A Decentralized Edge Computing Latency-Aware Task Management Method With High Availability for IoT Applications

MUHAMMAD BUKHSH, SAIMA ABDULLAH, AND IMRAN SARWAR BAJWA
Department of Computer Science and IT, The Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan
Corresponding author: Saima Abdullah (saima.abdullah@iub.edu.pk)

ABSTRACT Rapid growth of Internet of Things (IoT), and other intelligent devices, introduced different applications which offer real-time latency features; however, it is difficult to handle the large volumes of data produced during the computational process, to adequately complete tasks. The decentralized edge computing process handles the task at the user’s end to accomplish latency applications, but recent research adopted centralized methodologies for computing in the edge network, placing additional overhead for cluster management and grouping. In this paper, we formulate an edge nodes group on task arrivals with a decentralized technique to process jobs, in a parallel mode, to complete execution. In addition, high availability will be added to promise effective processing of IoT based applications executed in the edge computing system. In the edge node environment, where resources are restricted, there is a requirement for high availability methods, which can deliver system reliability according to the local device information, without the data of network topology. In this paper, our technique is to enhance network reliability with the help of the edge node’s local information, which is executed in the distributed edge computing network, while also proposing a high availability technique to enhance the overall IoT environment. Our proposed Latency Aware Algorithm for Edge Computing with High Availability (LAAECHA) detects edge nodes faults, repairs edge nodes and replaces edge nodes with backups, using a new algorithm in a decentralized mode. Our research results show that the proposed LAAECHA method is more effective than existing methods, ensuring latency-aware IoT applications achieve their deadlines, while significantly reducing network traffic as well as guaranteeing system availability and reliability of the IoT network.

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

I. INTRODUCTION
Internet of Things (IoT) can be defined as a set of interconnected devices, including the digitalization and capability of devices to transfer data over a network without any human interaction. All these devices have unique IDs. IoT is well managed, containing a group of numerous strategies, data analyses, machine learning processes and the ability to sense devices. IoT converts the behavior of traditional objects into acting as smart objects by exploring a wide variety of modern technologies. These modifications are implemented by embedding devices, choosing proper communication channels and using internet protocols. IoT establishes a system where data can easily flow among different devices, users and products. The main objective of IoT is to connect real-world things with virtual world things. So, the interconnection with anything can be done at anytime, anywhere in the world [1]. We can see multiple examples in which IoT applications, along with edge computing, play important role in our daily lives, including in smart homes, healthcare and social networks. Edge computing in IoT applications provide processing power and security to edge data [2]. With the progress of IoT and its transmission technology, there are a huge number of devices which are connected with each other by using the internet to create data in large numbers. This bulky data produces a problem of great concern i.e., high latency and requiring higher bandwidth. Data transmission between cloud centers and IoT devices will lead to high latency. Due to this...
reason, edge computing became famous as edge computing servers are installed at edge side and most communication is carried out locally [3]. Edge computing can furnish the services at the edge network to reduce the latency rate. It also helps to improve battery health and maximize its utilization. The performance of edge computing is based on two parameters; the first one is data transmission between edge servers and the second one is the efficiency of edge server processing. The edge server has great potential to provide low latency and reliability simultaneously [4].

Edge computing uses a decentralized approach in which application processing and performing jobs can be done locally. In this way, it minimizes the network traffic problem [5]. There is a need to trace a fault at local nodes, especially when a node failure occurs as processing and computation is done locally [6]. In edge computing, resources are available at the edge network, near end-devices. Computing resources are placed near these devices. This technique has a great impact on bandwidth and also reduces communication latency [7]. In cloud computing, centralized servers are used to first collect data, then perform processing and finally transfer data back to the devices. This approach creates certain problems i.e., a high bandwidth is required for transmission, which increases the cost of the system. So, there is a need to switch to edge computing, which uses a decentralized approach [8]. The task scheduling problem in edge computing is an important topic for discussion. IoT devices offload computational tasks to edge servers for the optimum use of energy. Task offloading requires extra energy; therefore, there is a need to see how the task offloading decision is taken by devices [9].

The studies show that the performance of collaborative edge computing is better than offloading due to the distribution of tasks. The allows for the allocation of tasks and data among edge devices which have processing power, and the capability of data storage and transmission [10]. Edge computing enables most processing to be done at the edge, which increases the system’s response time. It also makes the system flexible because most of the computation can be done at the edge and only a limited amount of information is sent to the cloud network [11].

Resources are accessible at the local network in edge computing. This provides additional dispensation to various applications that need less latency and decentralization. In edge computing, each node is called an edge node. The idea of offloading jobs to a remote server in edge computing fulfills the difference between insufficient processing capabilities and demands of high computation for resource intensive applications. Cloud job offloading fuels the issue of high latency and enhances data transmission, which leads to the need for a high bandwidth. Resources for edge nodes are restricted in edge computing in the sense of power, processing and storage capacity. These limitations create more chances for failure compared to other systems. So, there is a need to develop a fault tolerant system in order to ensure reliability. There is research focused on allowing for a fault tolerant application design, which observes services executed on other nodes as backup [12].

In cloud computing, centralized servers are used to collect data, then accomplish processing and finally transfer data back to the devices. This approach produces certain problems i.e., increase latency, high bandwidth is needed for transmission, which rises the cost of the system and also increase latency. So, there is a need to switch to edge computing, which uses a decentralized approach. The task scheduling problem in edge computing is an important topic for discussion. IoT devices offload computational tasks to edge servers for the optimum use of energy. Task offloading needs extra energy; therefore, there is a need to manage the task offloading decision is taken by devices. In this research paper, we have worked on a decentralized technique for making edge nodes group in the distribution mode to implement the latency-aware and resource concentrated IoT application. However, we offered a decentralized high availability approach to confirm reliability of tasks executed in a distributed manner on the edge nodes group so that is not discuss in existing studies. We have displayed high availability benchmarks for edge computing process, such as reliability and Mean Time to Failure (MTTF), to offer necessary redundancy during the design time for the high availability technique. We have also presented a high availability algorithm for the decentralized edge node grouping, backup node selection, fault detection, node repair and edge node replacement for network reliability in the distributed mode over the IoT network.

**Our Contributions Are:** We have developed the LAAECHA algorithm for edge nodes grouping and faults tolerance with high availability to process a resource concerted task to achieve its target. We have provided edge node repair and replacement for the LAAECHA algorithm to enhance the IoT network reliability and availability. We have also
proposed a mathematical model in the distributed edge executing environment for allocation of resources to a task. In the edge network, reliability standards are computed for edge nodes. To maintain reliability, a backup edge node selection process is presented to provide redundancy.

II. RELATED WORK

With the growth of 5G and the rapid development of IoT, 75.44 billion devices will be connected worldwide by 2025, this will create a high amount of information [13]. Recent IoT works led to practice cloud structures to manage this huge information [14], this demonstrates how costly it will be in terms of bandwidth and time latency. Edge computing was introduced to manage this situation and gained much attention in study [15]. The advancements of mobile phone technology, smart health, Raspberry Pi and smart cities aim to deliver facilities at edge networks by adding calculations on edge node network hardware [16], [17]. These characteristics expand responses and enhance security of data as they manage data locally and minimize network data.

