A Resilient Fog-IoT Framework for Seamless Microservice Execution

Md Whaiduzzaman*, Alistair Barros†, Ahmedur Rahman Shovon‡, Md Razon Hossain§, and Colin Fidge¶
Queensland University of Technology, Brisbane, Australia*†¶
Jahangirnagar University, Dhaka, Bangladesh ‡§
Email:*wzaman@juniv.edu, †alistair.barros@qut.edu.au, ‡shovon.sylhet@gmail.com, §hossainmdrazon@gmail.com, ¶c.fidge@qut.edu.au

***Abstract*—Microservices have been proposed as the software architecture style for fog-IoT network applications ecosystems. Recently, microservices have been extensively used in fog-IoT ecosystems. Here, master-worker fog-based frameworks have been widely adopted in the ecosystem to ensure resilience. However, the architecture's reliability, including the possibility of the master fog node failure, and service unavailability, is not reflected adequately in the literature. Therefore, we present a resilient master-citizen fog-IoT framework to ensure efficient resource management and overall system reliability. In this work, we develop a master fog node and layered citizen nodes in the distributed ecosystem. Our fault-tolerant fog-IoT-based microservice execution framework can ensure efficient recovery from a single point of failure, unavailability, and unexpected events in the master fog node, which means the network can continue working after a system failure. We use different fault-tolerance strategies and algorithms for selecting the master fog node that synchronizes with other citizen fog nodes and the upper layer cloud for efficient microservice execution. Finally, we developed and implemented a resilient fog-IoT network for providing uninterrupted services in the event of master fog node failure.**

***Index Terms*—Microservice, Fog Computing, IoT, Fault Tolarance framework**

I. Introduction

Fog computing is the new complement of the cloud computing paradigm and becoming popular day by day. [1]. Enterprise systems are now coming closer to the cloud with the help of the new trend of fog computing. [2]. Fog Computing, as proposed by Cisco [3] has shifted the centralized cloud computing model through a diffuse network of lightweight nodes as the underpinning infrastructure of the Internet of Things(IoT). It enables software to execute on small, low-powered devices of fog nodes connected closely to IoT objects at the granularity of microservices. With the emerging Fog computing and IoT, models the technological world has shifted to a new paradigm. Many applications, architectures, frameworks come together with mobile devices to complement this new paradigm [4, 5]. These IoT devices use many applications and programs to solve their tasks quickly. However, due to limited resources, these devices offload their jobs and use the services offered by Fog IoT environments [6, 7, 8].

The advantage of the layered architecture of the cloud fog IoT ecosystem is demonstrated and discussed in literature which can perform efficient task scheduling and faster application execution [9]. Recent trends include growing interest in using microservices architecture to address Fault Tolerance (FT) in IoT ecosystems[10]. However, there is a failure to address fault tolerance issues for fog IoT ecosystems in the current literature. We understand that a Master fog node plays a vital role in this architecture [8]; hence to avoid a single point of failure, we need planning and a proper fault tolerance mechanism in place to protect the essential Master fog node [11] in the framework.

The master-worker framework is introduced to accelerate the efficiency of the system. The master fog resides among the general-purpose fogs, either horizontally or vertically, and responsible for performing essential tasks such as API request scheduling, communication with the cloud, resource management. However, the failure of the master node hinders these tasks and, in the worst case, can shut down the whole system. Therefore, our proposed framework focuses on the fog layer to increase the resilience of the system.

We design the selection criteria considering the dynamic available resources of workers fog node to estimate the best master fog node selection. In addition, we periodically collect the available resources among the fog colony and decide the possible primary, secondary and tertiary master fog nodes. We place relevant algorithms and explain the all possible steps and explain the situations by comparing resource. In this process, we consider the essentials microservices movement and exceptions to essentials services running in the citizen fog nodes. We also consider the master fog nodes responsibility and database transfer strategy to secondary , tertiary and cloud to mitigate the strategics plan to recovery from the resilient fault tolerance system.

We ensure storing the snapshots of all fog devices' health statuses in each device synchronized with the master fog and the cloud application at a regular interval. Our framework analyze each fog node's score whenever we need to select a primary master fog. The score is calculated from the last thirty snapshots of data from each fog node. The scoring weight from the configuration file at run time can be modified based on the system requirements. Each fog node's score is calculated based on these factors and the configuration file's scoring weights. We selected the master fog and candidate master fog nodes from the top scores and selection score is synchronized with the cloud application if the master fog, the secondary master fog, and the tertiary master fog become unavailable. We assess our framework's fault tolerance for several scenario illustrates CPU usage and memory usage of the Kubernetes edge cluster. Overall, the system remains alive in all four scenarios and effectively handles the single point of failure.

The aim of this research is to design and validate process for master fog node replacement in a fault tolerant fog computing network. We focus on the Master fog node failure, a new master fog node, a secondary and tertiary master fog selection, and ensuring resiliency in the recovery process within in a minimal time. We propose a fault tolerance framework to provide resilience and support to the user when the system failure occurs or the system unavailability; still, the user enjoys the seamless connectivity and minimal downtime experienced by the framework.

Therefore, we address these research objectives:

RO1: To select contingent master fog nodes based on a periodic computational resource capacity estimation strategy in Fog-IoT ecosystems.

RO2: To prepare contingent master fog nodes for efficient enactment of master fog reallocation when a failure occurs

This research presents a fault-tolerant microservice cloud fog-IoT ecosystem framework to avoid single point of Master node failure and ensure uninterrupted service availability. We discuss the framework's internal components, and segregate the Master and citizen fog working component functionality. We also devise several algorithms to select the possible master fog node to recover the failure point. Our experimental results show that we can provide fault-tolerant seamless, resilient, uninterrupted services. Here, Table I presents the abbreviations used throughout the paper.

