A Resilient Fog-IoT Framework for Seamless Microservice Execution

Presented by
Dr. Md Whaiduzzaman

Table of Contents

Introduction

- Fog computing brings enterprise systems closer to the cloud [1, 2, 3]
- Layered architecture of the IoT-Fog-Cloud ecosystem performs efficient application execution [4, 5]
- Master-worker framework is introduced to accelerate the efficiency of the system [6]

Introduction (cont.)

- Recent trends include growing interest in using microservice architecture to address Fault Tolerance in IoT ecosystem [7]
- A fault tolerance mechanism to protect the essential master fog node is needed to avoid a single point of failure for microservice based architecture

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

Our Contributions

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

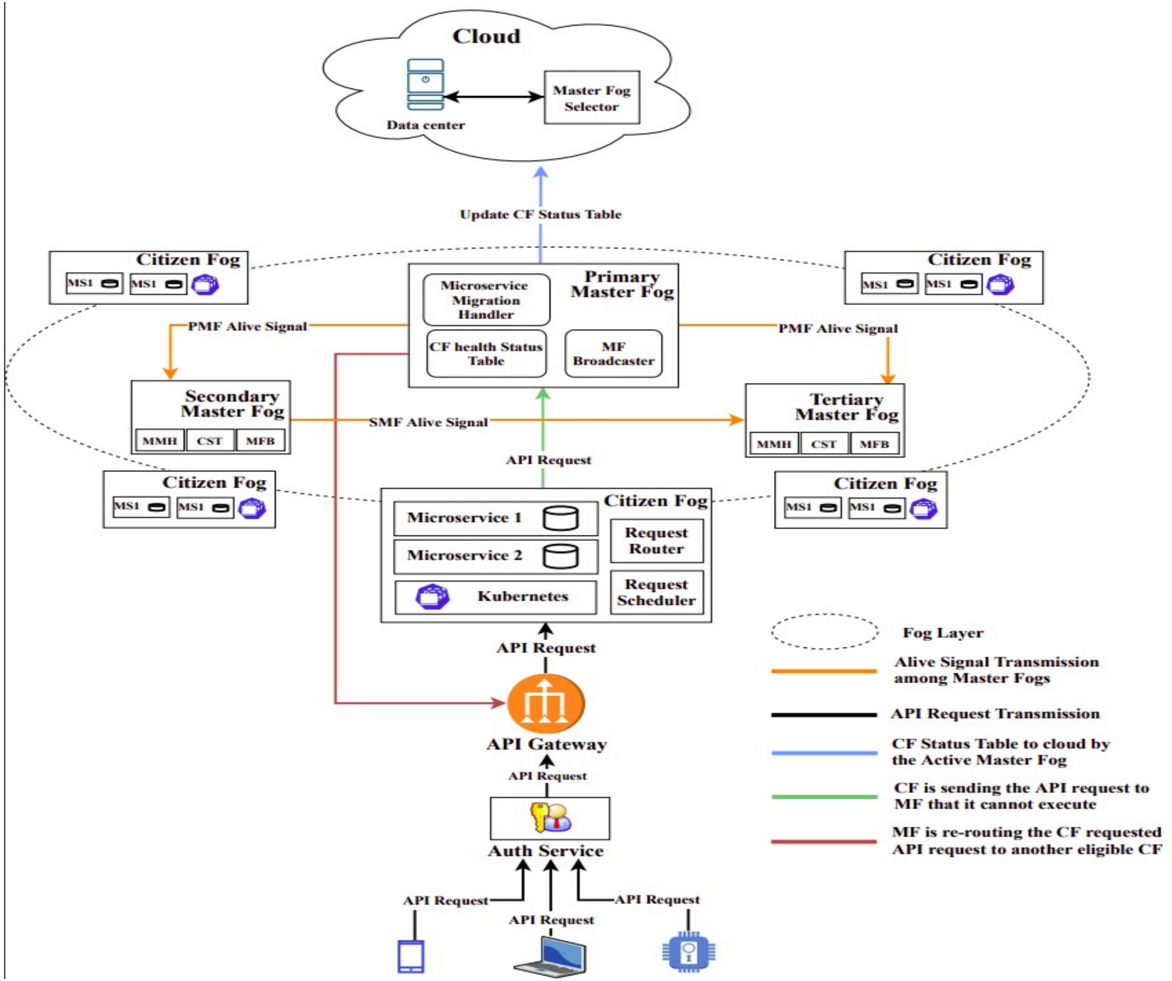


Figure 1: Overview of our Fault Tolerant Framework

Background

- A fault-tolerant architecture for IoT applications that uses master-citizen orchestration was proposed [6]
- Reduced Variable Neighborhood Search (RVNS)-based framework was introduced for fault-tolerant data transmission [8]
- A framework based on microservice architecture providing reactive and proactive fault tolerant support was developed [9]

Background (Cont.)