A. EDGE NODE RELATED GROUP FORMATION

The resources which are available to edge nodes are heterogeneous and their geographical position from different network devices varies due to routers, access points, servers and cell phones etc. [18]. Portability is also linked to IoT devices. The generated traffic of end devices lies upon time and the distribution technique according to population strength and activities which are performed by these devices. These characteristics have a great impact on the traffic burden, power and potential of edge devices. Due to these properties, IoT applications face some serious problems i.e., requiring a high bandwidth, latency, flexibility, capacity and capability to error handling. [19]. To overcome all these problems, edge devices and resources can be utilized in a distributed manner without any peripheral aid, just like the cloud [20].

It is ideal to create a set of edge devices to run and process such applications which need more resources and frequently surpass the edge node’s capabilities. Edge computing offers flexible resources that permit distributed computing and provide security for data from flaws often used in a centralized approach [21]. Some research, like cloudlet [22], femtoclouds [23] and edge computing, [24] have insisted on integrating resources for mobile devices. A device offloads the main parts of its processing to cloudlet when it detects edge support, rather creating parts for task completion [25]. The idea of cluster computing in femtoclouds requires computing environment for allocation of resources to a task. In the edge network, reliability standards are computed for edge nodes. To maintain reliability, a backup edge node selection process is presented to provide redundancy.

B. FAULT TOLERANCE

It is significant for edge computing to remain reliable and fault tolerant when an IoT application is running on a set of edge networks. It is a complex task to design an effective and fault tolerant system for an edge computing network, because of the huge diversity of edge devices, networks and data computing approaches, which are under discussion in. [31]. Technologies depending on container-based virtualization have been proposed by [32] for fault-tolerance. The work in [6] proposed fault tolerance and backups in edge cluster networks with the support of Container, Kubernetes, and Apache Kafka. In [33], each device preserves a reliable view of replicated services in the distributed fault tolerant system for the energetic IoT network. In the heartbeat guidelines, any error can be recovered within a few moments without any exterior interference. Authors in [34] explored an approach for scheming applications to endure in high-pressure situations within the edge node computing network. They have presented the perception of segments and used the term “reversibility” in favor of the application design phase to accept node failures in edge computing. The work in [35] emphasizes fault tolerance by providing the concept of crystal for fog application. Crystal’s framework allows programmers to develop application with crystal objects as a building segment. Hereafter, it can provide position accuracy, node flexibility and fault tolerant working in a diverse node’s environment. The research study in [36] deliberates the problem of fault tolerance and consistency for the system to provide a smart environment. Fault tolerance is accomplished through facilities, when a node fails, it is replaced by another suitable node placed near the damaged node.

The data replication approach has been used by [37] to confirm maximum data accessibility in high node failures.
TABLE 1. Comparison of existing work with respect to fault tolerance and availability.

| Paper          | Resource availability check | Decentralized Methodology | Device heterogeneity status | Fault tolerance                                      | Reliability check                              | Drawbacks                                                                 |
|----------------|------------------------------|---------------------------|----------------------------|------------------------------------------------------|-----------------------------------------------|---------------------------------------------------------------------------|
| [12]           | Yes, check resources        | Yes                       | Heterogeneity available    | Provide fault tolerance for both process and data    | Node and application-level measures           | Number of unwanted of replicas might exist, like node fault detection, recovery and damaged node replacement |
| [31]           | No                           | Client-server             | Heterogeneity not available| Provide fault tolerance for process, data and      | Not available                                | Practice external framework to manage fault tolerance                     |
|                |                              |                           |                            | communication                                       |                                               |                                                                           |
| [34]           | Yes, check resources        | Client-server             | Heterogeneity available    | Process and communication                           | Not available                                | Specific framework required for application development                   |
| [36]           | No                           | No                        | Heterogeneity not available| For both process and data                           | Available                                    | For recovery process, put checks on nodes                                  |
| [37]           | No                           | Yes                       | Heterogeneity available    | Only for service                                    | Not available                                | Check only one model for individually facility                            |
| [38]           | Yes, check resources        | Yes                       | Heterogeneity not available| Only for data                                        | Available                                    | Number of unwanted of replicas might be present                            |
| [40]           | No                           | No                        | Heterogeneity not available| Only for communication                              | Incomplete for one technique                 | Support just link reliability                                             |
| [42]           | Yes, check resources        | Yes                       | Heterogeneity available    | Provide fault tolerance data                         | Node level                                   | Number of unwanted of replicas might exist, like node fault detection and recovery |
| Proposed LAAECHA work | Yes, check resources | Yes                       | Heterogeneity available    | Fault tolerance available for data, edge node faults detection and recovery | Node level, group level and application-level measures proposed | Extra work required for network high availability with respect to fault tolerance for edge computing in the IoT network. |

in the IoT based network. They described advantages of information repetition in the same wireless sensor networks (WSN) for IoT network. Specifically, in edge computing networks, failure might occur and these failures might take the important portion of edge side resources offline, which enhances latency, causing total QoS. This paper displays a system proposal for fault tolerance in IoT, using a virtual service composition. We describe the comparison among virtual and real facilities, and display the network process regarding how to manage fault tolerance. For the virtual facility, two methods are used: linear RLS and non-linear MARS. Our practical work proves sensor usages of various modalities to be valuable for backup fault-tolerance [38]. The work in [39] demonstrated the failure of the storage area network (SAN) in IoT environments to support fault tolerance along with backups to cut the network offline for
They estimated a consistency of the storage area network with the use of a binary diagram-based technique for IoT networks.

According to the resources in edge computing, the computing scenarios of scheduling tasks are divided into four categories, and their composition and characteristics are analyzed in detail. According to where their execution takes place, computing tasks can be accomplished via local execution, partial offloading and full offloading. We formulate the optimization problem to minimize the delay time and energy consumption for computation offloading of an edge computing system with different queuing models, indicating its solution complexity [40]. As established above, different techniques and methodologies have been explored to manage fault tolerance in IoT environments for edge node computing. These methodologies generally depend on the centralized mode to handle fault tolerance which is not appropriate for an edge computing system due to wastage of resources and latency. Due to this, required high bandwidth, network cost and edge network life also decreased which is the main drawback of the existing studies. The present research is unique to support network high availability or data, they are generally incapable of delivering reliability parameters and edge nodes management for fault tolerance in the computing network. The investigations and assessments of various methodologies is explained in Table 1.

### III. PROPOSED LAAECHA MANAGEMENT FRAMEWORK

In this section, we will discuss how a number of subtasks are handled in the IoT application initially. After that, we will explore the network decentralized edge node computing model. Then reliability measurements and edge node high availability algorithm are examined to enhance the edge computing environment life.

