Received February 6, 2020, accepted March 1, 2020, date of publication March 10, 2020, date of current version March 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2979939

A Decentralized Latency-Aware Task Allocation and Group Formation Approach With Fault Tolerance for IoT Applications

MUHAMMAD MUDASSAR[1], YANLONG ZHAI[1], (Member, IEEE), LEJIAN LIAO[1], (Member, IEEE), AND JUN SHEN[2], (Senior Member, IEEE)
[1]School of Computer Science, Beijing Institute of Technology, Beijing 100081, China
[2]School of Computing and Information Technology, Faculty of Engineering and Information Science, University of Wollongong, Wollongong, NSW 2522, Australia

Corresponding authors: Yanlong Zhai (ylzhai@bit.edu.cn) and Lejian Liao (liaolj@bit.edu.cn)

This work was supported by the Natural Science Foundation of China under Grant 61602037 and Grant 61872079.

ABSTRACT Development of internet of things (IoT) and smart devices eased life by offering numerous applications targeting to provide real-time low latency services, but they also brought challenges in handling huge data generated from the powerful computations, to get a job done. Decentralized edge computing could help to achieve latency requirements of the applications by executing them closer to the user at edge of network, but most of the current studies actually deployed centralized approaches for cluster computing at edge, which put extra overhead of cluster formation and management. In this article, we propose to group heterogeneous edge nodes on task arrival with a more decentralized method and execute tasks in parallel to meet their deadline. On the other hand, to guarantee successful execution of critical IoT application running in an edge network, fault tolerance has to be significantly considered. For resource limited edge devices, there is a great need for efficient fault tolerance techniques, which can provide reliability based on device's local information, without worrying about overall network topology. In this article, our novel method is to increase task reliability being executed in distributed edge computing environment through finding reliability of an edge node locally, and by providing fault tolerance to increase overall application availability. Our proposed fault tolerance technique works in decentralized mode by executing new algorithms to handle above mentioned problems. Our experiment results show that our approach is effective as well as providing desired goals of achieving deadline for latency-aware IoT applications, with staggering decrease in overall network traffic along with ensuring reliability and availability.

INDEX TERMS Edge computing, distributed computing, Internet of Things, fault tolerance.

I. INTRODUCTION

Internet of Things (IoT) is expanding day by day and making lives of individuals easier by offering many applications like smart cities, home automation, security surveillance, and smart health care at any time from any place. All of these devices and applications under umbrella of IoT are generating large amount of data, which can be very useful if analyzed properly well on time. Big data processing technologies usually use cloud-computing resources for big task processing, but accessing cloud resources from end user devices adds latency. Further, it requires high bandwidth and is not suitable for real-time systems. To address these issues, mobile edge computing [1] was proposed, which allows moving computations at edge of the network closer to the users and IoT end-points.

To guarantee low-latency execution for the deadline sensitive IoT applications, edge computing offers to process these applications in close proximity to end devices that act as both data generators and consumers. There are various characteristics of edge computing making it an appropriate platform for providing critical IoT services. In edge computing where resources are available inside local network, it offers more privilege to a number of applications that require: a) low-latency real-time results; b) decentralization; c) location-oriented; d) mobility; e) heterogeneity [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Maaz Rehan.

Besides the resource rich cloud computing paradigm, nodes in edge computing, referred as edge nodes, are resource-poor devices and heterogeneous in terms of resources along with the mobility property attached to them. IoT applications with resource intensive tasks, when executed on individual edge nodes, might hinder quality of service (QoS) and user satisfaction. This results as counterproductive rather than productive. In edge computing paradigm, the concept of task offloading to a remote server promise to fill the gap between inefficient processing capability of edge node and high computation demands of resource intensive applications [3], [4]. However, cloud task offloading worsens the problem of latency and increases data traffic routed to cloud.

Usually smart devices are co-located in proximity to other devices from time to time, suggesting that edge nodes can potentially collaborate to execute a resource intensive, real-time, latency-aware task by dividing its computations among available devices, as shown in Fig. 1. By distributing complex (compute-intensive) processing (e.g., speech recognition, object detection, planning navigation, machine learning) in multiple edge resources closer to IoT end devices might result in lower latency, and also a staggering cut down in overall network traffic can be achieved, as data will be restricted in local network rather than routing towards cloud in default mobile edge computing.

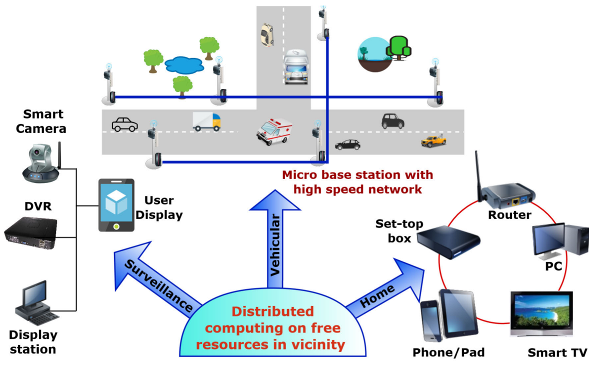


FIGURE 1. Distributed edge computing on available smart devices in the locality.

Techniques for resource leveraging at the edge [5] or clustering solutions [6] to group IoT devices might solve the above problem to some extent, but they rely on master-slave architecture or involve hierarchal grouping of devices. One common property of current grouping or clustering methods is that they operate in a non-distributed and centralized manner, which can cause problems in heterogeneous edge nodes environment. The cluster management in itself has a constant overhead in terms of cost and executions, and the failure of cluster head will make the entire cluster unavailable. Hence, there is a need for decentralized technique helping the edge-computing environment to leverage resources to process big tasks of IoT applications, and by dividing application tasks on a number of edge devices executing them in parallel to fulfill latency requirements.

Moving away from the cloud towards edge computing, and by pushing computations at edge of the network to execute tasks distributed, poses new challenges to meet IoT application requirements (e.g. availability and reliability). In the edge computing environment where the resources of edge nodes are limited in terms of energy, computation and storage of edge devices along with their mobility property make them make them more susceptible to failures than other systems. Hence, special care is required to ensure reliability of the task running in heterogeneous distributed edge network. It is of great importance to design a fault tolerant system for distributed edge networks with special focus on increasing the reliability, along with handling dynamic behavior of edge devices.

There exist research work investigating reliability and fault tolerance in edge computing and IoT infrastructure. Research work in [7]–[9], concentrated on providing fault tolerant application design by considering services running on other nodes as a backup, or using concept of reversibility functions to recover from a failed event. Reliability and availability modeling and analysis are important requirements for data storage to ensure robust design and operation. Research work in [10] evaluated reliability of mesh storage area network (SAN), while [11] used data replication techniques to ensure maximum data availability for high node failures in IoT systems. Some research work proposed solutions to address failure of communication links in IoT devices [12], [13]. To best of our knowledge, reliability and fault tolerance of tasks running on individual node in distributed edge network environment has yet been addressed fully.

The above research work provides insights into importance of fault tolerance for edge computing, especially when application is running in distributed mode on unreliable edge devices. Most of these tends to use some already existing technologies like MDFS, Apache Kafka, Kubernetes, etc., to provide fault tolerance, which will add an extra burden on resource limited nodes to configure and manage these external frameworks. Others focused on tackling fault handling for communication and routing of data. Device heterogeneity and resources availability are barely considered by most of studies. Only a few of them provided measures or modeled reliability parameters. Our work at first finds reliability parameters of edge nodes to ensure resources availability, and by providing decentralized fault tolerance methodology, we have tried to enhance QoS for an IoT application.

