I. J. Computer Network and Information Security, **2026, 4, 106-129**
Published Online on August 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2026.04.06

Adaptive Context-Aware Replication Mechanism for Distributed IoT Environments

Mrinal Kanti Mahato
Department of CS, Nistarini College, Purulia, 723101, India
Department of Computer Science and Engineering, Maulana Abul Kalam Azad University of Technology, Nadia, 741249, India
E-mail: mrinalmahato22@gmail.com
ORCID iD: https://orcid.org/0009-0009-8638-8299

Bikash Choudhury*
Department of CSE, Ramkrishna Mahato Government Engineering College, Purulia, 723103, India
E-mail: bikashchoudhury@rkmgec.ac.in
ORCID iD: https://orcid.org/0000-0002-5962-0656
*Corresponding author

Tanushree Garai
Department of CSE, Government College of Engineering & Ceramic Technology, Kolkata, 700010, India
E-mail: tanushree.garai@gcect.ac.in
ORCID iD: https://orcid.org/0009-0007-3675-3943

Sudip Kumar Adhikari
Department of CSE, Kalyani Government Engineering College, Nadia, 741235, India
E-mail: sudipadhikari@ieee.org
ORCID iD: https://orcid.org/0000-0001-5174-397X

Himadri Nath Saha
Department of Computer Science, Surendranath Evening College, Kolkata, 700009, India
E-mail: contactathimadri@gmail.com
ORCID iD: https://orcid.org/0000-0001-8864-4297

Received: January 27, 2026; Revised: March 11, 2026; Accepted: April 22, 2026; Published: August 8, 2026

Abstract: The rapid evolution of the Internet of Things (IoT), supported by the convergence of cloud, edge and mist computing layers, opens new avenues for delivering reliable and responsive services to distributed smart devices. However, ensuring efficient and adaptive service replication in such resource-constrained and dynamically changing IoT environments remains a significant challenge. To tackle this, we introduce Elastic Context-Aware Replication (ECAR), an intelligent replication strategy tailored for IoT systems. ECAR dynamically redistributes services across the IoT continuum by leveraging both physical and logical contextual information. Unlike traditional replication schemes, ECAR continuously adapts to real-time workload fluctuations and network conditions, ensuring low latency and efficient resource usage. ECAR's effectiveness is demonstrated through a comparative evaluation involving diverse IoT deployment scenarios, including Cloud-Intensive Replication (CIR), Cloud-Edge-Intensive Replication (CEIR) and Cloud-Edge-Access-Intensive Replication (CEAIR), alongside two existing replication strategies, Group-Delay-Aware Replication (GDAR) and Combined Context-Aware Replication (CCA). The evaluation shows ECAR achieving up to 18% reduction in service drop rates, 82% improvement in allocation efficiency and 82% better resource utilization. These results underline ECAR's effectiveness in supporting scalable, reliable and latency-aware service delivery for IoT deployments.

Index Terms: Service Replication, Internet of Things, Context-Aware, Proactive Sensing.

1. Introduction

Cloud computing has gained significant attention for its technical and economic benefits, particularly in enabling on-demand capacity management. The exponential expansion of IoT has intensified the need for responsive and secure computing, as the massive data generated by IoT devices demands real-time processing. However, relying solely on cloud infrastructures has revealed limitations such as latency issues, bandwidth constraints and concerns over data privacy and security [1]. Edge computing brings computation closer to end-users, complementing cloud computing with reduced latency, enhanced security and improved responsiveness [2, 3]. Initially, IoT devices were passive data collectors in cloud and edge architectures; however, mist computing has evolved them into active participants leveraging their proximity to users [4].