#### A. TASK MANAGEMENT

For task $\omega_k$ execution on the edge nodes group, characteristics about task are $\omega_k = \{i_k, \mu_k, a_k, e_k\}$, where $i_k$ represents $i^{th}$ edge node, which received the execution request. $\mu_k$ is workload, $a_k$ is data size and $e_k$ is deadline of $k$. The task $\omega_k$ is divided into $b$ subtasks (denoted as $\delta_i$) $\omega_k = \{\delta_1, \delta_2, \ldots, \delta_b\}$, with no restrictions on them. These subtasks can be completed in parallel on various edge nodes without a sequence of working. The important equation variables are displayed in Table 2. To process a subtask circulated on the edge nodes network and for a fair process, the necessary resources are delivered in ($e_k$) time to complete task earlier than its target. In this paper, we are assuming the CPU period needed by every task as a resource. The task $\omega_k$ needed $c_i$ millions of instructions (MI) to finish one subtask. The subtask $\delta_i$ load will be:

$$\mu \delta_i = \sum_{i=1}^{n} c_i.$$  \hspace{1cm} (1)

| Symbol | Description |
|--------|-------------|
| $\omega_k$ | Task |
| $i_k$ | Task is received on $i^{th}$ edge node |
| $\mu_k$ | Currently assigned workload of task |
| $\mu_{\text{car}}$ | At time $t$ currently workload of task |
| $\mu_{\text{seed}}$ | Work at time $t$ still to be assigned |
| $a_k$ | Data size linked with task |
| $e_k$ | Task deadline |
| $t_{\text{odi}}$ | At edge node $e$ CPU time needed to perform a subtask |
| $\delta_b$ | Subtask of task |
| $N_i$ | Instructions needed to execute a subtask |
| $\nu_b$ | Edge node speed |
| $\text{car}\omega_k$ | At time $t$ current assigned resources for task $\omega_k$ |
| $\Re_{\text{car}}$ | At $t$ cumulative resources assigned to task |
| $D_{\text{req}}\omega_k$ | At $t$ current demand of resources by task |
| $\text{Str}_{\omega}$ | All resources share by task |

| Related to group and reliability measures |
|------------------------------------------|
| $E$ | Node related to edge network |
| $\pi_{\omega}$ | Edge node neighbor |
| $\pi_{\delta}$ | Backup of edge node |
| $G_{\omega}^i$ | $i^{th}$ edge nodes group to process $\omega$ |
| $E_{\text{res}}$ | Edge node organizer |
| $U$ | Edge node network |
| $G$ | Group of edge node |
| $\Re_{\delta}$ | All resources needed in $\omega$ |
| $\Re_{\text{req}}$ | At $\delta^{th}$ edge node network resources presented |
| $T_{\text{so}}$ | Subtask $\delta_i$, total completing time |
| $\tau_{\omega}$ | Task $\omega$ time from start to end |
| $A_g$ | Edge nodes failure ratio |
| $r$ | Probability of failures ($t$) |
| $\pi_{\text{back}}$ | Backups edge node number |
| $\pi_{\text{req}}$ | Required backups number |
| $\text{Rrs}$ | Reliability level required |
| $R\pi_{\delta}^g(t)$ | Group member $g_i \in G_{\delta}^i$ ($\omega$) reliability with $\eta$ backups |
| $\xi_i$ | Event that groups edge node $i$ not fail ($t$) |
| $\rho$ | Weight allocated to various resources of edge node in the network |
| $\xi_i$ | Resources at edge node $i$ available capacity |
| $\Lambda$ | Arrival rate of subtask |
| $S\delta$ | Service rate of subtask |
| $T_{\text{trans}}$ | Successful transmission time |
| $T_{\text{req}}$ | Request time period |

138998
The CPU time ($t_{\text{ew}}$) needed to process a $\omega_k$ on the edge network $e$ based on speed ($\omega_{pe}$).

$$t_{\text{ew}} = \frac{\mu_{\text{si}} \delta_{\text{si}}}{\delta_{\text{pe}}}$$ (2)

For task $\omega_k$, which has $b$ subtasks executed on $y$ edge nodes in a distributed way, $\text{car}_{\omega_k}(t)$ represents resources (CPU time) assigned at $t$. $\text{Rcar}_{\omega_k}(t)$ resources for task $k^{th}$ at $t$ will be:

$$\text{Rcar}_{\omega_k}(t) = \int_0^t \text{car}_{\omega_k}(t) \, dt$$ (3)

The complete workload ($\mu_k$) of $\omega_k$ is same to the summation of workload of $\delta$, and $\mu_{\text{car}}$ is the $\mu_k$ for $\omega_k$ allocated to $y$ edge nodes till $t$. The needed $\mu$ will:

$$\mu_{\text{reqd}} = \mu_k - \mu_{\text{car}}$$ (4)

$\text{Dcr}_{\omega_k}(t)$ indicates the current resources demand at $t$. $\text{Str}_{\omega_k}(t)$ specifies complete available resource for the $k^{th}$ $\omega_k$ at $t$. The degree of fairness $\text{fd}_{\omega_k}(t)$ of $k^{th}$ $\omega_k$ at $t$ is demarcated as the resources for $\omega_k$ used in the edge network distribution process related to the cloud.

$$\text{fd}_{\omega_k}(t) = \frac{\int_0^t \text{car}_{\omega_k}(t) \, dt}{\int_0^t \max \{\text{Dcr}_{\omega_k}(t), \text{Str}_{\omega_k}(t)\} \, dt}$$ (5)

The network fairness degree $\text{fd}_{\omega_k}(t) = 1$ displays complete resources available for $k^{th}$ $\omega_k$ at $t$, and the required network resources assigned to finish the $\omega_k$. In disparity $\text{fd}_{\omega_k}(t)$, less than 1 means not a fair execution. It is clear if $k^{th}$ $\omega_k$ is processed on the cloud, all the required resources are available when $\omega_k$ is allocated to the cloud; hence, $\text{fd}_{\omega_k}(t)$ is equal to 1 because $\text{car}_{\omega_k}(t) = \max \{\text{Dcr}_{\omega_k}(t), \text{Str}_{\omega_k}(t)\}$ are available at all times.

B. DECENTRALIZED MODE EDGE NODE GROUP FORMATION

For the management of large tasks competently on the edge environment, we suggest to process it in a distributed mode on the group of edge nodes $G^e_i(\omega)$, where $i$ indicates the group number and $e$ means edge nodes and $\omega$ relates to task to be executed in the group. $E^{orz}$ is an edge node which is represented as an organizer which will receive a $\omega_k$ to process. The organizer edge node will communicate in the group with edge nodes to finish the allocated work and give output to the end nodes.

Our proposed LAAECHA algorithm processes in the decentralized mode because the $E^{orz}$ edge node is responsible for making the group of edge nodes and after that, each node selects a subtask according to its local information within accessible resources. Once an edge node agrees to process a task, it also includes itself as a member of the edge nodes group, cooperating to execute a task in the distribution mode. The $E^{orz}$ edge node can process subtasks by itself; it can also decide to assign a task to another node. When a task $\omega$ reaches the $E^{orz}$ to process in a distributed way, the $E^{orz}$ node will broadcast to the edge environment $U$ to make a $G^e_i(\omega)$.

The edge nodes in the group will collectively process subtasks of $\omega = \{\delta_1, \delta_2, \ldots, \delta_b\}$ and give output back to the organizer node. Each edge node in the network knows its resources ($\text{Raei}$) to process the task. We supposed that a subtask will totally be executed, when it is received by an edge node.

