International Journal of Data Informatics and Intelligent Computing (IJDIIC)
Vol. 4, No. 4, December 2025, pp. 52~69
DOI: 10.59461/ijdiic.v4i4.241

Optimized Fault-Tolerant Placement of Microservices in Distributed Fog Networks

Shally Gupta[1], Nanhay Singh[2]
[1]Department of Information Technology, Delhi Technological University, India
[2]Department of Computer Science & Engineering, NSUT East Campus, India

Article Info

Article history:

Received November 02, 2025
Revised December 15, 2025
Accepted December 21, 2025

Keywords:

Microservices
Fog Computing
Fault Tolerance
Load Balancing
System Reliability

ABSTRACT

We propose a fault-tolerance aware placement algorithm for microservices in distributed fog environments that combines a reactive Dynamic Connections Load Balancer (DCLB) with a proactive, fault-threshold driven migration and backup-predetermination mechanism. The DCLB places services only on healthy nodes by using active-connection-based load metrics, while the proactive module preassigns backup nodes for critical services and triggers rapid migration when a node crosses a fault threshold. We evaluate the approach using extensive iFogSim simulations on a smart-home management microservice composition under controlled failure injections. This paper presents a fault–tolerance–aware microservice placement strategy that integrates health-aware primary placement with proactive backup node predetermination. Experimental results show that the strategy effectively limits service disruption, achieving an average service downtime of less than 30% of the total simulation time across failure scenarios. System reliability remains consistently high, with values exceeding 0.85 in single-failure cases and remaining above 0.75 under multiple failures. Resource utilization across fog nodes remains balanced, with moderate memory usage and stable energy consumption. These results demonstrate that proactive, health-aware placement improves service continuity and resilience in fog-based microservice deployments.

This is an open access article under the CC BY-SA license.

Corresponding Author: Shally Gupta (e-mail: shally.gupta@dtu.ac.in)

1. INTRODUCTION

Fog computing has emerged as a crucial paradigm that extends cloud capabilities to the network edge, enabling decentralized computation, reduced latency, and real-time responsiveness for Internet of Things (IoT) applications. By integrating microservice-based architectures where applications are decomposed into small, independent, and loosely coupled services, fog systems can achieve high scalability, modularity, and adaptability [1].

However, ensuring the reliability and availability of microservices in dynamic fog environments remains a key challenge [2]. The unpredictable behavior of fog nodes, including frequent failures, resource variability, and mobility, can lead to temporary service disruptions and degraded performance. Maintaining seamless microservice availability in the face of such failures is essential for supporting mission-critical IoT applications [3].

Several studies have explored the role of fog computing in enabling efficient, decentralized processing and real-time analytics in IoT and smart city applications [4][5]. Research efforts have also focused on fault tolerance in distributed systems, employing both proactive and reactive mechanisms to ensure service continuity. Proactive approaches, such as backup node predetermination, reduce recovery time but add resource overhead, while reactive techniques dynamically reallocate workloads after failures but may cause short-term downtime. Achieving an optimal balance between these strategies is vital for resilient and resource-efficient service management.

Recent works have addressed fault-tolerant architectures in fog and edge computing. Tuli et al. [6] proposed Dragon, a decentralized recovery mechanism for edge federations, while [7] analyzed proactive and reactive recovery schemes for fog-to-cloud (F2C) systems. Ray et al. [8] introduced a proactive reliability enhancement model for cloud federations. Despite these efforts, existing solutions often overlook node health and fail to integrate dynamic load balancing with fault-aware redundancy, resulting in inefficient resource utilization and higher service downtime.

This research addresses these limitations by proposing a Fault-Tolerance Aware Placement Algorithm (FTAPA) for microservices in fog computing. The proposed approach combines reactive and proactive strategies:

Reactive layer: employs a Dynamic Connections Load Balancer (DCLB) to assign microservices to healthy nodes based on active connections and resource availability.

Proactive layer: predetermines backup nodes using fault-tolerance thresholds to enable rapid failover and minimize downtime.

The major contributions of this work are as follows:

- Development of a connection-aware load balancing mechanism that enhances reliability and optimizes resource utilization by prioritizing healthy nodes.
- Introduction of a fault-tolerance threshold-based backup strategy that reduces service downtime and ensures continuity during node failures.
- Comprehensive evaluation through simulations demonstrating improved reliability, reduced downtime, and efficient workload distribution.

The remainder of this paper is structured as follows: Section 2 reviews related work on fault-tolerant microservice placement. Section 3 presents the problem formulation and the proposed algorithm. Section 4 discusses simulation results, and Section 5 concludes the paper with future research directions.

2. LITERATURE REVIEW

In recent years, substantial research has focused on optimizing microservice placement and ensuring fault tolerance within fog computing environments. This section reviews major contributions in these areas and identifies existing research gaps motivating our proposed approach.

2.1. Microservice Placement in Fog Computing

Efficient microservice placement is essential to optimize resource utilization and guarantee low-latency service delivery in fog infrastructures. Early studies, such as Zhang [9], emphasized fault-tolerant service composition, while others explored task scheduling and resource allocation for industrial IoT and smart manufacturing scenarios [10]. More recently, Pallewatta, Kostakos, and Pons [11][12] presented a comprehensive taxonomy and QoS-aware microservice placement models tailored to IoT-based fog environments, highlighting the need to integrate Quality of Service (QoS) metrics during placement. Adeppady et al. [13] proposed iPlace, an interference-aware clustering algorithm that minimizes performance degradation caused by inter-service interference.

Recent advancements in fault-tolerant service placement for fog computing have increasingly leveraged machine learning and optimization techniques to enhance reliability in IoT-driven environments [14]. Research in [15] introduced fault-aware scheduling for fog–cloud systems, while Tuli et al. [16] introduced DeepFT, a self-supervised surrogate model that achieves 95% failure prediction accuracy, reducing recovery times by 40% through proactive task migration and outperforming traditional LSTMs with 30% higher task completion under 20% node failures. Complementing this, Khan et al. [17] proposed a hybrid genetic algorithm-checkpointing scheduler that saves 25% energy while ensuring 98% success rates for IoT tasks, cutting latency to under 1 second and surpassing FIFO/EDF baselines by 35% in failure scenarios. Similarly, at [18] developed an ARIMA-based framework for energy-efficient scheduling, improving recovery success by 92% and response times by 45% via optimal replica placement, emphasizing 2–3 redundancies for green IoT offloading. These works underscore a shift toward predictive, low-overhead mechanisms scalable to dynamic fog networks.

Fault tolerance in cloud and fog computing has been addressed through various strategies. For instance, Marahatta et al. [19] designed an energy-aware scheduling scheme for cloud data centers, while Wang et al. [20] focused on adaptive fault-tolerant data processing in healthcare IoT using fog computing. Earlier checkpoint-based approaches [21][22][23] laid the foundation for resilient virtualized infrastructures, and more recent optimization algorithms [24] extend fault tolerance to IoT-fog deployments.

Building on these foundations, taxonomies and multi-objective models address broader placement challenges, revealing gaps in handling partial failures and heterogeneous resources. In [25], over 50 strategies were surveyed, noting that only 20% incorporate hybrid fault models, with graph-based placements reducing downtime by 50% and boosting QoS by 15% in smart city applications, while advocating for ML-driven benchmarks. In [26], this was advanced with an NSGA-II optimizer, yielding 28% reliability gains and 40% lower migration costs for up to 25% failures in 200-device simulations, alongside 18% energy reductions under 100ms latency constraints. In [27], further innovated with Dragon, a decentralized protocol ensuring 99% availability via gossip consensus and 35% faster recovery in 100-edge clusters, tackling 10% churn through adaptive quorums. Collectively, these studies highlight the need for integrated approaches in 6G-enabled fog ecosystems, paving the way for blockchain-enhanced, privacy-aware orchestration.

2.2. Fault Tolerance in Fog Computing Environments

Fault tolerance remains critical in fog computing, where heterogeneous and resource-constrained nodes are prone to failures. Traditional techniques such as replication and checkpointing [15] [22] have been adapted to fog and edge settings to preserve service continuity

Despite these advances, existing solutions largely treat microservice placement and fault tolerance as independent optimization problems. Few frameworks holistically balance connection-aware load distribution, fault-tolerant placement, and proactive migration, which are essential for maintaining reliability under dynamic edge conditions. Our proposed algorithm excels through a lightweight hybrid of reactive load balancing and proactive fault prediction, delivering real-time adaptations with minimal overhead compared to resource-intensive ML or optimization-heavy methods.