In this paper, we have contributed by analyzing decentralized methodology for organizing edge nodes in a group to execute the latency-aware and resource intensive IoT application in distributed mode. Moreover, we have presented a decentralized fault tolerance technique to ensue reliability of tasks running in distributed on the group of edge nodes. We have modeled fault tolerance metrics for edge computing such as reliability and Mean Time to Failure (MTTF) to provide necessary redundancy during the design time of our fault tolerance technique.

Our main contributions in this article are:

- We have proposed an algorithm for decentralized grouping of edge nodes to execute a resource intensive task distributed to meet its deadline requirements.
- Presented a mathematical model for fair resource allocation to a task in distributed edge executing environment.
- Reliability parameters are measured for nodes in edge network.
- For error prone heterogeneous edge network, an efficient decentralized fault tolerance methodology is provided.
- Algorithm for backup node selection based on reliability model to provide redundancy.
- We provided thorough evaluation for our proposed techniques to validate their effectiveness.

II. RELATED WORK

With 5G on the horizon and very fast evolving concept of IoT, the connected devices are projected to amount to 75.44 billion worldwide by 2025 [14] and this will be producing immense volume of data. Current IoT solutions tend to use remote cloud infrastructure to process this voluminous data [15], [16], but it will add cost in terms of bandwidth, and at same time the latency will be increased. To address this edge computing was proposed and it received much interest in literature [17]–[19]. With improvements of processing capacity in smart devices such as smart phones, smart TVs, and Raspberry Pi, smart gateways aim to provide services at edge of the network by pushing computations on these edge devices [20]–[22]. This improves responsiveness, increase data security as keeping data more local and hence reducing internet traffic.

A. EDGE NODE GROUPING

Edge nodes are heterogeneous in terms of available resources and geographical location ranging from servers, routers, access points, mobile phones, set top boxes etc. [23]. Another property related to IoT devices is mobility, traffic generated by end devices is very dependent on time and spatially distributed according to the population density and activities. These properties significantly affect the load and potential of edge devices [24] and make IoT applications suffer from a lot of problems including excessive bandwidth utilization, delay, scalability and fault tolerance. To handle these issues edge devices and resources can be operated in a distributed manner without external support like from cloud [25]. This distributed operation of edge devices can leverage the computing capabilities of edge nodes and help to face challenges mentioned above.

It is desirable to form a group of edge devices to execute applications that require large resources and often transcend the capabilities of single edge node. Edge computing can provide elastic resources that allow for distributed data processing and protects the data from the drawbacks of traditional centralized architecture [26]. Research efforts such as Cloudlet [27], Femtoclouds [5] and Fog computing [23] have argued for combining resources of mobile devices. For Cloudlet solutions, when a device finds edge support via Cloudlet, it offloads most of the processing to Cloudlet instead of splitting the task among the available edge devices [3]. The clustering concept in Femtoclouds needed to be managed using specialized controllers in a centralized manner [28]. Fog computing, which provides dedicated computing servers at edge to meet requirements of end users at a specific location, will increase the cost and specialized management will be required to account for changes in user demands [29]. These limitations hinder the qualities of these systems despite their apparent advantages.

Research work in [30] proposed graph-based clustering technique for mobile edge computing (MEC) to consolidate as many communications as possible at the edge. They created MEC clusters by partitioning geographical area to localize communications inside the cluster and reducing load to cloud. Edge Mesh [31], focuses on enabling distributed intelligence in IoT, it distributes the whole application into subtasks which are distributed among edge devices, but it requires that devices have to form a mesh network with every device in edge network. H. Guo *et al.* [25] proposed transparent computing based architecture for scalable and manageable IoT applications. Authors in [6] have proposed clustering approaches in both bottom to top and top to bottom ways depending on the need of an IoT application. All of these research works have used centralized methods for management of different resources like OS, application and data. A decentralized methodology is required for leveraging resources at edge of an IoT network to execute big tasks while satisfying their deadlines.

B. RELIABILITY AND FAULT TOLERANCE

When an IoT application is executing on a group of edge nodes, it is essential for edge computing to be reliable and fault tolerant. Designing an efficient fault tolerant system for edge computing environment is a complex task, mainly because of the extremely large variety of edge devices, networks and data computing methodologies [35]. Technologies based on container-based virtualization have been used by [8] for fault-tolerance systems. Javed *et al.* [33] explored fault tolerance and automated recovery in a small edge cluster by using Containers, Kubernetes, and Apache Kafka. In [36] a distributed fault tolerant system for dynamic IoT environment is proposed, where each device maintains consistent view of duplicated services. In combination with heartbeat protocol, recovery from failure can be achieved within few seconds without any external intervention.

In current shift towards massive use of smart devices, emergent needs for IoT applications have further increased importance of fault tolerance for nodes in edge computing. Authors in [7] proposed an approach for designing applications capable of surviving in the high-stress environment of edge computing paradigm. They have introduced concept of

TABLE 1. Comparison of fault tolerance research work.

Paper/Feature	Considering available resources	Approach	Considering device heterogeneity	Fault tolerance for	Reliability measurements	Unsolved problems
DRAW [11]	Yes	Decentralized	No	Data	No	A large quantity of unwanted replicas might exist
Fault tolerant smart city [9]	No	Decentralized	Yes	Service	No	Consider only single replica for each service
k-Out-of-n [34]	No	Centralized	No	Data and process	Yes	To perform recovery, n number of nodes must be working
DFT based Mesh SAN [10]	No	Centralized	No	Communication	Partially for one technique	Provides only link reliability
CEFIoT [35]	No	Client-server	No	Data, process and communication	No	Use external cloud frameworks to handle fault tolerance for each service
Crystal [36]	Yes	Client-server	Yes	Process and communication	No	Use external tools for reliability, application development requires to follow specific framework
Our work	Yes	Decentralized	Yes	Process and data	Node level and application level measures	Extra work required for network fault tolerance

phase and used the 'reversibility' in context of application design to tolerate node failure. The study in [34] focuses on providing fault tolerance by providing abstraction of crystal for fog application development. Crystals allow application developers to easily build a sustainable fog application by using crystal instances as a building block. Hence, it can support location transparency, mobility and fault tolerant distributed processing over heterogeneous fog nodes. Research work in [9] discusses the issue of reliability and fault tolerance for fog platforms to support smart city applications. Fault tolerance is performed using fog services, when a fog node fails the services running on this node it is replaced by similar service available on another healthy fog node located close to failed node. The data replication technique has been used by [11] to ensure maximum data availability under high node failures in IoT based systems. They address the benefits of data replication in a homogeneous wireless sensor networks (WSN) for an IoT-based system.

In particular, failure in edge network might occur at any time, and unlike traditional distributed computing environments, these failures might lead to a significant portion of edge side resources going offline, resulting in delay and effecting the overall QoS. Gia *et al.* [13] propose an architecture supporting fault tolerance and scalability for health care; their approach covers malfunction of sink node hardware and traffic bottlenecks. Authors in [37] have proposed fault tolerance to handle failures for routing paths in IoT environment. They have defined neighborhood relationships among nodes to exchange information with each other to find optimal configurations in heterogeneous IoT environment.