Service-Oriented Middleware (SOM) serves as a fundamental framework for managing the complex, distributed nature of IoT environments [5]. It acts as a middleware layer that enables seamless communication, coordination and decision-making among heterogeneous IoT devices, ensuring efficient interaction within large-scale networks. By integrating various core functionalities of Service-Oriented Architecture (SOA) [6], such as service discovery, server selection and service replication [7], SOM enhances interoperability and resource management across diverse IoT infrastructures. SOM facilitates real-time data processing and adaptive decision-making through its ability to unify different IoT components into a cohesive system. Additionally, SOM employs multiple agent-based mechanisms, where installed agents across different nodes collaborate to achieve a global objective that optimizes system performance and responsiveness in dynamic IoT ecosystems. Maximizing the fulfillment of QoS requirements for a larger number of requests is a primary goal for any service provider. Replication plays a crucial role in achieving this objective across diverse architectures, spanning from traditional cloud systems [8] to evolving cloud-edge frameworks [9]. While efficient resource utilization sustains availability, replication serves as a fundamental mechanism to enhance resource accessibility and reliability. For enhancing resource availability, replication has remained a cornerstone from early file-based cloud service models [10] to peer-to-peer frameworks [11] and modern cloud-edge systems [12]. Now, replica distribution throughout a large IoT network becomes pivotal due to uneven demand and changing requirement patterns across different regions. In this paper, we focus on the challenge of service replication in a comprehensive three-tier architecture (cloud-edge-access) and introduce a novel dynamic replication scheme that optimizes resource efficiency while enhancing data availability across the network according to the demand disparity across the regions. Finding a node for replication in a three-tier network becomes a challenge due to its wide search space. We introduce a context-aware replication strategy designed to reduce the search space and accelerate the replication process. The effectiveness of the proposed approach is validated through comprehensive experimentation and analysis, confirming its capability to address the dynamic requirements of IoT environments. The key contributions of this work include:

1) To reduce the search space, we propose a procedure that divides the complete network into two different context networks. *Logical-Context:* Based on the service consumption or provision similarities, nodes in the network are segregated. *Physical-Context:* Based on the physical proximity, nodes are segregated. Next, nodes in the network are grouped based on the *Combined-Context,* which is a combination of the *Logical-Physical* context. The complete procedure is discussed in Section 4.
2) Next, a proactive distributed replication strategy is introduced that considers the requirement inconsistency in different physical-context and runs over the combined-context network to reduce the service drop rate or increase the service availability. To address the replication issue, we formulate it as a Distributed Constraint Distribution Problem (DCDP) and solve it using the Min-Max algorithm, as discussed in Section 5.
3) To evaluate the performance of our proposed context-aware proactive replication scheme, a mathematical model has been formulated for a three-tier IoT architecture. This model incorporates both reactive and proactive replication strategies and quantitatively measures key performance indicators such as service drop rate, response time, service completion time and resource utilization elaborated in Section 7.

The comprehensive review of related literature is presented in Section 2. Following that, Section 3 explains our system's architecture, encompassing computation and communication models. Section 4 elaborates on the conceptual underpinnings of our contextual framework. Subsequently, Section 5 delves into the practical implementation and intricacies of our proposed replication scheme, describing its design and deployment. Next, the performance of our scheme in Section 7 offers insights into its efficacy through rigorous evaluation. The outcomes of this evaluation are presented in Section 8, showcasing the advantageous facets of our Proposed Replication Scheme (ECAR). Finally, Section 9 encapsulates the key findings and contributions of our research.

2. Related Work

The increasing demand for high-quality service delivery across the World Wide Web has necessitated replication as a fundamental strategy, with various approaches addressing distinct challenges. Centralized replication techniques have traditionally prioritized reliability through static replication, particularly in high-performance computing environments,

optimizing performance for contiguous data requests but struggling with non-contiguous access patterns and lacking adaptability [13]. Similarly, static replication in Hadoop Distributed File Systems (HDFS) has improved reliability but hindered performance when uniformly applied to all data files, prompting the introduction of predictive analysis and erasure coding solutions, albeit with limited applicability beyond HDFS [14]. In geographically distributed systems, adaptive automation frameworks such as AWARE [15] have been introduced to mitigate heterogeneous latencies, although their effectiveness remains constrained under rapidly fluctuating network and workload conditions. A group delay-aware service replication and task offloading framework in Mobile Edge Computing (MEC) [16] minimizes average response time while enforcing group-based QoS constraints, including maximum delay and inter-user delay synchronization.
The problem is formulated as an Integer Linear Programming (ILP) model and efficiently solved using linear relaxation and rounding techniques, achieving near-optimal performance with reduced computational complexity.