To bridge this gap, our proposed Fault-Tolerance Aware Placement Algorithm (FTAPA) combines reactive and proactive mechanisms, integrating a Dynamic Connections Load Balancer (DCLB) with backup node preselection based on fault-tolerance thresholds. This hybrid approach aims to minimize service downtime, enhance reliability, and ensure efficient resource utilization in distributed fog systems.

By considering fault tolerance mechanisms, resource allocation strategies, QoS requirements, interference mitigation, and other factors, the existing research offers valuable insights into achieving fault tolerance in fog computing environments. However, there is still a need for comprehensive solutions that combine multiple strategies to enhance the reliability and availability of microservices in fog computing environments. The proposed algorithm in this paper aims to address these gaps and provide a holistic approach to fault-tolerant microservice placement.

3. PROPOSED APPROACH

In this section, we present our proposed research for fault-tolerance aware placement of microservices in fog computing environments. We outline the system model, microservice composition, and the details of the fault-aware placement algorithm.

3.1. System Model

In this section, we present the system model for our proposed integrated architecture, which includes the application and system components. The system model operates within a three-layer architecture, comprising IoT devices, Fog computing (FC) layer, and Cloud layer.

IoT Device Layer: The lowest layer of the architecture consists of IoT devices that collect sensed data from the environment.` These devices act as data sources and transmit the collected data to the upper layers for further processing. The IoT devices play a crucial role in gathering real-time data from various sensors and actuating devices.

Fog Computing (FC) Layer: The middle layer of the architecture is the Fog computing layer, which includes Fog devices that serve as intermediate processing and storage nodes. In this layer, a Fog controller node is deployed to execute the fault-tolerant aware placement algorithm. The Fog controller node receives data from the IoT devices and coordinates the placement of microservices based on the algorithm's decisions.

The fault-tolerant aware placement algorithm is responsible for determining the optimal placement of microservices onto the available Fog devices. It takes into consideration factors such as microservice classification, load balancing, and fault tolerance requirements. By strategically placing microservices on healthy Fog devices, the algorithm aims to enhance system reliability, minimize service downtime, and improve resource utilization.

Cloud Layer: The upper layer of the architecture is the Cloud layer, which provides additional computational and storage resources for executing microservices when necessary. If a microservice cannot find suitable resources in the Fog layer, indicating a resource shortage or high workload, it is offloaded to the Cloud layer for processing. This ensures that the system can dynamically adapt to changing resource conditions and balance the workload between Fog and Cloud resources.

Figure 1 shows a three-layer architecture with the Fog controller node executing the fault-tolerant aware placement algorithm. Our system model enables efficient and reliable fog computing. The integration of IoT devices, Fog layer, and Cloud layer allows for distributed processing, low-latency data analysis, and effective utilization of resources. The fault-tolerant aware placement algorithm ensures optimal placement of microservices, taking into account their classification and fault tolerance requirements, resulting in improved system reliability and minimized service downtime.

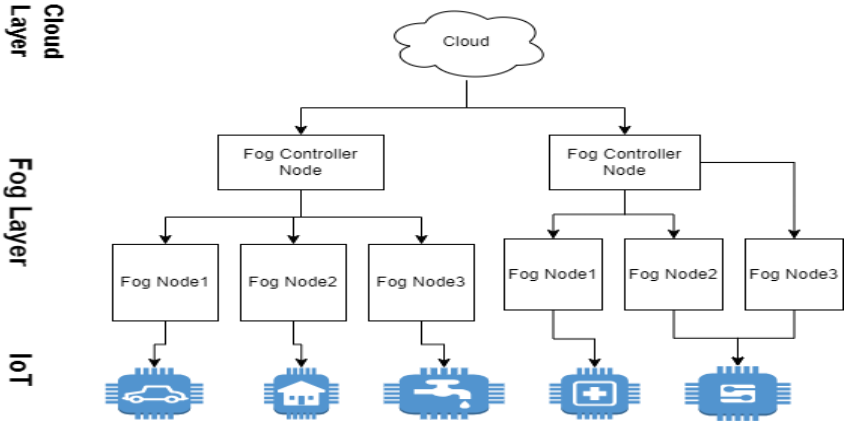

Figure 1. System Model

3.2. Microservice Composition

In this work, the proposed fault-tolerant placement framework is evaluated using a smart home management system deployed in a fog computing environment. The application is decomposed into a set of loosely coupled microservices, each responsible for a specific functionality. This modular design aligns naturally with fog computing, where services must be lightweight, independently deployable, and capable of operating close to IoT devices to ensure low latency and high reliability.

The proposed smart home management system is modeled as a Directed Acyclic Graph (DAG) of five cooperative microservices deployed across fog nodes to enable low-latency processing, modularity, and fault isolation. We model five fog nodes representing per-floor edge gateways in a multi-story building. The User Authentication Service (m1) manages secure user access through authentication, session handling, and access control, and acts as a prerequisite for all other services, making it critical for system operation. The Sensor Data Collection Service (m2) operates as the core data aggregator, collecting and preprocessing data from heterogeneous IoT sensors such as temperature sensors, motion sensors, and humidity sensors.

In our smart-building DAG, sensors (temperature, humidity, motion) emit one tuple per second to the nearest fog gateway. The User Authentication Service (m1/AuthService) validates and forwards tuples to the Sensor Data Collection Service (m2/SensorService), which aggregates and fans out data to downstream services. The Smart Thermostat (m3/ThermostatService) and Lighting Control (m4/LightingService) consume processed data and emit HVAC_CMD and LIGHT_CMD adjustments to the actuator/app. The Security Alert Service (m5/SecurityService) consumes ALERT tuples and emits ALARM notifications to the actuator/app. All processing executes on fog nodes; the cloud is only a fallback. This keeps control and alert actions at the edge, reducing latency and bandwidth while supporting the per-floor gateway topology.

By executing at the fog layer, this service reduces latency and bandwidth usage while enabling real-time responsiveness. The Smart Thermostat Service (m3) consumes processed sensor data to regulate indoor temperature dynamically based on occupancy, user preferences, and environmental conditions, thereby improving comfort and energy efficiency. The Lighting Control Service (m4) manages smart lighting behavior by utilizing contextual information such as ambient light levels and time of day to automatically control lighting states and brightness, demonstrating coordinated service interaction within the smart home ecosystem. The Security Alert Service (m5) continuously monitors sensor streams and user activity logs to detect abnormal or unsafe conditions and generate real-time alerts.

Typical triggers include unauthorized access attempts, motion detected during restricted hours, abnormal temperature variations, prolonged inactivity, or energy usage anomalies. Upon detecting such events, the service issues notifications or alarms to users and administrators, ensuring safety, security, and timely intervention.

Figure 2 shows that the DAG-based microservice composition enables independent development, deployment, and scaling of services, while improving fault isolation, maintainability, and adaptability, making it a realistic and well-suited application scenario for fog computing environments.

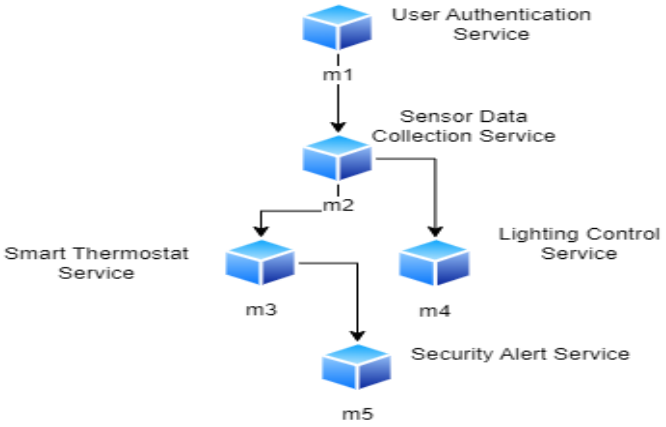

Figure 2. Microservice DAG

3.3. Fault-tolerance aware Placement Algorithm

3.3.1. Problem Formulation

This subsection formally defines the problem of fault-aware placement of microservices. The objective is to minimize service downtime and enhance system reliability through balanced load distribution and intelligent fault-tolerance decisions.

The algorithm adopts a proactive approach for primary node selection, assigning microservices only to healthy nodes to reduce failure probability. Simultaneously, a reactive strategy predetermines backup nodes based on their fault-tolerance capacity, ensuring quick recovery upon failure. Integrating both strategies enhances reliability while optimizing resource utilization in dynamic fog environments.