In this paper, our contributions are as follows:

1) We design a resilient master fog node selection process that provides seamless execution in a fog-IoT eco system.
2) We implement our developed master fog selection algorithm that ensures uninterrupted services in the case of master fog node failure.
3) We experiment with practical data and validate that our system can run smoothly and seamlessly in a fault-tolerant environment.

Essentials abbreviations used in the paper are enlisted in Table I. The rest of the paper is organized as follows: Section 2 represents the background and related works; Section 3 presents the fog-IoT framework overview; Section 4 presents the system design and modeling formulation; Results and discussion are highlighted in Section 5; and finally, Section 6 concludes our work with possible future research directions.

II. Related Work

This section discusses related works, issues, problems, and possibilities of several previous research works in fault tolerance and resilient networks in the cloud, fog, and IoT environments.

TABLE I: Essential Abbreviations

Short Form	**Full Form or definition**
PMF	Primary Master Fog
SMF	Secondary Master Fog
TMF	Tertiary Master Fog
CF	Citizen Fog
WF	Worker Fog
MS	Microservices
IoT	Internet of Things
CPU	Central Processing Unit
FT	Fault Tolereant
FTSM	Fault Tolerant Scheduling Method
IoTEF	Intenet of Things Edge Cloud Federation
CEFIoT	Cloud and Edge Fog Tolerant IoT
FTIoT	Fault Tolerant IoT
TBFC	Tree Based Fog Computing

Asad Javed et al. [11] proposed a novel fault-tolerant architecture CEFIoT for IoT applications by adopting state-of-the-art cloud technologies and deploying them for edge computing. They solved the data fault tolerance issue by exploiting the Apache Kafka publish/subscribe platform as the unified high-performance data replication solution. Also, the authors point out Masternode, Worker, or citizen fog node with Kubernetes orchestration. Umar Ozeer et al. [12] transferred state saving techniques based on an uncoordinated checkpoint, messages log, and function call record for stateful IoT applications in the fog, taking into account the specifics of the environment. Kun Wang et al. [13] proposed a Reduced Variable Neighborhood Search (RVNS)-based sEnsor Data Processing Framework (REDPF) to enhance data transmission and processing speed reliability. Functionalities of REDPF include fault-tolerant data transmission, self-adaptive filtering, and data-load-reduction processing.

Alexander [14] proposed a framework based on a microservices architecture that provides reactive and proactive FT support with two microservices: Real-Time FT, complex event processing and analyzing stream data for rapid error recovery; and Predictive FT, using machine learning to learn fault patterns and mitigate future faults before they occur. Shu-Ching et al. [15] proposed a protocol to achieve all fault-free nodes with minimal message exchanges and tolerate the maximum number of a dormant and malicious faulty component named IFCAP. Jitender Grover et al. [16] proposed a novel agent-based reliable and fault-tolerant hierarchical IoT-cloud architecture, which is distributed over four levels (cloud-fog-mist-dew) based on the end IoT device's processing power and distance. Nader et al. [17] investigated the issues of reliability and fault tolerance for fog platforms supporting IoT-based smart cities. Oma et al. [18] proposed a fault tolerance tree-based fog computing (FT tree-based fog computing (FTBFC) model using minimum energy and execution time to a new fault-tolerance strategy. Asad et al. [19] proposed a new Internet of Things Edge-Cloud Federation (IoTEF) architecture for multi-cluster IoT applications by adapting Cloud and Edge Fault-Tolerant IoT (CEFIoT) layered design.

We found three layers of Fog IoT and Cloud framework ensure faster processing and efficient resource management from the above discussion. Several pieces of research show the potentiality and introduced the fault tolerance system in several ways. However, there is a lack of a combined efficient Master Fog node three-layer system and a resilient fault tolerance system. We are motivated to harness the efficient resource utilization framework of three-layer architecture with Master Fog node IoT ecosystem incorporating the fault tolerance capability to achieve an improved resilient efficient fault-tolerant system.

III. Fog-IoT Framework

This section presents our overall fog-IoT and cloud framework overview and discusses how they are tied up with each other and work together. Figure 1 shows the overview of our new framework. A three-layer fog-IoT framework consists of IoT devices in the first layer, fog devices in the second layer, and cloud in the third layer.

A. First Layer: IoT and end devices

The first layer consists of IoT and end devices, making API requests to the Fog layer. Each request goes through the *Auth* service and API gateway. The Auth service then authenticates the requests, and the API gateway routes the request to the corresponding citizen fog.

B. Second Layer: Fog nodes

The second layer consists of two types of fogs, a Master Fog (MF) and a few Citizen Fog (CF) nodes. Citizen fogs are the general-purpose fog nodes, and microservices run in these citizen fogs. CFs has two more components, Request Scheduler (RS) and Request Router (RR). The scheduler receives the request and schedules the API requests for the microservices. If the CF has overflowed with API requests, the API request sends to the master fog.

There are three kinds of master fogs, Primary Master Fog (PMF), Secondary Master Fog (SMF), and Tertiary Master Fog (TMF). The Cloud selects these master fogs from citizen fogs using the master selection algorithm (Algorithm 1). Only one master fog is active at a time; at the beginning, the PMF stays active. If PMF does not respond for a specific time, an SMF takes over and announces itself as the master fog. If both the PMF and SMF are not responding, TMF announces itself as the master fog. If all three master fogs are not responding or are down, cloud re-selects the master fogs from the rest of the working citizen fogs using Algorithm 1.

Each master fog has three main components, Microservice Migration Handler (MMH), MF Broadcaster (MFB), and CF health Status Table (CST). Master fogs are a special kind of node; therefore, when a citizen fog is selected as the master fog, the MMH handles existing microservices' migration. Moreover, the MFB handles the confirmations and announcement of the active status of the master fog. The CST keeps track of the resource and computation status of all the citizen fog. If there is one or multiple eligible CF, a request is sent to the CF with the most resource and computing ability. The Master fog also sends the CST to the cloud application.