Distributed approaches have been explored to manage hotspots and enhance query efficiency, particularly within Peer-to-Peer (P2P) environments, where replication distance and replica count influence update frequency and maintenance costs [11]. However, critical factors such as network traffic and device mobility are often overlooked, limiting cost optimization potential. Swarm intelligence-based methods attempt to minimize overhead by clustering nodes and distributing replicas, yet their effectiveness is restricted by dynamic variations in network conditions and node behavior [17]. Traditional State Machine (SM) and Deferred Update (DU) replication techniques, while effective for large transactions and CPU-intensive operations respectively, face challenges in achieving efficient hybrid integration [18]. A decentralized offloading framework combining distributed auctions with deep reinforcement learning has also been proposed to allocate tasks based on reputation in federated IoT settings, enhancing fairness and autonomy, although assuming stable offloading channels and reliable peer feedback.

Dynamic replication strategies have been developed to overcome the limitations of static methods by providing greater flexibility under varying network conditions. In edge computing frameworks, core network loads have been reduced by replicating data across fog layers, although reliance on static load balancing restricts responsiveness to real-time workload fluctuations [19]. In fog-radio access networks (F-RANs), virtual machine migration introduces replication overheads. Though optimization techniques improved placement efficiency, they often fail to account for user mobility and dynamic traffic demands [20]. A reliability-aware task replication algorithm was proposed to utilize delay samples and MEC server statuses, achieving significant delay reductions while minimizing redundant replication [21]. A proactive, context-aware service replication scheme for ad hoc IoT scenarios is proposed that manages replications dynamically [22]. It employs a dual-threshold-based sensing mechanism to predict replication needs in advance and leverages both physical context (node stability and proximity) and logical context (service similarity and usage patterns) to identify suitable replication nodes. A decentralized optimization approach is used for efficient service allocation, improving response time, service availability and resource utilization. These works collectively highlight the importance of integrating QoS-aware optimization and context-driven proactive strategies to design efficient and scalable service replication mechanisms in diverse edge and IoT scenarios. In addition, a dynamic service function chaining (SFC) classification framework for NFV/SDN networks has been introduced [23], addressing the Dynamic Chain Request Classification Offloading (D-CRCO) problem through a hybrid eviction and split-distribution approach. An ILP model and a heuristic (DNN) were designed to maximize the number of served SFC requests while respecting TCAM size, link capacity, and service availability constraints, significantly enhancing classification scalability and resource efficiency.

Machine learning-based approaches have increasingly been integrated into replication and resource management strategies to enhance adaptability and efficiency. In HDFS, clustering-based dynamic replication uses usage patterns to apply different replication policies to different clusters. This improves storage efficiency and data retrieval speed [24]. However, the approach has limited adaptability in highly dynamic workload environments. Reinforcement learning (RL) models have been utilized to adjust replication and scaling decisions in real-time environments, eliminating manual threshold calibration and improving responsiveness to workload variations [25]. RL is considered to adaptively manage replication without prior knowledge of the workload [26]. Combining with predictive analytics, an RL algorithm has been proposed to forecast system failures and adjust replication policies in real time [27]. ML- and RL-based approaches can enhance performance but incur high training overhead and depend on substantial historical data. In this context, a joint task offloading and resource allocation strategy for multi-user and task-dependent MEC systems was proposed in [28]. The approach used a Deep Q-Network (JTOCRA-DQN) to solve a weighted delay-energy optimization problem. It also included action space refinement and dynamic task prioritization, which helped reduce the overall system cost. Furthermore, an energy-efficient task offloading and multi-cache placement strategy for mobile augmented reality applications in a three-layer MEC system was proposed in [29]. The approach used a block coordinate descent-based algorithm to jointly optimize service caching, computation caching, and task offloading. This helped reduce both service delay and energy consumption significantly. Additionally, a resource and delay-aware fine-grained service offloading mechanism for collaborative edge computing has been introduced [30], employing network-adaptive service graph reconstruction and graph neural network-based matching algorithms to balance resource utilization, eliminate dependency conflicts, and reduce service transmission delay with strong scalability in dynamic environments. With the emergence of AI, replication strategies also leverage clustering with ML. A dynamic replication policy on HDFS considered machine learning clustering to categorize files into several (hot, warm and cold) groups [24]. Different