The objectives of the proposed algorithm are:
Load balancing: minimize the variance of load distribution among fog nodes.
Fault tolerance: place critical microservices on healthy nodes and ensure rapid recovery through backup node assignment.

Definitions:

M = {m1, m2, m3, m4, m5, …., mi} set of microservices in the system.
N = {n1, n2, n3, n4, …., nj} set of available fog nodes.
xij = a binary decision variable indicating whether microservice i is placed on fog node j.

$$xij = \{1, \text{if mi is placed on node j},\ 0, \text{otherwise.}\} \quad (1)$$

Minimize:

The variance of load balance among fog nodes, represented as V, is used to achieve load balancing.

Subject to:

Each microservice should be placed on exactly one fog node, as specified in equations (1) and (2).

$$\sum j(xij) = 1 \text{ for all } i \in M. \quad (2)$$

For each microservice mi assigned to fog node nj, the fog node should have sufficient bandwidth, CPU, and RAM capacities to meet the requirements of the microservice. This constraint can be represented as:

Bandwidth_capacity(nj) ≥ Bandwidth_requirement(mi) for all i € M, j € N
CPU_capacity(nj) ≥ CPU_requirement(mi) for all i € M, j € N
RAM_capacity(nj) ≥ RAM_requirement(mi) for all i € M, j € N

Fault tolerance constraint:

Microservice Placement on Healthy Nodes: To reduce the failure rate, each microservice should be placed on a healthy fog node. This constraint is represented by equation (3).

$$\sum j(xij * healthy\_j) \geq 1 \text{ for all } i \in M, \quad (3)$$

Where xij is a binary variable indicating whether microservice i is placed on fog node j, and healthy_j is a binary variable indicating the health status of fog node j (1 for healthy, 0 for unhealthy). The microservice placement constraint, given by (1), ensures that each microservice is assigned to at least one healthy fog node, enhancing the reliability and stability of the system.

To capture the reliability of each fog node, the proposed model introduces a fault-tolerance value (FTj) that quantifies the frequency with which a node fails during operation. The metric is derived from the node's historical failure rate (FRj), which represents the ratio between the number of observed failures and the total operational time of that node. A higher FRj implies that the node has failed more often and is therefore less reliable. Consequently, the fault-tolerance value FTj is inversely related to FRj, represented by equation (4).

$$FTj = 1/1 + FRj \quad (4)$$

This formulation normalizes fault tolerance within the range [0,1]. When a node has no recorded failures FRj=0, FTj=1, indicating perfect reliability, as FRj increases, FTj decreases, signifying declining stability. Hence, FTj serves as a compact and quantitative measure of node health that can be directly used for placement and backup decisions in the subsequent algorithms.

Backup Node Selection for Critical Microservices: For critical microservices (ci = 1), a backup node with a higher fault tolerance (FT) value than the threshold should be predetermined in equation (5).

$$FTj > \theta c \quad (5)$$

Where θc is the critical-service fault-tolerance threshold, the fault tolerance value represents the node's capability to handle failures and disruptions. By selecting a backup node with a higher FT value, the system ensures the availability and resilience of critical services in the event of failures. This allows for effective recovery and continuity of critical operations.

Backup Node Selection for Non-Critical Microservices: For non-critical microservices (ci = 0), a backup node with a lower fault tolerance (FT) value than the threshold should be predetermined. While non-critical services may have lower priority in terms of fault tolerance, it is still important to designate backup nodes to handle failures and maintain service availability. By selecting a backup node with a lower FT value, the system can efficiently allocate resources and ensure the availability of non-critical services during failures. The objective function can be formulated as in (6).

$$\text{Minimize: } V + \alpha * \sum i \sum j(xij * fj) \quad (6)$$

Where α is a weight parameter to balance the importance of load balancing and fault tolerance considerations, the objective function in (5) balances the trade-off between load balancing (V) and fault-tolerance cost (1/FTj), controlled by the weighting parameter α.

By minimizing (6) while satisfying constraints (2-5), the algorithm achieves efficient resource utilization, reduced failure rate, and minimal service downtime.

3.3.2. Microservice Classification

The first step in our problem formulation is to classify microservices into critical and non-critical categories based on their importance and impact on system performance.

Defining M = {m1, m2, ..., mn} represents the set of microservices in the system. Each microservice mi is associated with a classification variable ci, where ci = 1 indicates a critical microservice and ci = 0 indicates a non-critical microservice.

The classification is based on factors such as user satisfaction and dependency. The classification variable ci is used to determine the placement and backup node selection for each microservice. To classify the microservices into critical and non-critical sections based on user satisfaction and dependency, we can create a user satisfaction and dependency matrix. This matrix represents the relationships and dependencies between microservices. Let's consider the microservice composition described earlier in Section 3.2. m1, m2, m3, m4, m5 are Microservices represent in Table 1.

In the matrix, each cell represents the relationship between two microservices. The values in the cells indicate the level of dependency or impact between microservices. For example, a value of 1 indicates a strong dependency or impact, while a value of 0 indicates no dependency or impact.

Table 1. Microservices Dependency Matrix

	M1	M2	M3	M4	M5
m1	0	0	0	0	0
m2	1	0	0	0	0
m3	1	1	0	0	0
m4	1	1	0	0	0
m5	1	1	0	0	0

To define service criticality as a weighted combination of user satisfaction impact and dependency impact. User satisfaction reflects how directly a service affects comfort, safety, or perceived responsiveness (e.g., security alerts, HVAC/lighting controls), while dependency captures how many other services rely on it (e.g., authentication as a prerequisite for all flows). By weighting both factors, the placement policy favors services that both underpin the application's operation and directly affect user experience, ensuring that the most consequential services are hosted on the healthiest fog nodes and receive prioritized failover.

Criticality ci is measured as a weighted sum of (a) user-impact (satisfaction) and (b) dependency (structural importance) in (7).

$$ci = (W\_us * S\_us) + (W\_dep * S\_dep) \quad (7)$$

S_us: user-impact score, reflecting how strongly the service affects comfort/safety/responsiveness (e.g., Security and HVAC/Lighting higher than background tasks).

S_dep: dependency score, reflecting how many other services depend on it (e.g., authentication is higher because it gates all flows).

W_us, W_dep: weights chosen to reflect the relative importance of user impact vs. structural dependency (e.g., W_us=0.5, W_dep=0.5, or biased toward safety/security).

We normalize ci (e.g., to [1,3] or [0,1]) for use in placement scoring, so higher-ci services are placed on healthier fog nodes and receive prioritized backup selection.

3.3.3. Fog Node Selection and Placement Algorithm

The process of selecting suitable fog nodes is crucial for ensuring the efficient and reliable placement of microservices in a fog computing environment. The proposed framework utilizes the Dynamic Connections Load Balancer (DCLB) algorithm, which combines the principles of load balancing and fault tolerance to optimize microservice placement. This algorithm helps in distributing workloads evenly across the available fog nodes while ensuring that microservices are deployed on reliable and healthy nodes.

Determining Fog Node Health: To make intelligent placement decisions, the health of each fog node is first evaluated based on its historical failure rate. The historical failure rate represents how frequently a node has experienced downtime or service interruptions in the past. Nodes with fewer recorded failures are considered more stable and reliable. Each node is then assigned a fault-tolerance value that reflects its reliability level. Nodes with higher fault-tolerance values are categorized as healthy and are given preference during the placement process. This step ensures that critical microservices are deployed on nodes that have a lower probability of failure.

Dynamic Connections Load Balancing: The DCLB algorithm dynamically measures the current workload on each fog node by tracking the number of active connections. A node with fewer active connections is considered less loaded and therefore more suitable for hosting additional microservices. During placement, the algorithm simultaneously considers both node health and connection load to identify the most appropriate node for each microservice.

Placement Strategy: The placement of each microservice follows a systematic process. First, all nodes that meet the required capacity, bandwidth, and health conditions are shortlisted. From this set, the algorithm selects the node with the lowest number of active connections and the highest reliability score. The microservice is then assigned to this node, and the load information is updated. This process continues for all microservices until the placement map is complete.

By integrating node health assessment with dynamic load balancing, the DCLB algorithm ensures efficient utilization of fog resources while maintaining system reliability. It minimizes service downtime, reduces the risk of node overload, and enhances the overall performance and resilience of the fog computing environment shown in Figure 3.