Chen *et al.* [32] proposed a fault-tolerant and energy-efficient data allocation and task scheduling algorithm for mobile devices. They evaluated communication costs for joining and leaving a network of mobile devices due to failures or mobility of devices. Node failure in the IoT environment might lead to data loss, and user might no longer have access to valuable information stored on that node. Research work in [10] modeled failure behavior of mesh storage area network (SAN) for IoT to provide fault tolerance through redundancy to decrease system down time. They have evaluated reliability of SAN using binary decision diagram based method. Authors in [38] applied check-pointing technology to realize failure recovery on large-scale IoT applications, but check-pointing puts load of extra computations even when system is working smoothly. The above research works related to fault tolerance and reliability, have concentrated either on data or communication links in IoT networks. The reliability and fault tolerance of nodes in edge computing is not yet adequately investigated. In this work, we have find reliability parameters of edge nodes to ensure resources availability, and by providing decentralized fault tolerance methodology, we have made efforts to enhance QoS for an IoT application.

As discussed above, various approaches have been investigated and proposed to handle fault tolerance in IoT systems in general and specific for edge computing. The approaches usually focus on using centralized methodologies to manage fault tolerance which is not suitable for edge computing environment. The research works are also limited to provide fault tolerance for data or communication, and mostly are unable to provide measurements for reliability of the fault tolerant system for edge computing. The analysis and comparison over different methods is clarified in detail with Table 1.

III. METHODOLOGY

In this section, we will discuss how an IoT application (now onwards will be referred as a task) is managed by using a number of subtasks at first. Then we will present our methodology for decentralized edge node grouping. Latterly reliability parameters are discussed along with the algorithm for providing fault tolerance.

TABLE 2. Symbols and their descriptions used in this paper.

Symbol	Description
Task management	
ω_k	task (refers to an IoT application)
i_k	i^{th} edge node where task is received
μ_k	workload of task
μ_{cur}	currently assigned workload at time t
μ_{need}	workload still to be assigned at time t
d_k	size of data associated with task
t_k	deadline of task
t_{δ_i}	CPU time required to execute a subtask on edge node e
δ_m	a subtask of ω_k
c_i	Number of instructions to complete a sub task
s_e	speed of edge node
γ	number of edge nodes on which subtasks($\delta_m \in \omega_k$) are running
cur_{ω_k}	current assigned resources at time t
$\Re cur_{\omega_k}$	cumulative resources assigned to ω_k at time t
D_{ω_k}	current demand of resources by ω_k at time t
S_{ω_k}	total resource share of ω_k
Group formation and Reliability	
e	edge node
e_q	neighbor of edge node
e_b	backup edge node
$G_i^{\mathrm{e}}(\omega)$	i^{th} group of edge nodes to execute ω
E^{org}	organizer node
U	edge network
g	edge node that belongs to group
$\Re_\omega$	total resources required by ω
$\Re_{e_i}$	Resources available at i^{th} edge node
τ	Total execution time of a subtask δ_i
τ_ω	time from start of ω to its completion
λ_g	failure rate of edge node
r	probability of number of failures in a time t
η	number of stand by backups
η_{req}	number of required backups
R_{req}	required reliability level
$R_{g_i}^{\eta}(\tau)$	reliability for group member $g_i \in G_i^{\mathrm{e}}$ with η number of backups
ξ_i	event that node i of group does not fail during τ
ρ	weight assigned to different resources of a node
S_i	denotes available capacity of resource at edge node i

A. TASK MANAGEMENT

To process a task ω_k on the group of edge nodes, properties related to the task are $\omega_k = \{\mathrm{i_k}, \mu_\mathrm{k}, \mathrm{d_k}, \mathrm{t_k}\}$ where $\mathrm{i_k}$ denotes i^{th} edge node on which processing request is received, μ_k is workload of the task, $\mathrm{d_k}$ is size of data associated with task and $\mathrm{t_k}$ is deadline of ω_k. The task is divided into m number of independent subtasks (denoted as δ_i) $\omega_k = \{\delta_1, \delta_2, \ldots \delta_\mathrm{m}\}$, with no constraints on them (such that they can be executed independently of each other). These subtasks can be executed in parallel on different edge nodes without concerning the order of execution. The variables and their descriptions are shown in Table 2.

To execute a task distributed on edge nodes, a fair execution means that all the required resources are provided well in time ($\mathrm{t_k}$) to get the job done before its deadline. In this paper, we are considering CPU time required by each subtask as a resource. For any task ω_k that consists of m number of subtasks, let c_i be million of instructions (MI) required to complete a single subtask. The workload of a subtask δ_i will be:

$$\mu_{\delta_i} = \sum_{i=1}^{n} c_i. \tag{1}$$

The CPU time (t_{δ_i}) required to execute a subtask δ_i on the edge node e depends on its speed (s_e) as MIPS (millions of instructions per second). That is,

$$t_{\delta_i} = \frac{\mu_{\delta_i}}{s_e}. \tag{2}$$

For any task ω_k which consists of m number of subtasks running as distributed on γ number of edge nodes, let cur_{ω_k} *(t)* denote currently assigned resources (CPU time) at time t. Let $\Re cur_{\omega_k}(t)$ denote cumulative resources for k^{th} task at time t. Thus

$$\Re cur_{\omega_k}(t) = \int_0^t cur_{\omega_k}(t)dt. \tag{3}$$

The total workload (μ_k) for the task ω_k is equal to sum of workload of m subtasks, and μ_{cur} is the workload for tasks assigned to γ nodes until time t. The required workload will be:

$$\mu_{need} = \mu_k - \mu_{cur}. \tag{4}$$

Let $D_{\omega_k}(t)$ denote the current demand of resources at time t (the demand depends on μ_{need}). Also, let $S_{\omega_k}(t)$ denote total resource share for the k^{th} task at time t. The fairness degree $fd_{\omega_k}(t)$ of k^{th} task at time t is defined as normalization result of the amount of resources a task ω_k utilizing in distributed execution environment of edge network with respect to cloud execution.

$$fd_{\omega_k}(t) = \frac{\int_0^t cur_{\omega_k}(t)dt}{\int_0^t max\{D_{\omega_k}(t), S_{\omega_k}(t)\}dt} \tag{5}$$

The fairness degree $fd_{\omega_k}(t) = 1$ shows the absolute resources fairness for k^{th} task at time t, and all the needed resources are allocated to complete the task efficiently. In contrast $fd_{\omega_k}(t) < 1$ indicates unfair. It is obvious that for k^{th} task, if it is executed on cloud all the needed resources are provided as soon as task is assigned to cloud (as cloud is rich in term of resources), hence, $fd_{\omega_k}(t) = 1$ because it has cur_{ω_k} (t)= $max\{D_{\omega_k}(t), S_{\omega_k}(t)\}$ at any time.

B. DECENTRALIZED EDGE NODE GROUPING

To handle a big task efficiently on edge network we have proposed to execute it in a distributed mode on the federation of edge nodes $G_i^{\mathrm{e}}(\omega)$, where i is the ith group and e is edge nodes in the group and ω is task to be processed. We have specified an edge node as organizer E^{org}, which has a task to perform or which receives request from a user as shown in Fig. 2. This organizer node will collaborate with other edge nodes to complete assigned task and send results to the user. Our algorithm works in decentralized way because this organizer will only serve as initiator of making federation of edge nodes and then each edge node decides by itself to execute a subtask based on its local information about available resources. Once a node decides to execute a subtask it also becomes a part of federation of edge nodes collaborating to complete a task distributed. The E^{org} can execute some subtask by itself, while it can not compute a solution to decide either a node will be assigned a task or not.

