program directly. SQLite has outstanding performance, with low-resource consumption, low latency, and overall simplicity of configuration and management.

As a common agent component in the WSN communication environment, message brokers that implement communication protocol would deal with messages from endpoints that generate massive data. The communication between EN-1 and EN-2 shown in Figure 2 is achieved by implementing message brokers on all the edge nodes. Since most of the IoT communication brokers such as Message Queuing Telemetry Transport (MQTT) brokers do not provide any mechanism for logging historical data, a script to log sensed data to SQLite is written based on python benefitting from the multiple programming language adaptation feature of SQLite. The script will log data on a collection of topics, which includes message time, message topic, and message payload. Considering the repeatability of the sensed data, the script only logs changed data from the status sensor, which indicates that if a status sensor sends its status as "ON" once per second, then it could result in 3600 times of "ON" messages logged every hour. However, the script only logs one message. The script uses the main thread to get the data (on message callback) and a worker thread to log the data. A queue is used to move the messages between threads. Once the data is placed in the queue, the worker will take it from the queue and log it into a disk. The worker is started at the beginning of the script.

2.3. Abnormal Data Type Abstract

In previous work [20], four abnormal sensor data types were defined for the rapid response system testing in the warehouse scenario, which were (i) rapid-growth, (ii) slow-growth-diffusion, (iii) slow-growth-nondiffusion, and (iv) error data on a single node. The classification was based on the data change rate and considered the diffusivity of the sensed object. However, to loosely couple the outlier detection approach with the specific sensing data type, the abnormal data types are summarized into three abstracts which could improve the portability of the system for adapting to various application scenarios. To support this, an enhanced outlier detection mechanism is also proposed. The following section presents the outlier type abstracts and related detection mechanisms.

Sensing information that is collected by sensors is classified into two forms, (i) numerical data and (ii) status data. In the detection and rapid response system, our monitoring method for numerical data anomalies is mainly to record comparison results, while the detection method for state data anomalies is to record state changes. The detection of numerical data is commonly based on a certain preset threshold, which specifies whether it is an outlier. Two methods are considered for setting a threshold: constant value and prediction formula. The constant value is applied for data with high stability such as temperature data from an incubator. Prediction formula, which requires preset formula or complex prediction calculations, is often used for sensor data with uncertain change trends and large overall fluctuations. Road vehicle flow monitoring sensor data is a typical example. Besides exceeding the threshold, data jitter within the boundary threshold is considered as a type of outlier as well. The core performance of data jitter is the sharp increase/decrease in the dispersion of values, i.e., the degree of deviation of the sensing data from the average value increases under the premise of the constant sampling interval. In the domain of WSN, sensor data has a strong relation with time series. Hence, the processing of sensor data needs to consider both the timing factor and the data volume. Considering the premise that the sensor nodes have limited computing capacity and limited storage space, we simplified the calculation of the real-time standard deviation of data jitter into the calculation of data increments, and set a threshold for the increments, thereby transforming data jitter anomalies into threshold anomalies.

As for the status data, an expect-states pattern, which holds all effective statuses of the sensor data, is used for the outlier detection. One of the typical status sensors is the bistate switch. In an application scenario where the bistate switch is required to be normally 'ON', the expect-states pattern will hold the 'ON' state only, which indicates the 'OFF' state is an outlier.

Based on the three outlier data abstracts presented above, a rapid response strategy for the edge nodes in the system to justify the abnormal types is proposed in the following section.

2.4. Rapid Response Strategy

Within a target monitoring area, there are two primary cases in which sensor nodes may generate abnormal sensing data: one is sudden environmental change, the other is error data caused by a broken sensor or irruption. A rapid response is only expected to be triggered by the first case, which could save time for emergency interventions and reduce the potential for business losses. In contrast, a rapid response caused by the second case will lead to a waste of resources. The aims of this strategy are to initially process the data near the edge, at the data generation end, quickly identify the types of abnormal data, and make response strategies. It avoids the system being overly sensitive caused by responses to the raw data; in the meanwhile, it could reduce the waiting time on uploading redundant data to the cloud as well as the waiting time on the instructions from the center server.