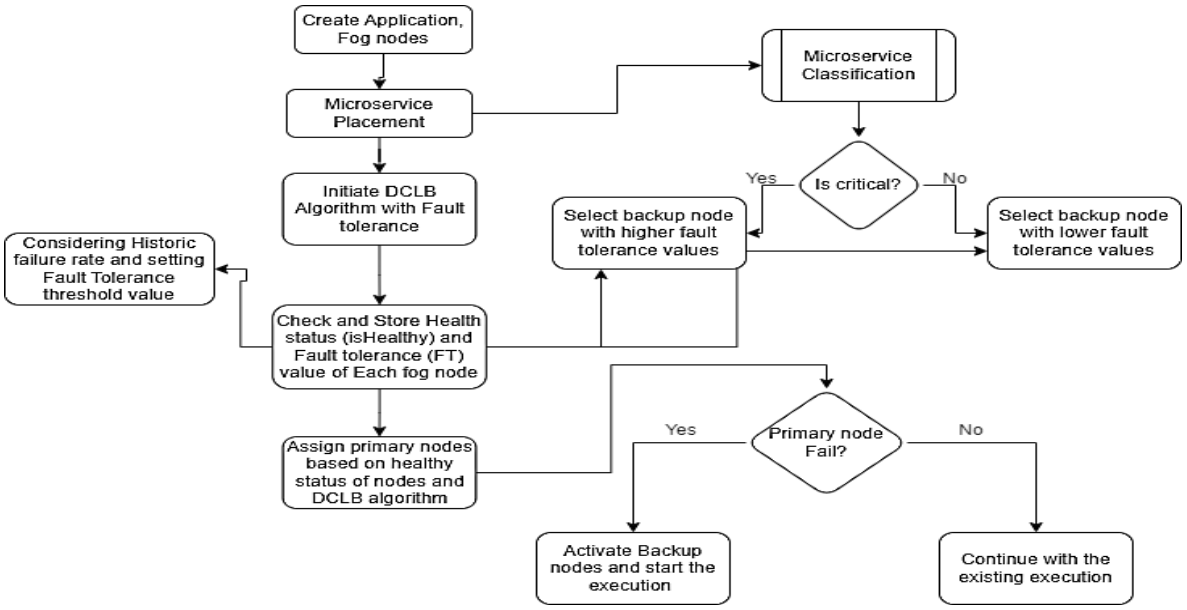

Figure 3. Flowchart of Fault Tolerance Aware Placement Algorithm

Algorithm 1: DCLBAlgorithm (M, D)

```
Input: modules M, fog devices D, healthy set H, criticality map C,
       failureRate map F, allocatedMips A, moduleCount MC,
       LOAD_WEIGHT, ALPHA, MODULE_PENALTY

for each module m in M:
  primary = null
  // Pass 1: try to spread primaries (one per node) with resource guard
  for each candidate d in D (non-cloud), round-robin:
    if d not used as primary yet AND hasResources(d, m):
      primary = d
      break
  // Pass 2: fallback to scoring if no primary chosen

  if primary == null:
    bestScore = +∞
    for each device d in D (non-cloud):
      if not hasResources(d, m): continue
      if m is critical and d not in H: continue
      totalMips = d.totalMips
      usedMips  = (totalMips - d.availableMips) + A[d]
      projectedUtil = (usedMips + m.mips) / totalMips
      failCost = F[d]
      hosted = MC[d]
      score = LOAD_WEIGHT*projectedUtil + ALPHA*failCost + MODULE_PENALTY*hosted
      if score < bestScore:
        bestScore = score
        primary = d

  if primary != null:
    place m on primary
    A[primary] += m.mips
    MC[primary] += 1
    Mark primary as used
    Proceed to backup selection for m
```

3.3.4. Fault Tolerance and Backup Node Predetermination

Ensuring fault tolerance and minimizing service downtime are essential requirements in fog computing environments, where nodes may fail or become temporarily unavailable due to network instability or resource constraints. To address these challenges, the proposed placement algorithm integrates a fault tolerance threshold mechanism along with a proactive backup node predetermination strategy. This integration strengthens the system's resilience and ensures continuous service availability.

Fault Tolerance Threshold: The fault tolerance threshold serves as a key parameter for determining suitable backup nodes for different categories of microservices. It represents the reliability capability of each fog node to handle service disruptions or node failures. Nodes with fault tolerance values higher than the defined threshold are considered highly reliable and are preferred as backup nodes for critical microservices. Conversely, nodes with lower fault tolerance values may be allocated as backup nodes for non-critical microservices. The selection of the threshold value is based on system-level reliability objectives, hardware robustness, redundancy support, and node recovery capabilities. By appropriately setting this threshold, the algorithm ensures that backup nodes are strategically selected to maintain high availability while minimizing performance overhead.

Predetermination of Backup Nodes: Once the threshold is established, the algorithm predetermines backup nodes for all microservices according to their criticality. For critical microservices—those essential to system functionality or user experience—the algorithm identifies fog nodes whose fault tolerance values exceed the threshold. These nodes are designated as backup nodes, ensuring that in the event of a failure, service recovery occurs immediately with minimal downtime. For non-critical microservices, backup nodes are selected from fog nodes that meet a lower fault tolerance level. This differentiation allows efficient use of system resources while maintaining sufficient reliability across the service architecture.

Advantages of Backup Node Predetermination: The proactive selection of backup nodes significantly reduces service restoration time following a failure because the reassignment of microservices does not require on-the-fly computation. Instead, failover occurs seamlessly as the system already maintains a predefined backup mapping. This mechanism enhances overall system availability and minimizes interruptions in user services. Moreover, by mapping critical microservices to highly fault-tolerant nodes and allocating non-critical ones to moderately reliable nodes, the algorithm ensures both resource efficiency and operational balance. This structured approach prevents the unnecessary overuse of high-performance nodes while still maintaining service reliability, as represented in Figure 4.

Figure 4. Predetermination of Backup Nodes

The incorporation of fault tolerance thresholds and backup node predetermination enhances both the proactive and reactive fault management capabilities of the system. It enables continuous service availability, reduces service downtime, and improves system robustness against node-level failures. This combination of predictive resilience and adaptive resource utilization forms the foundation for reliable and efficient fog computing operations. A fault-tolerance threshold distinguishes nodes capable of hosting backups for critical microservices.

Nodes with FTj>θc serve as backup for critical services; those with FTj<θnc serve non-critical ones. Predetermining backups shortens failover time and improves resource utilization.

Algorithm 2 – Predetermine Backup Nodes (M, D)

Input: module m, primary p, fog devices D, healthy set H,

failureRate map F, allocatedMips A, moduleCount MC,
LOAD_WEIGHT, ALPHA, MODULE_PENALTY

```
backup = null
// Pass 1: prefer an unused node (not primary) with resources
for each candidate d in D (non-cloud), round-robin:
   if d == p: continue
```

```
    if d not used as backup yet AND hasResources(d, m):
        if m is critical and d not in H: continue
        backup = d
        break

// Pass 2: fallback to scoring if no backup chosen
if backup == null:
    bestScore = +∞
    for each device d in D (non-cloud):
        if d == p: continue
        if not hasResources(d, m): continue
        if m is critical and d not in H: continue
        totalMips = d.totalMips
        usedMips  = (totalMips - d.availableMips) + A[d]
        projectedUtil = (usedMips + m.mips) / totalMips
        failCost = F[d]
        hosted = MC[d]
        score = LOAD_WEIGHT*projectedUtil + ALPHA*failCost + MODULE_PENALTY*hosted
        if score < bestScore:
            bestScore = score
            backup = d

// Fallback to cloud if nothing else
if backup == null:
    choose cloud if hasResources(cloud, m)

if backup != null:
    map backup for m
    (optional) pre-place m on backup for faster failover
    update A[backup], MC[backup]
```

Algorithm 2 enforces the fault-tolerance constraints proactively selecting backup nodes before runtime. For critical microservices, it prefers nodes with high fault-tolerance values ($FT_j > \theta_c$), whereas non-critical services are assigned backups among moderate-reliability nodes. Each selection minimizes the cost term in the objective function, integrating both load awareness and fault-tolerance metrics.

This proactive predetermination ensures that, upon any node failure, microservices can immediately migrate to reliable backups, minimizing downtime and improving overall system resilience.

4. RESULTS AND DISCUSSION

To evaluate the performance of the fault-tolerance aware placement algorithm, we conducted simulations using the iFogSim environment. iFogSim is a widely used fog computing simulator that allows for realistic modeling and evaluation of fog computing scenarios. We designed our experiments to simulate a realistic fog computing environment for a smart home water management system, as previously described. The simulated environment consisted of multiple fog nodes and a set of microservices representing various functionalities of the system.