- An architecture distributed over four levels (cloud-fog-mist-dew) was introduced based on IoT device’s processing power and distance [10]
- Issues of reliability and fault tolerance in IoT based smart cities were investigated [11]
- IoT-Edge-Cloud federation architecture for multi-cluster IoT applications was developed by adapting Cloud-Edge-IoT fault tolerant layered design [12]

Background (Cont.)

- Fog-IoT-Cloud framework ensure faster processing and efficient resource management
- There is a lack of a combined efficient Master Fog node three-layer system and a resilient fault tolerance system
- Efficient resource utilization of three-layer architecture with fault tolerant Master Fog node IoT ecosystem is needed for seamless microservice execution.

Framework Design

- Our proposed framework has three layers:
  - IoT and end devices
  - Fog nodes
  - Cloud

Framework Design (Cont.)

- First layer - IoT and end devices:
  - Consists of IoT and end devices
  - Each request goes through the Auth service and API gateway
  - Auth service authenticates the requests
  - API gateway routes the request to the corresponding citizen fog

Framework Design (Cont.)

- Second layer – fog nodes:
  - Consists of two types of fog nodes – Master Fog and Citizen Fog
  - Cloud selects these master fogs from citizen fogs using master selection algorithm

Framework Design (Cont.)

- Second layer – Master fog node:
  - Master fog nodes has three categories: Primary Master Fog, Secondary Master Fog, Tertiary Master Fog
  - Each master fog node has three components: Microservice Migration Handler, Master Fog Broadcaster, and Citizen Fog Health Status Table
  - Only one master fog is active at a time

Framework Design (Cont.)

- Second layer – Master fog node responsibilities:
    - Citizen fog orchestration
    - Request handling from citizen fogs and scheduling
    - Automatic application deployment
    - Citizen fog failure handling
    - Citizen fog health status management
    - Communication with the cloud and other master fogs to share state information

Framework Design (Cont.)

- Second layer – Citizen fog nodes:
    - Citizen fogs are the general-purpose fog nodes
    - Microservices run in citizen fogs
    - Citizen fog has two components: Request Scheduler and Request Router

Framework Design (Cont.)

- Third layer – Cloud:
  - The Cloud application is responsible for selecting the master fogs in two scenarios: setting up the framework initially, when all three types of master fogs are unresponsive
  - Cloud receives the alive signals from all the master fogs at a regular interval

System Design and Modeling

- Master Fog Selection Process:
  - Master fog is selected from eligible citizen fogs
  - To be eligible as a master fog, the citizen fog must migrate its existing microservices
  - The weight of master fog selection criteria changes dynamically based on system status and fog node priority

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

Table 1: Master fog selection criteria

System Design and Modeling (Cont.)

- Master Fog Selection Process - Exception Criteria:
    - Fog nodes dedicated to perform critical tasks
    - Fog nodes with authorization service
    - For nodes performing any prioritized service

System Design and Modeling (Cont.)

- Master Fog Selector:
  - The cloud component, Master Fog Selector executes Master Fog Selection algorithm
  - Calculates weighted points for citizen fogs
  - Declares the citizen fog with the by descending scores as the primary master fog, secondary master fog, and tertiary master fog
  - Stores citizen fogs health status information

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

Table 2: List of notations used in algorithms

```
foreach CF in CFList do
    if CF not in exceptionList then
        criteriaList = ⟨TRAM, ARAM, TROM,
          AROM, PU, CS, CU⟩

        foreach criteria in criteriaList do
            weightedPointOf_criteria ← calculate
              point of the criteria using Equation 1 *
              weight_criteria
            SPP ← weightedPointOf_criteria

        weightedPointOfIS ← calculate point of the
          CF_IS using Equation 2 * weight_IS
        weightedSPP ← SPP * weight_SPP
        totalPoint ← weightedSPP +
          weightedPointOfIS

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

Algorithm 1: Master Fog Selection Algorithm

System Design and Modeling (Cont.)

- Master Fog’s Fault Tolerant Scenarios:
  - Scenario 1: Everything is working fine
  - Scenario 2: Primary master fog is unavailable
  - Scenario 3: Both the primary and secondary master fog are unavailable
  - Scenario 4: All three master fogs are unavailable

System Design and Modeling (Cont.)

- MF Fault Tolerant Scenario 1 - Everything is working fine:
  - The best-case scenario
  - The primary master fog is responding correctly

System Design and Modeling (Cont.)

- MF Fault Tolerant Scenario 2 - Primary master fog is unavailable:
    - Primary master fog broadcasts alive signals to all master fogs and cloud
    - If the secondary master fog does not receive three consecutive signals from primary master fog, it declares itself as the primary master fog
    - To avoid multiple master fog ambiguity, the cloud synchronizes with all master fogs

