Reliability-Aware Proactive Placement of Microservices-Based IoT Applications in Fog Computing Environments

Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya

***Abstract*—The fog computing paradigm is rapidly gaining popularity for latency-critical and bandwidth-hungry IoT application deployment. Meanwhile, MicroService Architecture (MSA) is increasingly adopted for developing IoT applications due to its high scalability and extensibility. For mission-critical IoT services in fog, reliability remains one of the most critical QoS requirements due to less dependability of fog resources. Granular microservices with independent deployment and scaling exhibit great potential in utilising resource-constrained fog resources to improve reliability through redundant placement. However, current research on service placement lacks reliability-aware holistic approaches that combine the MSA features and failure characteristics of fog resources under independent and correlated failures. Hence, we analyse MSA and formulate the reliability-aware placement problem by modelling composite services as k-out-of-n serial-parallel systems in a throughput-aware manner for placement under fog resource failures. Our proposed Reliability-aware Placement Method (RPM) is a hierarchical policy combining improved PSO and NSGA-II algorithms. We integrate it with Monte Carlo reliability calculations to produce redundant placements reaching a trade-off between reliability and cost. The performance results reveal that compared to the benchmarks, our algorithm shows significant improvements in reliability satisfaction (up to 25%) and time to first failure (up to 40%), thus providing a robust placement method.**

***Index Terms*—Edge/Fog computing, fog service placement, Internet of Things, microservice architecture, redundancy, reliability.**

I. Introduction

FOG computing paradigm provides cloud-like services at the edge of the network by utilising distributed, heterogeneous and resource-constrained computing resources that reside between the edge of the network and the cloud while maintaining seamless connectivity between them [1]. Hence, Fog computing has emerged as a feasible solution for deploying latency-critical and bandwidth-hungry IoT applications. As IoT applications include highly safety-critical and mission-critical services (i.e., smart healthcare, intelligent transportation, Industrial Internet of Things (IIoT) etc.), high reliability is a crucial requirement [2]. Moreover, the heterogeneity of fog resources and their resource-constrained and geo-distributed nature results in lower dependability compared to the powerful, robust and centralised cloud servers [3], [4]. Thus, application deployment within fog computing environments should incorporate reliability awareness to minimise the application unavailability caused by fog device failures (i.e., hardware, software, power, network, etc.) while satisfying multiple other Quality of Service (QoS) requirements such as deadline, budget and throughput.

Over the years, two main approaches have been introduced to maintain application reliability: proactive failure avoidance and reactive failure recovery techniques. For IoT services with stringent latency requirements, reactive algorithms that focus on healing after faulty events are insufficient to ensure the higher level of availability required to meet low and ultra-low latency expectations, which fall within millisecond deadline limits [5], [6]. Hence, proactive methods driven by redundant placement are identified as viable solutions. In cloud environments, redundant placement is limited by the high costs incurred by deploying multiple copies of the application. In fog environments, this is further restrained by the limited availability of computing resources.

Under such challenges, the shift in IoT application development from monoliths to microservices has the potential to improve the proactive redundant placement within fog environments due to their fine-grained design. According to MicroService Architecture (MSA), complex applications are designed and developed as a collection of small and modular components known as 'microservices' that communicate with each other using lightweight communication protocols to provide end-user services [7]. Microservices are independently deployable and scalable units that are packaged using lightweight container technologies like Docker [8]. Such characteristics of microservices have made them the most suitable application model for deployment within distributed, heterogeneous and resource-constrained fog devices [9], [10]. Their ability to support independent scalability, including both vertical and horizontal scalability, enhances the chances of throughput and reliability-aware redundant placement within resource-limited fog devices (i.e., Raspberry Pis, small-cell base stations, nano data centres, edge servers etc.) with heterogeneous failure characteristics.

While MSA presents potential improvements to the reliability-aware proactive placement of IoT applications, they

Manuscript received 10 August 2023; revised 11 April 2024; accepted 23 April 2024. Date of publication 29 April 2024; date of current version 5 November 2024. Recommended for acceptance by W. Gao. *(Corresponding author: Samodha Pallewatta.)*

The authors are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3052, Australia (e-mail: ppallewatta@student.unimelb.edu.au; vassilis.kostakos@unimelb.edu.au; rbuyya@unimelb.edu.au).

Digital Object Identifier 10.1109/TMC.2024.3394486

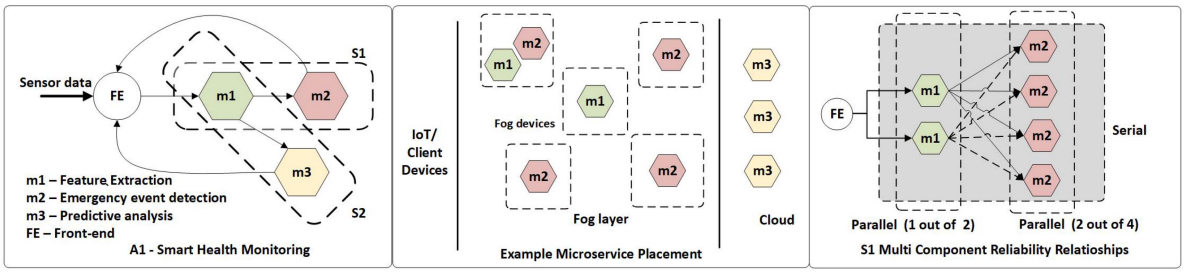

Fig. 1. A scenario of usecase in the context of smart heath monitoring.

also introduce critical challenges that must be addressed when designing placement policies. The granularity of microservices with well-defined business boundaries results in complex interactions among microservices to create 'services'. Here we use the term 'services' to denote business functionalities accessed by the end-users, which consist of one or more interconnected microservices giving rise to composite services. Furthermore, this results in each microservice-based application being a composition of multiple services with heterogeneous QoS requirements (i.e., latency-critical, latency tolerant, high bandwidth consuming etc.) where some microservices are shared among various services. This enables per-service QoS definitions which can be used with batch placement to utilise edge and cloud resources in a balanced manner [11].

Microservices-based application placement falls under Fog Service Placement Problem (FSPP) [12], [11], where each application service is deployed to provide shared access to a large number of users. Thus, concepts such as throughput-aware service scalability and load sharing are important aspects of FSPP, which sets it apart from DAG-based workflow scheduling and task offloading problems studied in the existing literature [12]. Existing research on FSPP mainly focuses on QoS parameters such as latency, cost and throughput. Thus, reliability-aware placement has a lot of room for improvement, especially for IoT applications developed using MSA. Existing works lack proper analysis of the potential of microservices-based IoT application architecture to introduce novel placement algorithms that enable the proactive redundant placement to improve the reliability of the services under both independent and correlated failures of the fog resources. Thus, there's scope for research to focus on these characteristics and utilise them to get the best out of the federated edge and cloud environments to improve reliability while satisfying other QoS parameters such as latency, cost and throughput. To further highlight this idea, we present an IoT use case modelled using MSA and examine its reliability-aware placement.

A. Motivational Scenario

We consider a use case of a smart health monitoring application (see Fig. 1) to demonstrate how MSA features can be utilised in achieving high reliability in fog applications.

Due to the granularity of microservices, QoS requirements can be defined at the composite service level. Thus, $A1$ can be represented as a composition of two composite services: a latency-critical emergency event detection service (service *S1* consisting of microservices, *m1* and *m2*) and a latency tolerant, computationally intensive analysis service (service *S2* consisting of microservices, *m1* and *m3*) [11]. The loosely coupled nature of the microservices enables dynamic deployment of microservices across fog layer resources and cloud resources in a QoS-aware manner. In our example scenario, $m1$ and $m2$ are deployed in the fog layer to accommodate the low latency requirement of $S1$, whereas $m3$, which only contributes to the latency tolerant service $S2$, is placed within cloud data centres. It improves fog resource utilisation, thus allowing more fog resources to be allocated for services with stringent latency requirements.

Being a latency-critical service, *S1* has high-reliability expectations so that in case of an emergency, the application can react within the stringent latency expectations of the service. As services like *S1* have latency requirements in the millisecond range, in case of fog resource failures, the effect on the service would be adverse if only reactive fault-tolerance methods were employed. Thus, such application services can benefit from proactive reliability ensured by redundant placements [5]. However, this is limited by the heterogeneity and resource-constrained nature of the fog devices. The independently deployable and scalable nature of the microservices can be utilised to overcome this challenge. To this end, microservice instances packaged as lightweight Docker containers can be scaled horizontally or/and vertically in throughput and reliability-aware manner. Example use case indicates that to support user requests, at least one instance of $m1$ and two instances of $m2$ are required. Failure characteristics of the fog devices can be used to improve this placement further so that redundant microservice instances are deployed to improve the service reliability. For example, the number of redundant placements can be increased if their deployed devices have low reliability (four instances of $m2$ and two instances of $m1$ depending on the failure characteristics of the fog devices they are deployed on). Hence, with MSA, each composite service is represented as a serial-parallel hybrid system, with each horizontally scaled microservice being a k out of n load-balanced sub-system of the end-user service. Here, k is the minimum number of microservice instances that can cater for the incoming user request volume, determined in a throughput-aware manner, whereas n is dynamically determined by integrating knowledge of the failure characteristics (i.e., independent and correlated failures) of fog devices to ensure availability of at least k instances during application run time.

Thus, it is evident that MSA can provide the flexibility required to utilise resource-constrained fog resources to improve

TABLE I
COMPARISON OF EXISTING RESEARCH

Work	Research Problem	Environment	Application Model	QoS			Failure Characteristics		Scalability			Batch Placement
				Reliability	Throughput	Other	Type	Repairable	Redundancy	Replica Calc.	Load Balance	
[14]	Workflow Scheduling	Cloud	DAG Workflows	✓	-	Latency	Independent	✓	-	-	-	-
[15]				✓	-	Latency, Cost	Ind., Corr.(Network)	-	✓	Static - (PB)	-	-
[16]				✓	-	Latency	Independent	✓	✓	Static - (PB)	-	-
[17]				✓	-	Latency	Ind., Corr. (Network)	-	✓	Static - (PB)	-	-
[19]	Task Offloading	Edge-Only	Independent Tasks	✓	-	Latency, Bandwidth	Independent	✓	-	-	-	✓
[18]				✓	-	Latency, Cost	Independent	✓	-	-	-	✓
[20]	FSPP	Fog (Edge-Cloud)	MSA	✓	-	Latency, Cost	Correlated	-	✓	Dynamic	✓	-
[12]				-	-	Latency	-	-	✓	Dynamic	✓	✓
[5]				-	✓	Latency	-	-	✓	Dynamic	✓	-
[11]				-	✓	Latency, Cost	-	-	-	Dynamic	✓	✓
[21]			Monolith	-	✓	Latency	-	-	-	Dynamic	✓	✓
[22]				-	✓	Latency	-	-	-	Dynamic	✓	✓
Our	FSPP	Fog	MSA	✓	✓	Latency, Cost	Ind., Corr.	✓	✓	Dynamic	✓	✓

the reliability of the deployed applications by introducing robust placement policies that combine MSA features with the failure characteristics of fog resources.