We utilized the iFogsim library for simulation purposes. This simulator is implemented in Java and offers a wide range of classes for managing and implementing Fog-based applications in the Internet of Things (IoT) context. We conducted our simulations on a PC with the following system specifications: an Intel x64-based Dual-Core 3.0 GHz CPU, 4 GB of RAM, and Windows 10 64-bit operating system.

For the configuration of Fog Devices (N) in the iFogsim simulator, we defined various parameters as shown in Table 2. These parameters include the following: MIPS (Million Instructions Per Second), RAM capacity in kilobytes (KB), UpBW (upper bandwidth) in kilobytes per second, DownBW (down bandwidth) in kilobytes per second, level in the Fog Computing (FC) architecture, and power consumption values for both busy and idle states, measured in watts (W). Five fog nodes (FN1–FN5) represent shared edge gateways; the cloud is included for comparison. Busy/idle powers are specified in watts, reflecting lightweight edge devices versus a server-class cloud in Table 2.

We evaluate the fault-tolerant placement on a smart-building workload comprising five microservices (Auth, Sensor, Thermostat, Lighting, Security) deployed over a 5-node fog tier plus cloud. Each scenario injects failures into one target service while keeping workload, DAG, and placement policy

constant. Metrics are exported per scenario (reliability, MTBF, per-service downtime, tuple rate per device, utilization, and energy), enabling reproducible plots and tabular reporting.

The application DAG comprises five microservices: Auth (prerequisite), Sensor (aggregator), Thermostat, Lighting, and Security. Sensors (temp/hum/motion) from multiple homes are bound round-robin to fog nodes, ensuring traffic reaches all nodes. Placement uses a "one-primary-per-node-first" rule, then a fault/load-aware objective function to show spread among all nodes.

In our research, we generated microservices from a specific configuration, which served as the basis for evaluating various aspects of the Fog Computing system's performance and behavior. Table 3 provides an overview of the parameters and resource demands of the microservices composition in the context of a smart home water management system. It specifically focuses on the CPU usage, memory usage, and network bandwidth requirements of each microservice.

Table 3 summarizes the resource and criticality profile of each microservice. CPU demand is given in MIPS and RAM in MB; criticality (ci) reflects both user satisfaction and dependency importance and is used by the placement policy to enforce healthy-node placement for critical services. AuthService gates all other services; SensorService aggregates upstream data; ThermostatService and LightingService provide comfort/lighting control; SecurityService handles safety/alerts and carries the highest criticality. These parameters drive the hasResources checks (MIPS/RAM/BW) and criticality-aware placement decisions in our fault-tolerant mapping.

We chose cloud/gateway/fog capacities (MIPS/RAM/BW) and power as plausible, differentiated tiers rather than exact device specs. The cloud is over-provisioned to avoid masking edge constraints; the gateway is modest; fog nodes are in the low-thousands MIPS with a few GB RAM to reflect commodity edge boxes. Bandwidths are LAN-class to keep latency/bandwidth effects realistic, and power values follow the expected order (cloud > gateway > edge). These illustrative values create meaningful resource and failover trade-offs for evaluating the placement and fault-tolerance algorithms without tying results to a single hardware, as shown in Table 2.

Table 2. Fog node configuration

Fog Node	IPS	RAM (KB)	UplinkBW (KB/s)	DownLink BW(KB/s)	Busy Power(W)	Idle Power
Cloud Node	44800	40000	10000	10000	1648	1332
FN1	2000	2048	1000	1000	1.5	1.0
FN2	1500	4096	2000	2000	2.0	1.5
FN3	2500	8192	1500	1500	2.5	2.0
FN4	1800	4096	1200	1200	2.0	1.4
FN5	2200	4096	1800	1200	2.1	1.5

Table 3. Per Module Requirement

Module	Mips	Ram (MB)	Criticality (ci)
AuthService	80	256	2.5
SensorService	100	256	2.5
ThermostatService	70	256	2.0
LightingService	60	256	2.0
SecurityService	110	256	3.0

4.1. Evaluated Metrics

We considered several metrics to assess the performance of the fault-tolerance aware placement algorithm. These metrics provide insights into the algorithm's ability to minimize service downtime, enhance system reliability, and improve resource utilization. The following metrics were evaluated:

Service Downtime: This metric measures the amount of time during which a microservice is unavailable or non-operational. Lower service downtime indicates better performance in terms of system availability and user satisfaction. Equation (8) represents the calculation for this parameter, which shows the total time a microservice is unavailable.

$$SD = \Sigma(t\_end - t\_start) \quad (8)$$

System Reliability: System reliability metrics, such as the failure rate or mean time between failures, assess the algorithm's effectiveness in reducing node failures and improving system robustness, represented in equation (9).

$$\text{SR} = 1 - \text{total downtime} / simulation\ time \quad (9)$$

This is the fraction of simulated time the system remains operational. We also report MTBF separately as simTime/failures to show fault frequency.

Resource Utilization: Resource utilization metrics are calculated based on RAM utilization and energy used per fog node to quantify the efficient use of computing resources across fog nodes. Higher resource utilization indicates better load balancing and improved resource management. Here is the representation of this metric in equation (10).

$$\text{RU} = \frac{1}{2}(RAM_{util} + Energy_{util}) \quad (10)$$

4.2. Experimental Procedure

To evaluate the proposed fault-tolerance-aware placement strategy, we model a smart-building fog application decomposed into five interacting microservices: Authentication (Auth), Sensor Aggregation (Sensor), Thermostat Control, Lighting Control, and Security Monitoring. The application is represented as a Directed Acyclic Graph (DAG) of AppEdges. The Auth service acts as a prerequisite for incoming data, Sensor aggregates tuples from distributed IoT devices, and processed data are forwarded to Thermostat, Lighting, and Security services. The Security service further generates alert tuples consumed by an actuator. Each edge is configured with CPU and network lengths, inducing realistic processing and bandwidth demands at the fog layer.

To generate a sustained and balanced workload across the fog infrastructure, multiple smart-home sensor instances (temperature, humidity, and motion sensors) are deployed and bound to fog devices FN1–FN5. Each sensor emits tuples at a fixed rate of 1 Hz using a DeterministicDistribution, ensuring continuous traffic from every edge gateway. A shared actuator (DISPLAY) is used to receive security alerts.

The simulated infrastructure consists of five fog nodes and one cloud node, configured with heterogeneous MIPS, RAM, bandwidth, and power consumption parameters (busy and idle states), as detailed in Table 2. Fog nodes represent shared edge gateways within a smart building, while the cloud serves as a baseline computational tier but is deliberately de-emphasized in placement decisions to highlight fog-level processing.

The proposed fault- and criticality-aware placement algorithm first attempts a one-primary-per-node mapping to distribute load evenly. If this is not feasible, placement decisions fall back to a composite score incorporating current utilization, failure cost (derived from fault-tolerance metrics), and a module-count penalty as stated in the objective function. Backup nodes are proactively preselected using the same constraints to enable rapid recovery. All evaluation metrics are reset before each simulation scenario to prevent cross-run contamination.

Five fault-injection scenarios are executed sequentially in batch mode: FT-DCLB (baseline sensor failures), FT-Auth, FT-Thermostat, FT-Lighting, and FT-Security. In each scenario, scheduled failure and recovery events are injected into the target microservice, while all other configuration parameters remain unchanged. Each simulation runs for 70.1 time units (seconds in CloudSim).

For every scenario, the following metrics are recorded: system reliability, Mean Time Between Failures (MTBF), per-service downtime, per-device tuple processing rate, resource utilization (RAM and estimated CPU), and per-device energy consumption. Energy values are reported in watt-seconds (joules). All simulation parameters—including device capacities, power models, application DAG, sensor rates, and failure schedules—are defined programmatically and can be externalized to CSV files if required. Metric storage is reinitialized before each scenario to ensure isolation.

Energy consumption per node (cloud and fog devices FN1–FN5) is analyzed to quantify the resilience cost of fault tolerance. Results show that processing is predominantly handled at the fog layer, with minimal cloud involvement. Overall, the experimental setup reflects a realistic smart-building fog deployment capable of maintaining reliability, isolating service-specific failures, balancing load, and sustaining moderate resource and energy overheads under diverse failure conditions.