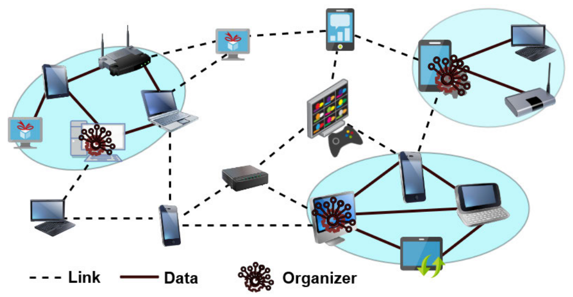


FIGURE 2. Decentralized grouping of edge nodes to perform a task.

When a task ω arrives at E^{org} to execute in such a distributed fashion, the organizer node will announce in the edge network U to form a group $G_i^e(\omega)$. The nodes in the group will collaboratively execute subtasks of $\omega = \{\delta_1, \delta_2, \ldots \delta_m\}$, and will send results back to E^{org}. Each edge node knows about how much of its resource ($\Re_{ei}$) it can collaborate to execute the task. We assume that when a subtask is received by an edge node, it will completely execute it.

Our algorithm for group formation and task distribution is shown in Algorithm 1. Edge node E^{org} will send a request for group formation, and edge nodes in the locality replying to this request are added to a queue θ (lines 3 to 7). If the nodes in the queue have some resources then they will take a subtask from organizer node, meanwhile this node is added to the group that is executing ω collaboratively (lines 9 to 15). More nodes are added to the queue while considering only the neighbor nodes of the nodes present in the group (line 18 to 21). This process continues until the required resources by task are fulfilled.

Algorithm 1 Group Formation and Distributed Processing

```
Require: Organizer node $E^{org}$; $G(\omega) = \{\ \}$; Set of edge
    nodes in the network U; Empty queue $\theta$; Resources of
    the task $\Re_\omega$; Resources of edge node $\Re_{ei}$
Ensure: Group $G_i^e(\omega)$ of edge nodes to execute task $\omega$
 1: for each $E^{org} \in U$ do
 2:     $\theta \leftarrow$ E$^{org}$
 3:     E$^{org}$ will send join request to all edge nodes (u∈U)
 4:     nodes (ú∈U)in locality reply to $E^{org}$
 5:     for each ú∈U do
 6:         $\theta \leftarrow$ ú
 7:     end for
 8:     while $fd_{\omega_k} \neq 1$ do
 9:         if $\theta \neq \phi$ then
10:             for each $e_i \in \theta$ do
11:                 if $\Re_\omega \cap \Re_{ei} \neq \phi$ then
12:                     $e_i$ pull $\delta_i \in \omega$
13:                     $\Re_\omega = \Re_\omega - \Re_{ei}$
14:                     $G(\omega) \leftarrow e_i$
15:                 end if
16:             end for
17:         else
18:             for each $e_i \in G(\omega)$ do
19:                 if (neighbor $e_q \in e_i) \cap G(\omega) == \phi$ then
20:                     $e_i$ will send $RReq = (E^{org}, \Re_\omega)$ to $e_q$
21:                     $\theta \leftarrow e_q$
22:                 end if
23:             end for
24:         end if
25:     end while
26:     $G_i^e(\omega) \leftarrow G(\omega)$
27: end for
```

C. RELIABILITY PARAMETERS

When an application is running in error-prone edge network both data and task reliability are critically important. To ensure reliability of edge node, we have considered MTTF (mean time to failure) as a metric for node failure rate and we have measured reliability for group $G_i^e(\omega)$, which is executing the distributed task collaboratively. The reliability of group is defined as probability to complete the task ω successfully. First we will consider the case of edge node failure while working on the reliability of $G_i^e(\omega)$. The node failure in edge network is independent and identically distributed, hence failure process of edge nodes can be modeled using Poisson's process.

As discussed in Section III-A the task ω is divided into m number of subtasks, to complete a subtask $\delta_i \in \omega$ on an edge node $g \in G_i^e(\omega)$, let τ denote time duration starting from subtask δ_i assignment to result delivery. In addition, let $\tau_{\delta_i}^T$ denote data transfer time from organizer to edge node g, and $\tau_{\delta_i}^E$ is execution time and $\tau_{\delta_i}^R$ is transfer time for the results. Then time duration of the subtask τ on g becomes:

$$\tau = \tau_{\delta_i}^T + \tau_{\delta_i}^E + \tau_{\delta_i}^R \tag{6}$$

If λ_g is the failure rate of edge node then according to Poisson's process, probability of r failures in time τ will be:

$$p_r(\tau) = \frac{(\lambda_g \tau)^r e^{-\lambda_g \tau}}{r!} \tag{7}$$

In this paper, we are using standby backup for subtask δ_i, which we name as replication factor (RF). The durability of δ_i depends on different replication schemes, like RF2 (replication factor 2) or RF3. Replication scheme RF1 can tolerate single node failure (as it works with one standby backup) while RF2 can tolerate two node failures.

In case of RF0 (there is no backup node for δ_i) we say g is reliable if there happens to be no failure during τ shown as following:

$$R_{g_i}^0(\tau) = p_0(\tau) = e^{-\lambda_g \tau} \tag{8}$$

The mean time to failure (MTTF) of g can be defined as:

$$MTTF_g = \int_0^\infty R_{g_i}^0(t)dt = \frac{1}{\lambda_g} \tag{9}$$

this shows that the lower the failure rate, the higher MTTF will be, and system reliability will be increased. For RF1 reliability of g will be equal to sum of probability (no node fails during τ) along with probability (one node fails during τ), because we are using standby mode.

$$R^1_{g_i}(\tau) = p_0(\tau) + p_1(\tau) = e^{-\lambda_g \tau}(1 + \lambda_g \tau) \tag{10}$$

The respective MTTF will be:

$$MTTF_g = \int_0^\infty R^1_{g_i}(t)dt = \frac{1+1}{\lambda_g} \tag{11}$$

for a task with η number of stand by backups (replication scheme RFη) reliability will become:

$$R^\eta_{g_i}(\tau) = \sum_{r=0}^{\eta} \frac{(\lambda_g \tau)^r e^{-\lambda_g \tau}}{r!} \tag{12}$$

The relative MTTF will be

$$MTTF_g = \int_0^\infty R^\eta_{g_i}(t)dt = \frac{\eta+1}{\lambda_g} \tag{13}$$

Our aim is to increase the overall reliability of subtask during time τ, and according to Eq. 13 if we increase RF it will increase reliability of δ_i. So, for time τ the required reliability level R_{req} can be achieved if $MTTF \geq \tau$, and it actually depends on η number of standby backup nodes. The backup node selection problem, therefore, becomes the task of selecting η_{req} number of nodes on which to put backup of δ_i, so that R_{req} is reached. Hence for time τ we have to ensure following:

$$R^\eta_{g_i}(\tau) \geq R_{req} \tag{14}$$

To ensure reliability of task running on $G^e_i(\omega)$ for time τ_ω (time from the start of ω to its completion), we have to ensure that there will be no process or data loss. This can be calculated by finding probability of no node failure in the group for τ_ω, because subtasks of ω are running on g number of edge nodes in parallel, if any node $g \in G^e_i(\omega)$ for time τ_ω fails the task will be failed. As node failure in edge network happens following Poisson process, we can estimate the probability that a node fails during t_k given that node failure rate is λ_g.