```
Acting Node: Secondary Master Fog
iterator ← 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 ← iterator + 1
            else if iterator = c then
                isMasterFog ← true
                Declare SMF As Master Fog
            else
                iterator ← 0
```

Algorithm 2: Master Fog Fault Tolerant Sc 2 - PMF Failed

System Design and Modeling (Cont.)

- MF Fault Tolerant Scenario 3 - Both the primary and secondary master fog are unavailable:
  - Tertiary master fog gets activated
  - Synchronizes with the cloud master fog selector

```
Acting Node: Secondary Master Fog
iterator ← 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 ← iterator + 1
            else if iterator = c then
                isMasterFog ← true
                Declare SMF As Master Fog
            else
                iterator ← 0
```

Algorithm 3: Master Fog Fault Tolerant Sc 3-PMF, SMF Failed

System Design and Modeling (Cont.)

- MF Fault Tolerant Scenario 4 - All three master fogs are unavailable:
  - The cloud handles this disaster scenario
  - If it does not receive any signal from any of the master fogs for three consecutive times, it pings them
  - If no master fog responds back, the cloud starts the master selection procedure from the rest of the citizen fogs

Acting Node: Cloud application
$PMFResponse \leftarrow$ Check if Primary MF is alive
$SMFResponse \leftarrow$ Check if Secondary MF is alive
$TMFResponse \leftarrow$ Check if Tertiary MF is alive
if $PMFResponse$, *and* $SMFResponse$ *are false* **then**
    if $TMFResponsen\ is\ false$ **then**
        Add PMF, SMF, and TMF to $exceptionList$
        Start Master Fog selection process using $Algorithm$ 1
    else
        Approve TMF as Master Fog

Algorithm 4: Master Fog Fault Tolerant Sc 4-All MF Failed

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 $f \in \mathcal{F}$

Table 3: List of model notations

System Design and Modeling (Cont.)

- Model:
  - Set if resource parameters:

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

  - Set of positive and negative parameters:

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