Our proposed LAAECHA algorithm for grouping and $\omega_k$ allocation is offered in Algorithm 1. The $E^{orz}$ edge node will broadcast a message for grouping, and the edge nodes which are in the surroundings will be entered into a queue $\emptyset$ (step 3 to 7). After group formation, the edge nodes in the queue with rich resources will take a task for execution from the $E^{orz}$ edge node (step 9 to 15). Many other edge nodes are included in the group when seeing only nearby edge nodes available (step 18 to 21). This procedure is repeated until necessary resources for executing the tasks are available.

C. PARAMETERS FOR RELIABILITY

Both data and reliability are very significant when an application is working in an edge network. For edge node reliability, MTTF is a metric for failure for edge nodes, considering group $G^e_i(\omega)$ reliability, which processes the scattered job. The group reliability means the probability of finishing the task $\omega$ effectively. The first manages edge node failure during execution on the reliability of $G^e_i(\omega)$. The edge nodes failure process can be considered with Poisson’s process, where edge node failure in the system is not bounded.

As described in task management, the $\omega$ is distributed into $b$ subtasks, to accomplish a $\delta_i \in \omega$ on $g \in G^e_i(\omega)$, where $t$ represents initial time length for subtask $\delta_i$ assignment to the complete subtask. Further, $t_{\frac{T}{\delta i}}$ indicates the organizer node to edge node $g$ data transfer time, $t_{\frac{E}{\delta i}}$ means subtask execution time and $t_{\frac{R}{\delta i}}$ relates to subtask transfer time for the output. Now period of $\delta \tau$ on edge node $g$ will be:

$$t = t_{\frac{T}{\delta i}} + t_{\frac{E}{\delta i}} + t_{\frac{R}{\delta i}}$$ (6)
This shows that if MTTF will be high, the failure rate becomes low and network reliability will increase. For the replication factor probability sum, g reliability will be equal. Reliability will be:

\[ R_{r}^{g}(t) = p0(t) + p1(t) = e - \lambda gt(1 + \lambda g t) \]  
(10)

The relevant MTTF will be:

\[ MTTF_{g} = \int_{0}^{\infty} R_{r}^{g}(t) dt = \frac{1 + \lambda g}{\lambda g} \]  
(11)

When \( \omega \) have \( P \) standby edge node, reliability becomes:

\[ R_{r}^{g}(t) = \sum_{r=0}^{n} \frac{(\lambda g t)^{r} e^{-\lambda g t}}{r!} \]  
(12)

The MTTF becomes:

\[ MTTF_{g} = \int_{0}^{\infty} R_{r}^{g}(t) dt = \frac{n + 1}{\lambda g} \]  
(13)

This paper’s goal is to enhance the subtask reliability with respect to \( t \), and equation 13 shows if the replication factor is enhanced, the \( \delta_{i} \) reliability will be boosted. So, the \( R_{rs} \) during time \( t \) can be accomplished if MTTF is greater or equal to \( t \), this depends on \( P \) edge nodes standby backups. For the standby edge node selection issue, \( \omega \) select \( n_{req} \) number of nodes from backup of \( \delta_{i} \), so that required reliability \( R_{rs} \) is achieved. For \( t \) required, reliability will be:

\[ R_{g}^{n}(t) \geq R_{rs} \]  
(14)

Group \( G_{i}^{g}(\omega) \) guarantees reliability of task execution for \( \tau_{\omega} \), we have supposed no data or process loss during this time period. Subtasks of \( \omega \) are executed on \( g \) in parallel mode, through a probability calculation, there will be no edge node failures in the group during \( \tau_{\omega} \), if any edge node \( g \in G_{i}^{g}(\omega) \) fails during this time \( \tau_{\omega} \), the task will be incomplete. According to Poisson process, we can assess the probability of edge node failure when \( t_{k} \) the edge node \( r \) rate is \( \lambda_{g} \).

\[ p(\tau_{\omega}) = (\lambda_{g} \tau_{\omega}) e - \lambda_{g} \tau_{\omega} \]  
(15)

When \( n \) is number of edge nodes in edge group, then the edge node failure probability among \( \tau_{\omega} \) is \( 1 - P(\tau_{\omega}) \), this shows that the first edge node in group \( G_{i}^{g}(\omega) \) will not fail, and that the second edge node will not fail in group \( G_{i}^{g}(\omega) \), and \( n^{th} \) edge node will also not fail. The event name, \( \tilde{E}_{i} \), where \( t \) relates to an edge node, will not fail,

\[ \tilde{E}_{i} = 1 - P(\tau_{\omega}) \]  
(16)

The probability of accumulative during \( \tau_{\omega} \) when in \( G \) no edge node fails is:

\[ P(\tilde{E}_{1}, \tilde{E}_{2}, \ldots, \tilde{E}_{n}) = P(\tilde{E}_{1})P(\tilde{E}_{2}) \ldots P(\tilde{E}_{n}) \]  
(17)

The group \( R \) will be:

\[ R_{G}(\tau_{\omega}) = \prod_{i=1}^{n} (1 - P(\tau_{\omega})) \]  
(18)
TABLE 3. Neighbor nodes status.

| Position     | Status                                           |
|--------------|--------------------------------------------------|
| Candidate    | n edge node has sufficient resources to          |
|              | complete the task                                |
| Serving      | n edge node already member of same group         |
| Exhausted    | No resources are available on n edge node        |
|              | complete a task                                  |

D. FAULT TOLERANCE

A distributed fault tolerance method is needed to increase the performance of edge nodes with resource limited devices. In the proposed LAAECHA method, edge devices work individually, with no need for global connection in edge networks. The edge node resources computation will be:

$$R_{ni} = \sum_{i=0}^{n} (P \times \text{Available Capacity} (S_i))$$  \hspace{1cm} (19)

$S_i \in \{\text{CPU\%}, \text{RAM\%}, \text{Bandwidth \%}\}$ indicates residual reserves, and $P$ specifies the weight allocated to every resource. In this situation, we have allotted a 30-weight value to bandwidth, 30 to RAM and 40 to CPU. These numbers are given after considering all important factors in the IoT application. These parameters are also suggested by earlier research in [38]. We maintain selective edge nodes from nearby edge nodes in the group $G_i^\omega$, that can execute related task with the increase of node failure. A proper replacement has many resources to provide as low latency performance as possible. When choosing edge node $n$ as the nearby node, edge node $g$ remains cautious with its plan. Each edge node state is shown in Table 3.

- The chances of neighbor’s edge nodes have more precedence for being allocated a backup edge node due to free resources.
- Neighbor edge nodes along with serving states are working edge nodes which provide resources to work as edge node $g$. These edge nodes have no chances for acquiring a backup node.
- Neighbor’s nodes in an exhausted state have no more resources for further tasks. These edge nodes also have no chances for receiving a backup node.

Every edge node $g$ has a small neighbor edge node set $n$ in the edge IoT network. The neighbor edge nodes frequently change data associated with their local resources to remains up-to-date. However, node failure in the edge network is identified by the periodic exchange of information. In the edge network, to guarantee the desired reliability level and manage backups on $\eta_{req}$ edge nodes network, the edge node selection and backup procedure are available in Algorithm 2.