Table 1 lists the proposed outlier type abstracts and specifications. With the hierarchical edge computing architecture illustrated in this paper, we distribute three abstracts of outlier type detection to different tiers of EN. Based on the principle that the lower the EN's grade, the lower the computing complexity, EN-1 simply detects the status changes and numerical data that exceed the threshold with a constant value. EN-2 processes the data exceeding the threshold with prediction formula in coherence with historic data retrieved from the local micro database. The computing that involves real-time standard deviation based on massive data and related prediction will be allocated to the higher tiers of EN even to the Cloud. The detailed process is proposed as pseudo-codes in Algorithm 1. * r indicates compare result in both cases. Since thresholds may either be upper boundary or lower boundary, we use Boolean variable r to denote whether data value exceeds the boundary. ** Data gradient set: $Grad(n) = \{grad(n,i), n \in N, i\text{:timestamp}\}$. Data gradient: $grad(n,i) = [d(n,i + Tc) - d(n,i)]/Tc$, $n \in N$, i:timestamp.

Algorithm 1 Rapid response strategy

```
Inputs:
    Status pattern set: SP
    Threshold: TH
    Prediction formula: f
    Data set: D(n) = {d(n,i,tp,payload), n∈N, i:timestamp,tp:datatype}
    Node set: N = {$n_1$,$n_2$, . . . , $n_k$,k:number of nodes in the area}
Process on EN-1:
    detect d (n,i,tp,payload)
    write (d,Database)
    if tp == status:
        compare (payload, SP) => r *
        if r == true:
            report (“status change”)
    else:
        compare (payload, TH) => r
        if r == true:
            report (“exceeding threshold”)
            generate M=(n,i)
            activate EN-2
Process on EN-2:
    observe M
    retrieve ($D_{nb}$,Database)
    compare (f(m),TH) ==> r
    if r == true:
        report (“exceeding threshold”)
    compute Grad (m) **, where m∈$D_{nb}$
    #{grad (m,i)>0}/#{$N_{nb}$} ==> r
    If r == true:
            Report (“Jitter”)
    else:
            activate EN-3
```

Table 1. Outlier type abstracts and specifications.

Outlier Types	Quantitative Reference	Allocated EN
Exceeding threshold	Constant value Prediction formula	EN-1 EN-2 and u-Database
Jitter	Real-time standard deviation Increment	EN-3 and u-Database EN-2 and u-Database
Status change	Status pattern	EN-1

3. Implementation

In this section, the proposed method is evaluated using software-defined networks. The platform, hardware settings, response accuracy are reported.

3.1. Implementation Platform and Hardware

Considering that massive sensor nodes in IoT applications cannot be completely simulated in a laboratory scenario for research experiments, we selected an open-source flow-based tool and platform called Node-RED [21] for implementing the architecture proposed in this paper. Node-RED is used as a platform to integrate components at multiple layers in the network. It provides a convenient connection with the web interface to visualize the data flow and configure the network. In our simulation, ENs communicate by subscribing to a selected topic on a broker, and the messages/data coming into the topic could be observed on the web interface. The Node-RED is run on both the Raspberry Pi (Rpi) machines and laptops as the initial setup of the network. With Node-Red running on the equipment, we could manage and configure the network and component connection using a browser. Thus, all the platform and hardware were set up as listed in Table 2. Besides the physical sensors, python scripts were used to simulate sensing data for various scenarios that currently cannot be implemented in the laboratory.

Table 2. Simulation platform and hardware.

Implementation	Networking (Functionalities)
a. SN, AN *: (a) Arduino board (b) Python code (simulation)	Data collection Data formatting (JSON)
b. EN *: edge computing tasks (a) MQTT broker (on Rpi, PC) (b) CoAP MA (on PC)	Response tasks – Modularised – Reconfigurable – Cooperative
c. Web-based Client: (a) MQTT Lens (b) CoAP client	Data visualisation
d. Network: (a) Node-RED simulator	-
e. Cloud: (a) IBM cloud (plan)	-

* SN: sensor node, AN: actuator node, EN: edge node.

As is shown in Table 2, Raspberry Pi and PC were used as edge nodes and the MQTT broker, as well as CoAP MA, were deployed on edge. Users could access the web-based client to retrieve raw data and preprocessed data. The network for the whole system was simulated based on Node-RED simulator. The cloud is planned to apply IBM open-source cloud as some of the IoT application components are integrated and free to use. The block diagram in Figure 3 shows the architecture of the simulation environment, which contained the components listed in Table 2. Figure 4 illustrates an example Node-red flow that implements the functions of EN-1.