$$\mathcal{H}_f^n = \langle \psi, I \rangle$$

System Design and Modeling (Cont.)

- Model (cont.):
  - Positive point factor for each parameter:

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

  - Positive point factor for each parameter:

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

System Design and Modeling (Cont.)

- Model (cont.):
  - Weighted sum of the points:

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

  - Weighted point factor for each node:

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

Framework Implementation

- Raspberry Pi 400, Raspberry Pi 4 B 8GB, Raspberry Pi 4 B, and Raspberry Pi 3 B+ were used as master and citizen fogs
- The cloud applications were deployed in Amazon Web Services
- Fog devices were interconnected using a singular network and use MQTT protocol
- RESTful API were used to connect the fog nodes with cloud applications over HTTPS protocol

Framework Implementation (Cont.)

- 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

Results

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

Results (Cont.)

- RO1 - To select contingent master fog nodes based on a periodic computational resource capacity estimation strategy in Fog-IoT ecosystems:
  - Stores the snapshots of all fog devices' health statuses
  - Health statuses are synchronized with the master fog and the cloud application at a regular interval

Results (Cont.)

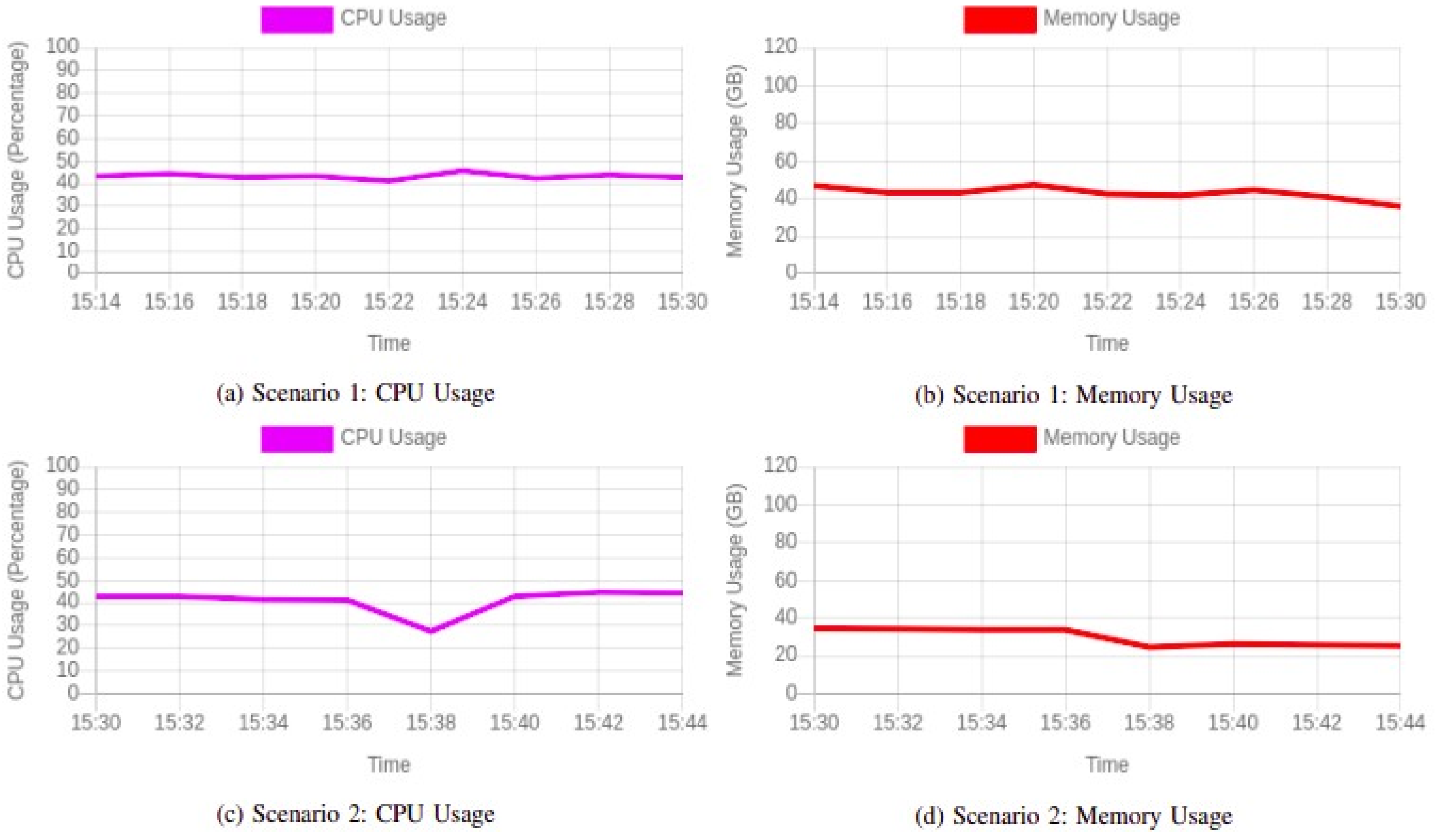
- RO1 (Cont.):
  - The framework analyzes each fog node's score whenever at master fog selection process
  - The score was calculated from the last thirty snapshots of data from each fog node
  - The scoring weight was collected from the configuration file at run time, and it could be modified based on the system requirements

Results (Cont.)

- RO2 - To prepare contingent master fog nodes for efficient enactment of master fog reallocation when a failure occurs:
  - The framework's fault tolerance are assessed for all four scenarios
  - The fault tolerance of any master fog node occurrences is only considered

Results (Cont.)