The proposed LAAECHA high availability policy generates edge nodes backups for each edge node in the group, without relying on a central point. Firstly, we formed a neighbor nodes list of $g$, which have no exhausted nodes. Then, we sorted the remaining resources at every edge node.

Algorithm 2. Proposed LAAECHA for Selection of Edge Node and Placement of Backup

```
Require: Edge nodes group $G_i^\omega$ performing a task; Group member edge node $g$; Neighbor Resource $Rrs$; Edge node state; edge nodes exhausted. Ensure: Edge node $g$ reliability

1: for each $g \in G_i^\omega$ do
2:     glist $\leftarrow$ edge node $g$ neighbor
3:     avail_edge_node $\leftarrow$ glist-Exhausted
4:     sort avail_edge_nodes with respect to $Rrs$
5:     for each $e \in$ avail_edge_nodes do
6:         if $e \notin G_i^\omega$ then
7:             backup_edge_node $\leftarrow$ e
8:             avail_edge_node = avail_edge_node - e
9:     end if
10:    end for
11:   for each $e \in$ avail_edge_node do
12:       backup_edge_node $\leftarrow$ e
13:   endfor
14:   while backup_stor < $\eta_{req}$ do
15:       e $\leftarrow$ backup_stor .pop
16:       replica $\leftarrow$ proces (g) & data (g)
17:       copy (replica, e)
18:       backup_stor $\leftarrow$ e
19:   end while
20: end for
```

After that, we prepared a candidate list which is available in the serving nodes. When the backup node list is created, subtask $\delta_i$, processes and data are copied to the backup edge nodes. The edge node replacement quantity is based on $\eta_{req}$, as mentioned earlier.

For the recovery process, every node in the group knows about the tasks allocated to its nearby nodes, along with their edge node backup list and the data from its $E_{orz}$ edge node. All data is distributed timely to the nearby edge nodes. When edge node $g_i$ does not obtain data from its nearby node $e_q$, the nearby edge node is treated as a dead node. After the fault discovery, edge node $g_i$ will check the list of backup nodes and choose the first edge node from the backup list, edge node $e_q$, and send a message to this edge node for recovery. However, using Algorithm 2, subtask $\delta_i$ is assigned to a backup edge node, backup edge node $e_q$, which executes the subtask and forwards the results to $G_i^\omega$. The same process is repeated if any organizer node fails.

E. DECENTRALIZED EDGE NODE REPAIR

The placement of a large number of edge nodes in an IoT environment is necessary for gathering huge volumes of data. In a huge IoT environment, the edge nodes responsible for data gathering also send data to the base station. All edge nodes are activated with some restricted resources. The IoT network lifetime decreases due to the processes of recharging, replacement and recovery of failed edge nodes. The detection
of faulty edge nodes, changing faulty edge nodes with back-ups and recovery of these faults in edge nodes, can enhance the IoT network lifetime.

The task \( \omega_k \) is divided into \( b \) number of subtask (nominated as \( \delta_k \)), \( \omega_k = \{ \delta_1, \delta_2, \ldots, \delta_b \} \), with no limitations, when task \( \omega_k \) is executed on edge nodes group. Several task \( \omega_k \), which are coming from various devices are categorized as \( n \). Then, for task \( \omega_k = \{ \delta_1, \delta_2, \ldots, \delta_b \} \), \( T_{tran} \), is the successful transmission time and \( T_{rep} \) is the request time period, as in first step of proposed LAAECHA algorithm. Every subtask of task \( \omega_k \) can be donated as subtask \( \delta \). Moreover, the subtask arrival and service rate are signified as \( A_k = \frac{1}{T_{rep}} \) and \( S_k = \frac{1}{T_{min}} \) respectively. To measure the system traffic intensity \( \rho \) by putting traffic \( \rho \) in all IoT subgroups, see in proposed LAAECHA algorithm step 4:

\[
\rho = \frac{A_k}{S_k} = \frac{T_{tran}\delta}{T_{rep}\delta} \quad (20)
\]

When edge node \( g \) fails it might be due to faults in hardware or software, which results in the system starting the edge node repair process instantly. The edge node down time and recovery times are supposed to be dispersed. Subtask \( \delta \) failure rate and repair rate is represented as \( \xi \delta \) and \( \eta \delta \) in the Algorithm 2. In the stable IoT system, the transmission rate is larger than the summation of all tasks’ coming rates. Here, we compute the effective transmission rate at each organizer node within the IoT subgroup, which are:

\[
\text{Effective Transmission Rate} = \frac{T_{tran}\delta}{(\xi \delta + \eta \delta)} \quad (21)
\]

\[
\rho = \sum_{\delta=1}^{b} \frac{A_k}{S_k} = \sum_{\delta=1}^{b} \frac{(T_{tran}\delta)}{T_{rep}\delta} < 1 \quad (22)
\]

According to the service rate, if a faulty edge node is repaired, then the service node starts its execution; otherwise, the backupNodeSelection process is initiated which is offered in the proposed algorithm in points 7 to 11. Afterwards, when estimating the system traffic, if the system traffic intensity is greater or the \( \rho \) value is equal or greater than 1 in step 14, then some subtask request times should be changed and reallocated to preserve the system in the best position. Request time of subtasks should be higher to acquire the re-calculated system network traffic \( \rho < 1 \). We can analyze that it is in the unstable system between quality of service and system stability.

In the LAAECHA algorithm, overall system traffic intensity will be calculated at each request time period. Firstly, all subtasks have equal request frequency, denoting the user demand request ratio in the start. Here, \( T_{min} = \text{start} \) is a request period for all the \( \delta \) subtasks in the system. When the network is not unstable, and has no scheduling scheme, then the request time value is altered by the procedure. The overall performance of the system is affected by the traffic intensity, so it measures the start in the system. The request time period will gradually rise through the rate of the smallest request time period \( T_{min} \), which is presented in the proposed LAAECHA algorithm on line 16.

The whole traffic system will be re-calculated by the new rate of the request period, and the subtasks will be re-organized by the new traffic intensity rate; then, total traffic system intensity will be checked repeatedly. While waiting for the circle ends, traffic system intensity becomes less than 1, making the whole system stable. The proposed LAAECHA algorithm reduces the subtask transfer period by continuing the quality of service of various sizes of data within subtasks from the organizer at the receiver’s end.

It reorganizes the subtask demand to assure that the system replies and that it remains stable while seeing the edge node repair portability. Nowadays, the edge devices demand has increased due to updates in IoT edge technology, the IoT network performance, reliability and availability of generating valuable challenges. The IoT system performance slows down due to link failures, edge nodes and the main edge node linking various edge nodes to the sink. It is very important to describe this process which properly detects the failures of edge nodes, repairs these edge nodes and restores the system for the best performance.

The proposed algorithm has following 4 major processing in order to the time complexity of each operation mention in subsequent section.

In First operation depends on distance of two factors first factor is the number of non-broker nodes and second factor is number of broker nodes.