$$p(\tau_\omega) = (\lambda_g \tau_\omega)e^{-\lambda_g \tau_\omega} \tag{15}$$

For n node in the group, the probability that no node fails during τ_ω is $1 - p(\tau_\omega)$, this means first node in $G^e_i(\omega)$ will not fail and second node in $G^e_i(\omega)$ will not fail and so on, and n^{th} will not fail. Let us name this event as ξ_i that node i does not fail during,

$$\xi_i = 1 - p(\tau_\omega) \tag{16}$$

The cumulative probability that no node fails in the group during τ_ω is

$$\begin{aligned} P(\xi_1, \xi_2, \ldots..\xi_n) &= P(\xi_1)P(\xi_2)\ldots.P(\xi_n) \\ &= \prod_{i=1}^{n}(1 - p(\tau_\omega)) \end{aligned} \tag{17}$$

So the reliability of the group will become

$$R_G(\tau_\omega) = 1 - \prod_{i=1}^{n}(1 - p(\tau_\omega)) \tag{18}$$

TABLE 3. State of neighbor node.

State	Current Status
Candidate	q has enough resources to execute the task
Serving	q is already part of same group as that of g
Exhausted	q has no more resources to serve a task

D. FAULT TOLERANCE

Considering resources limited edge devices, a decentralized local fault tolerance technique is required to enhance task execution at edge. Our method allows edge devices to operate independently, without having global knowledge of the overall edge network.

At first, all edge nodes have to keep knowledge about their available resources locally. Resources calculation at each edge node can be performed as following

$$R_{ni} = \sum_{i=1}^{n}(\rho \times AvailableCapacity(S_i)) \tag{19}$$

where $S_i \in \{CPU\%, RAM\%, Bandwidth\%\}$ denotes the amount of respective remaining resource, and ρ indicates the weights assigned to each resources. For this work, we have assigned weight value of 40 to CPU, and 30 for RAM and Bandwidth each. We have assigned these values via considering importance level of each factor for an IoT application scenario. Similar values are also suggested by previous studies [39].

For each edge node in group $G^e_i(\omega)$, we want to keep a subset of selected edge nodes from its neighbors. To provide robustness this subset should contain edge nodes that can complete the relevant application tasks in case of node failure. A suitable replacement is one that has as many resources to offer low latency execution as possible. While selecting a node q as a neighbor node the edge node g has to be careful about its state. Table 3 shows state of each node.

- Neighbors in the Candidate state have high priority to become a backup node, because these nodes have some free resources.
- Neighbors in Serving state are active nodes providing resources to serve similar tasks as that of g, which is serving (being both are part of the same group $G^e_i(\omega)$). These nodes are given less priority to become a backup node.
- Neighbors in Exhausted state usually have exhausted their resources. These nodes will not be used as backup nodes.

Each node g in the edge network keeps a small neighbor set Q. The neighbors will periodically exchange information related to its local resources with each other to remains up-to-date. Meanwhile, this periodic exchange will also help to identify node failure. To ensure required reliability level according to Eq. 14, we have to place backup on η_{req} edge nodes. The node selection and backup placement is given in the Algorithm 2.

Algorithm 2 Candidate Node Selection and Backup Placement for Reliability

```
Require: Group of nodes $G_i^e(\omega)$ executing a task; Members of group g; Knowledge about resources $R_{e_q}$ of each neighbor; State of node; Exhausted nodes
Ensure: Reliability for node g
1:  for each $g \in G_i^e(\omega)$ do
2:      list ← neighbors of g
3:      available_nodes ← list - Exhausted
4:      sort available_nodes according to $R_{e_q}$
5:      for each $e \in available\_nodes$ do
6:          if $e \notin G_i^e(\omega)$ then
7:              backup_node ← $e$
8:              available_nodes = available_nodes − $e$
9:          end if
10:     end for
11:     for each $e \in available\_nodes$ do
12:         backup_node ← $e$ ($\because$ insert $e$ at end of list)
13:     end for
14:     while |backup_store| < $\eta_{req}$ do
15:         $e$ ← backup_node.pop
16:         replica ← process(g) & data(g)
17:         copy (replica, $e$)
18:         backup_store ← $e$
19:     end while
20: end for
```

Our fault tolerance methodology creates backup nodes for each node that belongs to the group, without relying on any central entity (based on information received by neighbor nodes). Firstly, we have created a list of neighbor nodes of g, which does not include any exhausted node. Afterward we have sorted this list according to available resources at each node. Secondly, we have prioritized candidate nodes present in the list over serving nodes. After the list for backup nodes is created, we have copied process and data related to subtask δ_i, which executing on g, to the backup nodes. The number of replicas depends on η_{req}, as discussed previously.

For recovery, firstly, each node in the group will keep track of tasks assigned to its direct neighbors along with a list of backup nodes, as well as information about organizer node. This information is shared periodically between neighbors. When a group member node g_i does not receive information from one of its neighbor e_q, the neighbor is considered as dead. After fault detection node g_i will look in the list of backup nodes (previously shared by e_q), and select first node from the list as backup node e_b and send recovery message to this node. As the backup node has already been assigned task δ_i using Algorithm 2, e_b completes the task and sends results back to G^{org}.

IV. SIMULATION AND RESULTS

To evaluate our proposed algorithms for decentralized edge node grouping and fault tolerance, we have simulated a smart battlefield scenario game, where soldiers wear sensors to collect information and receive instructions. Sensors worn by soldiers include bio-sensors to monitor vital signs, sensors to monitor battlefield information and sensors for position-related information, as shown in Fig. 3. Sensors collect real-time data of battlefield, and it is required to process this data immediately to react properly in a war scenario. The overall job processing for a soldier has different task types, like moving object detection, sound detection, target detection, and monitoring as well range detection to hit target, weapon selection based on target type, information about other team members. All tasks are required to be executed without any latency, so edge computing is the best option. Also single edge node might not be able to process the entire job of a solider well in time, hence, decentralized computing could help to distribute tasks to nearby edge nodes and get the results quickly.

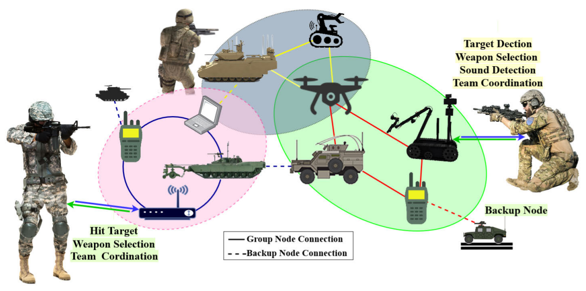


FIGURE 3. Simulation of battlefield scenario with edge group computing and fault tolerance.

We carried out our simulation using iFogSim [40] as a simulation tool, which runs on top of CloudSim [41] simulator. This simulator allows running application scenarios using edge devices with different configurations and enables to measure different application related statistics, as well as device and network-related measures.