The following are the general responsibilities of an MF:

- Citizen fog Orchestration.
- Request handling from CFs and scheduling.
- Automatic application Deployment.
- Worker fog failure handling.
- Citizen fog health status management.
- Communication with the cloud and other MFs to share its own state.

C. Third layer: Cloud

The Cloud application is responsible for selecting the master fogs in two cases. The first case is setting up the framework initially, and the second case is when all three master fogs are not responding. The Master Fog Selector (MFS) uses the CST data and runs Algorithm 1 to select the master fogs. This CST data in the cloud is continuously updated and synchronized by the MF. Moreover, the cloud also receives the alive signals from all the MFs.

IV. System Design and Modeling

This section discusses, outlines, and explains our system's design concepts, parameters, metrics, components, algorithms, and model design.

A. Master Fog Selection Process

The Master fog is responsible for CF orchestration, scheduling, and resource management. Therefore, the MF needs to have a higher resource and computation ability. For simplicity, the MF does not have any microservice running end device application execution. To be eligible as an MF, the CF must migrate its existing microservices. We assign each MF selection criteria as a weight that dynamically changes based on the system status and fog node priority.

TABLE II: Master Fog selection criteria

Category	**Criteria**
HW Configuration	Total RAM
	Total ROM
	Number of Processing Units
	CPU Maximum Clock Speed
Snapshot Data	Current Available RAM
	Current Available ROM
	CPU Usage

1) Exception Criteria: There can be fog nodes dedicated to some unique or critical tasks and should not be considered for the Master Fog node. For instance, an authorization service, or particular database, or a dedicated fog node for any particular important service. These CFs will not be included in the MF candidate list.

The cloud component, Master Fog Selector (MST), executes our Master Fog Selection algorithm (Algorithm 1). For each citizen fog, it calculates weighted points for each criteria by using Equation 1 and Equation 2 in Section 4.3 below. Subsequently, it calculates the sum of weighted points and declares the CF with the highest total as the Primary master fog, the CF with the second-highest total is considered the secondary master fog, and the third one is considered the tertiary master

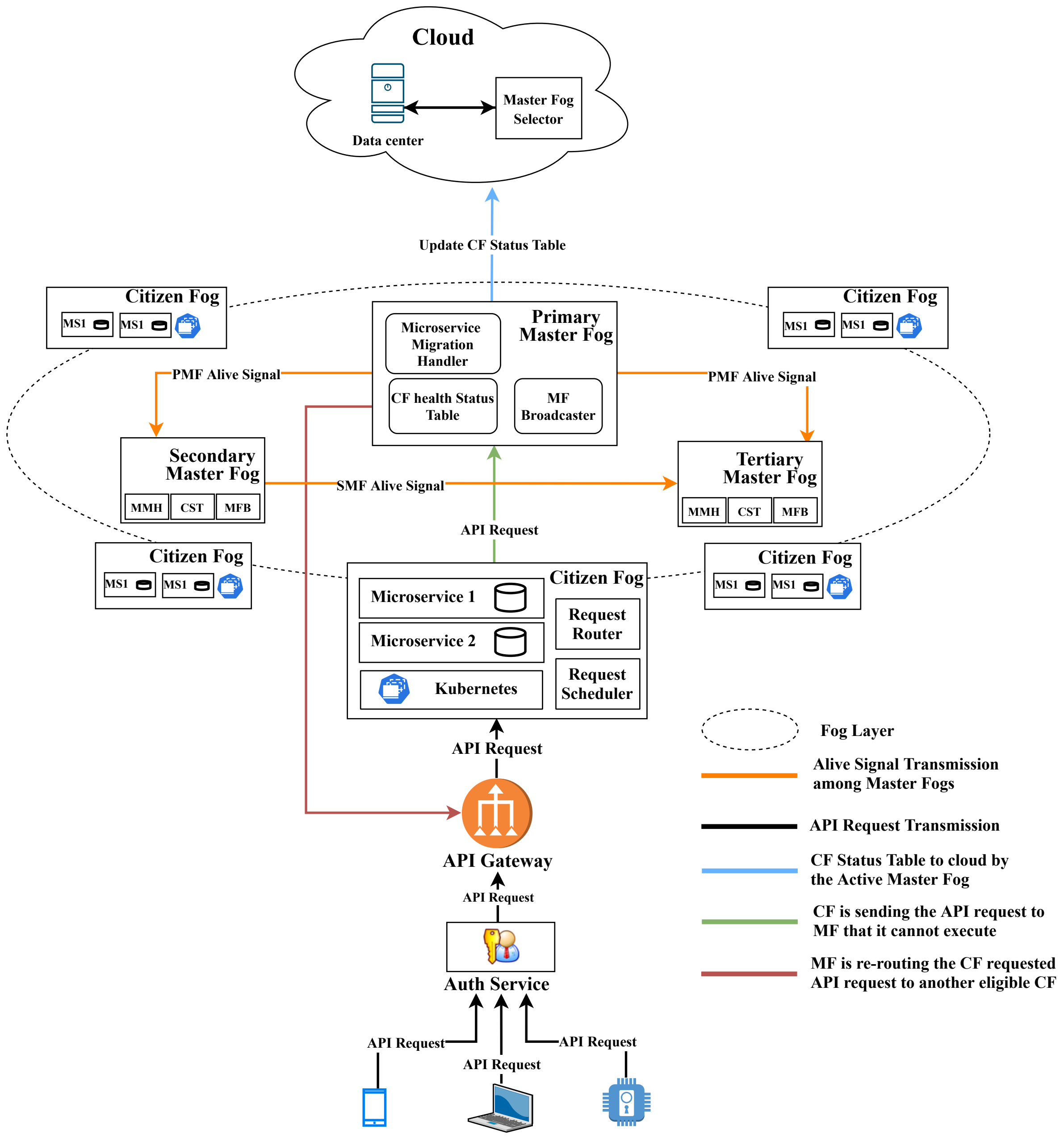


Fig. 1: Overview of our Fault Tolerant Framework

fog. The primary master fog is activated immediately as the master fog and starts to send the alive signal to the other two inactive master fogs. This selection algorithm executes in two scenarios. The framework sets up for the first time when the MST uses the pre-existing fog node resource status. Again, the master fog node sends the citizen fog's health status table to the cloud whenever there is an update in the table. MST uses this data to calculate and select master fogs.