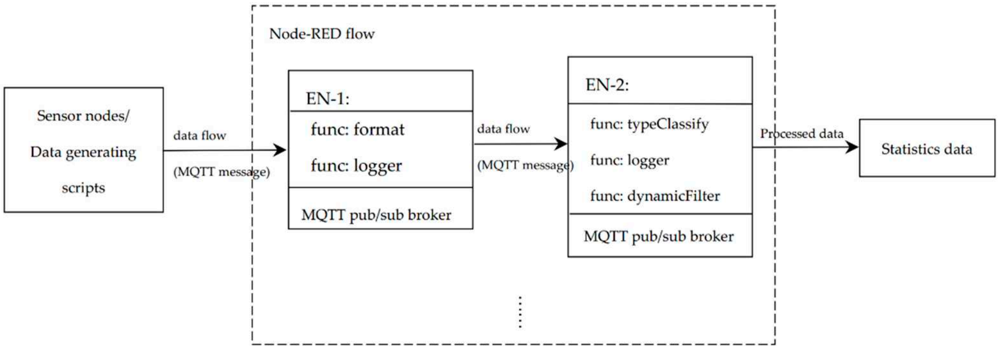


Figure 3. Architecture of the simulation environment.

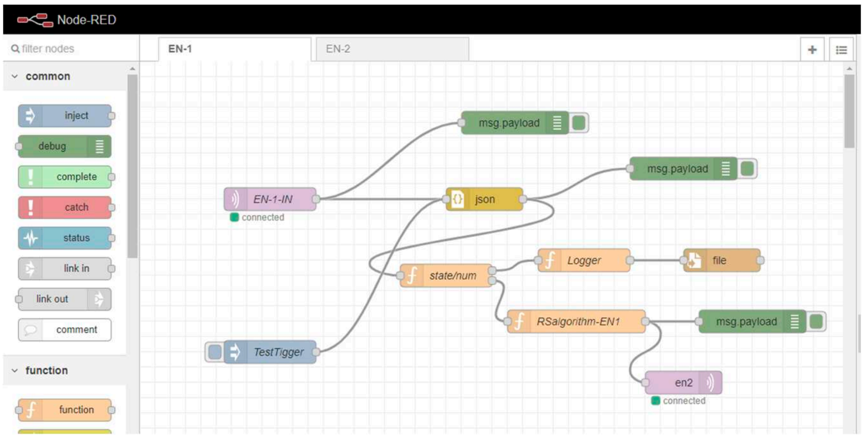


Figure 4. Example Node-RED flow that implements the functions of EN-1.

3.2. Case Study Model

The experiment scenario is a cognitive warehouse which could recognize and respond to unexpected cases as illustrated in Table 1 rapidly. Temperature was selected as the sensing data type during the case study. The sampling rate of sensor data was configured as once per ten seconds, once per thirty seconds, and once per minute for three groups of repeating tests. The network scale was configured as 1000 nodes, 5000 nodes and 10,000 nodes, respectively.