B. Proposed Approach and Contributions

The above use case demonstrates that proper utilisation of MSA characteristics can potentially improve the reliability of mission-critical IoT services through proactive and dynamic redundant placement of microservices in a "reliability and throughput aware" manner. Research that emphasises the said characteristics is still in its early stages and has much room for improvement. Existing research lacks in multiple areas, such as utilising microservice features (i.e., granular design, independent deployment and scalability, balanced deployment between fog and cloud, per-service QoS-awareness), overcoming challenges of the MSA (i.e., complex interaction patterns among microservices), application batch placement to prioritise mission-critical services, consideration of multiple failure types (i.e., independent failures, correlated failures) and dynamic redundant placement of microservice. In this work, we aim to address these shortcomings by proposing a holistic placement approach that improves the reliability of the services under multiple reliability-related metrics, such as availability and time to first failure. The **key contributions** of our work are:

1) In order to capture MSA characteristics, we model the microservices-based application services as *k out of n* serial-parallel systems and formulate the placement problem to capture reliability, throughput awareness, and cost at the composite service level. The problem formulation captures both independent and correlated failures within repairable fog environments and, dynamically calculates and places redundant microservice instances proactively.
2) Based on the problem formulation, we propose a hierarchical placement algorithm to place microservice replicas within fog environments proactively. Our proposed algorithm operates at two levels; Particle Swarm Optimisation based Throughput-aware Scalable Placement (TSP), Genetic Algorithm based Reliability-aware Redundant Placement (RRP), which together provide a robust placement method under failures in fog resources. Furthermore, a Monte Carlo-based approach is incorporated to calculate reliability-related parameters.
3) We improve the performance of the algorithm by introducing multiple novel processes: an availability-aware fitness function for TSP, an availability-aware heuristic redundancy placement for the initialisation of RRP and a reliability-aware dominant selection method for RRP.
4) We implement our policy using iFogSim2 [13] simulated fog environment and evaluate against multiple benchmarks based on reliability satisfaction, time to first failure and deployment cost.

II. RELATED WORK

In this section, we summarise current works in cloud and fog environments (see Table I) related to reliability-aware placement and proactive redundant placement, considering multiple placement problems such as FSPP, DAG workflow scheduling and task offloading. We also make a qualitative comparison between existing approaches and our work.

Multiple works consider reliability in cloud environments for the deployment of workflows, where the majority focus on scientific workflows. Rehani et al. [14] propose a DAG workflow scheduling algorithm that considers the reliability of repairable cloud resources for assigning tasks to VMs. They model the cloud failures and repairs using Weibull distribution and use Monte Carlo Failure Estimation to accurately calculate the time to failure and time to repair for each cloud resource. Tang et al. [15] consider a multi-cloud scenario to improve the reliability of the DAG-based scientific workflows to reach a trade-off between cost and reliability using the hazard rates of VMs and their connected links. Zhu et al. [16] also present a fault-tolerant DAG placement by proposing primary-backup copy placement (PB) with one replica per task deployed as a backup. Their work assumes no simultaneous failures among devices and considers only one host fails at a time. [17] extends this to consider network failures that can result in simultaneous failures of the hosts and propose a placement algorithm to place the primary and its backup copy in different subnets to overcome such failures.

Works such as Yao et al. [18], Liu et al. [19] and Aral et al. [20] focus on reliability-aware scheduling within edge computing environments. [18], [19] consider task-offloading problem considering failures of the edge VMs. [18] considers independent tasks whereas [19] models the application dataflows as DAGs. Both of these works assume the VM failures to be repairable and independent of each other. [18] tries to achieve a trade-off between cost and reliability, whereas [19] aims to balance reliability and network usage. Aral et al. [20] introduce a Bayesian

Network-based approach to model and detect correlated failures among edge nodes and combine it with link failure probabilities to calculate the joint failure probability of edge devices. When the minimum required replica count for each single-component service is provided as input, [20] outputs a redundant placement to minimise the joint failure probability of the replicas. [11], [12], [21], [22], and [5] explore the effect of replica placement to improve the performance of the fog application services. [21], [22] consider monolith applications, whereas [5], [11], [12] model the applications following MSA. [21], [22] and [11] place the minimum number of required microservice replicas to satisfy the throughput requirements of the services but do not consider redundant placements to handle uncertainty. [5] tries to overcome the throughput uncertainty of the services where some of the microservices have multiple candidates, whereas [12] proposes a method to evenly distribute microservices across the fog resources to improve service availability.

Qualitative Comparison: DAG workflow scheduling in the cloud [14], [15], [16], [17] and IoT application offloading in the edge [18], [19], both consider workloads with ephemeral life cycles where the problem is addressed from the user perspective such that the DAGs/tasks are deployed to be used by a particular user, and after the execution, each task is removed from the environment giving way to the following tasks in the queue. In contrast to this, our work considers the Fog Service Placement Problem (FSPP) described in many previous works such as [5], [11], [12], where the placement is addressed from the application provider's perspective where applications are used by a large number of users and process continuous requests, making their life cycle perpetual. This makes it infeasible to adapt former approaches to reliability-aware FSSP. Furthermore, throughput-awareness, horizontal/vertical scaling, and load balancing become essential aspects of the FSPP, which are not considered in [14], [18], [19], etc. Moreover, MSA creates composite services with complex interaction patterns among microservices. Existing works like [18], [20], [21], [22] consider independent tasks or single component services, thus failing to capture the effect of such dependencies in modelling system reliability. [5], [12] consider complex interactions among microservices along with redundant placement of microservices but do not consider failure characteristics of the edge/fog nodes to improve the reliability of the placement. Among the works that consider failures within fog environments, some consider independent failures [14], [16], whereas works like [20] consider correlated failures. [17] considers both independent and correlated failures but limits it to network failures that can be isolated at the subnet level.

Based on the above analysis, existing works lack holistic approaches that capture all the above characteristics. To this end, in our work, we consider MSA characteristics (i.e., composite services, microservice interaction patterns, independent scalability, load balancing, etc.) and propose a reliability-aware redundant placement approach for application batch placement under fog resource failures (both independent and correlated failures). We further improve the robustness of the algorithm by dynamically calculating the number of microservice replicas in a "throughput and reliability-aware" manner while reaching a trade-off between reliability and cost.

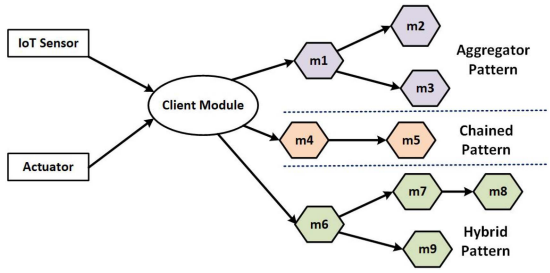

Fig. 2. Microservices-based Application Model.

TABLE II
NOTATIONS

Symbol	Definition
D	Devices available for microservice placement
A	Set of all requested applications for placement.
M_a	Set of all microservices of application $a \in A$.
S_a	Set of all services defined for application $a \in A$.
A_s	Set of services within A applications.
M^s	Set of microservices of service s.
$\overline{M}^s$	Set of critical microservices of service s.
n_m	Number of deployed instances of microservice m.
k_m	Number of instances required to satisfy the throughput demand for microservice m.
P^s	Set of data paths in service $s \in A_S$.
df_p^s	Set of all data flows in path $p \in P^s$.
$R_{mm'}$	Access rate among microservices m & m'.
l_s	makespan requirement of service $s \in S_a$.
r_s	Throughput requirement of service $s \in S_a$.
γ_d	Resource availability of device $d \in D$
Γ_m	Resources required by microservice $m \in a$ to support an access rate of r_m.
v_s^l	makespan violation of service $s \in S_a$.
$x_{m_i}^d \in \{0,1\}$	Equals to 1 if i^{th} instance of microservice m is mapped to $d \in D$, 0 otherwise.

III. System Model and Problem Formulation

A. Microservices-Based Application Model

Microservices-based applications can be modelled using a Directed Acyclic Graph (DAG) [11] where vertices denote microservices and edges represent the interactions among microservices with direction from client microservice towards the invoked microservice (Fig. 2). Each application, $a \in A$, is depicted as a collection of microservices, data flows among them, and a set of composite services providing end-user requested functionalities denoted as $< M_a, df^a, S_a >$. Each microservice is defined based on its resource requirements; $< \Gamma_m, r_m >$ where Γ_m can be a combination of multiple resources such as CPU, RAM and storage requirements of microservice $m \in M_a$ to support the request rate of r_m. This acts as the basic deployment unit of each microservice, which can be independently scaled (horizontally and vertically).

The granularity of MSA supports complex interactions, thus creating various composite service patterns (i.e., *Chained*, *Aggregator* and *Hybrid*) with diverse data flow representations (i.e., chained pattern as a single chain, aggregator pattern where multiple data paths are invoked and results are aggregated to return a single response, etc.). These data flow characteristics affect the end-to-end latency of the composite services. Thus, we represent each service $s \in S_a$ by a tuple containing the set of all microservices of the service and all possible data paths within the service: $< M^s, P^s >$.

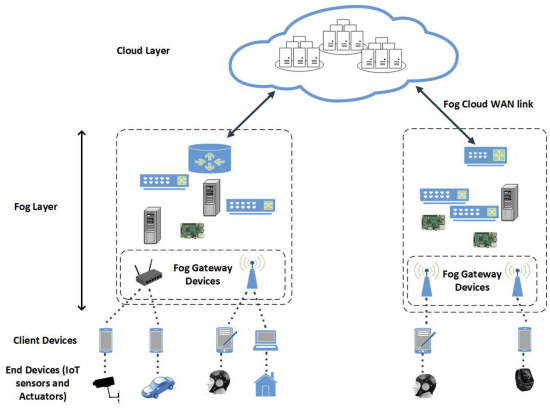

Fig. 3. An overview of the fog architecture.

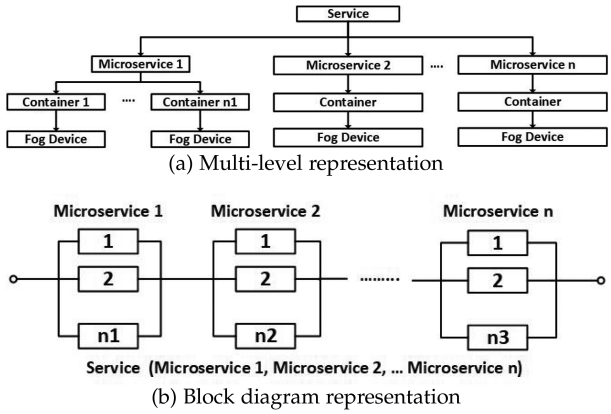

Fig. 4. Multi-component system reliability model.