B. *Fault Tolerant Scenarios*

Our framework can handle the MF's unavailability scenarios as follow:

Scenario 1: Everything is working fine. This is the best-case scenario where the PMF is responding correctly.

Scenario 2: Primary Master Fog is unavailable. The primary master fog broadcasts alive signals with a particular time interval to the SMF, the TMF, and the cloud. If the SMF does not receive three consecutive signals from PMF, it declares itself as the primary MF in the network and starts to send its own alive signal. To avoid multiple MF ambiguity, the cloud synchronizes with other MFs. Algorithm 2 states this fault-tolerant process.

Scenario 3: Both the Primary and Secondary MF are unavailable. In this scenario, the TMF gets activated and

TABLE III: List of notations which are being used in Algorithm 1, 2, 3, and 4

Notations in Algorithm	Description
CF	A particular citizen fog
$CFList$	List of citizen fogs
PMF	Primary Master Fog
SMF	Secondary Master Fog
TMF	Tertiary Master Fog
$exceptionList$	List of citizen fogs that are not allowed to be master fog
$CFPointerMap$	List of selected CF mapped to corresponding Total Point
$sortedCFList$	Sorted $CFPointerMap$ according to Total Point in Descending Order
$TRAM$	Total RAM
$ARAM$	Available RAM
$TROM$	Total ROM
$AROM$	Available ROM
PU	Processing Unit
CS	Maximum CPU clock Speed
CU	CPU Usages
IS	Image Size of the migrating services in a particular CF
SPP	Sum of weighted Points of resource and Performance criteria
c	Number of signals needed to identify whether any other MF is alive or not
$isMasterFog$	Defines whether self (PMF/SMF/TMF) is announced as Master fog or not. Default value is False

Algorithm 1: Master Fog Selection Algorithm

```
foreach $CF$ in $CFList$ do
    if $CF$ not in $exceptionList$ then
        $criteriaList$ = $\langle TRAM, ARAM, TROM, AROM, PU, CS, CU \rangle$

        foreach $criteria$ in $criteriaList$ do
            $weightedPointOf_{criteria}$ ← calculate point of the $criteria$ using Equation 1 * $weight_{criteria}$
            $SPP \leftarrow weightedPointOf_{criteria}$

        $weightedPointOfIS$ ← calculate point of the $CF_{IS}$ using Equation 2 * $weight_{IS}$
        $weightedSPP \leftarrow SPP * weight_{SPP}$
        $totalPoint \leftarrow weightedSPP + weightedPointOfIS$

        Add $totalPoint$ to $CFPointerMap[CF]$
$sortedCFList$ ← Sort $CFPointerMap$ In Descending Order
$primaryMasterFog \leftarrow sortedCFList[0]$
$secondaryMasterFog \leftarrow sortedCFList[1]$
$tertiaryMasterFog \leftarrow sortedCFList[2]$
```

Algorithm 2: Master Fog Fault Tolerant Scenario 1, PMF Failed

```
Acting Node: Secondary Master Fog
$iterator \leftarrow 0$
while System is alive do
    Send Self Active Message To Tertiary MF
    if $isMasterFog$ then
        Send CF Status to Cloud
        Send Self Active Message To Cloud
    else
        if $iterator < c$ then
            $PMFResponse$ ← Check if PMF is alive
            if $PMFResponse$ is false then
                $iterator \leftarrow iterator + 1$
            else if $iterator = c$ then
                $isMasterFog$ ← true
                Declare SMF As Master Fog
            else
                $iterator \leftarrow 0$
```

Algorithm 3: Master Fog Fault Tolerant Scenario 2, Both the PMF and SMF Failed

```
Acting Node: Tertiary Master Fog
$iterator \leftarrow 0$
while System is alive do
    if $isMasterFog$ then
        Send CF Status to Cloud
        Send Self Active Mesage To Cloud
    else
        if $iterator < c$ then
            $PMFResponse$ ← Check if Primary MF is alive
            $SMFResponse$ ← Check if Secondary MF is alive
            if $PMFResponse$ and $SMFResponse$ are false then
                $iterator \leftarrow iterator + 1$
            else if $iterator = c$ then
                if $PMFResponse$ and $SMFResponse$ are false then
                    $isMasterFog$ ← true
                    Declare TMF As Master Fog
                else
                    $iterator \leftarrow 0$
            else
                $iterator \leftarrow 0$
```

follows the fault-tolerant Algorithm 3. Similar to the PMF, SMF also broadcasts an alive signal with cloud and the TMF. If the TMF does not receive three consecutive signals from both the PMF and SMF, it asks the cloud for confirmation. If the cloud confirms that the other two MFs are down, the TMF declares itself as MF and starts sending a message.