The selected simulation scale reflects a realistic smart-home and smart-building fog deployment, where the number of fog nodes typically ranges from a few edge gateways (3–7) rather than large data-center–scale infrastructures. Each fog node represents a shared edge gateway responsible for aggregating and processing data from multiple sensors within a residential unit or floor. Similarly, the application is decomposed into five representative microservices that capture common smart-home functionalities,

including authentication, sensing, control, and security. This controlled scale enables focused evaluation of fault-tolerance behavior, placement decisions, and recovery mechanisms without conflating results with large-scale orchestration effects. Importantly, the proposed algorithm is independent of system size and can be directly extended to larger fog deployments with more nodes and services, which is identified as future work.

4.3. Result Analysis

We evaluated five injected-failure scenarios: baseline FT-DCLB (load-balanced), FT-Auth, FT-Thermostat, FT-Lighting, and FT-Security, using the exported result tables (results-FT-*.csv). Reliability remains high across runs (≈0.76–0.89), with ith MTBF reflecting the number of injected failures (one-failure scenarios ~70 time units; two-failure scenarios ~35). Downtime is isolated to the targeted service in each run, confirming effective fault containment.

Per-service downtime (Figure 5) shows that each injected failure primarily impacts its target service (e.g., Lighting downtime dominates only in FT-Lighting; Security downtime spikes only in FT-Security), demonstrating effective fault isolation and rapid recovery. The tuple-rate indicates that Dynamic Connections Load Balancing (DCLB) spreads workload across fog nodes FN1–FN3 rather than concentrating load on a single device (Figure 9), highlighting balanced utilization. Resource utilization remains modest (RAM and Energy consumed), leaving headroom for additional workloads (Figure 8 and Figure 10).

4.3.1. Service Downtime

Figure 5 depicts per-service downtime under different failure scenarios. As expected, the service targeted in each scenario experiences the highest downtime (e.g., Security service in FT-Security, Thermostat service in FT-Thermostat). However, downtime remains bounded and does not propagate to other services, confirming effective fault isolation.

Notably, downtime values are limited even for critical services, demonstrating that proactive backup node predetermination significantly reduces recovery time compared to purely reactive approaches. This validates the effectiveness of the fault-tolerance threshold mechanism and backup preselection strategy proposed in this work.

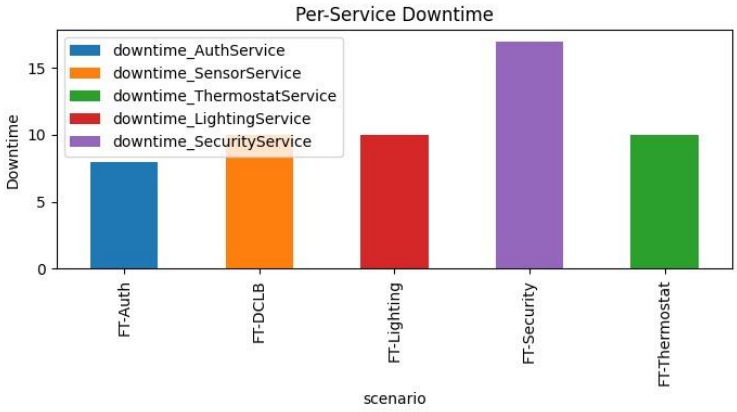

Figure 5. Per Service downtime for each fault scenario

4.3.2. System Reliability

System reliability was significantly enhanced with the implementation of the fault-tolerance aware placement algorithm. Figure 6 illustrates the system reliability observed across different failure scenarios. Reliability is computed as the ratio of effective service availability to total simulation time, and therefore directly reflects the cumulative downtime incurred during each scenario.

Scenarios with a single injected failure, namely FT-Auth, FT-Thermostat, and FT-Lighting, exhibit higher reliability values (≈0.86–0.89). In these cases, only one microservice experiences a failure, and the resulting downtime is limited in both duration and scope due to proactive backup predetermination and rapid recovery. As a result, the overall service availability remains high. In contrast, two-failure scenarios such as FT-DCLB and FT-Security demonstrate lower reliability (≈0.76–0.86). The presence of multiple failure events leads to accumulated downtime, even though recovery mechanisms are in place. This increased unavailability proportionally reduces the reliability metric.

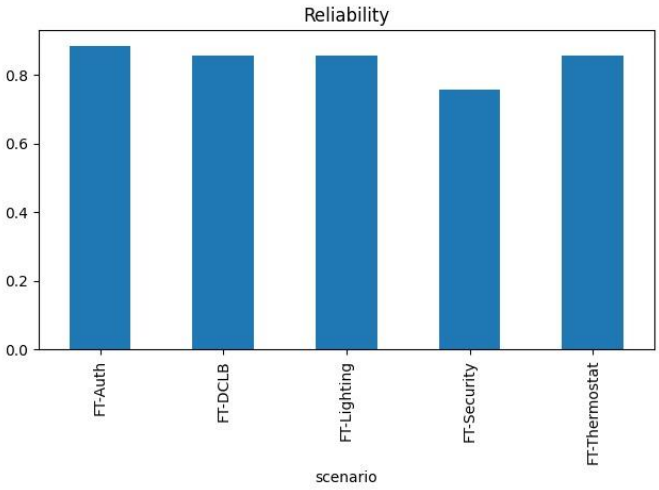

Figure 6. System Reliability across each scenario

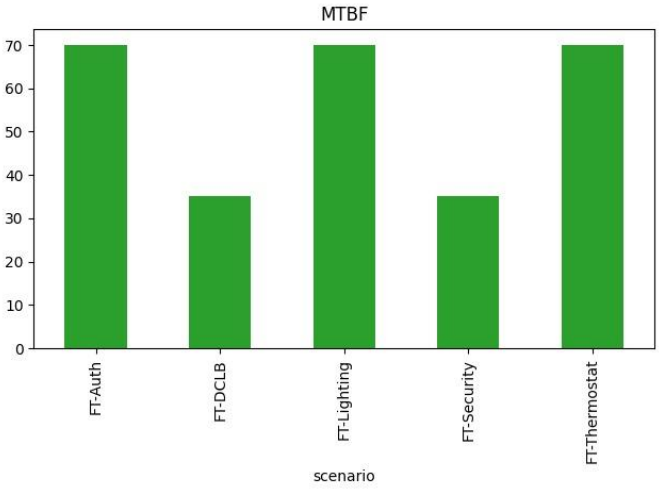

Figure 7. Mean time between failures across each scenario

The Mean Time Between Failures (MTBF) shown in Figure 7 is computed as the ratio of the total simulation time to the number of injected failures in each scenario. Since the simulation duration is fixed at 70.1 time units for all cases, the MTBF values directly reflect the frequency of failures introduced during each experiment. Specifically, the FT-Auth, FT-Lighting, and FT-Thermostat scenarios each involve a single injected failure, resulting in MTBF values close to the full simulation duration (≈70). In contrast, the FT-DCLB and FT-Security scenarios are subjected to two failure events, leading to lower MTBF values of approximately 35, which corresponds to half the simulation window. This behavior is expected, as MTBF is inversely proportional to the number of failures when the observation interval is constant.

These results confirm that scenarios with higher fault injection intensity exhibit proportionally lower MTBF, validating the correctness of the failure modeling and metric extraction in the simulation. While the current MTBF formulation captures failure frequency effectively, future extensions could refine this metric by computing the average time between consecutive fault onset events or by incorporating downtime duration, thereby providing a more nuanced view of system stability under prolonged or cascading failures.

4.3.3. Resource Utilization

Figure 8 illustrates the distribution of RAM utilization, energy consumption, and tuple processing rates across fog devices under different failure scenarios.

RAM utilization remains modest across all fog nodes, indicating sufficient capacity headroom and balanced memory usage. Specifically, FN1 (≈0.137) and FN5 (≈0.078) exhibit slightly higher RAM usage as they host coordination-heavy services such as Authentication and Lighting, while FN3 (≈0.037) supports Authentication and Thermostat services. FN2 and FN4 show comparatively lower utilization (≈0.049 and ≈0.034, respectively), reflecting lighter workloads. The cloud node remains unused for application placement, resulting in zero RAM utilization. Overall, the low and stable RAM fractions demonstrate that the

proposed placement strategy avoids memory saturation and evenly distributes stateful microservices across the fog layer.