B. Fog Computing Environment Model

The fog environment is represented by a hierarchical architecture consisting of three main layers: IoT/client devices, fog layer and cloud layer (Fig. 3). The fog layer, which resides between end devices and the cloud, contains heterogeneous, resource-constrained, distributed devices that provide computational, networking and storage closer to the edge of the network. We model the fog layer as clusters of such fog nodes managed by multiple service providers. Client devices access fog resources through gateway devices such as wireless access points and base transmission systems using Wireless Local Area Network (WLAN) technologies. These fog clusters maintain seamless connectivity with the cloud with Wide Area Network (WAN) links through fog-cloud gateways. Intra-cluster communication is established using high bandwidth Local Area Network (LAN) to achieve high throughput and low latency within the fog clusters. As fog devices are heterogeneous in resource availability, we characterise each device ($d \in D$) based on its resources (γ_d). γ_d can be a combination of resources including, but not limited to, CPU, RAM and storage. Moreover, in this work, we also consider the failure characteristics of the fog devices, detailed in the following sections.

C. System and Failure Characteristics

In this section, we analyse microservices-based application architecture and fog environments to create a reliability model.

1) Reliability Analysis of Microservices Applications: A failure is an event that causes a system to become unable to perform its intended task reliably [23]. A system can consist of one or more components, where system reliability depends on the failure and repair characteristics of these components. Thus, for the microservices-based application placement, we identify the system boundaries, decompose the system to identify its components and their failure characteristics, and afterwards model their effect on system performance. Fig. 4(a) depicts the multi-level representation of the system.

For the reliability modelling of a microservices-based fog applications, we consider each end-user service as a separate system with reliability requirements realised at the service level. Each service consists of one or more independently deployable and scalable microservices with data dependencies among them. Accordingly, we formulate the block representation of the system (Fig. 4(b)) to analyse the effect of component failures on the system performance. For a service S with $\overline{M}^s$ critical microservices ($\overline{M}^s \subset M^s$), each microservice $m \in \overline{M}^s$ can be horizontally and vertically scaled to meet the user demand by utilising resource-constrained fog resources. Service failure occurs when the service is unable to maintain the expected level of QoS (i.e., deadline and throughput) due to the failure of one or more microservice instances belonging to the service.

If microservice m requires a minimum of k instances to support the expected throughput demand, m is considered to be operating as expected if a minimum of k instances out of the deployed n are running without failures. Furthermore, to maintain service availability, all critical microservices of the service should be running without failures. For the chained, aggregator and hybrid interaction patterns discussed in section III-A, this results in a serial relationship among critical microservices of the service, where the failure of one or more critical microservices results in degrading the service performance or making the service unavailable until the system is restored.

Hence, for a microservices-based IoT application, reliability can be analysed per each composite service by modelling the service as a *serial-parallel hybrid system* of its critical microservices and their replicas. Following this model, we analyse the effect of underlying fog resource reliability on the availability of the service under two main resource failure types: independent and correlated.

2) Independent Failures: Independent failures in distributed computing environments include failures of servers/nodes due to factors such as hardware failures (i.e., disk failures) and software/OS failures (i.e., kernel failures, firmware failures etc.) that occur individually and independently among nodes. In literature, such failures are analysed using failure probability density functions (i.e., Weibull, Lognormal, Poisson etc.) of each node defined independently [14], [19]. Using this information, the reliability of multi-component systems can be analysed based on metrics such as Time To Failure (TTF) and availability [20].

Within fog and cloud environments, computation nodes can be repaired after failures or deployed containers can be redeployed

or migrated to working nodes upon the failure of the current nodes. As a result, in analysing the reliability of such systems, TTF can be identified as an essential metric. By maximising the TTF of services, we can minimise the number of times the microservice instances have to be redeployed or migrated to maintain service QoS, thus improving service reliability in mission-critical scenarios. At the same time, Service Level Agreements (SLAs) of the services include the reliability of the service in terms of expected average uptime availability. For microservices-based IoT applications, this can be defined at the composite service level. Hence, in this work, we create the reliability model considering both TTF and availability.

1) TTF Calculation: Based on the proposed serial-parallel hybrid reliability model of a service, the *TTF* of service S can be defined as,

$$TTF(S) = \min[TTF(m); \forall m \in \overline{M}^s] \tag{1a}$$

For each microservice, the *TTF* is determined by considering the **k-out-of-n load balancing system** represented by its instances. For microservice $m \in \overline{M}^s$, if I_m is the set of $|I_m| = n_m$ instances, the *TTF* of m is defined as,

$$TTF(m) = \min[TTF(I'_m); \forall I'_m \subset I_m] \tag{1b}$$

where $|I'_m| \geq (n_m - k_m + 1)$.

As we consider failure of each microservice instance due to the underlying host failures, failure of m occurs when the fog devices that host $n_m - k_m + 1$ instances or more of the microservice fail.

If $f[d_{m_i}]$ indicates failure event of the device $d \in D$ hosting instance m_i of microservice m, $T(\bigcap_{m_i \in I'_m} f[d_{m_i}])$ would depict the time when joint failure of all microservice instances (m_i) of I'_m occurs. Accordingly, $TTF(m)$ can be reduced to the minimum time to joint failure of the devices as follows,

$$TTF(I'_m) = min[T(\bigcap_{m_i \in I'_m} f[d_{m_i}])] \tag{1c}$$

2) Availability Calculation: Based on the proposed reliability model, the availability of service S can be defined as,

$$\text{AV}(S)_{t1,t2} = \frac{1}{(t2 - t1)} \int_{t1}^{t2} Av_S(t)dt \tag{2a}$$

$$Av_S(t) = \begin{cases} 1 & \text{Up}(I_{m,t}) \geq k_m; \forall m \in \overline{M}^s \\ 0 & \text{otherwise} \end{cases} \tag{2b}$$

(2a) defines mean availability of the service S within [t1,t2] time period in terms of service uptime. Function $Av_S(t)$ denotes if the service is in up or failed status at time t. In (2b), function $\text{Up}(I_m, t)$ calculates the number of running instances of microservice m at time t. Above two equations together calculate the average uptime availability of the service S following *k out of n* load balancing model.

3) Correlated Failures: Correlated or dependent failures, also known as Common Cause Failures (CCF), indicate one or more components of the system failing simultaneously due to a common cause. Within distributed computing environments, this can be due to failures of shared power supplies, virtual networks, network component failures, software updates, etc. [17], [20]. Such failures affect the redundant placement decisions as deploying redundant instances within a group of servers that belong to the same Common Cause Failure Group (CCFG) reduces its effectiveness. Considering this, we propose a Discorrelation Index (DI) for each microservice as follows:

$$DI(m) = \frac{\sum_{\forall g \in G} min[\frac{|I_m \setminus F_G(g, I_m)|}{k_m}, 1]}{|G|} \tag{3a}$$

(3a) considers each sub system (microservice) having a parallel relationship among its components (microservice instances). Here, $F_G(g, I_m)$ returns the instances that belong to the same CCFG ($g \in G$) and calculates the *k out of n* instance satisfaction under CCF. Based on this, calculations for each service S can be represented as follows:

$$DI(S) = \frac{\sum_{\forall m \in \overline{M}^s} DI(m)}{|\overline{M}^s|} \tag{3b}$$

D. Throughput-Aware Minimum Instance Calculation

In the k out of n parallel model derived for each microservice, k can be determined in a throughput-aware manner where the throughput requirement is defined per service (r_s for service S). We take the microservice definition proposed in our application model (Section III-A), where the resource requirement for the microservice is defined to support a certain request rate. We consider this as the base microservice instance to be deployed as a Docker container and calculate the number of instances required to support the incoming request volume. For each microservice in the DAG representation, its expected incoming request rate (r'_m) is calculated using the following equations:

$$r'_m = \sum_{\forall m' \in CM(m)} R_{m'm} \tag{4a}$$

$$R_{m'm} = \begin{cases} r_s & \text{m}' \text{ is Client Module} \\ \alpha.r'_{m'} & \text{otherwise} \end{cases} \tag{4b}$$

The access rate of the microservice m is calculated by identifying all incoming edges of m and adding their request rates (4a). To achieve this, the function $CM(m)$ outputs the client microservices of m based on the DAG representation of the application. $\alpha \in [0, 1]$ indicates the difference in rates between incoming and outgoing requests of m'. Afterwards, the minimum instance count for the microservice m is calculated as,

$$k_m = \frac{r'_m}{r_m} \tag{4c}$$

E. Service Latency Model

Due to the granularity of the MSA, deadlines can be defined at the composite service level, where the latency of each service depends on the data flow pattern of the service. Considering multiple service composition patterns, the deadline violation of service S with a deadline of l_S can be calculated based on the latency of the longest data path of the service. Considering each data path within the service ($p \in P^S$), function $L(df_p^S)$ calculates the total latency of the datapath p of service S for

the proposed placement. Due to distributed nature of the fog resources, the total latency consists of network latency ($L_{nw}(df_p^S)$) and processing latency ($L_{proc}(df_p^S)$), where network latency is a combination of transmission latency and propagation latency among different fog/cloud nodes where the microservices are deployed.

$$v_S^l = \max\{L(df_p^S); \forall p \in P^S\} - l_S \tag{5a}$$

$$L(df_p^S) = L_{nw}(df_p^S) + L_{proc}(df_p^S) \tag{5b}$$

F. Pricing Model

Cloud service providers support container deployment through serverless compute engines (i.e., AWS Fargate, Azure Container Instances etc.) where pricing is calculated based on the requested virtual CPU (vCPUs), memory and storage and flexibility is provided to configure each separately. In our work, we use the above on-demand pricing model to determine the price of deploying microservices within fog and cloud servers using container technology. For a service S having a set of M^s microservices,

$$C(S) = \sum_{\substack{\forall m \in M^s \\ \forall d \in D}} \sum_{i=1}^{n_m} x_{m_i}^d C_m^d \tag{6}$$

where, C_m^d indicates the total cost of deploying microservice m on device d. $x_{m_i}^d \in \{0,1\}$ is a binary variable which is set to 1, if the i^{th} instance of the microservice m is deployed on device d. According to the above equation total cost for service S is calculated as the total cost for deployment of all microservices instances.