---

**Algorithm 3: Proposed LAAECHA for Edge Node Repair and Edge Node Backup Selection**

1. **for** \( \delta \)th subtask: \( sub \delta \) (\( T_{rep}, T_{trans} \))
2. Initial request time = \( T_{min} \) get Failure rate, Repair rate \( A_k, S_k \)
3. \( T_{rep} = \sum_{\delta=1}^{b} T_{rep} \) for \( \delta = \{\delta_1, \delta_2, \ldots, \delta_b\} \) at broker
4. **for** \( \delta \)th traffic intensity \( \rho \) do
5. **for all** \( T_{rep} = T_{min} \), get effective transmission rate \( T_{trans} \)
6. \( T_{trans} = \frac{T_{trans} \delta}{(\xi \delta + \eta \delta)} \)
7. **if** node get repaired then
8. **continue**;
9. **else**
10. **call** backupNodeSelection
11. **end if**
12. \( \rho = \frac{T_{trans} \delta}{T_{rep} \delta} \) for \( \delta = \{\delta_1, \delta_2, \ldots, \delta_b\} \)
13. while \( T_{trans} \delta \gg T_{rep} \delta \)
14. \( \delta \) sort sub\( \delta \) in descending order
15. **while** \( \rho > 1 \) do
16. **for all** \( \delta = \{\delta_1, \delta_2, \ldots, \delta_b\} \), \( T_{repaired} = T_{repaired} + \frac{T_{min} \delta}{\delta} \)
17. \( \rho = \sum_{\delta=1}^{b} \frac{T_{trans} \delta}{T_{rep} \delta} \)
18. **while end**
19. **end while**
20. **end while**
21. Request sub\( \delta \) in descending \( \rho \) order,
22. **end for**
Cost: Number of non-broker nodes $\times$ broker nodes
Mathematical Bound
brokerNodes = $BN$, nonBrokerNodes = $NBN$

$$T(n) := O(BN \times NBN)$$

In 2nd operation check the cost of the messages forwarding from non-broker nodes to broker nodes.
Cost: Number of non-broker nodes: $T(n) := O(BN)$
In operation 3 messages transmitting from broker node to backup storage point.
Cost: Number of non-broker nodes:

$$T(n) := O(BN)$$

In operation 4 the cost depends on number of messages sending from broker to sink, which proportionally to the number of non-broker nodes.
Cost: Number of non-broker nodes:

$$T(n) := O(BN)$$

The maximum cost of proposed algorithm:

$$T(n) := O(BN \times NBN)$$

A group formation, task distribution and replacement algorithm in operation 1 calculating the distance of nodes from the broker is:
Cost: for each cycle:
numberOfCycle: = $NC$
brokerNodes: = $BN$
nonBrokerNodes: = $NBN$

$$T(n) := O(\text{NC} (BN \times NBN))$$

In operation 2 check the cost of messages forwarding from non-broker nodes to broker nodes in the system.
Cost: Number of non-broker nodes.
Operation 3 cost depends on number of messages sending from broker to sink, which proportionally to the number of non-broker nodes.
Cost: Number of non-broker nodes.
The maximum cost of A group formation, task distribution and replacement algorithm:

$$T(n) := O(\text{NC} (BN \times NBN))$$

A group formation, task distribution and replacement algorithm have extra cost depending on replacement of the edge node in the cluster is considered. The time complexity of LAAECHA is improving against group formation, task distribution and replacement algorithm, while during replacement, LAAECHA repair its faulty edge nodes in minimum time in the network but group formation, task distribution and replacement algorithm replaced the resources which need more time and message communication time increased. Compared algorithm cost is coming in multiple and its impact will give very worst time complexity after some rounds when edge nodes become faulty.

IV. RESULTS AND SIMULATIONS
To appraise our LAAECHA algorithm related with edge node management and faults, we have supposed players attached with sensors in a simulated game environment scenario to gather data and guidelines. In this scenario, various types of sensors are deployed to collect data about position, vital signs, environment and local information. These sensors gather real-time information of the field, it is essential to execute this information instantly to respond on time in the game scenario. There are different tasks for each player, like hitting the target, sound detection, range detection, object movement detection, weapon selection detection, team members information as well as monitoring information. All these jobs need to be processed in real-time without latency, so edge computing is the best for the above scenario. However, all these tasks might not be executed by a single node, so for better results, a decentralized edge computing system is needed to divide tasks for execution by nearby edge nodes to achieve quick and effective results.

![FIGURE 3. Simulation with edge node grouping and LAAECHA.](image-url)

We carried out our simulation using iFogSim as a simulation tool. This simulator tool enables running application scenarios using edge devices with various configurations and allows to measure various application interrelated statistics, as well as device and network-related measures. We have tested our proposed LAAECHA algorithm related to the distribution processing job with a decentralized edge computing system on the group of edge nodes with the help of the general game scenario to manage the edge nodes high availability. However, in the real-time game scenario, high availability management is very important for quick requests and response. In this scenario, the simulation has been repeated with different players and various subtasks for execution. To analyze the proposed LAAECHA algorithm, this game scenario is best for this situation, because the task for the player is easily divided into subtasks, with task execution dependent on the edge node. Due to this, decentralization will lead, and allow, the decentralized grouping of edge nodes and
faulting tolerance to process excellently. In the edge network, every player is linked to a random node, called an organizer edge node. In the distribution environment, edge devices will communicate with the organizer edge nodes for task execution according to their available resources. The requirements for subtask execution are available in below table, and every player’s task distribution is shown in Figure 4. On x-axis number of tasks are shown and on y-axis number of players are shown.

Figure 5 demonstrates the system topology. In the simulation, a healthy edge node is named cloud, there are organizer nodes connected with every end device. In this simulation, there are 8 organizer nodes because we testing 8 players. 72 edge nodes are linked to different organizer nodes with different task execution capacities. There are a large number of sensors which are connected with edge nodes in the network. The cloud and edge node parameters, CPU, upload link, download link and RAM are described in detail in below table. The various parameters which are used in the simulation are discussed in Table 5.

We have been careful about CPU requirements with every task for the purpose of task allocation. A particular subtask for each player will be assigned to those nodes which are fulfilling CPU requirements, adding those nodes to the group as well. With the proposed high availability scenario, each node in the network will also choose its backup edge node. We have analyzed the proposed method by processing tasks allocated to a player i.e., LAAECHA, edge and cloud’s conventional approach. We have checked each player by assigning them various subtasks, examining different parameters for evaluation i.e., network usage, application latency and deadline missing ratio. Figure 6 describes the application latency results with proposed approaches. When we executed the application on the cloud, latency was very high. This is

| Subtask of Task       | C_i needed |
|-----------------------|------------|
| Target hit            | 860        |
| Selection of weapons  | 660        |
| Computation of range  | 850        |
| Detection of sound    | 750        |
| Detection of target   | 1000       |
| Team members difference | 600      |
| Coordination of team  | 550        |

![Task Allocation](image)

![Application Latency](image)

![Comparison of application latency](image)
due to the requirement for all data to be moved to cloud for execution; however, processing with edge node provides a low latency. When we use the LAAECHA high availability approach, latency decreases significantly when compared to existing approaches.