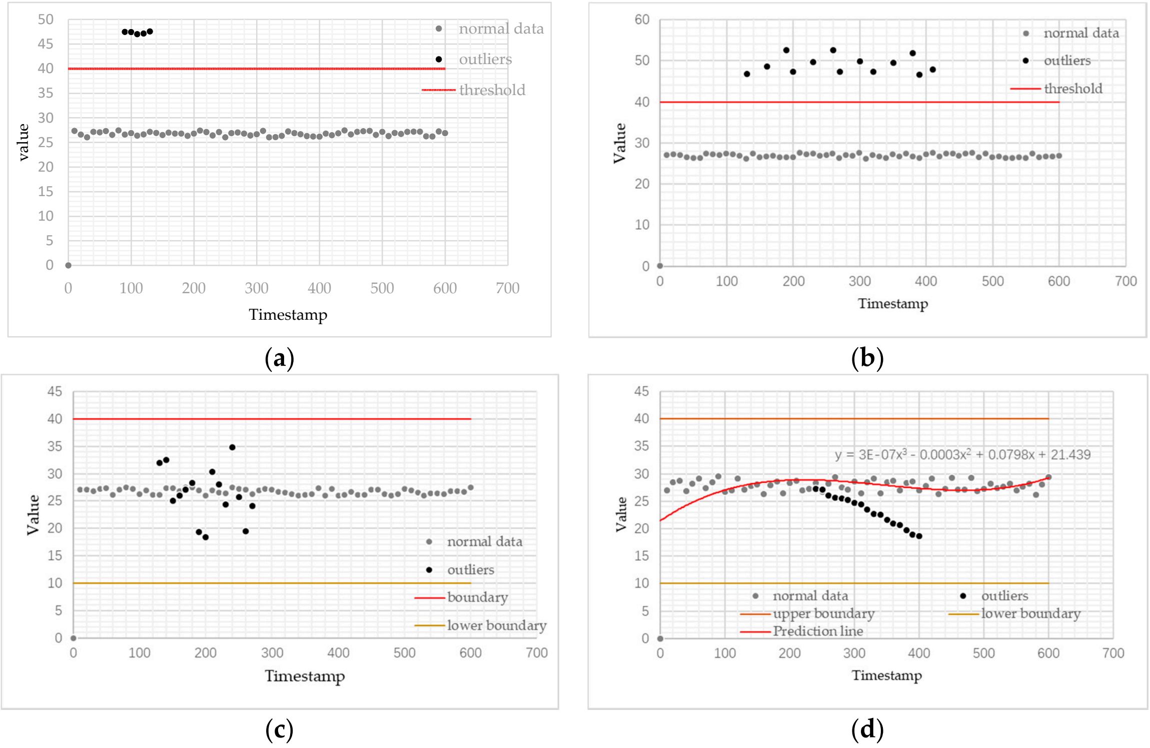
When setting the network scale as 1000 nodes, the nodes were grouped into 50 LANs which kept generating sensed data. Every 20 sensor nodes in the same group were considered as neighbors and connected with EN in Star topology. Besides the normal data (set as 26 ± 1 centigrade for initialization), three types of outlier data were inserted into the data set randomly, which included over the boundary (constant and predicted), jitter, and error. Figure 5 illustrates four example cases during our experiments with fixed sampling rate as 10 per second and fixed network scale as 1000 nodes. Figure 5a–c judge outliers based on the constant threshold. The difference between (a) and (b) was the duration that outliers been detected. Both of these cases would be reported as "exceeding threshold" while the case in (b) needed to be further processed by EN-2. The case shown in Figure 5c indicated jitter happened to the sensor node. The single node's value did not exceed preset boundaries, but it broke the stability of the sensed data, which could be considered as one of the general outlier types. The case shown in Figure 5d denoted the threshold based on the prediction formula. Data which deviated from the expected scope would be considered as outliers.

Such data flows were tested under three different network scales with two different sampling rates by the system. The outcomes and related analysis are presented in the next section.

3.3. Outcomes and Analysis

To normalize the analysis, we only tested and compared the numerical sensor data during the experiment. Instead of practical environmental data, our dataset was designed for testing the proposed architecture and algorithm performances in the simulation. During the test under the setting of 1000-node network scale and $1/10_{sec}$ sampling rate, 50 groups of data were generated and tested corresponding to the WSN in 50 LANs. Packs of sensor nodes for each corresponding edge node were configured as 20, which was a fixed parameter during the whole experiment. In other words, every 20 sensor nodes in the same group were considered as neighbors that shared data at their corresponding edge node. Four types of numerical outliers were inserted into a different dataset for each group, which were 'error', 'exceed threshold', 'jitter', as well as 'dynamic threshold'. The system average response time and accuracy were used to reflect the system performance. The system response time was calculated by $(T_{response} - T_{detect})/f_{sampling}$, where $T_{response}$ denoted the timestamp that the system responded to the outlier, T_{detect} denoted the timestamp that the outlier was detected, and $f_{sampling}$

denoted the sampling rate of the sensed data. The accuracy was calculated by dividing the number of outliers that were detected correctly by the number of original datasets on each end node.



(**a**) (**b**) (**c**) (**d**)

Figure 5. Example cases for system testing. (**a**) Outliers exceed a constant threshold in a short period (**b**) Outliers exceed a constant threshold in a long period (**c**) Outliers act as jitter within boundary threshold (**d**) Outliers deviate prediction line.

The comparison of the statistical testing results with different parameters are shown in Figures 6 and 7. The original data are shown in the tables in Appendix A.