G. Problem Formulation

Based on the system model, we formulate the reliability-aware placement problem as a multi-objective optimisation. As proactive redundant placement of microservices is limited by the cost of resource allocation and resource availability in fog environments, the placement problem aims to reach a trade-off between maximising reliability (7) and minimising the cost (8). Based on the proposed reliability model, the reliability of the services is represented as a composite of three metrics: TTF, availability and DI. Furthermore, the placement aims to satisfy three constraints: resource constraints (9a), service deadline (9b) and throughput requirements of the services (9c).

$$\max \mathrm{P}(A_s) = \sum_{\forall S \in A_s} [TTF(S), AV(S), DI(S)] \tag{7}$$

$$\min C = \sum_{\forall S \in A_s} C(S) \tag{8}$$

Subject to,

$$\sum_{\forall a \in A \forall m \in M_a} \sum_{\forall m_i \in I_m} x_{m_i}^d \Gamma_m \leq \gamma_d; \forall d \in D \tag{9a}$$

$$V_S^l = 0; \forall S \in A_s \tag{9b}$$

$$n_m \geq k_m; \forall m \in M_a; \forall a \in A \tag{9c}$$

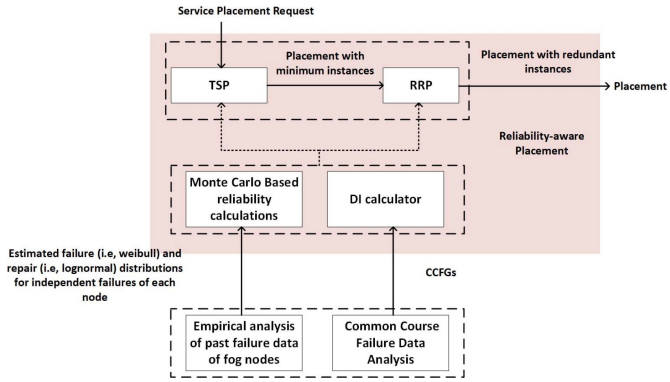

Fig. 5. Reliability-aware placement process.

As application placement within fog environments has to utilise the limited fog resources and achieve a proper balance between fog layer resource usage and cloud usage, the batch placement of applications contributes to prioritising services based on heterogeneous QoS requirements. Thus, we formulate our placement problem to support the placement of a set of applications A, where all the available services are depicted by A_s.

IV. Reliability-Aware Placement Method

A. Overview

Based on the problem formulation, we propose a **Reliability-aware Placement Method (RPM)** for the proactive redundant placement of microservices-based IoT applications. Fig. 5 presents a high-level representation of the method. Our approach consists of four main processes:

- Monte Carlo Simulation-based Service Reliability calculation process: It uses empirical data derived from past failures of the devices to calculate time to failure $(TTF(S))$ and availability $(AV(S))$ metrics based on independent failures.
- DI calculation process: It calculates DI using data on CCFGs derived from common course failure data.
- Throughput-aware Scalable Placement **(TSP)** - It generates initial microservice placement with the minimum number of microservice instances to satisfy the throughput demand.
- Reliability-aware Redundant Placement **(RRP)** - It extends TSP to accommodate the redundant deployment of microservices to improve reliability in a cost-aware manner

Monte Carlo reliability calculation and DI calculation provide service reliability-related metrics considering independent and correlated failures of the fog devices (i.e. TTF, Availability, DI). These metrics are used by TSP and RRP, which create a hierarchical approach for throughput, reliability and cost-aware redundant placement of a batch of IoT applications.

Our proposed approach assumes the availability of previous failure data of the fog resources and meta-data derived from them. This includes data related to both independent failures and correlated failures. Previous works such as [24], [25] use publicly available failure and repair data of cloud data centres to derive statistical parameters for failure and repair distributions

using empirical analysis. In our approach, such parameters derived for each fog node are provided as metadata to Monte Carlo-based reliability calculation process to derive reliability metrics based on independent failures of fog devices. To identify the possibility of correlated failures among devices, the *CCF* analysis also can be conducted using past failure data to identify spatial and temporal dependencies among fog nodes. [20] proposes a method based on a Dynamic Bayesian Network to identify fog nodes that can fail together. Using such approaches, fog devices that belong to the same *CCFG* can be determined to be used as input for calculating DI by the DI calculator process.

Our placement method (RPM denoted in Fig. 5) uses these data to propose a redundant placement method following the reliability model proposed specifically for microservices-based IoT applications. Thus, the process of deriving statistical parameters and dependency information from past failure data is out of the scope of this work. We base our policy on the derived metadata with the flexibility of updating the methods used to extract the metadata.

B. Monte Carlo Simulation-Based Service Reliability

Due to non-constant failure/repair rates of the components, the use of Markov chains and Bayesian Networks for reliability analysis becomes impractical [26]. For such repairable systems, Monte Carlo Simulation is better suited. Monte Carlo Simulation performs a virtual experiment that simulates random walks within the stochastic environment using random number generation from known probability distributions [26]. When the parameters for the failure and repair distributions of each fog node are estimated from past failure data, the Monte Carlo method uses values drawn from a uniform random variable $U(0,1)$ together with the Inverse Cumulative Distribution Function (ICDF) of the distribution to generate failure and repair times repeatedly to create histories of the system that are used to derive failure and repair times within a considered time duration.

Data centre failure and repair data analysis presented in [24], [25] shows that server failures best fit the Weibull distribution while repair times can be best modelled using Lognormal distributions. Thus, in our work, we consider these distributions to model failure and repair times of the fog nodes. However, the use of Monte Carlo Simulations to determine reliability metrics makes the approach easily adaptable to any distribution due to its use of the inverse transform method.

As most of the failures in fog resources are repairable, the effect of the repair/maintenance actions on the status of the fog nodes needs to be considered. Kijima [27] analyses such systems and proposes a model based on the system repair condition known as *general renewal process* which models general or imperfect repair of the components where the failed system is returned to a state between new and prior to the most recent failure by introducing a virtual age to the component. For a component having virtual age $V_{i-1} = v$ after the $(i-1)^{th}$ repair, the CDF for the time to i^{th} failure T becomes,

$$F(T|V_{i-1} = v) = \frac{F(T+v) - F(v)}{1 - F(v)} \tag{10}$$

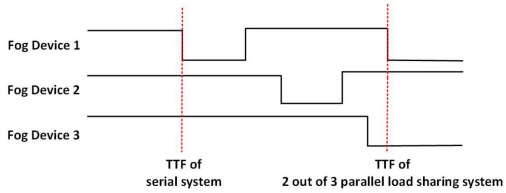

Fig. 6. Monte Carlo based TTF calculation.

For failures following the weibull distribution this results in the ICDF,

$$t_1 = \eta \sqrt[\beta]{-ln(1-U)} - t' \tag{11}$$

where t' is the time elapsed since last failure of the component from the historical data. For $i \geq 2$, ICDF is calculated as,

$$t_i = \left[\eta \sqrt[\beta]{\left(\frac{v_{i-1}}{\eta}\right)^{\beta} - ln(1-U)}\right] - v_{i-1}; i \geq 2 \tag{12}$$

where virtual age is calculated using repair degree q [28], [29] as follows:

$$v_{i-1} = q(t_1 + t' + t_2 + \cdots + t_{i-1}); 0 \leq q \leq 1 \tag{13}$$

Parameter q enables the system reliability measurements to be adjusted based on repair characteristics. q indicates the remaining damage after the repair, where $q = 0$ and $q = 1$ represent the two extreme cases of perfect repair and minimal repair, respectively [30].

Accordingly, we propose Algorithm 1 to calculate the expected $TTF(S)$ and $AV(S)$ of each service using Monte Carlo simulations. Fig. 6 shows a visual representation of how the algorithm calculates TTF for a service. For clarity, Algorithm 1 is presented as a combination of conducting Monte Carlo simulations (lines 3–20) and calculating relevant metrics using resultant events (lines 21–26). However, it's important to note that Monte Carlo simulation is a less frequently process that needs to be done as new empirical data become available or periodically. Calculated events are stored and used by placement policy which is a more frequent process. Due to this approach, Monte Carlo simulations can be carried out in cloud servers, thus overcoming the computation complexity of the process and mitigating its effect on the placement algorithm.

Results of the Monte Carlo simulation is used by TSP and RRP algorithms to generate reliability-aware placement of microservices.

C. Stage 1 - Throughput-Aware Scalable Placement

Throughput-aware Scalable Placement (TSP) is the first stage of our hierarchical placement policy (see Algorithm 2). TSP outputs a reliability-aware, scalable placement based on the throughput requirements of the services, but does not focus on the deployment of redundant microservice instances. TSP aims to achieve following objectives :

- O1: Place the minimum number of microservices required to satisfy the throughput requirement of each service.

Algorithm 1: Monte Carlo Based Service Reliability.

Input: Placement P for service S, Estimated failure and Repair distributions for each fog device
Output: $TTF(S)$, $AV(S)$, $Events$
$\overline{M}^s \leftarrow S$.getMicroservices();
$D \leftarrow P$.getAllMappedDevices() ;
$Events \leftarrow \{\}$
for d in D **do**
  $i \leftarrow 0$;
  for $i \leq simTimes$ **do**
    set $t \leftarrow 0.0$; $status \leftarrow UP$; $currentEvent \leftarrow$ first event ;
    while $t \leq T_{\max}$ **do**
      $u \leftarrow sample(U(0,1))$;
      if $status = UP$ **then**
        $\Delta t \leftarrow$timeToNextFailure$(d, u, Events.get(d))$ ▷ *This is calculated using (11), (12)* ;
      else
        $\Delta t \leftarrow$timeToRepair(d, u); ▷ *This is calculated using the ICDF of Lognormal distribution*
      end if
      $Events$.updateAverage$(d, currentEvent, \Delta t)$;
      $t \leftarrow t + \Delta t$; $currentEvent \leftarrow$ next event ;
      $status \leftarrow (status = UP)?DOWN : UP$
    end while
    $i \leftarrow i + 1$;
  end for
end for
for m in $\overline{M}^s$ **do**
  $D_m \leftarrow P$.getMappedDevices(m); $n_m \leftarrow$ no of min instances ;
  $ttf_m \leftarrow$ calculate time to $(n_m - k_m + 1)$ or more simultaneous failures based on $Events$ related to D_m ;
end for
$TTF(S) \leftarrow minimum(ttf_m; \forall m \in \overline{M}^s)$;
$AV(S)_{0,t} \leftarrow$ calculateAvailability$(Events)$; ▷ *AV(S) is calulated applying (2) to the calculated Events*
return $TTF(S)$, $AV(S)$, $Events$

Algorithm 2: TSP Algorithm.