For the reliability test with the LAAECHA method, we checked the subtasks’ deadline missing ratios. Figure 7 illustrates that the proposed approach minimized the deadline missing ratio compared to the other three approaches i.e., execution with high availability, cloud execution and execution without fault tolerance. It is observed that our

| TABLE 5. Different parameters for simulation setup. |
|---------------------------------------------------|
| **Nodes**                                         |
| Cloud                                            |
| Central Processing Unit (Millions of Instruction Per Seconds) | 45000 |
| Random Access Memory (MB)                         | 41850 |
| Upload link (Mbps)                                | 1200  |
| Download Link (Mbps)                              | 12000 |
| Delay among organizer and cloud                   | 90    |
| Edge nodes and organizer nodes                    |
| Central Processing Unit (Millions of Instruction Per Seconds) | 400 to 1800 |
| Random Access Memory (MB)                         | 2000  |
| Upload link (Mbps)                                | 9000  |
| Download Link (Mbps)                              | 9000  |
| Delay among organizer and cloud                   | 1     |
| Sensors                                          |
| Upload link (Mbps)                                | 9000  |
| Download Link (Mbps)                              | 9000  |
| Delay among organizer and cloud                   | 1     |
simulation results are better when the LAAECHA approach is used. The LAAECHA approach also reduces the missing deadlines of tasks. In conclusion, it will enhance application availability and reliability. The performance of the cloud is not ideal due to high latency and missing deadlines. Our proposed LAAECHA method also reduces network usage for the decentralization approach for application execution on the edge nodes group with high availability and without fault tolerance. Compared to cloud-based execution, data transferred in the network will also be minimized. When LAAECHA is activated, it saves bandwidth in the edge side execution. This is due to when any node in the network fails, subtasks are executed on backup nodes.

If fault tolerance is not initiated, then failed tasks are accomplished in the cloud because there is no backup node, which causes high network traffic. Our next comparison is energy consumption of the executing application using the LAAECHA method with and without edge node high availability.

Figure 9 clearly shows that the energy consumption for the LAAECHA approach is optimal when compared to existing fault tolerance approaches. When we used the edge with the high availability approach, faulty edge nodes were not recovered and network resources were wasted. When we used the edge without the fault tolerance approach, tasks were executed on the cloud when an edge node failed, which required extra energy consumed by the cloud.

All tasks and subtasks are completed in the network when the high availability is activated. Backup nodes will complete the task if edge node fails. Figure 10 shows the effect of the increasing rate of edge node failures in the application availability using LAAECHA, with and without the fault tolerance approach. For application availability, the proposed algorithm processes the tasks within the specified time period. We can observe that when LAAECHA is not present in the edge network, the rate of node failures increases and application availability reduces. But the availability of the application increases for higher values of Rrs, as it will enhance replication factor.

Figure 11 displays multiple replications with diverse reliability levels of Rrs. Our methodology created a smaller number of backups to save resources, when there is medium level of reliability and node failure rate is low. However, when the edge node failure percentage is higher, a high replication factor is achieved. A high value replication factor results in a higher failure rate, which indicates that the network can still find a backup within the edge network. It helps to execute a task within its timeframe, but this consumes more resources when there are more replicas where the network has to choose among reliability level and resource utilization.

To evaluate reliability within our proposed LAAECHA algorithm, we have edge node repair ratios for subtasks.
Figure 12 displays that our algorithm supports minimizing the edge nodes faults and repairing faulty edge nodes in the best way to enhance network availability when compared to edge node processing with and without fault tolerance, and the cloud computing approach. It is confirmed that, when the LAAECHA approach is executed, performance is high with a low number of dead edge nodes in the edge network. As a result, the LAAECHA approach will enhance application availability and reliability. On the other hand, cloud execution shows low performance due to a large number of dead nodes; therefore, most of the tasks lose their resources and network availability decreases.

From the results discussed above, we revealed that grouping edge nodes facilitated the application process in the time period by reducing missing deadline ratios, it also reduced overall network traffic. The simulation results enabled the evaluation of various structures with respect to edge computation, such as subtasks deadline’s missing percentages, fault tolerance, failure rate analysis and availability with analysis of the replication factor. The results achieved with the simulation are very conclusive, particularly in demonstrating an incredible reduction in network traffic detected with an overall increase in the rate of reliability. The decentralized edge node grouping can be made at the start of a task, and the task can be accomplished with available resources on edge devices.

V. CONCLUSION AND FUTURE WORK

The Internet of Things is supporting human beings in every field of life, whether it is in the agricultural sector, electrical sector, industrial sector, transport sector or electronic media sector. The proposed LAAECHA approach used a decentralized method to control edge nodes’ task execution with intensive resources to minimize task latency with the help of the high availability system to guarantee the availability and reliability of the edge network. This paper’s approach has two important characteristics over conventional cluster-based approaches for group formation with distributed node execution. First, there is no need for global information to select a cluster head, the edge node group formation starts when the job reaches the organizer edge node, which results in this node communicating with other nodes to process jobs and further edge nodes included in the group for parallel task execution to accomplish deadlines. Second, IoT applications with huge jobs can be processed in a better way locally, which means overall network traffic will be reduced in the edge network. Reliability and availability are the key drivers of IoT systems, so in decentralized high availability methodology, there is a focus on maximizing the reliability level with edge devices heterogeneity. Our proposed LAAECHA approach results showed that our methodologies are effective, the edge node grouping algorithm, backup edge node selection algorithm and faults management have the best performance. When compared to existing approaches for huge task execution, the proposed LAAECHA method achieved the best improvement and performance regarding edge nodes management, missing deadline ratio, network traffic and task latency. The energy efficiency and base station/ sink faults detection, repairing and replacement not discussed in proposed work. These issues may be considered in future, for network high availability with respect to fault tolerance and energy efficiency for edge computing in the IoT environment.