Scenario 4: All three MFs are unavailable. The cloud follows Algorithm 4 and handles this disaster scenario. The cloud subscribes to each MF's alive signal, and if it does not receive any signal from any of the MFs for three consecutive times, it pings all three MF to be sure that the MFs are down. If no MF responds back, the cloud starts the MF selection procedure from the rest of the CFs.

```
Algorithm 4: Master Fog Fault Tolerant Scenario 3, All the MFs Failed
1  Acting Node: Cloud application
2  $PMFResponse \leftarrow$ Check if Primary MF is alive
3  $SMFResponse \leftarrow$ Check if Secondary MF is alive
4  $TMFResponse \leftarrow$ Check if Tertiary MF is alive
5  if $PMFResponse$, and $SMFResponse$ are false then
6      if $TMFResponsen$ is $false$ then
7          Add $PMF$, $SMF$, and $TMF$ to $exceptionList$
8          Start Master Fog selection process using $Algorithm$ 1
9      else
10         Approve $TMF$ as Master Fog
```

C. Model

In this section, we present a mathematical model to represent the master fog selection procedure. Here, we consider that all master fog nodes can handle all resources and reside within the same local area network.

TABLE IV: List of the Model Notations

Symbol	Description
f	Individual fog node
$\mathcal{F}$	Set of fog nodes where $f \in \mathcal{F}$
$\mathcal{H}_f$	Set of MF selection parameters of a fog node f
$\mathcal{H}_f^p, \mathcal{H}_f^n$	Set of positive and negative MF selection parameters respectively where $\mathcal{H}_f^p \subset \mathcal{H}_f, \mathcal{H}_f^n \subset \mathcal{H}_f$
$\mathcal{P}(\chi)$	Positive point factor of a resource parameter where $\chi \in \mathcal{H}_f^p$
$\mathcal{Q}(\chi)$	Negative point factor of a resource parameter where $\chi \in \mathcal{H}_f^n$
$\omega(\chi)$	Weight of a resource parameter where $\chi \in \mathcal{H}_f$
$\mathcal{W}_f$	Total points of f
M_p	Total memory capacity (in Bytes) of random access memory (RAM)
M_r	Total memory capacity (in Bytes) of read only memory (ROM)
P	Processing unit
τ	Maximum CPU clock speed
ψ	Current CPU usage
I	Respective image size of all microservices in $f \in \mathcal{F}$

a) Environment and Notations: In our model, there is a set of fog nodes $\mathcal{F}$, where each fog node $f \in \mathcal{F}$. The master fog selection procedure depends on different parameters, and we denote the set of parameters as $\mathcal{H}_f$, where $\mathcal{H}_f^p$ and $\mathcal{H}_f^n$ represents the set of positive and negative parameters relatively ($\mathcal{H}_f^p \subset \mathcal{H}_f, \mathcal{H}_f^n \subset \mathcal{H}_f$). The rest of the notations are presented in Table IV.

b) Model: Here, we denote a fog node as

$$f = \langle M_p, M_r, P \rangle$$

where M_p is RAM, M_r is ROM, and P is the number of processing core.

Now, before calculating the eligibility of a master node, we identify the set of parameters to consider. Here, we denote the set of considered resource parameters as

$$\mathcal{H}_f = \langle M_p, M_p^a, M_r, M_r^a, P, \tau, \psi, I \rangle$$

where, M_p^a is the relative available RAM, M_r^a is the relative available ROM, τ is the relative maximum CPU clockspeed, and ψ is the relative CPU usage of the fog nodes $f \in \mathcal{F}$. Here, I is the relative image size of the microservices in a fog node. Here, relative measurement can vary based on different environment configurations (Homogeneous or Heterogeneous). For example, an administrator can sort nodes based on the parameters and calculate relative parameter values for other nodes based on the maximum or median parameter value (e.g., available memory size in bytes).

Now, not all high parameter values indicate high influences while calculating the weighted point factors. That means a few parameters have negative influences. Now, as $\mathcal{H}_f$ contains both positive and negative influencing resource parameters, we divide the positive and negative parameters as separate sets as

$$\mathcal{H}_f^p = \langle M_p, M_p^a, M_r, M_r^a, P, \tau \rangle$$
$$\mathcal{H}_f^n = \langle \psi, I \rangle$$

where, $\mathcal{H}_f^p$ is the set of positive influencing resource parameters, and $\mathcal{H}_f^n$ is the set of negative influencing resource parameters.

Now, we want to calculate the point factor $\mathcal{P}(\chi)$ for each resource parameter $\chi \in \mathcal{H}_f^p$ as

$$\mathcal{P}(\chi) = \begin{cases} 100 & \text{if } \chi = \chi_{\max} \\ (\chi * 100)/\chi_{\max} & \text{if } \chi < \chi_{\max} \end{cases} \quad (1)$$

where $\chi_{\max} = \max\left(\bigcup_{f \in \mathcal{F}} \chi\right)$ and $\chi \in \mathcal{H}_f^p$

Similarly, we want to calculate the point factor $\mathcal{P}(\chi)$ for each resource parameter $\chi \in \mathcal{H}_f^n$ as

$$\mathcal{Q}(\chi) = \begin{cases} 100 & \text{if } \chi = \chi_{\min} \\ (\chi_{\min} * 100)/\chi & \text{if } \chi > \chi_{\min} \end{cases} \quad (2)$$

where $\chi_{\min} = \min\left(\bigcup_{f \in \mathcal{F}} \chi\right)$ and $\chi \in \mathcal{H}_f^n$

M_p, M_p^a, M_r, M_r^a, P, τ, and ψ are the resource attributes of f. The weighted sum of the points of these resource attributes is $\mathcal{T}_f$.

$$\mathcal{T}_f = \sum_{\chi \in H_f^p} [\mathcal{P}(\chi) \cdot \omega(\chi)] + \sum_{\chi \in H_f^n} [\mathcal{Q}(\chi) \cdot \omega(\chi)] + \mathcal{P}(\psi) \cdot \omega(\psi)$$

However, even if a fog node f has higher resources, it can have microservices that migration can be overhead. Therefore we provide weights to $\mathcal{T}_f$ and image size I. Now, the weighted point factor $\mathcal{W}_f$ for each fog node $f \in \mathcal{F}$ is

$$\mathcal{W}_f = [\mathcal{T}_f \cdot \omega(\mathcal{T})] + [\mathcal{P}(I) \cdot \omega(I)]$$

Now, to choose the potential master fog, our model elects the fog node with the maximum weighted point factor, i.e., $\max \mathcal{W}_f$.