We have considered this game scenario as generalizable case to test our proposed methodologies of decentralized distributed task execution on a group of edge nodes, and handling fault tolerance for edge nodes. Because in battlefield there is a great need to handle fault tolerance. The simulation scenario has been repeated with different numbers of players acting as soldiers, with each player having a different number of subtasks to process. To evaluate our proposed methodologies this game scenario suits best, because the overall job of a player can be easily divided into number of independent subtasks, and each edge node can decide at its own either to process a subtask or not. This will lead to decentralization, hence allow evaluating our decentralized edge node grouping and faulting tolerance effectively. Each player is connected with a random node in the edge network, which will also act as organizer node for that player. Different edge devices will collaborate with organizer nodes based on their available resources to execute different tasks in distributed environment. Processing requirements for subtasks (δ_i) are given in Table 4, and task allocation for each player is shown in Fig. 4.

TABLE 4. Processing requirements of subtasks.

Subtask (δ_i)	CPU needed (c_i)
Hit target	850
Weapon selection	650
Range calculation	830
Sound detection	700
Target detection	975
Distance from team members	580
Team coordination	500

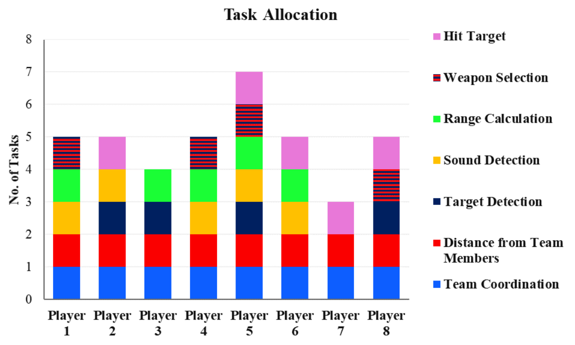


FIGURE 4. Task allocation map for each player.

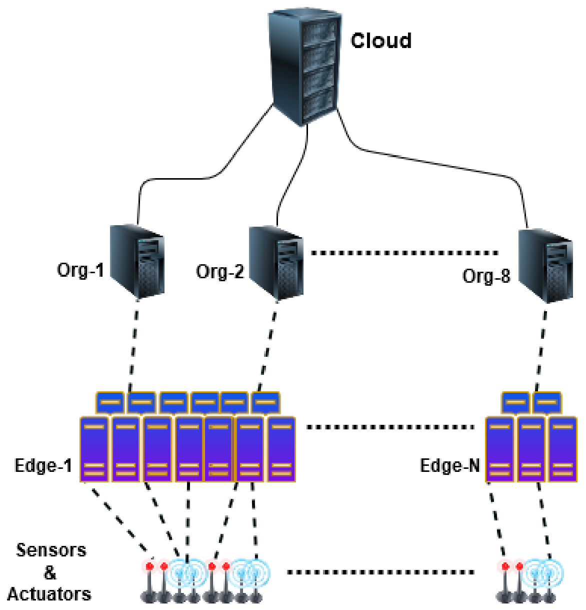


FIGURE 5. iFogSim network topology with cloud, organizer and edge nodes.

Fig. 5 illustrates the iFogSim network topology. For our simulation, a powerful node is configured as cloud, and there exists an organizer node associated with each player. In our work, we are testing for eight different players hence there are 8 organizer nodes. A number of edge nodes are also placed, both organizer nodes and edge nodes have different processing capacities, which are randomly assigned to each node. Sensors and actuators are also attached to edge nodes. To ensure diversity of simulation, parameters have been varied as the summary in Table 5.

TABLE 5. Simulation environment setup.

Number of nodes	Cloud	1
	Organizers nodes	8
	Organizers nodes	72
Cloud	CPU (MIPS)	40000
	RAM (MB)	40960
	Upstream link (Mbps)	1000
	Downstream link (Mbps)	10000
	Delay (ms) (between cloud and organizer node)	100
Organizer and edge nodes	CPU (MIPS)	500 to 2000
	RAM (MB)	2048
	Upstream link (Mbps)	10000
	Downstream link (Mbps)	10000
	Delay (ms) (between ogaranizer and edge node)	2
Sensors and actuators	Upstream link (Mbps)	10000
	Downstream link (Mbps)	10000
	Delay (ms) (between edge and sensor & actuators)	1

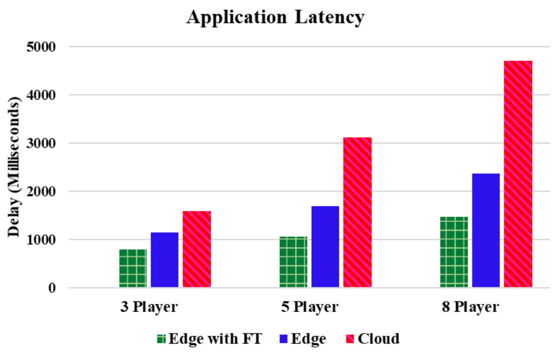


FIGURE 6. Application latency comparison.

For task allocation, we have considered CPU requirements by each task. Edge node from the network fulfilling CPU requirements will be assigned the specific subtask of a player, and nodes are added to the group as well. Each node will also select its backup node by following our fault tolerance methodology. We have compared our approaches for executing tasks assigned to a player, namely Edge with FT (execution on group of edge nodes with fault tolerance), Edge (execution on group of edge nodes without fault tolerance), and traditional approach of cloud placement (Cloud). We have tested for different numbers of players, and each player has been assigned with a different number of subtasks. We have chosen application latency, deadline missing ratio, and network usage as metrics for evaluation.

Fig. 6 illustrates results related to application latency for both of our approaches and cloud placement. The latency is very high when application is executed on cloud, because all of the data required to be transferred to cloud for processing, while executions on group of edge nodes result in lower latencies. Specifically latency is less when fault tolerance is active. That is because in simple grouping approach, when an edge node fails the allocated task its data will be transferred to cloud for further processing.

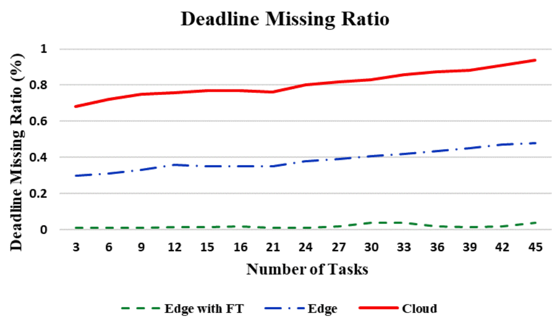


FIGURE 7. Deadline missing ratio.

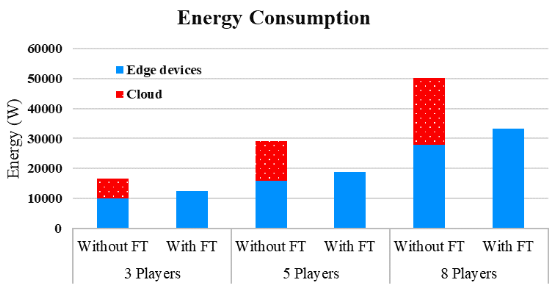


FIGURE 9. Energy consumption in proposed edge grouping with FT and without FT.