- RO2 (Cont.):
    - Time vs aggregated CPU usages percentages and time vs aggregated memory usages of the Kubernetes edge cluster are used as factors of fault tolerance evaluation



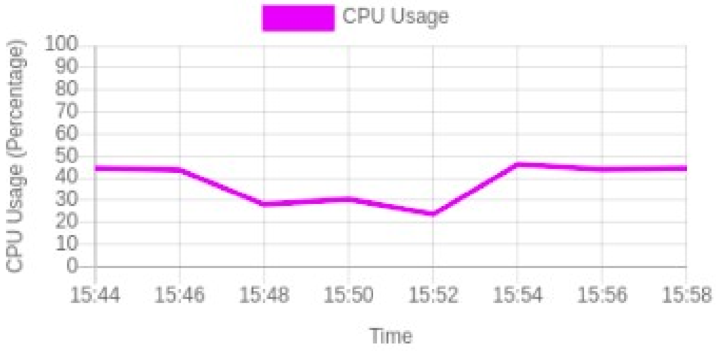
(a) Scenario 1: CPU Usage

(b) Scenario 1: Memory Usage

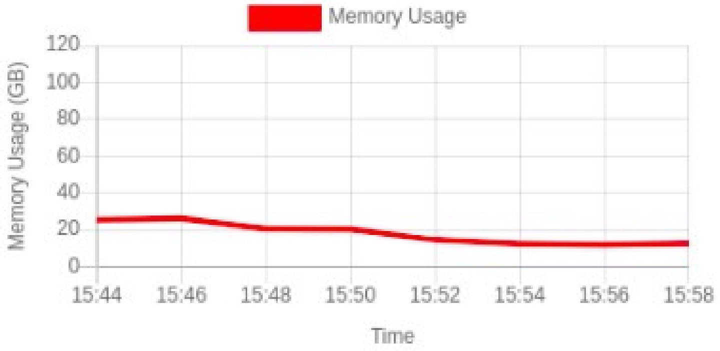
(c) Scenario 2: CPU Usage

(d) Scenario 2: Memory Usage

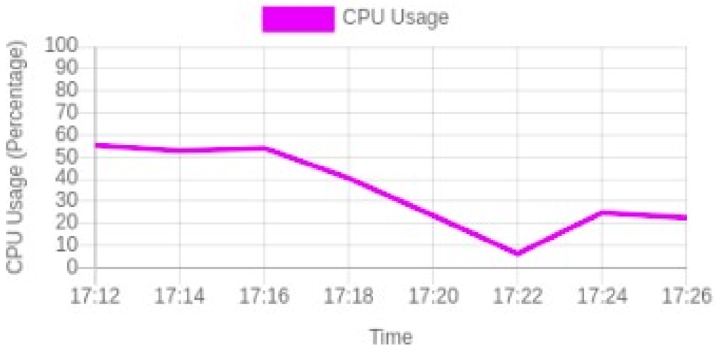
Figure 2: Fault tolerance assessment for scenario 1 and 2



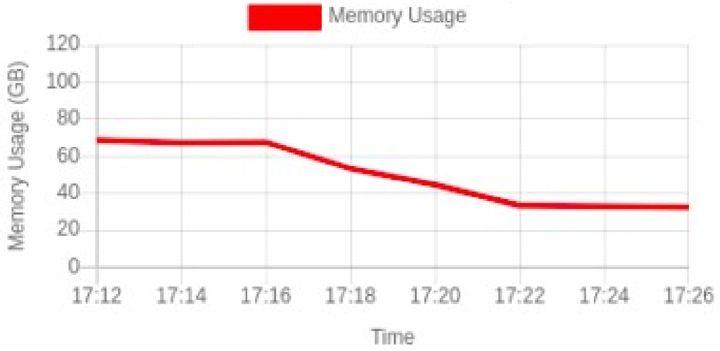
(e) Scenario 3: CPU Usage



(f) Scenario 3: Memory Usage



(g) Scenario 4: CPU Usage



(h) Scenario 4: Memory Usage

Figure 3: Fault tolerance assessment for scenario 3 and 4

Conclusion

- This research provides a fault-tolerant framework considering the initial concept of having a Master Fog node and its impact on Cloud Fog IoT eco systems for microservices execution
- Introduces a master fog selection process which is evaluated against 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

References

[1] 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.

[2] 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.

[3] 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.

References (Cont)

[4] 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.

[5] 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.

[6] Asad Javed, Keijo Heljanko, Andrea Buda, and Kary Framling. 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.

[7] Mahyar Tourchi Moghaddam and Henry Muccini. Faulttolerant iot. In International Workshop on Software Engineering for Resilient Systems, pages 67 84. Springer, 2019.

References (Cont)

[8] 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.

[9] 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.

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

References (Cont)

[11] 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.

[12] Asad Javed, Jer´ emy Robert, Keijo Heljanko, and Kary ´ Framling. Iotef: A federated edge-cloud architecture for ¨ fault-tolerant iot applications. Journal of Grid Computing, pages 1–24, 2020.

THANK YOU