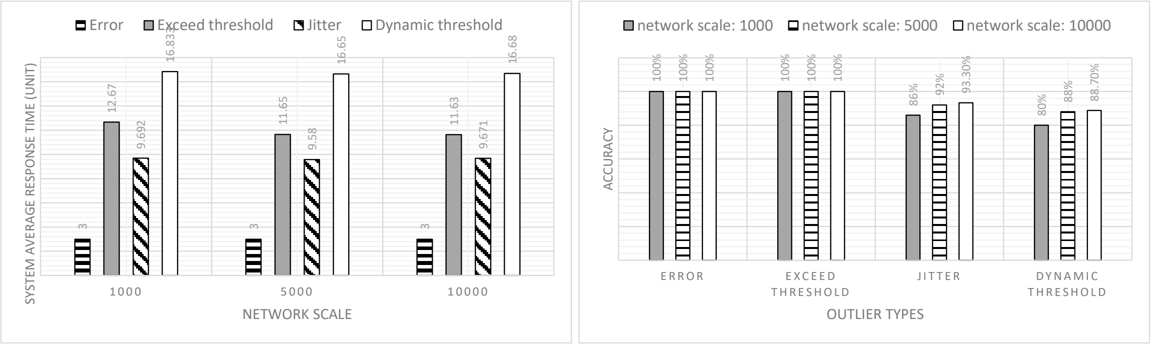


Figure 6. The comparison of the system average response time and accuracy under different network scales and fixed sampling rate (sampling rate = 1/10$_{\text{sec}}$).

By analyzing the statistical outcomes shown in Figures 6 and 7, it can be observed that cases related to the errors and exceeding the threshold could be 100% detected by the system. However, the system average response time had an obvious difference. The results would not be affected by the network scale and the sampling rate. As introduced in Sections 2.3 and 2.4, any data process that required the retrieval of historical data from the local database needed to be handled on EN-2. Therefore, extra data retrieval time and increment calculation led to a relatively long system average response time for the exceed threshold cases. The cases relating to the jitter had above eighty percent accuracy with a 9.69 sampling unit average response time which indicated that the edge system could detect the jitter-type data exception successfully within 10 sampling units. Eighty percent of the tested outliers defined by the dynamic threshold could be detected by the edge system within 17 sampling units. The sampling interval did not impact the system average response time and the response accuracy due to the unity of the sampling rate that was preset in the algorithm and the sampling interval configured in the simulation environment.

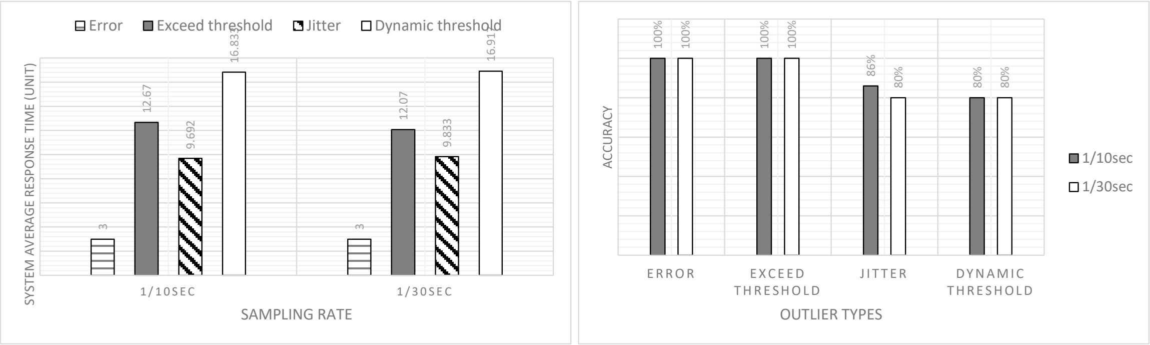


Figure 7. The comparison of the system average response time and accuracy under different sampling rates and fixed network scale (network scale = 1000).

4. Discussion

This paper has proposed a rapid response system architecture designed for data outlier detection, which involves the concept of hierarchical edge computing and brings the advantages of edge computing—its low latency to the edge of the network. For the case study of distributed warehouse management in logistics, an algorithm for distinguishing and rapidly responding to emergency cases was proposed. Three types of outlier abstracts were emphasized in this paper in order to promote the portability of the system. The performance of the system was evaluated by a software-defined simulation, with a focus on accuracy and rapidness of the Grade-1 and Grade-2 edge nodes in the system. Applying the rapid response algorithm showed that above eighty percent of the abnormal cases could be detected and responded to, in less than twenty sampling units.