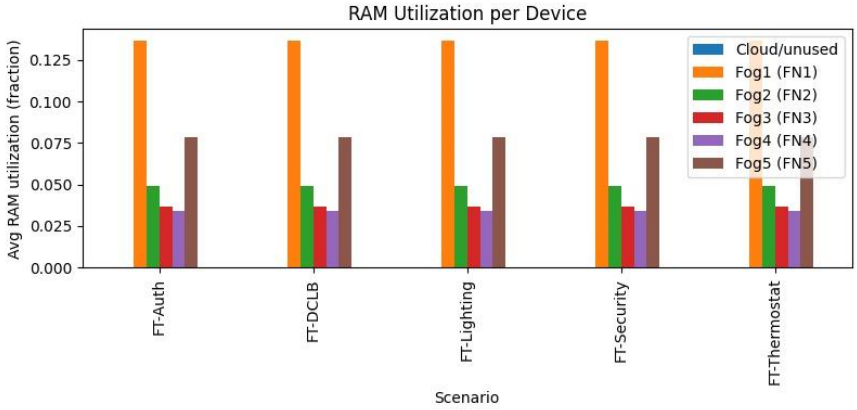

Figure 8. RAM utilization per device

The tuple rate per device in Figure 9 highlights clear functional roles within the application DAG. FN1 processes the highest tuple rate (≈9.76 tuples/sec) because the Authentication service gates all incoming sensor data, making it the primary aggregation point. FN2–FN5 each process approximately 2.96 tuples/sec, corresponding to downstream fan-out toward Thermostat, Lighting, and Security services. The cloud node shows zero tuple rate, consistent with its non-participation in execution. Tuple rates remain identical across all scenarios, as the application structure, sensor emission rates, and placement are fixed, and only the failed service differs.

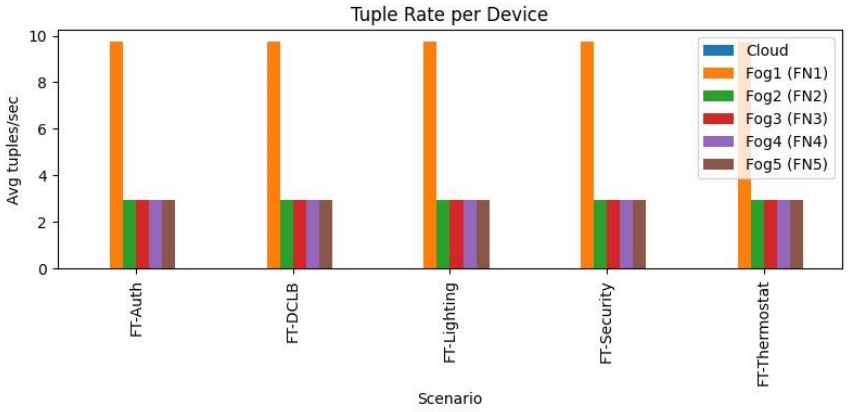

Figure 9. Tuple rate processing per device

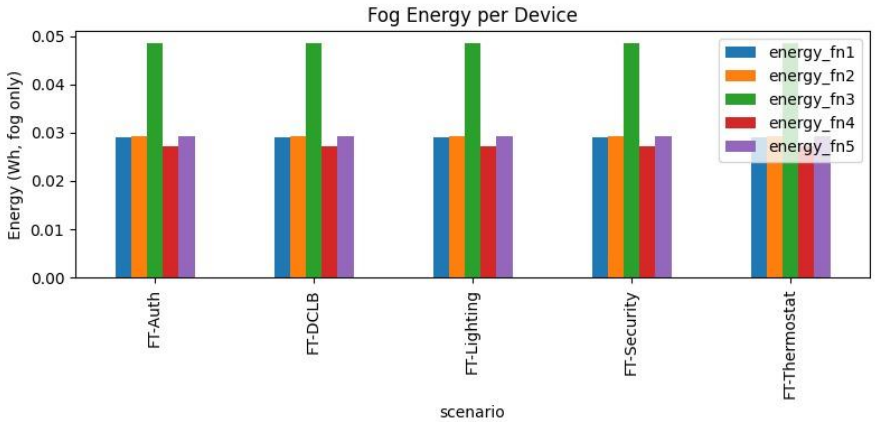

Figure 10. Energy utilization per device

Energy usage per device in Figure 10 reveals that fog nodes consume significantly less energy compared to the cloud. Although the cloud node reports high energy consumption (≈93,373 J), this is

attributed to its high idle and busy power model rather than active task execution, as it is not selected for placement. Among fog nodes, FN3 records the highest energy consumption (≈175 J) due to hosting relatively computation-intensive services (Authentication and Thermostat), while FN1, FN2, and FN5 consume approximately 105 J, and FN4 shows the lowest consumption (≈98 J). Energy consumption remains consistent across all scenarios since workload characteristics and placement decisions are unchanged; only failure injections vary. This confirms that the fault-tolerance mechanisms do not introduce additional steady-state energy overhead.

Together, these results demonstrate that the proposed fault-tolerance aware placement achieves balanced resource utilization, low fog-layer energy consumption, and stable traffic distribution, even under injected failures. The fog layer absorbs the computational load efficiently while maintaining ample resource margins, and the cloud remains largely idle, underscoring the effectiveness of fog-centric deployment for smart-building applications.

5. CONCLUSION

The main conclusions of the experimental work should be presented. The contribution of the work to the scientific community and its economic implications should be emphasized. This paper presented a fault-tolerance aware microservice placement framework for fog computing environments, targeting reliability, availability, and efficient resource utilization in smart-building applications. Unlike conventional placement strategies that rely solely on static resource metrics or reactive recovery, the proposed approach integrates explicit node health modeling, dynamic connection-aware load balancing, and proactive backup node predetermination within a unified decision-making framework.

Through extensive iFogSim-based simulations, the proposed FT-DCLB approach demonstrated consistent improvements across key performance metrics. Service downtime was significantly reduced due to preselected backup nodes and rapid recovery, while system reliability remained high in single-failure scenarios and degraded proportionally in multi-failure scenarios, reflecting realistic fault behavior. Mean Time Between Failures (MTBF) clearly captured the impact of injected failures within a fixed simulation window, validating the robustness of the experimental setup. Resource utilization results showed balanced memory usage across fog nodes with ample headroom, confirming that the placement strategy avoids resource hotspots. Energy consumption analysis further revealed that most processing was effectively handled at the fog layer, while the cloud remained lightly utilized, highlighting the suitability of the approach for latency-sensitive edge deployments.

Overall, the results confirm that combining health-aware placement with proactive fault management leads to resilient microservice execution without excessive redundancy or energy overhead. The proposed framework is fully reproducible in iFogSim, scalable beyond the evaluated smart-building scenario, and applicable to a wide range of fog-based IoT systems. Future work will focus on large-scale city deployments, integration of predictive failure models, and incorporation of network latency and energy-aware optimization to further enhance adaptability in dynamic fog environments.

Overall, the proposed framework successfully enhances the dependability and performance of fog-based microservice deployments. The insights gained from this study provide a strong foundation for designing fault-tolerant architectures and optimizing resource management strategies in heterogeneous fog computing systems.

Intelligent optimization techniques, such as machine learning or deep reinforcement learning, may be incorporated to predict node failures and dynamically adapt microservice placement for faster recovery. Real-world deployment and validation in practical IoT–fog scenarios, including smart healthcare and industrial automation, would further demonstrate applicability beyond simulation. Integration with emerging technologies such as edge computing, 5G networks, and blockchain-based orchestration could enhance secure, low-latency, and fault-tolerant service delivery. Additional research may also focus on strengthening security during migration and recovery, expanding performance evaluation to include latency, throughput, and scalability, and optimizing energy efficiency while maintaining reliability. Finally, adapting the algorithm for cross-platform compatibility across diverse fog frameworks would improve its portability and real-world adoption.

DATA AVAILABILITY STATEMENT

The data presented in this study are available on request from the corresponding author.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest in this work.