Input: Placement Requests and Meta-data
Output: Microservices to devices mapping
Calculate the number of instances per microservice (4);
Set iteration count $i \leftarrow 1$;
  ▷ *Prioritize microservices based on deadline of the composite services they belong to*
Place all in cloud and calculate deadline violation (5a)
$ToFogM \leftarrow$ deadline violated; $ToCloudM \leftarrow$ deadline satisfied
  ▷ *Construct a random swarm of N particles under deadline and resource constraints*
$Particles \leftarrow$ initialise$(N, ToFogM, ToCloudM)$
while$i \leq Iterations$ **do**
  Calculate fitness of each particle using **AFF** ;
  Update $pBest$ and $gBest$;
  Select exemplar dimensions for each particle;
  Update velocity of each particle;
  ▷ *Update position using deadline-resource constrained prioritised construction*
  for $p \in Particles$ **do**
    for $m \in ToFogM$ **do**
      $D' \leftarrow$ eligibleFogDevices(m,p.velocityMatrix);
      Try to place m in a $d \in D'$ s.t resource constraints satisfied
      if not placed **then** $notPlaced.add(m)$
    end for
    for $m \in notPlaced$ **do**
      Try to place m in a $f \in fogDevices$ s.t resource constraints satisfied
      if not still placed **then** Place in $cloud$
    end for
    Place $ToCloudM$ in $cloud$
  end for
  Set $i \leftarrow i + 1$;
end while
return $gBest$ of the swarm

- O2: Dynamically identify microservices to be placed within fog layer based on the latency requirements of their services.
- O3: Generate a reliability-aware placement. The generated placement has two characteristics : high reliability considering possible fog environment failures, and higher potential to further improve the reliability through redundant placements in Stage 2.

At this stage, since the number of exact instances per each microservice is calculated using (4a)–(4c), we use a Particle Swarm Optimisation (PSO) based meta-heuristic to achieve throughput-reliability aware placement under resource and deadline constraints. In our previous work [11], we examined the adaptability of Set-based Comprehensive Learning Particle Swarm Optimisation (S-CLPSO) for microservices-based application placement to satisfy throughput, latency and cost requirements and introduced multiple approaches to improve its ability to achieve quicker convergence and reach the global optimum. Thus, in this stage, we adapt the improved S-CLSPO algorithm but extend and further improve it to solve the reliability-aware placement problem as follows:

1) Availability-aware fitness function (AFF): This function is introduced to satisfy O3 described above. Being the first stage of the policy, the aim of TSP is to provide an output that has the potential to be further improved with redundant placements in the next stage. To this end, we introduce a novel fitness function (14a) with 3 metrics: 1) TTF of each service, 2) a novel Availability Score for each microservice (14b), (14c) which is introduced by modifying (2) to calculate the mean number of active instances during service failure, thus aiming to minimise the simultaneous failures among its instances and improve the possibility of finding redundant placements during Stage 2 of the algorithm, and 3) DI of the placement which is also used to minimise simultaneous failures. For each particle, fitness is calculated as the summation of reliability, $\rho(S)$ of all the services

considered for placement.

$$\max \rho(S)_{t1,t2} = \left[\frac{TTF(S)}{t2 - t1} + \sum_{\forall m \in \overline{M}^s} AS(m).DI(m) \right] \tag{14a}$$

$$\text{AS}(m)_{t1,t2} = \frac{1}{(t_{fail})} \int_{t1}^{t2} As_m(t)dt \tag{14b}$$

$$As_m(t) = \begin{cases} \frac{\text{Up}(I_{m,t})}{k_m} & \text{Up}(I_{m,t}) < k_m \\ 0 & \text{otherwise} \end{cases} \tag{14c}$$

2) Multiple constraint handling: Each particle has to satisfy three main constraints to be considered a valid placement: throughput requirement of the service (O1), resource constraints of fog devices and deadline of the services (O2). The throughput requirement is handled at the start of the algorithm (line 1) by calculating the minimum number of instances (k_m) required. Other constraints are handled at the particle construction during the initial swarm creation (line 5) and the position updates conducted in each iteration (lines 11–22). To achieve deadline satisfaction, first, the deadline stringent microservices are identified (lines 3–4) and prioritised for placement within fog under resource constraints. For initialisation (line 5), the algorithm constructs particles through random assignment of microservice to devices such that the constraints are satisfied. To further improve the convergence, we seed the initial swarm with a reliability-aware heuristic placement that sorts fog devices based on their time to first failure and map the $ToFogM$ to devices with the highest time to failures. For the particle position update process, a velocity-aware position update method is implemented with deadline-resource constrained construction of particles to ensure the satisfaction of the constraints. Position update is conducted in a prioritised manner, starting with latency-sensitive microservices (lines 12–20). *eligibileFogDevices()* (line 13) method finds eligible devices in a velocity-aware manner where devices with equal or higher velocity compared to the current placed device are selected as eligible devices for the subsequent placement. This prioritises latency-critical microservices for placement within the fog, thus maximising the deadline satisfaction of the placement.

3) Updating $pBest$ and $gBest$: Due to resource constraints, fog may not be able to accommodate all latency-critical services in some particles. Hence, constructed particles, while satisfying resource constraints, may not be able to satisfy the deadline requirements after position updates. To mitigate the effect of such scenarios, $pBest$ and $gBest$ selection consider deadline satisfaction of the placement before comparing the fitness values.

The final placement generated from TSP is fed as the input to the Stage 2 of the proposed method discussed in the next section.

D. Stage 2 - Reliability-Aware Redundant Placement

During this stage, the placement generated from TSP is used as the input to the Reliability-aware Redundant Placement (RRP) algorithm (see Algorithm 3) to create redundant microservice deployments to improve the reliability further. RRP aims to achieve following objectives:

Algorithm 3: RRP Algorithm.

Input: TSP, $ToFog$, $ToCloud$ and Meta-data
Output: Microservices to devices mapping
1: Initialise population of N chromosomes using **AHI**
2: Calculate fitness using (16)
3: calculateDominants($population$) using **RDS**
4: $fronts \leftarrow$ calculateFronts($population$)
5: $crowdingDist \leftarrow$ calculateCrowdingDistance($population, fronts$)
6: **while** $i \leq Iterations$ **do**
7: $childChromosomes \leftarrow \{\}$ ▷ 2 N chromosomes
8: **while** $childChromosomes \leq N$ **do**
9: $orderedParents \leftarrow$ order($populations, fronts, crowdingDist$)
10: $parents \leftarrow$ tournamentSelect($orderedParents$)
11: $children \leftarrow$crossover($parents$)
12: $childChromosomes$.add($children$)
13: **end while**
14: mutate($childChromosomes$)
15: Calculate fitness using (16)
16: $population \leftarrow population \cup childChromosomes$
17: calculateDominants($population$) using **RDS**
18: $fronts \leftarrow$ calculateFronts($population$)
19: $crowdingDist \leftarrow$ calculateCrowdingDistance($population, fronts$)
20: $ordered \leftarrow$ order($populations, fronts, crowdingDist$)
21: $population \leftarrow$ get 1^{st} N chromosomes
22: calculateDominants($population$) using **RDS**
23: $fronts \leftarrow$ calculateFronts($population$)
24: $crowdingDist \leftarrow$ calculateCrowdingDistance($population, fronts$)
25: **end while**
26: **return** $population.best + TSP$

- O1: Dynamically place redundant microservice instance to satisfy the reliability requirements of the services.
- O2: Achieve lower deployment costs while ensuring the reliability demands.

As the number of redundant instances is not known prior to algorithm execution but decided based on the optimisation objectives, we propose an algorithm by improving NSGA-II proposed in [31]. NSGA-II is a genetic algorithm for multi-objective optimisation where each placement can be depicted as a 2D chromosome. This representation enables the count of instances to be adjusted flexibly to reach a trade-off between reliability and cost, thus satisfying O1. We make multiple improvements to adapt the NSGA-II algorithm to our specific placement problem as follows:

1) Availability-aware Heuristic Initialisation (AHI): This heuristic is used to populate the initial population in a reliability-aware manner (Algorithm 4) to achieve faster convergence by having a strong population as the starting point of the algorithm. To achieve this, AHI first calculates alternative fog devices for each device based on how they complement each other from a reliability perspective (lines 3–9). We introduce a alternative device score ($Alt\_Score_{d_1,d_2}$) based on TTF improvement

($ttf_{ext}^{d_1,d_2}$) and availability improvement ($av_{ext}^{d_1,d_2}$) as follows:

$$Alt\_Score_{d_1,d_2} = ttf_{ext}^{d_1,d_2} + av_{ext}^{d_1,d_2} \tag{15a}$$

$$ttf_{ext}^{d_1,d_2} = \begin{cases} \frac{ttf_{d_1 \cup d_2} - ttf_{d_1}}{t2-t1} & \{d_1,d_2\} \not\subset g; \forall g \in G \\ 0 & \text{otherwise} \end{cases} \tag{15b}$$

$$av_{ext}^{d_1,d_2} = \begin{cases} \frac{\left[\int_{t1}^{t2} Av_{d_1 \cup d_2}(t)dt - \int_{t1}^{t2} Av_{d_1}(t)dt\right]}{t_{f,d_1}} & \{d_1,d_2\} \not\subset g; \forall g \in G \\ 0 & \text{otherwise} \end{cases} \tag{15c}$$

(15) calculate the reliability improvement of deploying microservice instances on both d_1 and d_2 compared to deploying only on d_1, where t_{f,d_1} indicates the total failure duration of d_1 alone. To maintain the diversity among the generated chromosomes, results of the heuristic are made random by changing the order of considered mappings from TSP (line 13) and changing the order of the alternative devices (lines 19–20) to select the best alternative device out of a portion of the devices selected from D'.

2) Chromosome fitness and Reliability-aware Dominant Selection (RDS): We define the fitness of the chromosomes using 3 parameters including availability (16a), TTF (16b) and cost (8) of the placement. Based on the problem formulation in section III-G, the final fitness values are created as follows:

$$f_1 = \left[1 - \frac{\sum_{\forall S \in A_s} (Max[\rho_s - AV(S), 0])/\rho_s}{S_{num}^v}\right] DI(A_s) \tag{16a}$$

$$f_2 = \frac{\sum_{\forall S \in A_s} TTF(S)}{S_{num}.T} DI(A_s) \tag{16b}$$

where ρ_s indicates the reliability expectation of the service in terms of average uptime availability and S_{num}^v denotes the number of reliability expectation violated services. To maximise the reliability satisfaction while reducing the cost, we propose RDS (Algorithm 5) for dominant selection where higher priority is given to satisfying ρ_s using f_1 (lines 1–4) and non-dominated sorting is used for f_2 and cost (lines 5–10). O2 is satisfied through this process.

3) Generation of new population: RRP uses tournament selection, single-point crossover with random point selection and a custom mutation process to evolve the current population into the next. The mutation operator randomly selects between replica growth and replica removal. The device for replica growth is chosen by selecting a microservice placement and making a tournament selection on Alt_Score values of its alternative fog devices. Resource constraints are validated afterwards, and chromosomes undergo a mending process in case of violation by moving microservice instances from resource-violated fog devices.