A handful of the jitter cases that were not detected by the system were mainly caused by the 'break' during the jitter period. The system judged that the case did not satisfy the predefined jitter condition. One of the potential approaches for improving this is to use a machine learning model for EN-3 so that decisions are made for the lower layers. Importing artificial intelligence models onto Grade-3 edge nodes is one of the avenues to improve the response accuracy, which could enable decisions to be made on the edge of the cloud. The same situation happens to the cases related to the dynamic threshold. Once the deviation of outlier data is not significant, the system may skip the abnormal cases. Thus, to implement the entire system architecture as proposed in this paper, a clear direction for future research is the implementation of Grade-3 edge nodes, which potentially focuses on the short-time prediction. Besides the edge computing layer, the interaction and interoperation between the edge and the Cloud are also a valuable direction to extend our research.

Furthermore, some practical issues in the WSN communication environment such as message collisions, dead nodes, hotspots, etc., are valuable but have not been considered at the current research stage. We consider them as a potential research direction and will do more investigation and experiments to enhance our research outcomes in the future.

Author Contributions: Conceptualization, Y.W. and K.L.M.; methodology, Y.W.; software, Y.W.; validation, Y.W.; formal analysis, Y.W.; investigation, Y.W.; resources, Y.W.; supervision, K.L.M., S.-U.G., P.W.; writing—original draft preparation, Y.W.; writing—review and editing, K.L.M., K.L., D.H. All authors have read and agreed to the published version of the manuscript.

Funding: The research was funded by Xi'an Jiaotong-Liverpool University, Suzhou, China, the Research Development Fund (RDF-14-03-12), and XJTLU Key Program Special Fund (KSF-P-02).

Acknowledgments: Ka Lok Man wishes to thank the AI University Research Centre (AI-URC) and Research Institute of Big Data Analytics (RIBDA), Xi'an Jiaotong-Liverpool University, Suzhou, China, for supporting his related research contributions to this article through the XJTLU Key Program Special Fund (KSF-P-02) and RIBDA research subsidy fund.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Tables of the Statistical Outcomes with Various Sampling Rates and Network Scales

Table A1. Statistical outcomes of the experiment (sampling rate = 1/10$_{\text{sec}}$, network scale = 1000 nodes).

Outlier Types	Number of Original Data Sets	Number of Outliers Being Detected Correctly	System Average Response Time (Unit)	Accuracy
Error	5	5	3	100%
Exceed threshold	15	15	12.67	100%
Jitter	15	13	9.692	86%
Dynamic threshold	15	12	16.833	80%

Table A2. Statistical outcomes of the experiment (sampling rate = 1/10$_{\text{sec}}$, network scale = 5000 nodes).

Outlier Types	Number of Original Data Sets	Number of Outliers Being Detected Correctly	System Average Response Time (Unit)	Accuracy
Error	25	25	3	100%
Exceed threshold	75	75	11.65	100%
Jitter	75	69	9.58	92%
Dynamic threshold	75	66	16.65	88%

Table A3. Statistical outcomes of the experiment (sampling rate = 1/10$_{\text{sec}}$, network scale = 10,000 nodes).

Outlier Types	Number of Original Data Sets	Number of Outliers Being Detected Correctly	System Average Response Time (Unit)	Accuracy
Error	50	50	3	100%
Exceed threshold	150	150	11.63	100%
Jitter	150	140	9.671	93.3%
Dynamic threshold	150	133	16.68	88.7%

Table A4. Statistical outcomes of the experiment (sampling rate = 1/30$_{\text{sec}}$, network scale = 1000 nodes).