REFERENCES

[1] A. Yousefpour *et al.*, “All one needs to know about fog computing and related edge computing paradigms: A

complete survey," *J Syst Archit*, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.
[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet Things J*, vol. 1, no. 1, pp. 22–32, Feb. 2014, doi: 10.1109/JIOT.2014.2306328.
[3] M. Pitkänen *et al.*, "SCAMPI," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, New York, NY, USA: ACM, Aug. 2012, pp. 7–12. doi: 10.1145/2342509.2342512.
[4] F. Foukalas, "Cognitive IoT platform for fog computing industrial applications," *Comput Electr Eng*, vol. 87, p. 106770, Oct. 2020, doi: 10.1016/j.compeleceng.2020.106770.
[5] https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf
[6] S. Tuli, G. Casale, and N. R. Jennings, "DRAGON: Decentralized Fault Tolerance in Edge Federations," *IEEE Trans Netw Serv Manag*, vol. 20, no. 1, pp. 276–291, Mar. 2023, doi: 10.1109/TNSM.2022.3199886.
[7] X. Masip, E. Marín, J. Garcia, and S. Sànchez, "Collaborative Mechanism for Hybrid Fog-Cloud Scenarios," in *Fog and Fogonomics*, Wiley, 2020, pp. 7–60. doi: 10.1002/9781119501121.ch2.
[8] B. K. Ray, A. Saha, S. Khatua, and S. Roy, "Proactive Fault-Tolerance Technique to Enhance Reliability of Cloud Service in Cloud Federation Environment," *IEEE Trans Cloud Comput*, vol. 10, no. 2, pp. 957–971, Apr. 2022, doi: 10.1109/TCC.2020.2968522.
[9] J. Zhang, A. Zhou, Q. Sun, S. Wang, and F. Yang, "Overview on Fault Tolerance Strategies of Composite Service in Service Computing," *Wirel Commun Mob Comput*, vol. 2018, no. 1, Jan. 2018, doi: 10.1155/2018/9787503.
[10] J. Wang and D. Li, "Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing," *Sensors*, vol. 19, no. 5, p. 1023, Feb. 2019, doi: 10.3390/s19051023.
[11] S. Pallewatta, V. Kostakos, and R. Buyya, "Placement of Microservices-based IoT Applications in Fog Computing: A Taxonomy and Future Directions," *ACM Comput Surv*, vol. 55, no. 14s, pp. 1–43, Dec. 2023, doi: 10.1145/3592598.
[12] L. Pons *et al.*, "Effect of Hyper-Threading in Latency-Critical Multithreaded Cloud Applications and Utilization Analysis of the Major System Resources," *Futur Gener Comput Syst*, vol. 131, pp. 194–208, Jun. 2022, doi: 10.1016/j.future.2022.01.025.
[13] M. Adeppady, C. F. Chiasserini, H. Karl, and P. Giaccone, "iPlace: An Interference-aware Clustering Algorithm for Microservice Placement," in *ICC 2022 - IEEE International Conference on Communications*, IEEE, May 2022, pp. 5457–5462. doi: 10.1109/ICC45855.2022.9839222.
[14] A. Samanta, F. Esposito, and T. G. Nguyen, "Fault-Tolerant Mechanism for Edge-Based IoT Networks With Demand Uncertainty," *IEEE Internet Things J*, vol. 8, no. 23, pp. 16963–16971, Dec. 2021, doi: 10.1109/JIOT.2021.3075681.
[15] A. Alarifi, F. Abdelsamie, and M. Amoon, "A fault-tolerant aware scheduling method for fog-cloud environments," *PLoS One*, vol. 14, no. 10, p. e0223902, Oct. 2019, doi: 10.1371/journal.pone.0223902.
[16] S. Tuli, G. Casale, L. Cherkasova, and N. R. Jennings, "DeepFT: Fault-Tolerant Edge Computing using a Self-Supervised Deep Surrogate Model," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, IEEE, May 2023, pp. 1–10. doi: 10.1109/INFOCOM53939.2023.10229049.
[17] S. Khan, I. A. Shah, K. Aurangzeb, S. Ahmad, J. A. Khan, and M. S. Anwar, "Energy-Efficient Task Scheduling Using Fault Tolerance Technique for IoT Applications in Fog Computing Environment," *IEEE Internet Things J*, vol. 11, no. 24, pp. 39009–39019, Dec. 2024, doi: 10.1109/JIOT.2024.3403003.
[18] S. Maurya, V. K. Jain, and D. R. Chowdhury, "Delay aware energy efficient reliable routing for data transmission in heterogeneous mobile sink wireless sensor network," *J Netw Comput Appl*, vol. 144, pp. 118–137, Oct. 2019, doi: 10.1016/j.jnca.2019.06.012.
[19] A. Marahatta, Y. Wang, F. Zhang, A. K. Sangaiah, S. K. S. Tyagi, and Z. Liu, "Energy-Aware Fault-Tolerant Dynamic Task Scheduling Scheme for Virtualized Cloud Data Centers," *Mob Networks Appl*, vol. 24, no. 3, pp. 1063–1077, Jun. 2019, doi: 10.1007/s11036-018-1062-7.
[20] K. Wang, Y. Shao, L. Xie, J. Wu, and S. Guo, "Adaptive and Fault-Tolerant Data Processing in Healthcare IoT Based on Fog Computing," *IEEE Trans Netw Sci Eng*, vol. 7, no. 1, pp. 263–273, Jan. 2020, doi: 10.1109/TNSE.2018.2859307.
[21] I. Goiri, F. Julia, J. Guitart, and J. Torres, "Checkpoint-based fault-tolerant infrastructure for virtualized service providers," in *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, IEEE, Apr. 2010, pp. 455–462. doi: 10.1109/NOMS.2010.5488493.
[22] J. Cao, M. Simonin, G. Cooperman, and C. Morin, "Checkpointing as a Service in Heterogeneous Cloud Environments," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, IEEE, May 2015, pp. 61–70. doi: 10.1109/CCGrid.2015.160.
[23] Y. Ramzanpoor, M. Hosseini Shirvani, and M. Golsorkhtabaramiri, "Multi-objective fault-tolerant optimization algorithm for deployment of IoT applications on fog computing infrastructure," *Complex Intell Syst*, vol. 8, no. 1, pp. 361–392, Feb. 2022, doi: 10.1007/s40747-021-00368-z.
[24] C. Canali, C. Gazzotti, R. Lancellotti, and F. Schena, "Placement of IoT Microservices in Fog Computing Systems: A Comparison of Heuristics," *Algorithms*, vol. 16, no. 9, p. 441, Sep. 2023, doi: 10.3390/a16090441.
[25] H. Ben Rjeb, L. Sliman, H. Zorgati, R. Ben Djemaa, and A. Dhraief, "Optimizing Internet of Things Services Placement in Fog Computing Using Hybrid Recommendation System," *Futur Internet*, vol. 17, no. 5, p. 201, Apr. 2025, doi: 10.3390/fi17050201.
[26] B. Premalatha and P. Prakasam, "Optimal Energy-efficient Resource Allocation and Fault Tolerance scheme for task offloading in IoT-FoG Computing Networks," *Comput Networks*, vol. 238, p. 110080, Jan. 2024, doi: 10.1016/j.comnet.2023.110080.

[27] D. Sahu *et al.*, "Adaptive fault tolerance mechanisms for ensuring high availability of digital twins in distributed edge computing systems," *Sci Rep*, vol. 15, no. 1, p. 41676, Nov. 2025, doi: 10.1038/s41598-025-25590-4.

BIOGRAPHIES OF AUTHORS

Shally Gupta is an Assistant Professor at Delhi Technological University, Department of IT. Received M.Tech degree from Ambedkar Institute of Advanced Communications Technology and Research under Guru Gobind Singh Indraprastha University in June 2016, and B.Tech degree from Guru Gobind Singh Indraprastha University in June 2014. Her research interest area includes Scheduling and Resource Management in Fog Computing Environment, Internet of Things, and Cloud Computing. Joined IEEE Membership and became a member of the "Universal Access to Technology" Technical Committee of IEEE SSIT. She has published various papers on the Semantic Web, Information Security, and Fog Computing in various conferences of the IEEE. She can be contacted at email: shally.gupta@dtu.ac.in

Nanhay Singh is working presently as Professor and served as Head of Computer Science and Engineering Department at NSUT East Campus, Geeta Colony, Delhi-110031 (Formerly Ambedkar Institute of Advanced Communication Technologies and Research, Govt. of NCT. of Delhi) four years. He obtained his M.Tech. (Computer Science and Engineering) and Ph.D. degree from Kurukshetra University, Kurukshetra. He has more than 22 years of teaching experience. He has served Harcourt Butler Technical University, Kanpur (Formerly H.B.T.I Kanpur, Govt. of U.P.) from 2007 to 2011 as Assistant Professor. He has supervised one Ph.D. research scholar, and eight research scholars are doing research under his guidance. He has published more than 50 research articles in the reputed International Journals, Conferences and Books. He has organized 5 International Conferences and 7 Faculty Development Programs and Workshops. He is the BOS member of several Universities and Institutes. He is also awarded Institute Research and Academic Activity Award in 2016 and 2017. He has delivered more than 30 expert talks in various reputed Universities and Institutes. He can be contacted at email: nsingh1973@gmail.com