D. Framework Implementation

We implemented our framework using the Internet of Things (IoT) devices and Amazon Web Services (AWS). The IoT devices carry out responsibilities as master and citizen fogs. The fog devices are interconnected using a singular network and use the MQ Telemetry Transport (MQTT) protocol for communication between the master fog and citizen fogs. The RESTful API performs the communication between the fog nodes and cloud applications over HTTPS protocol. We used the following models of Raspberry Pi devices as fog nodes: Raspberry Pi 400, Raspberry Pi 4 B 8GB, Raspberry Pi 4 B, and Raspberry Pi 3 B+. Every inbound request was filtered and authorized by the Auth Service and the API gateway. The microservices were deployed in the citizen fogs while the master fog maintained the citizen fogs health status in real-time. The candidate master fogs' check if the primary master fog is active regularly. The fog nodes are a part of the Kubernetes cluster to identify the health statuses individually and collectively.

V. Results and Discussion

We evaluated the implemented framework for a set of fog devices with different hardware configurations which are connected to a single network. In our scenario, the network is immobile. The citizen fog health status reports were synced with the cloud application for ensuring fault tolerance.

We answer the first question by looking into the related Work, fog-IoT framework, and system design. We provide the master fog selection process, fault tolerance scenarios, and several detailed algorithms (Section III). We also consider and present the design considerations, architecture, and implemented algorithms in the same section.

We answer the second question by discussing the implementation overview and corresponding results based on practical scenarios. Here, we discuss the fault-tolerant framework development based on the algorithms. Then we present the graphs of different scenarios with timing graphs of different system resources such as CPU, and memory, and network utilization before or after the master fog node failure. From these graphs, we show that the fault-tolerant achievement or system can still perform at almost the same level of performance or efficiency.

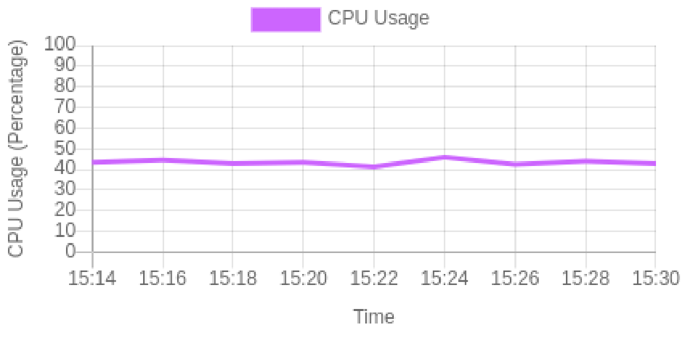
RO1: To select contingent master fog nodes based on a periodic computational resource capacity estimation strategy in Fog-IoT ecosystems.

We ensure storing the snapshots of all fog devices' health statuses in each device synchronized with the master fog and the cloud application at a regular interval. Our framework analyzed each fog node's score whenever we needed to select a primary master fog. The score was calculated from the last thirty snapshots of data from each fog node. The scoring weight was collected from the configuration file at run time, and it could be modified based on the system requirements. The list of criteria is shown in Table II. Each fog node's score was calculated based on these factors and the configuration file's scoring weights. We selected the master fog and candidate master fog nodes from the top ten devices. This selection score was synchronized with the cloud application if the master fog, the secondary master fog, and the tertiary master fog become unavailable.

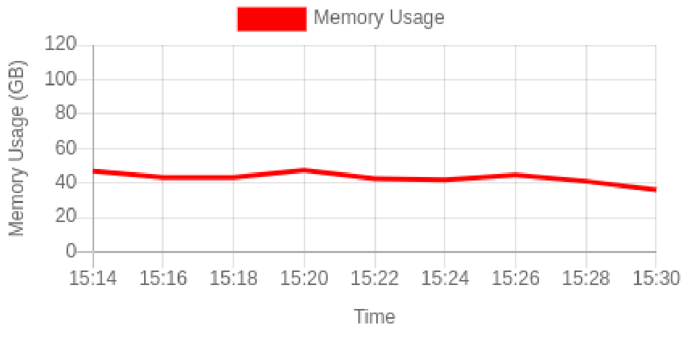
RO2: To prepare contingent master fog nodes for efficient enactment of master fog reallocation when a failure occurs

We assessed our framework's fault tolerance for each scenario described in Section IV-B. Fig. 2 illustrates CPU usage and memory usage of the Kubernetes edge cluster. As we consider the fault tolerance of any master fog node occurrences, we only consider the edge cluster. The x-axis shows the time period in MM:SS format and the y-axis shows the aggregated CPU usages percentage of our Kubernetes edge cluster for each scenario in Fig. 2a, 2c, 2e, and 2g. Our Kubernetes cluster had a total of 120 GB of random access memory. The memory usage graphs in Fig. 2b, 2d, 2f, and 2h represents the aggregated memory usage at a certain time for the listed scenarios. The data of these CPU usage and memory usage graphs were collected from the Kubernetes cluster dashboard.