Finally, RRP acquires the best chromosome of the final population by selecting the one with the highest weighted sum of the three objectives. To adjust the weighted sum as a maximisation objective, the cost is normalised using $(MaxCost - Cost)/(MaxCost - MinCost)$ for each chromosome. RRP combines the selected chromosome with TSP output and returns the final placement.

Algorithm 4: AHI Algorithm.

Input: Number of chromosomes (N) and Meta-data
Output: Initial population
1: $initPopulation \leftarrow \{\}$;
▷ *Calculate Per Device Alternatives*
2: $altDevices \leftarrow \{\}$ ▷ *Alternative devices and scores per device*
3: **for** $d \in fogDevices$ **do**
4: **for** $d' \in [fogDevices - \{d\}]$ **do**
5: $altScore \leftarrow$ calculateAtlScore(d, d') ▷ *Use (15)*
6: **if** $altScore \neq 0$ **then**
$altDevices$.add$(d, d', altScore)$
7: **end for**
8: Order $altDevices$.get(d) in descending fitness score
9: **end for**
10: **for** $n \in N$ **do**
11: $ordered \leftarrow (n \leq N/2)$?TRUE:FALSE ;
12: $P \leftarrow$ fog layer placement from TSP (list of $\{m, d\}$)
13: shuffle(P)
14: **for** $(m, d) \in P$ **do**
15: $D' \leftarrow altDevices$.get(d)
16: **if** $ordered$ is TRUE **then**
17: $d' \leftarrow$ choose device with highest $altScore$ from first device of D' s.t resource constraints are met
18: **else**
19: shuffle(D')
20: $d' \leftarrow$ choose device with highest $altScore$ from first x devices of D' s.t resource constraints are met
21: **end if**
22: **if** d' is null **then**
23: $d' \leftarrow$ select random device from $fogDevices$ s.t resource constraints are met ;
24: **end if**
25: $initPopulation$.getChromosome (n). place (m, d')
26: **end for**
27: **end for**
28: **return** $initPopulation$

Thus, TSP and RRP collectively produce a reliability, throughput and cost aware placement of microservices by incorporating knowledge on failure characteristics of the fog environment through Monte Carlo based reliability calculation and DI calculation.

V. Performance Evaluation

A. Experimental Configurations

For the evaluations, we use iFogSim2 [13] simulated fog environment. iFogSim2 provides support for modelling hierarchical fog-cloud architecture and microservice application architecture along with microservices-related functions such as horizontal scalability, load balancing and dynamic service discovery, which are essential in modelling and simulating our reliability-aware

Algorithm 5: RDS Algorithm.

Input: Chromosomes C_i and C_j
Output: **TRUE** if C_i dominates C_j, **FALSE** otherwise
if $C_i.f_1 > C_j.f_1$ **then**
  dominates $\leftarrow$ **TRUE**
else if $C_i.f_1 < C_j.f_1$ **then**
  dominates $\leftarrow$ **FALSE**
else
  if ($C_i.f_2 \geq C_j.f_2$ AND $C_i.cost \leq C_j.cost$) AND ($C_i.f_2 > C_j.f_2$ OR $C_i.cost < C_j.cost$) **then**
    dominates $\leftarrow$ **TRUE**
  else
    dominates $\leftarrow$ **FALSE**
  end if
end if
return *dominates*

deployment scenario. Furthermore, the simulator is easily extendable to simulate failure scenarios of the fog nodes.

We model the fog environment according to the architecture presented in section III-B. Network parameters of the fog environment include bandwidth and latency among different devices of the fog architecture. We extract these values from previous studies on network performance of edge networks following novel communication technologies as follows: WLAN communication (150 Mbps, 2 ms) based on WiFi-6 [32] and 5G [33], LAN connections (1 Gbps, 0.5 ms) based on gigabit Ethernet technology [34], and fog-cloud connections with WAN (30 ms, 100 Mbps) [11]. Fog device resources are defined using three parameters: CPU (1500-3000 MIPS), RAM (2-8 GB) and storage (32-256 GB) [35], [36]. These values represent resource availability of heterogeneous fog devices such as RaspberryPi, Dell PowerEdge, Jetson Nano, etc. The cost of the resources is modelled following the price model of AWS Fargate and extended to the fog layer with an increase factor of 1.2–1.5 as proposed in [37].

Due to the novelty of the fog computing paradigm, there's a lack of availability in fog computing reliability data. Hence, following previous reliability studies in the area [20], we create synthetic failure traces based on real-world failure data available for distributed systems. In our work, we use the failure characteristics presented in [25], which analyses Google cloud trace logs consisting of around 12,5000 servers monitored over 29 days. Failure characteristics of the fog devices in our simulated environment are modelled based on the results of the empirical analysis done on the said data set and fed to our placement algorithms. Failure and repair events during the simulation time are also synthesized accordingly.

Workloads used in the performance evaluation are synthetically generated following the microservices-based applications used in the literature [12], [38]. Workloads model multiple IoT applications such as smart health monitoring [39], smart parking [40], etc. and also follow general microservice composition patterns such as chained, aggregator, and hybrid patterns. Diversity among applications is ensured by varying microservice resource requirements in terms of CPU (300-900 MIPS), bandwidth (200-1500 bytes/packet), base request rate (100-200 requests/s) following previous IoT simulation benchmarks [11], [13].

B. Evaluation Overview

We conduct the evaluation of the proposed placement policy under two main criteria as follows:

1) RPM Algorithm Performance Evaluation: (results presented in Section V-C) - As explained in Sections IV-C and IV-D, we improved the convergence of the two meta-heuristic algorithms used in our placement policy through multiple novel approaches, namely AFF, AHI, RDS. To validate the effect of proposed improvements, we analyse the capability of the RPM algorithm to converge towards better placements by comparing it with multiple variants of the algorithm designed to capture the effects of the proposed improvements. To this end, we use multiple metrics that determine the fitness of the placement including Reliability (R.S: Reliability satisfaction, FTTF: First Time To Failure), Cost, and we also introduce a parameter representing Trade-off between reliability and cost (Trade Ratio). We use the results obtained in this stage to validate various improvements we introduced to S-CLPSO and NSGA-II algorithms.

2) RPM Algorithm Placement Evaluation: (results presented in Section V-D) - Here, we evaluate our proposed approach with baseline placement approaches (including previous representation works) to demonstrate the performance improvements due to different aspects considered in our proposed placement approach. To this end, we consider 3 main aspects: the use of proactive redundant placements to improve reliability, the effect of incorporating throughput-aware dynamic scalability of microservices and finally the impact of considering both independent and correlated failures.

Evaluation metrics used in this work are described below.

1) Reliability Satisfaction (R.S): Reliability Satisfaction calculates the uptime availability ($AV(S)$) of each deployed service S for the observed duration of time under Fog failures and compares it with the reliability expectation (ρ_s) of each service as follows:

$$R.S = \frac{\sum_{\forall S \in A_s} 100 - (Max[\rho_s - AV(S), 0]) * 100}{S^v_{num}} \quad (17)$$

2) First Time To Failure (FTTF): Indicates the average time to first failure of deployed services as a percentage of the total considered time duration. This follows the definition presented in (1a).

3) Cost of Deployment: Normalised total cost of deployment for all deployed microservices during the observed time period. This follows the definition presented in (6).

4) Trade Ratio: Reliability degradation per unit cost reduction. This metric is used to evaluate the algorithm based on its ability to converge to a result which can improve reliability in a cost-aware manner.

The above parameters are selected to cover the main objectives (reliability as the primary objective and cost of deployment as the secondary objective) of the proposed placement policy.

TABLE III
SIMULATION PARAMETERS

	Parameter	Value
Communication links (latency, bandwidth)	LAN	0.5ms, 1 Gbps
	WAN	30ms, 100 Mbps
	WLAN	2ms, 150 Mbps
Fog device resources	CPU (MIPS)	1500-3000
	RAM (GB)	2-8
	Storage (GB)	32-256
Cost Model parameters	CPU (Cloud)	$0.040480 per 150 MIPS per hour
	RAM (Cloud)	$0.004445 per GB per hour
	Storage (Cloud)	$0.000111 per GB per hour
	Increase factor for fog	1.2-1.5
QoS parameters	Reliability	above 99.95%
	Throughput	200-800 requests/s

TABLE IV
PARAMETERS FOR PLACEMENT ALGORITHMS

Parameter	TSP	RRP
No. of particles/ chromosomes in swarm	100	100
No. of iterations	300	300
ω_{min} - ω_{max}	0.4 - 0.9	
c	1.49445	-
m (refresh gap)	0	-

TABLE V
EVALUATION OF DIFFERENT VARIENTS (UNDER INDEPENDENT FAILURES)

	Scenario1				Scenario2				Scenario3			
Approach	R.S (%)	FTTF (%)	Cost (0-1)	Trade Ratio	R.S (%)	FTTF (%)	Cost (0-1)	Trade Ratio	R.S (%)	FTTF (%)	Cost (0-1)	Trade Ratio
RPM	98.813 ±0.201	93.533 ±0.652	0.531 ±0.005	0.036	98.577 ±0.205	94.51 ±0.503	0.609 ±0.005	0.049	98.239 ±0.307	91.784 ±0.823	0.713 ±0.007	0.069
No_AFF	95.249 ±0.435	83.019 ±0.906	0.472 ±0.007	0.139	93.073 ±0.494	81.654 ±0.899	0.524 ±0.007	0.218	91.314 ±0.799	73.317 ±1.634	0.624 ±0.01	0.324
No_AHI	97.54 ±0.344	91.005 ±0.773	0.539 ±0.007	0.086	97.664 ±0.263	92.641 ±0.577	0.592 ±0.006	0.085	97.35 ±0.404	90.62 ±0.917	0.731 ±0.01	0.127
No_RDS	90.498 ±0.594	76.085 ±0.882	0.314 ±0.002	0.192	88.161 ±0.405	76.321 ±0.603	0.373 ±0.001	0.254	80.992 ±0.565	64.307 ±0.957	0.367 ±0.002	0.364
No_CA	99.753 ±0.134	98.808 ±0.314	0.795 ±0.009	N/A	99.637 ±0.114	98.624 ±0.29	0.825 ±0.008	N/A	99.328 ±0.161	96.16 ±0.568	0.871 ±0.008	N/A

TABLE VI
EVALUATION OF DIFFERENT VARIENTS (INDEPENDENT AND CORRELATED FAILURES)