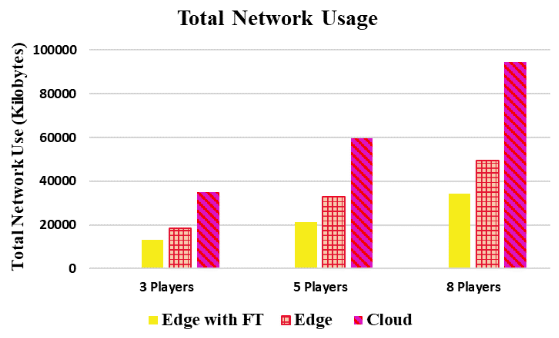


FIGURE 8. Decrease in network usage when application executed on group of edge nodes.

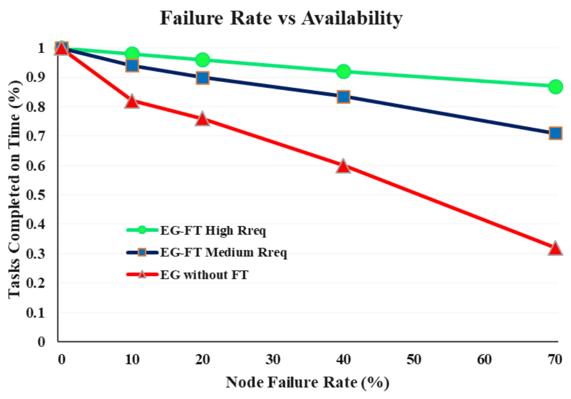


FIGURE 10. Effect of increasing node failure rate on application availability in terms of tasks completed on time, with high level of R_{req} (required reliability) and medium level of R_{req}.

To test for reliability with our proposed fault tolerance approach, we have tested deadline missing ratios for subtasks. Fig. 7 shows that our approach helps to reduce deadline missing ratio when compared with cloud execution approach as well as with edge side execution without fault tolerance. It is clear that, when fault tolerance is active, our approach is much better, with much smaller ratio of the number of tasks missing their deadlines. As a result, it will increase application reliability and availability. Execution at cloud shows poor performance because of higher latencies, resulting in most of application tasks missing their deadlines.

Our proposed methodologies also have impact on the total network use as shown by Fig. 8, when applications are executed in a decentralized way on group of edge nodes with or without fault tolerance, total amount of data transmitted in the network will be reduced if compared to cloud-based execution. Edge side execution results in saving the bandwidth especially when fault tolerance is active. This is because, even if edge node fails, the task is executed on a node present in edge network (backup node). If there is no fault tolerance, the failed tasks are completed on cloud as no backup is present, which will lead to increase in network traffic.

Next, we have compared the energy consumption of executing applications using edge grouping methodology without fault tolerance and with fault tolerance. It is clear from Fig. 9, that energy consumed with fault tolerance approach is less than that without fault tolerance approach. For without fault tolerance approach, when a node fails the task running on that node is executed in cloud, which results in additional energy consumed by cloud. When fault tolerance is active, all tasks are executed in edge network. And even in the case of node failure, the backup edge node will complete the task.

Fig. 10 shows results for the effect of increasing node failure rate on application availability for edge node grouping with fault tolerance activated (for high-reliability level and medium level) and without fault tolerance. The application availability is measured as the number of tasks completed on time. It can be observed that, as the node failure increases, application availability is reduced especially when fault tolerance is not present in edge computing. However, application availability is increased for high value of R_{req}, as it will increase replication factor (RF). This will help to achieve better results in fault handling.

Fig. 11 shows a number of replicas created as backup nodes with different reliability levels of R_{req}. When reliability level is medium and node failure rate is low, our methodology adjusts to create fewer backups to save resources. But when failure rate is increased, high replication factor is achieved to fulfill application reliability requirements. As for higher failure rate, a high value of RF will ensure that, the system can still find a backup inside edge network, hence helping to complete a task within its deadline. However, when there are more replicas, it will consume more resources, where the

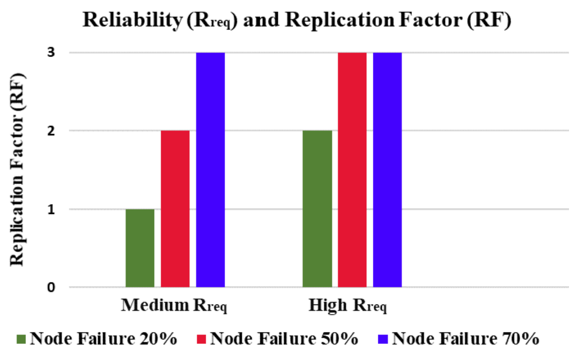


FIGURE 11. Replication factor (RF) achieved for different reliability levels (R_{req}) in the system against node failure rates of 20%, 50%, and 70%.

system has to trade-off between reliability level and resource utilization.

From the above results, we demonstrated that group formation of edge nodes helped to execute applications well in time with decrease in deadline missing cases, as well as decrease in overall data transferred over the network. For the results, the battlefield simulation scenario also helped to analyze different features related to edge computing like deadline missing ratios for subtasks, and regarding the fault tolerance, analysis of the failure rate and availability along with replication factor analysis. Results shown in figures related to these parameters made it clear that results obtained using this type of simulation scenario are very convincing, especially a staggering decrease in network traffic is observed with overall increase in reliability. The decentralized group formation can be performed on task arrival, and tasks can be executed on edge devices with relative available resources. This makes clear that we need not worry about clustering mechanism of edge nodes in advance, in order to run latency oriented IoT applications in distributed edge network.

V. CONCLUSION

In this paper, we have proposed a decentralized technique to leverage edge nodes to execute a resource intensive task inside the edge network to reduce the latency and by providing fault tolerance ensured availability in the error prone edge network. Compared with conventional cluster-based approaches to group different devices for distributed execution, the proposed approach has two desired features. Firstly, no global information is required to consider cluster head selection, whereas the edge node grouping starts as soon as task arrives at organizer node, which collaborates with other edge nodes to execute subtasks. With more devices added to the group, parallel execution of subtasks helps to achieve task deadline. Secondly, big tasks of IoT applications can be executed successfully within edge network, hence reducing the overall network traffic.

The distributed execution comes with the challenge of fault tolerance, especially when executing machines are mobile devices. We have presented a decentralized fault handling methodology, with particular focus on increasing reliability along with handling heterogeneity of edge devices. Simulation results show that our approaches are effective, and our decentralized handling of edge resources for task executions along with providing fault tolerance has the best overall performance. Compared with traditional cloud-based method for executing big tasks, our method achieves significant improvement in reducing deadline missing ratio, decreasing overall network traffic, as well as lowering latency.