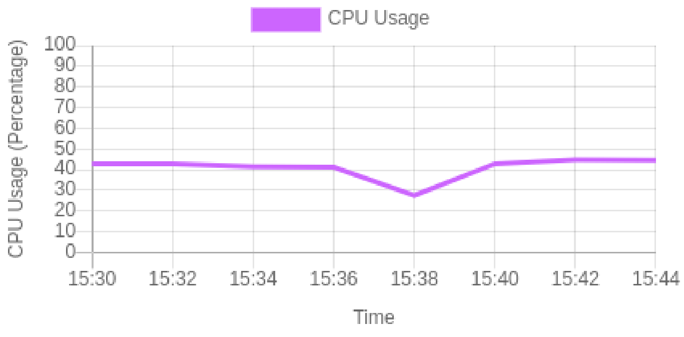
Fig. 2a and 2b show the best-case scenario of the framework when the master fog is active between the time frame of 15:14 to 15:30. There is no spike in CPU usage or memory usage within this time frame. Fig. 2c and 2d show the second scenario between the time frame of 15:30 to 15:44. The primary master fog becomes unavailable at 15:36, which causes a sharp fall in CPU usage and memory usage. The CPU usage turns to be similar to before the incident at 15:38 as the secondary master fog claims the primary master fog's responsibilities. However, the memory usage remains downward because of the loss of a fog node. Fig. 2e and 2f show the third scenario between the time frame of 15:44 to 15:58. The primary master fog becomes unavailable at 15:46, and the secondary master fog starts taking the master fog's responsibilities at 15:48. The secondary master fog then becomes unavailable at 15:50, causing the tertiary master fog to take responsibilities as the master fog node. The memory usage graph of this scenario is declining as the master fog nodes become available. In Fig. 2g we see a steady fall in CPU usage as all the three master fog nodes fail from 17:16, and eventually, as soon as the cloud handles the scenario, the CPU usage increase after 17:22. Fig. 2h indicates the decrease of the memory usage as the three master fog nodes sequentially become unavailable. Overall, the system remains



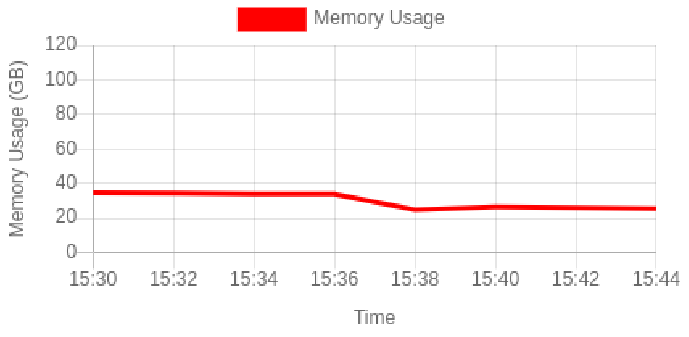
(a) Scenario 1: CPU Usage



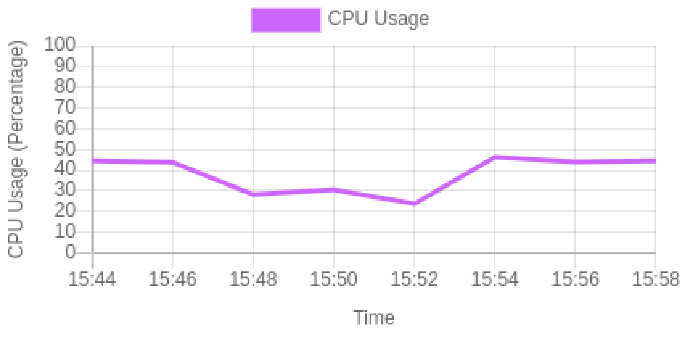
(b) Scenario 1: Memory Usage



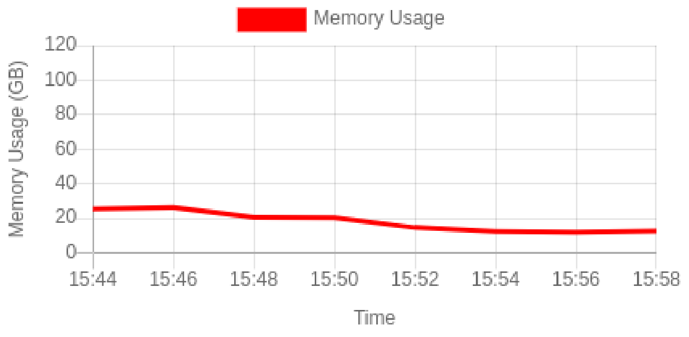
(c) Scenario 2: CPU Usage



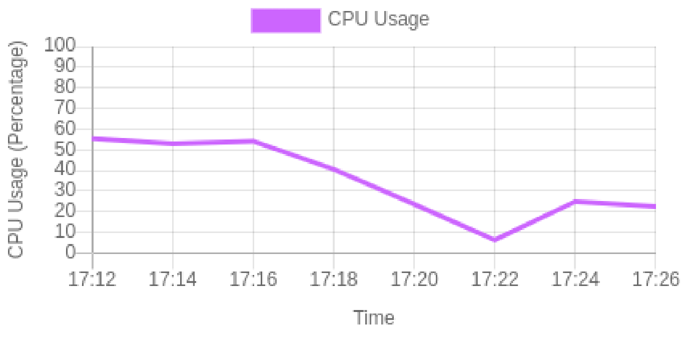
(d) Scenario 2: Memory Usage



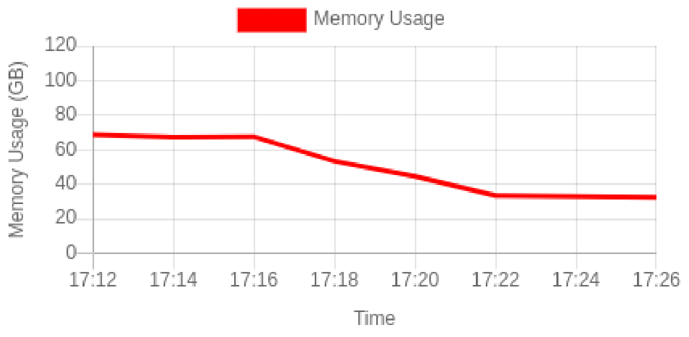
(e) Scenario 3: CPU Usage



(f) Scenario 3: Memory Usage



(g) Scenario 4: CPU Usage



(h) Scenario 4: Memory Usage

Fig. 2: Fault Tolerance Assessment for different scenarios

alive in all four scenarios and effectively handles the single point of failure.