	Scenario4				Scenario5				Scenario6			
Approach	R.S (%)	FTTF (%)	Cost (0-1)	Trade Ratio	R.S (%)	FTTF (%)	Cost (0-1)	Trade Ratio	R.S (%)	FTTF (%)	Cost (0-1)	Trade Ratio
RPM	98.305 ±0.29	92.474 ±0.812	0.633 ±0.015	0.064	98.894 ±0.196	96.237 ±0.443	0.762 ±0.01	0.103	99.417 ±0.14	95.968 ±0.563	0.592 ±0.011	0.025
No_AFF	97.032 ±0.427	89.297 ±0.92	0.643 ±0.013	0.148	96.606 ±0.379	92.438 ±0.576	0.728 ±0.009	0.397	98.832 ±0.222	93.734 ±0.743	0.566 ±0.012	0.083
No_AHI	96.395 ±0.474	88.094 ±0.943	0.619 ±0.015	0.164	97.449 ±0.333	93.001 ±0.622	0.694 ±0.013	0.178	99.124 ±0.201	95.914 ±0.586	0.632 ±0.014	0.188
No_RDS	87.252 ±0.724	69.859 ±0.925	0.291 ±0.002	0.238	88.161 ±0.405	76.321 ±0.602	0.323 ±0.001	0.235	87.798 ±0.525	72.487 ±0.809	0.233 ±0.001	0.278
No_CA	99.376 ±0.178	96.753 ±0.584	0.801 ±0.01	N/A	99.225 ±0.158	97.268 ±0.39	0.794 ±0.008	N/A	99.576 ±0.122	97.337 ±0.464	0.656 ±0.011	N/A

C. RPM Algorithm Performance Evaluation

1) Experiment Overview: In this section, we evaluate RPM's ability to converge to a solution that can reach a trade-off between cost and reliability. To this end, we consider multiple design decisions made in our proposed algorithm (**RPM**) and evaluate their effect on the performance of the placement. For the comparison, the following variants of the algorithm are used,

i) *No_AFF*: In this approach, the fitness function of the **TSP** uses (2a), (2b) to calculate the availability, instead of the Availability Score proposed in **AFF**.

ii) *No_AHI*: Creates random chromosomes for the initial population of RRP algorithm, without using Algorithm 4.

iii) *No_RDS*: In this approach, reliability and cost have equal priority during dominant chromosome selection. Hence, generic non-dominated sorting is used instead of our proposed **RDS** approach.

iv) *No Cost-awareness (No_CA):* Maximises reliability without having cost as a limiting factor for the redundant placement.

We carry out the experiments for 6 workloads covering both independent and correlated failures. The algorithm's search space depends on three main parameters: the number of composite services in the batch placement, the number of fog devices eligible for placement and the time duration considered. We create the workloads to capture performance with variations in all three parameters. All variants use the same parameters for the algorithms: TSP with 100 particles, 300 iterations and RRP with 100 chromosomes, 300 iterations. Based on the results, we compare each approach with RPM to evaluate its ability to reach a better trade-off between reliability and cost.

2) Results Analysis: We calculate the Trade-off Ratio of each approach with respect to the No_CA. The results are depicted in Tables V and VI where average R.S, FTTF and Cost are calculated with 95% confidence interval. The following analysis is conducted comparing different variants of the algorithm.

The aim of AFF is for the TSP (Stage 1) to produce a placement such that it is easier for the RRP (Stage 2) to find redundant placements that can improve the overall reliability of the final output. We can validate this by comparing No_AFF and RPM. Based on the results, it is evident that R.S and FTTF of No_AFF are lower than RPM for all considered workloads. Moreover, No_AFF does not provide sufficient cost advantage compared to RPM, which is further proven by the high trade-off ratio of the resultant placement. This shows that having AFF improves RRP's ability to find redundant placements that can easily enhance the reliability of the final placement while reducing the cost.

In RPM, we have introduced a heuristic to populate the initial population of RRP such that nodes selected for redundant placement try to complement the output from the TSP. The aim of introducing this method is to improve the convergence of the RRP by creating an initial population of better solutions. We verify this by comparing RPM with No_AHI, which randomly initialises the population. Results show that RPM can achieve higher reliability satisfaction and FTTF. The costs incurred by No_AHI vary depending on the scenarios showing slightly higher or lower cost values than RPM. However, No_AHI records a lower trade-off ratio demonstrating RPM's ability to reach a better trade-off between objectives.

In No_RDS, traditional non-dominated sorting gives equal priority to cost and reliability, which results in a lower cost at the expense of lower reliability (over 9% reliability violation for considered scenarios). Thus, for mission-critical services that usually expect higher availability (around 99.99%), this approach fails to achieve a proper balance. With our proposed RDS approach, the placement algorithm handles multi-objective optimisation while giving the reliability aspect higher priority than cost. Considering these factors, RPM can reach a better trade-off between reliability and cost for services with high-reliability requirements.

Based on the above analysis, the introduced improvements (AFF, AHI and RDS) ensure RPM's ability to converge towards a placement with higher reliability while minimizing the deployment cost as a secondary objective. Thus, we use RPM in the following section to provide reliability-aware placements under different scenarios for further evaluation. However, the algorithms are designed flexibly to switch between these variations easily depending on the kind of trade-off required.

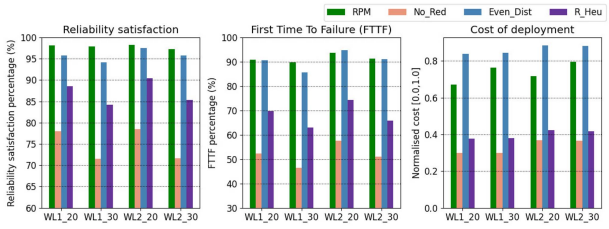

Fig. 7. Evaluation of proactive redundant placement.

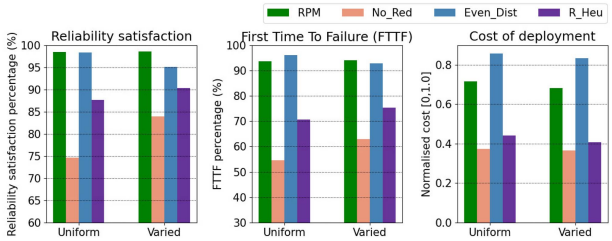

Fig. 8. Evaluation of throughput-aware scalability.

D. RPM Algorithm Placement Evaluation

1) Experiment Overview: In this section, we evaluate the efficiency of the placement generated by RPM under multiple aspects addressed by the algorithm: the effect of reliability-aware redundant placement, the impact of throughput-awareness, and finally, CCF consideration. To indicate the behaviour of the algorithms under different failure types, we start with independent failures in the first two experiments and add CCF to the final experiment to analyse the overall effect. We compare our approach with multiple alternative placement approaches as follows:

i) *No_Red*: Does not consider the redundant placement of the microservices but tries to place the minimum required microservice instances to maximise the reliability of the placement using TSP.

ii) *Even_Dist*: The placement method proposed in [12], where microservice instances are evenly replicated across the fog resources while maximising fog resource usage.

iii) *Reliability-aware Heuristic (R_Heu):* Uses the two heuristic approaches used in our placement policy to populate the initial populations of TSP and RRP algorithms. R_Heu represents an improved adaptation of primary-backup copy placement concept in [17] to our FSPP problem with load sharing.

2) Results Analysis: Effect of Redundant Placement: This section evaluates "proactive redundant placement" handled in stage 2 (RRP) of the hierarchical placement process. For this evaluation, we use two workloads (WL1 with six composite services and 30 devices, WL2 with 12 composite services and 60 devices) and consider two time periods (20 days, 30 days). Such a selection of workloads covers all three parameters that affect the solution space. Fig. 7 depicts the results of the different approaches.

Out of the approaches used in this comparison, all the approaches except No_Red utilise independent scalability of the microservices to replicate them across fog environments. Thus, in this scenario No_Red records the lowest reliability at a lower cost (see Fig. 7). Due to redundant placements, R_Heu records improved reliability compared to No_Red. However, being a heuristic approach, R_Heu lacks control over the number of redundant placements, which hinders it from achieving higher satisfaction compared to RPM. The reliability satisfactions of both of these approaches unacceptable for mission-critical services with stringent reliability expectations. Even_Dist approach shows reliability metrics closer to RPM, especially in WL2 where the number of fog devices is higher, allowing Even_Dist to deploy more replicas to ensure even distribution of instances. However, this approach incurs higher costs due to reliability-unaware replication and shows a higher reduction in reliability metrics as the considered time period increases.

Overall the results presented in Fig. 7 show that our policy is able to outperform other approaches in terms of reliability satisfaction while improving FTTF (up to 25% and 40% improvement in reliability satisfaction and FTTF, respectively). Although RPM incurs higher costs compared to No_Red and R_Heu due to higher flexibility in its replica placements, reliability and cost awareness of the algorithms allow it to reach higher reliability satisfaction (over 98%) while reducing the cost by more than 8% compared to Even_Dist which also use independent scalability of microservices for redundant placements. Thus, RMP limits the deployment cost of redundant placements as secondary objective. Above results demonstrates that RMP is able to utilise independently deployable and scalable nature of microservices to utilise limited Fog resources to improve reliability of mission-critical IoT services through proactive redundant placement of microservices.

Throughput-aware Scalability of the Placement: In this section, we evaluate how throughput awareness, together with MSA, contributes to higher performance (see Fig. 8). To this end, we use two workloads: a Uniform workload where all services have similar throughput requirements and a Varied workload having heterogeneous throughput requirements among services.

All considered approaches except Even_Dist incorporate throughput awareness into the placement. No_Red places the minimum required instances (k) to satisfy throughput requirements, whereas R_Heu deploys redundant microservice instances on top of that using the AHI algorithm. RPM formulates the problem as a k *out* n load balancing problem where n is determined robustly based on the failure characteristics of the environment. As a result, RPM reaches the highest reliability satisfaction in both scenarios (around 98.5% in both as shown in Fig. 8), adapting well to the heterogeneous throughput needs. Although No_Redundancy and R_Heu have lower performance due to limitations in proactive redundant placement, they show an increase in reliability metrics in the Varied scenario compared to Uniform. This is also a result of combining throughput and reliability awareness, where it's easier for these two approaches to ensure high reliability for low throughput services with less number of instances, which ultimately improves the average reliability compared to a uniform throughput scenario. Compared to the above three approaches, Even_Dist shows a considerable decline (98.4% in Uniform to 95.1% in Varied) in reliability metrics in the Varied scenario as this approach tries to replicate

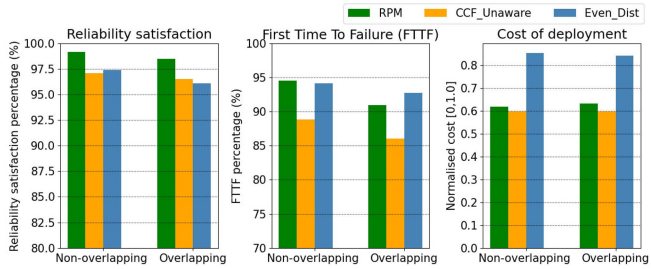

Fig. 9. Evaluation of CCF effect.

instances for all services evenly without prioritising the ones with higher throughput requirements.