REFERENCES

[1] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.
[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st ed Workshop Mobile Cloud Comput. (MCC)*, 2012, pp. 13–16.
[3] H. Flores, P. Hui, P. Nurmi, E. Lagerspetz, S. Tarkoma, J. Manner, V. Kostakos, Y. Li, and X. Su, "Evidence-aware mobile computational offloading," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1834–1850, Aug. 2018.
[4] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: From concept to practice and beyond," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, Mar. 2015.
[5] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 9–16.
[6] J. S. Kumar and M. A. Zaveri, "Clustering approaches for pragmatic two-layer IoT architecture," *Wireless Commun. Mobile Comput.*, vol. 2018, Apr. 2018, Art. no. 8739203.
[7] R. Paul, J. Melchior, P. V. Roy, and V. Vlassov, "Designing distributed applications using a phase-aware, reversible system," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 55–64.
[8] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. Lightweight virtualization: A performance comparison," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2015, pp. 386–393.
[9] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Towards fault tolerant fog computing for IoT-based smart city applications," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2019, pp. 752–757.
[10] L. Xing, M. Tannous, V. M. Vokkarane, H. Wang, and J. Guo, "Reliability modeling of mesh storage area networks for Internet of Things," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2047–2057, Dec. 2017.
[11] W. B. Qaim and O. Ozkasap, "DRAW: Data replication for enhanced data availability in IoT-based sensor systems," in *Proc. IEEE 16th Intl Conf Dependable, Autonomic Secure Comput., 16th Intl Conf Pervas. Intell. Comput., 4th Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 770–775.
[12] O. Kaiwartya, A. H. Abdullah, Y. Cao, J. Lloret, S. Kumar, R. R. Shah, M. Prasad, and S. Prakash, "Virtualization in wireless sensor networks: Fault tolerant embedding for Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 571–580, Apr. 2018.
[13] T. N. Gia, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fault tolerant and scalable IoT-based architecture for health monitoring," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, Apr. 2015, pp. 1–6.
[14] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester, "City of things: An integrated and multi-technology testbed for IoT smart city experiments," in *Proc. IEEE Int. Smart Cities Conf. (ISC)*, Sep. 2016, pp. 1–8.
[15] H.-L. Truong and S. Dustdar, "Principles for engineering IoT cloud systems," *IEEE Cloud Comput.*, vol. 2, no. 2, pp. 68–76, Mar. 2015.
[16] L. Wang and R. Ranjan, "Processing distributed Internet of Things data in clouds," *IEEE Cloud Comput.*, vol. 2, no. 1, pp. 76–80, Jan. 2015.
[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[18] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[19] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 1, pp. 108–119, Jan. 2017.

[20] X. Su, P. Li, J. Riekki, X. Liu, J. Kiljander, J.-P. Soininen, C. Prehofer, H. Flores, and Y. Li, "Distribution of semantic reasoning on the edge of Internet of Things," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, Mar. 2018, pp. 1–9.

[21] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the Internet of Things," *IEEE Pervas. Comput.*, vol. 14, no. 2, pp. 24–31, Jun. 2015.

[22] W. Zhang, H. Flores, and P. Hui, "Towards collaborative multi-device computing," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2018, pp. 22–27.

[23] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, Nov. 2015, pp. 73–78.

[24] Z. A. Qazi, P. K. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. R. Das, "KLEIN: A minimally disruptive design for an elastic cellular core," in *Proc. Symp. SDN Res. (SOSR)*, 2016. p. 2.

[25] H. Guo, J. Ren, D. Zhang, Y. Zhang, and J. Hu, "A scalable and manageable IoT architecture based on transparent computing," *J. Parallel Distrib. Comput.*, vol. 118, pp. 5–13, Aug. 2018.

[26] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C.-T. Lin, "Edge of things: The big picture on the integration of edge, IoT and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2018.

[27] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervas. Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[28] K. Habak, E. W. Zegura, M. Ammar, and K. A. Harras, "Workload management for dynamic mobile device clusters in edge femtoclouds," in *Proc. 2nd ACM/IEEE Symp. Edge Comput. (SEC)*, 2017, p. 6.

[29] H. Gedawy, K. Habak, K. Harras, and M. Hamdi, "An energy-aware IoT femtocloud system," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2018, pp. 58–65.

[30] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 2, pp. 787–796, Jun. 2018.

[31] Y. Sahni, J. Cao, S. Zhang, and L. Yang, "Edge mesh: A new paradigm to enable distributed intelligence in Internet of Things," *IEEE Access*, vol. 5, pp. 16441–16458, 2017.

[32] C.-A. Chen, M. Won, R. Stoleru, and G. G. Xie, "Energy-efficient fault-tolerant data storage and processing in mobile cloud," *IEEE Trans. Cloud Comput.*, vol. 3, no. 1, pp. 28–41, Jan. 2015.

[33] A. Javed, K. Heljanko, A. Buda, and K. Framling, "CEFIoT: A fault-tolerant IoT architecture for edge and cloud," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Feb. 2018, pp. 813–818.

[34] T. Jeong, J. Chung, J. W.-K. Hong, and S. Ha, "Towards a distributed computing framework for fog," in *Proc. IEEE Fog World Congr. (FWC)*, Oct. 2017, pp. 1–6.

[35] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.

[36] P. H. Su, C.-S. Shih, J. Y.-J. Hsu, K.-J. Lin, and Y.-C. Wang, "Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, Mar. 2014, pp. 45–50.

[37] M. Z. Hasan and F. Al-Turjman, "Optimizing multipath routing with guaranteed fault tolerance in Internet of Things," *IEEE Sensors J.*, vol. 17, no. 19, pp. 6463–6473, Oct. 2017.

[38] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel, "Fault-recovery and coherence in Internet of Things choreographies," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, Mar. 2014, pp. 532–537.

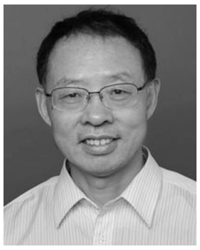
[39] Y. Li, N. T. Anh, A. S. Nooh, K. Ra, and M. Jo, "Dynamic mobile cloudlet clustering for fog computing," in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Jan. 2018, pp. 1–4.

[40] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "IFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw., Pract. Exper.*, vol. 47, no. 9, pp. 1275–1296, Sep. 2017.

[41] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

MUHAMMAD MUDASSAR received the master's degree from the School of Electrical Engineering and Computer Science (SEECS), NUST, Islamabad, Pakistan, in 2011. He is currently pursuing the Ph.D. degree in computer science with the Beijing Institute of Technology, Beijing, China. He worked as a Lecturer with the Department of Computer Science, COMSATS University Islamabad–Vehari, Pakistan. His research interests include distributed computing, big data processing frameworks, fault tolerance, and edge computing.

YANLONG ZHAI (Member, IEEE) received the B.Eng. and Ph.D. degrees in computer science from the Beijing Institute of Technology, Beijing, China, in 2004 and 2010, respectively. He was a Visiting Scholar with the Department of Electrical Engineering and Computer Science, University of California, Irvine. He is currently an Assistant Professor with the School of Computer Science, Beijing Institute of Technology. His research interests include cloud computing, big data, and edge computing.



LEJIAN LIAO (Member, IEEE) received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences. He has served as the Vice Dean of the school. He is currently a Professor with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He has published numerous articles in several areas of computer science. His main research interests include machine learning, natural language processing, and intelligent networks.

JUN SHEN (Senior Member, IEEE) received the Ph.D. degree from Southeast University, China, in 2001. He held positions at the Swinburne University of Technology, Melbourne, and the University of South Australia, Adelaide, before 2006. He is currently an Associate Professor with the School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia, where he has been the Head of Postgraduate Studies, and the Chair of the School Research Committee, since 2014. He has published more than 120 articles in journals and conferences in CS/IT areas. His expertise includes computational intelligence, web services, cloud computing, and learning technologies, including MOOC. He is also a Senior Member of ACM and ACS. He has been the Editor, the PC Chair, a Guest Editor, and a PC Member for numerous journals and conferences published by IEEE, ACM, Elsevier, and Springer. He is also a Current Member of ACM/AIS Task Force on Curriculum MSIS 2016.