VI. Conclusion

This research provides a fault-tolerant framework considering the initial concept of having a Master Fog (MF) node and its impact on Cloud Fog IoT eco systems for microservices execution. We established that a Master Fog node could play a vital part in this architecture for efficient service provisioning, resource management, task scheduling, and microservice execution with cloud and fog computing collaboration from the literature review. Then we discussed how the Master Fog plays the roles and how they usually behave in this network. Then, we provided an MF selection process and the Fault-tolerant system, which due to an unavailable and Master Fog node or a system down, can be resilient and provide regular services smoothly even if a fault took place. Finally, we provide an MF selection process, several fault-tolerant scenarios and demonstrated the system's availability and seamless microservices execution in this framework, even in the event of a system failure.

Acknowledgment

This work is supported through the Australian Research Council Discovery Project: DP190100314, "Re-Engineering Enterprise Systems for Microservices in the Cloud".

References

[1] Md Whaiduzzaman, Anjum Naveed, and Abdullah Gani. Mobicore: Mobile device based cloudlet resource enhancement for optimal task response. *IEEE transactions on services computing*, 11(1):144–154, 2016.

[2] Md Whaiduzzaman, Shelia Rahman Tuly, Nadia Haque, Md Razon Hossain, Alistair Barros, et al. Credit based task scheduling process management in fog computing. In *PACIS*, page 232, 2020.

[3] Amir M Rahmani, Tuan Nguyen Gia, Behailu Negash, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, and Pasi Liljeberg. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach. *Future Generation Computer Systems*, 78:641–658, 2018.

[4] Ahmedur Rahman Shovon, Shanto Roy, Tanusree Sharma, and Md Whaiduzzaman. A restful e-governance application framework for people identity verification in cloud. In *International Conference on Cloud Computing*, pages 281–294. Springer, 2018.

[5] Nishat Farjana, Shanto Roy, Md Julkar Nayeen Mahi, and Md Whaiduzzaman. An identity-based encryption scheme for data security in fog computing. In *Proceedings of International Joint Conference on Computational Intelligence*, pages 215–226. Springer, 2020.

[6] Md Whaiduzzaman, Khondokar Oliullah, Md Julkar Nayeen Mahi, and Alistair Barros. Auasf: An anonymous users authentication scheme for fog-iot environment. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2020.

[7] Md Whaiduzzaman, Abdullah Gani, and Anjum Naveed. Towards enhancing resource scarce cloudlet performance in mobile cloud computing. *Computer Science & Information Technology*, page 1, 2015.

[8] Md. Razon Hossain, Md. Whaiduzzaman, Alistair Barros, Shelia Rahman Tuly, Md. Julkar Nayeen Mahi, Shanto Roy, Colin Fidge, and Rajkumar Buyya. A scheduling-based dynamic fog computing framework for augmenting resource utilization. *Simulation Modelling Practice and Theory*, page 102336, 2021.

[9] Shreshth Tuli, Redowan Mahmud, Shikhar Tuli, and Rajkumar Buyya. Fogbus: A blockchain-based lightweight framework for edge and fog computing. *Journal of Systems and Software*, 154:22–36, 2019.

[10] Mahyar Tourchi Moghaddam and Henry Muccini. Fault-tolerant iot. In *International Workshop on Software Engineering for Resilient Systems*, pages 67–84. Springer, 2019.

[11] Asad Javed, Keijo Heljanko, Andrea Buda, and Kary Främling. Cefiot: A fault-tolerant iot architecture for edge and cloud. In *2018 IEEE 4th world forum on internet of things (WF-IoT)*, pages 813–818. IEEE, 2018.

[12] Umar Ozeer, Xavier Etchevers, Loïc Letondeur, François-Gaël Ottogalli, Gwen Salaün, and Jean-Marc Vincent. Resilience of stateful iot applications in a dynamic fog environment. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 332–341, 2018.

[13] Kun Wang, Yun Shao, Lei Xie, Jie Wu, and Song Guo. Adaptive and fault-tolerant data processing in healthcare iot based on fog computing. *IEEE Transactions on Network Science and Engineering*, 2018.

[14] Alexander Power and Gerald Kotonya. A microservices architecture for reactive and proactive fault tolerance in iot systems. In *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 588–599. IEEE, 2018.

[15] Shu-Ching Wang, Shih-Chi Tseng, Kuo-Qin Yan, and Yao-Te Tsai. Reaching agreement in an integrated fog cloud iot. *IEEE Access*, 6:64515–64524, 2018.

[16] Jitendcr Grover and Rama Murthy Garimella. Reliable and fault-tolerant iot-edge architecture. In *2018 IEEE SENSORS*, pages 1–4. IEEE, 2018.

[17] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Towards fault tolerant fog computing for iot-based smart city applications. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0752–0757. IEEE, 2019.

[18] Ryuji Oma, Shigenari Nakamura, Dilawaer Duolikun, Tomoya Enokido, and Makoto Takizawa. A fault-tolerant tree-based fog computing model. *International Journal of Web and Grid Services*, 15(3):219–239, 2019.

[19] Asad Javed, Jérémy Robert, Keijo Heljanko, and Kary Främling. Iotef: A federated edge-cloud architecture for fault-tolerant iot applications. *Journal of Grid Computing*, pages 1–24, 2020.