REFERENCES

[1] R. P. Singh, M. Javaid, A. Haleem, and R. Suman, “Internet of Things (IoT) applications to fight against COVID-19 pandemic,” Diabetes Metabolic Syndrome, Clin. Res. Rev., vol. 14, no. 4, pp. 521–524, Jul. 2020.
[2] C. Gong, F. Lin, X. Gong, and Y. Lu, “Intelligent cooperative edge computing in Internet of Things,” IEEE Internet Things J., vol. 7, no. 10, pp. 9372–9382, Apr. 2020.
[3] L. Cui, S. Yang, Z. Chen, Y. Pan, Z. Ming, and M. Xu, “A decentralized and trusted edge computing platform for Internet of Things,” IEEE Internet Things J., vol. 7, no. 5, pp. 3910–3922, May 2020.
[4] M. S. Elbamby, C. Perfecto, C. F. Liu, J. Park, S. Samarakoon, X. Chen, and M. Bennis, “Wireless edge computing with latency and reliability guarantees,” Proc. IEEE, vol. 107, no. 8, pp. 1717–1737, Aug. 2019.
[5] H. El-Sayed, S. Sankar, M. Prasad, D. Pathal, A. Gupta, M. Mohanry, and C. T. Lin, “Edge computing: The big picture of the integration of edge IoT and the cloud in a distributed computing environment,” IEEE Access, vol. 6, pp. 1706–1717, 2017.
[6] A. Javed, K. Heljanko, A. Buda, and K. Främling, “CEFiIoT: A fault-tolerant IoT architecture for edge and cloud,” in Proc. 4th World Forum Internet Things (WF-IoT), Feb. 2018, pp. 813–818.
[7] G. Premmsankar, M. D. Francesco, and T. Taleb, “Edge computing for the Internet of Things: A case study,” IEEE Internet Things J., vol. 5, no. 2, pp. 1275–1284, Apr. 2018.
[8] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, “A survey on the edge computing for the Internet of Things,” IEEE Access, vol. 6, pp. 6900–6919, 2018.
[9] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, “Cost efficient scheduling for delay-sensitive tasks in edge computing system,” in Proc. IEEE Int. Conf. Services Comput. (SCC), Jul. 2018, pp. 73–80.
[10] Y. Sahni, J. Cao, and L. Yang, “Data-aware task allocation for achieving low latency in collaborative edge computing,” IEEE Internet Things J., vol. 6, no. 2, pp. 3512–3524, Apr. 2018.
[11] C. Avasalcai, C. Tsigkanos, and S. Dustdar, “Decentralized resource auctioning for latency-sensitive edge computing,” in Proc. IEEE Int. Conf. Edge Comput. (EDGE), Jul. 2019, pp. 72–76.
[12] M. Mudassar, Y. Zhu, S. Jiao, and J. Shen, “A decentralized latency-aware task allocation and group formation approach with fault tolerance for IoT applications,” IEEE Access, vol. 8, pp. 49212–49223, 2020.
[13] 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.
[14] H.-L. Truong and S. Dustdar, “Principles for engineering IoT cloud systems,” IEEE Cloud Comput., vol. 2, no. 2, pp. 68–76, Mar. 2015.
[15] 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.
[16] X. Su, P. Li, J. Riekkii, 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. Pervasive Comput. Commun. (PerCom), Mar. 2018, pp. 1–9.
[17] W. Zhang, H. Flores, and P. Hui, “Towards collaborative multi-device computing,” in Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops), Mar. 2018, pp. 22–27.
[18] V. V. Shakhov, A. N. Yurgenson, and O. D. Sokolova, “Analysis of fault tolerance of wireless sensor networks,” in Proc. 15th Int. Sci.-Tech. Conf. Actual Problems Electron. Instrum. Eng. (APEIE), vol. 2, Oct. 2016, pp. 390–393.
[19] L. M. S. de Souza, H. Vogt, and M. Beigl, “A survey on fault tolerance in wireless sensor networks,” in Proc. 20th Int. Symp. on Wireless Pervasive Comput. (ISWPC), Sep. 2015, pp. 249–253.
[20] T. N. Gia, A.-M. Rahman, T. Wasterlund, 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.
[21] T. Yuan and S. Zhang, “Secure fault tolerance in wireless sensor networks,” in Proc. IEEE 8th Int. Conf. Comput. Inf. Technol. Workshops, Jul. 2008, pp. 477–482.

[22] H. Yetgin, K. T. K. Cheung, M. El-Hajjar, and L. H. Hanzo, “A survey of network lifetime maximization techniques in wireless sensor networks,” IEEE Commun. Surveys Tuts., vol. 19, no. 2, pp. 828–854, 2nd Quart., 2017.

[23] E. Moridi, M. Haghparast, M. Hosseinzadeh, and S. J. Jassi, “Fault management frameworks in wireless sensor networks: A survey,” Comput. Commun., vol. 155, pp. 205–226, Apr. 2020.

[24] H. Mohapatra and A. K. Rath, “Survey on fault tolerance-based clustering evolution in WSN,” IET Netw., vol. 9, no. 4, pp. 145–155, Jul. 2020.

[25] W. M. Elsayed, S. F. Sabbeh, and A. M. Riad, “A distributed fault tolerance mechanism for self-maintenance of clusters in wireless sensor networks,” Arabian J. Sci. Eng., vol. 43, no. 12, pp. 6689–6697, Dec. 2018.

[26] M. M. Jamjoom, “EEBFTC: Extended energy balanced with fault tolerance capability protocol for WSN,” Int. J. Adv. Comput. Sci. Appl., vol. 8, no. 1, pp. 253–258, 2017.

[27] F. T. Giuntini, D. M. Beder, and J. Ueyama, “Exploiting self-organization and fault tolerance in wireless sensor networks: A case study on wildfire detection application,” Int. J. Distrib. Sensor Netw., vol. 13, no. 4, Apr. 2017, Art. no. 155014771770412.

[28] D. Scazzoli, A. Kumar, N. Sharma, M. Magarini, and G. Verticale, “Fault recovery in time-synchronized mission critical ZigBee-based wireless sensor networks,” Int. J. Wireless Inf. Netw., vol. 24, no. 3, pp. 268–277, Sep. 2017.

[29] 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.

[30] S. Abdullah and K. Yang, “An energy efficient message scheduling algorithm considering node failure in IoT environment,” Wireless Pers. Commun., vol. 79, no. 3, pp. 1815–1835, Dec. 2014.

[31] 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.

[32] 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.

[33] 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.

[34] R. Paul, J. Melchior, P. Van 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.

[35] 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.

[36] 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.

[37] W. B. Quim and O. Oktasap, “DRAW: Data replication for enhanced data availability in IoT-based sensor systems,” in Proc. IEEE 16th Int. Conf. Dependable, Autonomic Secure Comput., 16th Int. Conf. Pervas. Comput., 4th Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech), Aug. 2018, pp. 770–775.

[38] S. Zhou, K.-J. Lin, J. Na, C.-C. Chuang, and C.-S. Shih, “Supporting service adaptation in fault tolerant Internet of Things,” in Proc. IEEE 8th Int. Conf. Service-Oriented Comput. Appl. (SOCA), Oct. 2015, pp. 65–72.

[39] 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.

[40] S. Chen, Q. Li, M. Zhou, and A. Abusorrah, “Recent advances in collaborative scheduling of computing tasks in an edge computing paradigm,” Sensors, vol. 21, no. 3, pp. 799, Jan. 2021.

[41] M. S. Hossain, F. B. Islam, C. I. Nwakamma, J. M. Lee, and D.-S. Kim, “Decentralized latency-aware edge node grouping with fault tolerance for Internet of Battlefield Things,” in Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC), Oct. 2020, pp. 420–423.

[42] J. Fang, R. Kang, and Y. Chen, “Reliability evaluation of non-repairable systems with failure mechanism trigger effect,” Rel. Eng. Syst. Saf., vol. 210, Jun. 2021, Art. no. 107454.

[43] K. K. Raghuvanshi, A. Agarwal, K. Jain, and V. B. Singh, “A time-variant fault detection software reliability model,” Social Netw. Appl. Sci., vol. 3, no. 1, pp. 1–10, Jan. 2021.

[44] T. Yuan, T. Yan, and S. J. Bae, “Superposed Poisson process models with a modified bathtub intensity function for repairable systems,” ISE Trans., vol. 53, no. 9, pp. 1037–1051, Sep. 2021.