replication factors are assigned in each group to reduce storage consumption while maintaining availability. A *K-Means++* based approach is used to extract contextual characteristics from HDFS logs to group files into clusters and assign tailored replication strategies [31]. Complementing these file-centric methods, the Smart Data Placement Strategy (SDPS) applies clustering at the node level [32]. Moreover, these approaches provide significant improvements over static replication, but clustering-based methods rely heavily on a narrow set of predefined features, thus limiting adaptability.

Other techniques based on indexing aim to minimize replication and ensure optimal service response times in IoT networks, although adapting to dynamically varying requests and device densities remains a significant challenge [33].

Beyond these core approaches, alternative methods have targeted replication challenges in evolving environments. Content delivery networks (CDNs) have been utilized to alleviate core network congestion by replicating content at surrogate servers [34], although static configurations limit their adaptability to shifting user demands. Predictive models have been employed to optimize client movement and replica placement in mobile environments, reducing latency and resource consumption [12]. Furthermore, recent innovations explore vehicles as fog nodes to minimize delay, though heavy reliance on accurate mobility predictions constrains their effectiveness [35]. Additionally, optimizing task allocation and CPU utilization remains critical for maintaining service reliability across extreme edge devices [36]. In summary, centralized, distributive, dynamic, and machine learning-based replication approaches each offer partial solutions to replication challenges, although inherent limitations persist. A dynamic three-layer replication framework, capable of adapting to network conditions, workload fluctuations, and evolving user demands, is required to comprehensively address these challenges, ensuring high availability, reduced latency, and efficient resource utilization across diverse computing environments.

A comparative evaluation of these existing methods alongside the proposed scheme is presented in Table 1.

Table 1. Comparison Between Literature Review and Proposed Work.

Paper	Cloud	Edge	Access	Replication	Context	Response Time	Optimization	Load Balancing	Resource Provisioning
[11]	✓	×	✓	✓	×	✓	✓	×	✓
[17]	×	×	✓	✓	✓	✓	×	×	✓
[34]	✓	×	✓	×	✓	✓	✓	×	✓
[35]	✓	×	✓	✓	×	×	×	×	✓
[19]	✓	✓	×	✓	×	✓	×	✓	✓
[36]	×	×	✓	✓	×	✓	✓	✓	×
[20]	×	✓	×	✓	×	×	✓	×	✓
[12]	✓	✓	×	×	✓	✓	✓	×	✓
[37]	×	✓	×	✓	×	✓	✓	×	×
[38]	×	×	✓	✓	✓	✓	✓	×	×
[16]	✓	✓	×	✓	×	✓	×	×	✓
[22]	✓	✓	✓	✓	×	✓	✓	×	✓
ECAR (proposed)	✓	✓	✓	✓	✓	✓	✓	✓	✓

3. Network and System Model

We consider a comprehensive three-tier core-edge-access IoT network and represent it as a hierarchical graph G = (*N,E*). The core (N_C), which is the highest layer in this hierarchy, comprises high computational servers known as core-cloud-servers (N_{Cs}), denoted as $N_{Cs} = \{c_1,c_2,c_3,...,c_n\}$ where $NCs \subseteq NC$. In the edge layer (N_E), some nodes function as edge-cloud-servers, denoted as $N_{Es} = \{e_1,e_2,e_3,...,e_n\}$, where $NEs \subseteq NE$. The lowest layer, the access network (N_D), contains all end-user devices, including low-powered sensors. Additionally, some nodes in this access network are designated as access-cloud-servers (N_{Ds}), represented as $N_{Ds} = \{d_1,d_2,d_3,...,d_n\}$, where $NDs \subseteq ND$. Initially, all services are provided by the core-cloud-servers (N_{Cs}), based on user demand, services are replicated to the edge-cloud-servers (N_{Es}) and/or access-cloud-servers (N_{Ds}).