Outlier Types	Number of Original Data Sets	Number of Outliers Being Detected Correctly	System Average Response Time (Unit)	Accuracy
Error	5	5	3	100%
Exceed threshold	15	15	12.07	100%
Jitter	15	12	9.833	80%
Dynamic threshold	15	12	16.917	80%

References

1. Internet of Things: Wireless Sensor Networks, International Electrotechnical Commission (IEC). Available online: https://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf (accessed on 16 July 2020).
2. Hodge, V.J.; O'Keefe, S.; Weeks, M.; Moulds, A. Wireless sensor networks for condition monitoring in the railway industry: A survey. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 1088–1106. [CrossRef]
3. Ojha, T.; Misra, S.; Raghuwanshi, N.S. Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges. *Comput. Electron. Agric.* **2015**, *118*, 66–84. [CrossRef]
4. Lu, W.; Gong, Y.; Liu, X.; Wu, J.; Peng, H. Collaborative energy and information transfer in green wireless sensor networks for smart cities. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1585–1593. [CrossRef]
5. Ghayvat, H.; Mukhopadhyay, S.; Gui, X.; Suryadevara, N. WSN- and IOT-Based Smart homes and their extension to smart buildings. *Sensors* **2015**, *15*, 10350–10379. [CrossRef] [PubMed]

6. Zhang, G.; Li, R. Fog computing architecture-based data acquisition for WSN applications. *China Commun.* **2017**, *14*, 69–81. [CrossRef]
7. Kantarci, B.; Mouftah, H.T. Trustworthy sensing for public safety in cloud-centric internet of things. *IEEE Internet Things J.* **2014**, *1*, 360–368. [CrossRef]
8. Martin Fernandez, C.; Diaz Rodriguez, M.; Rubio Munoz, B. An edge computing architecture in the internet of things. In Proceedings of the 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC), Singapore, Singapore, 29–31 May 2018; pp. 99–102. [CrossRef]
9. Mihai, V.; Dragana, C.; Stamatescu, G.; Popescu, D.; Ichim, L. Wireless Sensor network architecture based on fog computing. In Proceedings of the 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT), Thessaloniki, Greece, 10–13 April 2018; pp. 743–747. [CrossRef]
10. Elazhary, H. Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *J. Netw. Comput. Appl.* **2019**, *128*, 105–140. [CrossRef]
11. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the MCC '12: Proceedings of the First Edition of the Mcc Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012.
12. Caprolu, M.; di Pietro, R.; Lombardi, F.; Raponi, S. Edge computing perspectives: Architectures, technologies, and open security issues. In Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE), San Diego, CA, USA, 25–30 June 2019; pp. 116–123. [CrossRef]
13. Khan, L.U.; Yaqoob, I.; Tran, N.H.; Kazmi, S.M.A.; Dang, T.N.; Hong, C.S. Edge computing enabled smart cities: A comprehensive survey. *IEEE Internet Things J.* **2020**. [CrossRef]
14. Rausch, T.; Nastic, S.; Dustdar, S. EMMA: Distributed QoS-Aware MQTT middleware for edge computing applications. In Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, USA, 17–28 April 2018; pp. 191–197. [CrossRef]
15. Hong, C.-H.; Varghese, B. Resource management in Fog/Edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Comput. Surv.* **2019**, *52*, 1–37. [CrossRef]
16. Preden, J.; Kaugerand, J.; Suurjaak, E.; Astapov, S.; Motus, L.; Pahtma, R. Data to decision: Pushing situational information needs to the edge of the network. In Proceedings of the 2015 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision, Orlando, FL, USA, 9–12 March 2015; pp. 158–164. [CrossRef]
17. SQLite Home Page. Available online: https://www.sqlite.org/index.html (accessed on 23 June 2020).
18. MySQL. Available online: https://www.mysql.com/ (accessed on 24 June 2020).
19. PostgreSQL: The World's Most Advanced Open Source Database. Available online: https://www.postgresql.org/ (accessed on 24 June 2020).
20. Wang, Y.; Man, K.L.; Hughes, D.; Guan, S.; Wong, P. A rapid response approach applying edge computing for distributed warehouses in WSN. In *Advanced Multimedia and Ubiquitous Engineering*; Springer: Singapore, 2020; pp. 183–189.
21. Node-RED. Available online: https://nodered.org/ (accessed on 23 June 2020).

MDPI
St. Alban-Anlage 66
4052 Basel
Switzerland
Tel. +41 61 683 77 34
Fax +41 61 302 89 18
www.mdpi.com

Electronics Editorial Office
E-mail: electronics@mdpi.com
www.mdpi.com/journal/electronics

MDPI
St. Alban-Anlage 66
4052 Basel
Switzerland

Tel: +41 61 683 77 34
Fax: +41 61 302 89 18

www.mdpi.com

ISBN 978-3-0365-4275-1