Above results demonstrate that our proposed RPM approach achieves higher reliability due to incorporation of throughput aware dynamic calculation of redundant microservice instances. It improves the robustness of the algorithm to adjust to heterogeneous throughput requirements of the IoT services, thus allowing proper utilisation of limited fog resources to generate a scalable microservice placement using both horizontal and vertical scalability.

Effect of CCF: In this section, we evaluate the effect of considering common cause failures along with independent device failures. To assess the robustness of the proposed fitness functions, two main categories of CCFGs are considered: a non-overlapping scenario where device groups can be isolated and overlapping scenarios where devices can belong to multiple CCFGs in an overlapping manner (see Fig. 9).

For these two scenarios, RPM is compared with CCF_Unaware variation of the RPM algorithm and Even_dist approach. In both scenarios, RPM is able to take the effect of CCFGs into consideration for the placement decisions and hence, records the highest reliability satisfaction (up to 2.5% improvement). Because of CCFGs, RPM spreads redundant microservice instances across CCFGs such that failures of such groups would be isolated. This results in a slight increase in cost compared to CCF_Unaware (up to 3.5%), but still able to achieve around 20% cost reduction compared to Even_dist. This behavior demonstrates that by considering CCFs, RPM is able to place redundant microservice instances more efficiently by mitigating the effect of simultaneous failures of redundant instance.

Based on the experiments, it is evident that RPM provides a robust approach capable of delivering throughput-aware redundant placements under both independent and correlated failures of fog environments, while achieving a balance between reliability and cost. Moreover, the proposed algorithm is capable of navigating solution spaces of different sizes successfully and achieves higher reliability satisfaction compared to other baseline approaches. RPM also improves the cost of deployment as a secondary objective, thus providing an intelligent mechanism for utilising limited Fog resources.

VI. Conclusions and Future Work

We proposed a reliability model for microservices-based IoT applications, considering their placement within resource-constrained and heterogeneous fog devices where independent and correlated failures exist within the fog environments. Accordingly, we proposed a proactive redundant placement policy that utilises the independently deployable and scalable nature of the microservices to support the high-reliability requirements of the mission-critical IoT services in a throughput and cost aware manner. We implemented a hierarchical algorithm consisting of PSO and NSGA2-II algorithms and improved them with multiple approaches to improve the algorithm's performance. Moreover, we evaluated our approach through extensive experiments under two main aspects: performance improvements of the algorithm compared with multiple alternative approaches and efficiency of the resultant placement compared to multiple benchmark placement policies. The obtained results show that our policy can successfully navigate different solution spaces and provide robust placements that can achieve high reliability (up to 25% improvement in reliability) considering independent/correlated failures, throughput requirements of the services and cost of deployment.

Being novel distributed computing paradigm, fog computing lacks real-world commercial implementations or large scale test beds to extract large failure datasets to implement and evaluate the performance of the fog application scheduling algorithms at scale. Thus, in this work, following the evaluation approaches used by state-of-the-art fog computing research, we implemented our approach in a prominent simulator for simulating the application placement in fog environments. In order to reduce the implementation complexities when translating this work in to an real world fog computing control planes, we've proposed a modular framework which make failure data related calculations separate from the placement algorithms, so that they can be further optimised and run separately, to provide the output periodically to the placement algorithm. Moreover, this makes Monte Carlo simulation of failure data, a less frequent process that can be carried out in the Cloud to resolve the computation complexities of processing large data volumes.

Thus, in future work, we plan to implement our approach within a real-world Fog computing test-bed. Moreover, we plan to extend our approach to incorporate machine learning-based approaches to process past failure data of the fog environment to derive parameters related to the independent and correlated failures. We also plan to explore methods to obtain a trade-off between proactive and reactive placement methods to further reduce the deployment cost without compromising reliability.

References

[1] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*. Berlin, Germany: Springer, 2018, pp. 103–130.

[2] L. Xing, "Reliability in Internet of Things: Current status and future perspectives," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6704–6721, Aug. 2020.

[3] S. Bagchi, M.-B. Siddiqui, P. Wood, and H. Zhang, "Dependability in edge computing," *Commun. ACM*, vol. 63, no. 1, pp. 58–66, 2019.

[4] Z. Bakhshi, G. Rodriguez-Navas, and H. Hansson, "Dependable fog computing: A systematic literature review," in *Proc. 45th Euromicro Conf. Softw. Eng. Adv. Appl.*, 2019, pp. 395–403.

[5] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundant placement for microservice-based applications at the edge," *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1732–1745, May/Jun. 2022.

[6] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying microservice based applications with kubernetes: Experiments and lessons learned," in *Proc. IEEE 11th Int. Conf. cloud Comput.*, 2018, pp. 970–973.
[7] M. Fowler and J. Lewis, "Microservices a definition of this new architectural term," Mar. 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html
[8] J. Rufino, M. Alam, J. Ferreira, A. Rehman, and K. F. Tsang, "Orchestration of containerized microservices for IIoT using docker," in *Proc. IEEE Int. Conf. Ind. Technol.*, 2017, pp. 1532–1536.
[9] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments," in *Proc. IEEE/ACM 12th Int. Conf. Utility Cloud Comput.*, 2019, pp. 71–81.
[10] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Comput.*, vol. 3, no. 6, pp. 76–83, Nov./Dec. 2016.
[11] S. Pallewatta, V. Kostakos, and R. Buyya, "QoS-aware placement of microservices-based IoT applications in fog computing environments," *Future Gener. Comput. Syst.*, vol. 131, pp. 121–136, 2022.
[12] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Gener. Comput. Syst.*, vol. 97, pp. 131–144, 2019.
[13] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *J. Syst. Softw.*, vol. 190, 2022, Art. no. 111351.
[14] N. Rehani and R. Garg, "Reliability-aware workflow scheduling using Monte Carlo failure estimation in cloud," in *Proc. Int. Conf. Commun. Netw.*, 2017, pp. 139–153.
[15] X. Tang, "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2909–2919, Fourth Quarter 2021.
[16] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3501–3517, Dec. 2016.
[17] G. Yao, X. Li, Q. Ren, and R. Ruiz, "Failure-aware elastic cloud workflow scheduling," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1846–1859, May/Jun. 2023.
[18] J. Yao and N. Ansari, "Fog resource provisioning in reliability-aware IoT networks," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8262–8269, Oct. 2019.
[19] J. Liu et al., "Reliability-enhanced task offloading in mobile edge computing environments," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10382–10396, Jul. 2022.
[20] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1578–1590, Jul. 2021.
[21] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, and M. F. de Castro, "Optimized placement of scalable iot services in edge computing," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage.*, 2019, pp. 189–197.
[22] A. M. Maia, Y. GhamriDoudane, D. Vieira, and M. F. de Castro, "Dynamic service placement and load distribution in edge computing," in *Proc. 16th Int. Conf. Netw. Service Manage.*, 2020, pp. 1–9.
[23] W. R. Blischke and D. P. Murthy, *Reliability: Modeling, Prediction, and Optimization*. Hoboken, NJ, USA: Wiley, 2011.
[24] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Fourth Quarter 2009.
[25] P. Garraghan, P. Townend, and J. Xu, "An empirical failure-analysis of a large-scale cloud computing environment," in *Proc. IEEE 15th Int. Symp. High-Assurance Syst. Eng.*, 2014, pp. 113–120.
[26] E. Zio, "Monte carlo simulation: The method," in *The Monte Carlo Simulation Method for System Reliability and Risk Analysis*. Berlin, Germany: Springer, 2013, pp. 19–58.
[27] M. Kijima, "Some results for repairable systems with general repair," *J. Appl. Probability*, vol. 26, no. 1, pp. 89–102, 1989.
[28] A. Mettas and W. Zhao, "Modeling and analysis of repairable systems with general repair," in *Proc. Annu. Rel. Maintainability Symp.*, 2005, pp. 176–182.
[29] M. Yanez, F. Joglar, and M. Modarres, "Generalized renewal process for analysis of repairable systems with limited failure experience," *Rel. Eng. System Saf.*, vol. 77, no. 2, pp. 167–180, 2002.
[30] M. Tanwar, R. N. Rai, and N. Bolia, "Imperfect repair modeling using kijima type generalized renewal process," *Rel. Eng. System Saf.*, vol. 124, pp. 24–31, 2014.
[31] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Computation*, vol. 2, no. 3, pp. 221–248, 1994.
[32] S. Edirisinghe, C. Ranaweera, C. Lim, A. Nirmalathas, and E. Wong, "Universal optical network architecture for future wireless LANs," *J. Opt. Commun. Netw.*, vol. 13, no. 9, pp. D93–D102, 2021.
[33] D. Minovski, N. Ogren, C. Ahlund, and K. Mitra, "Throughput prediction using machine learning in LTE and 5G networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 3, pp. 1825–1840, Mar. 2021.
[34] A. Yousefpour et al., "FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5080–5096, Jun. 2019.
[35] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 3, pp. 940–954, Mar. 2022.
[36] M. Goudarzi, M. S. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2491–2505, May 2023.
[37] D. T. Nguyen, H. T. Nguyen, N. Trieu, and V. K. Bhargava, "Two-stage robust edge service placement and sizing under demand uncertainty," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1560–1574, Jan. 2022.
[38] S. Deng et al., "Optimal application deployment in resource constrained distributed edges," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 1907–1923, May 2021.
[39] A. A. Abdellatif, A. Mohamed, C. F. Chiasserini, M. Tlili, and A. Erbad, "Edge computing for smart health: Context-aware approaches, opportunities, and challenges," *IEEE Netw.*, vol. 33, no. 3, pp. 196–203, May/Jun. 2019.
[40] R. Ke, Y. Zhuang, Z. Pu, and Y. Wang, "A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on IoT devices," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 4962–4974, Aug. 2021.

Samodha Pallewatta is currently working toward the PhD degree with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia. Her research interests include encompass Fog/Edge computing, Internet of Things (IoT) and distributed systems. She is one of the contributors of the iFogSim simulator, used extensively for resource management research in Fog/Edge computing

Vassilis Kostakos is a professor with the School of Computing and Information Systems, University of Melbourne, Melbourne, Australia. His research includes Internet of Things, ubiquitous computing, human–computer interaction and social computing. He is a Marie Curie fellow, a fellow with the Academy of Finland Distinguished Professor Program, and a Founding Editor of the Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies.

Rajkumar Buyya is a Redmond Barry Distinguished professor and director with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored more than 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=167, g-index=360, 146500+ citations).