An agent a_i is deployed on each network node N_i, equipped with three distinct interfaces: the Service-Provider Interface (SI), the Service-Consumer Interface (CI) and the External-Agent Interface (EAI). Let SI_{ai} denote the SI, CI_{ai} denote the CI and EAI_{ai} denote the EAI for the agent a_i. The SI allows a_i to manage node-specific tasks, formally $SI_{ai} \rightarrow T(N_i)$, where $T(N_i)$ represents tasks specific to N_i. The CI enables a_i to intercept service requests generated by N_i as a client, expressed as $CI_{ai} \rightarrow R(N_i)$, where $R(N_i)$ represents service requests from N_i. Both SI_{ai} and CI_{ai} are local interfaces confined to N_i. The EAI, denoted as EAI_{ai}, allows a_i to communicate with other agents a_j across the network, represented as $EAI_{ai} \leftrightarrow a_j$ for $i = j$. The utility u_i of a server $server_i$ is derived from a combination of critical resource indicators: the remaining CPU cycles (R_{ci}), available energy reserves (R_{ei}), unused memory capacity (R_{mi}) and remaining bandwidth (R_{bi}) associated with node v_i. These parameters are normalized to form a resource vector $\mathbf{R}_i$, ensuring that the comparison across heterogeneous nodes remains consistent and unbiased [22]. Specifically,

$$R_i = (\overline{R}_{c_i}, \overline{R}_{e_i}, \overline{R}_{m_i}, \overline{R}_{b_i}) \tag{1}$$

where each $\left(\overline{R}_x = \frac{R_x}{\max\{R_x\}}\right)$ denotes the normalized value of the corresponding resource dimension. To capture the varying importance of different resources, we introduce a weight vector:

$$W = (w_c, w_e, w_m, w_b) \tag{2}$$

where, $w_c + w_e + w_m + w_b = 1$. The utility u_i of agent A_i (i.e., the server $server_i$) is then computed as a weighted sum:

$$u_i = w_c\overline{R}_{c_i} + w_e\overline{R}_{e_i} + w_m\overline{R}_{m_i} + w_b\overline{R}_{b_i} \tag{3}$$

This utility measure serves as a decision metric for selecting appropriate nodes for service replica placement. A higher utility implies that the server is better suited for hosting replicas, as it possesses a greater share of available resources relative to other candidates. The primary objective is to assign service replicas to nodes in a way that maximizes overall network utility, thereby promoting efficient resource utilization while meeting client-side QoS demands. To enhance responsiveness and reduce latency during both service discovery and allocation decision-making, the network infrastructure is logically partitioned into two operational contexts, as elaborated in the subsequent section.

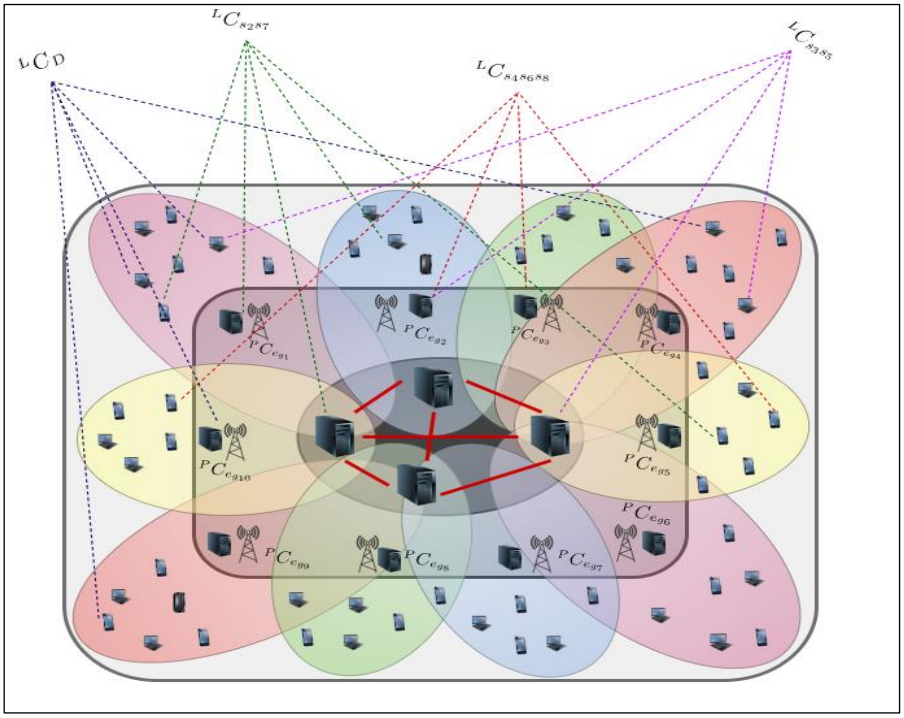

Fig. 1. Network and System Model.

4. Context Creation

4.1. Non-Functional Context Network

We partition the network nodes into multiple overlapping and relatively smaller sub-networks, referred to as Physical-Context *Networks* or *Non-Functional-Context Networks* (PCN). The nodes (core, edge and access) are grouped based on physical proximity and/or connectivity. PCNs are formed in a hierarchical network consisting of core, edge and access layers, using gateways in the edge network as reference points. Nodes in the access layer directly connected to a specific gateway, edge layer nodes connected to the same gateway and core layer nodes connected to the same gateway are grouped into the same PCN. This method ensures that all nodes in the network are included in at least one PCN based on their physical proximity and connection to the gateways, allowing for organized and efficient management of network resources. Overlapping PCNs are possible since nodes, especially in the core and edge layers, may connect to multiple gateways. Let $G \subseteq N_E$ be the set of gateways in the edge layer. Now, the connectivity function for the nodes in a gateway can be defined as $fP : N \rightarrow G$ that maps each node connected to the gateway. The set of access nodes connected to a gateway $g \in G$ is:

$$\dot{}^{P}\, C_D(g) = \{n \in N_D \mid f_P(n) = g\} \tag{4}$$

The set of edge nodes connected to a gateway $g \in G$ is:

$$\dot{}^{P}\, C_E(g) = \{n \in N_E \mid f_P(n) = g\} \tag{5}$$

The set of core nodes connected to a gateway $g \in G$ is:

$$\dot{}^{P}\, C_C(g) = \{n \in N_C \mid f_P(n) = g\} \tag{6}$$

A Physical Context Network (PCN) for a gateway $g \in G$ includes all nodes from the access layer, edge layer and core layer that are connected to g.

$$P_{C_N}(g) = P_{C_D}(g) \cup P_{C_E}(g) \cup P_{C_C}(g) \tag{7}$$

Since nodes in the core, edge and access layers can be connected to multiple gateways, PCNs can overlap. For two gateways $g_i, g_j \in G$:

$$P_{C_N}(g_i) \cap P_{C_N}(g_j) \neq \emptyset \tag{8}$$

Union of all PCNs covers all nodes in the network:

$$\bigcup_{g \in G} P_{C_N}(g) = N_D \cup N_E \cup N_C \tag{9}$$

Consider a network with core nodes $N_C = \{c_1,c_2,c_3, c_4,c_5,c_6\}$, edge nodes $N_E = \{e_1,e_2,e_3,e_4,g_1,g_2,g_3,g_4\}$ where g_1,g_2,g_3,g_4 are gateways and access nodes $N_D = \{d_1,d_2,d_3,d_4,d_5,d_6,d_7,d_8,d_9,d_{10},d_{11},d_{12},d_{13},d_{14},d_{15},d_{16}\}$. Next, the connectivity functions are defined as: $f_P(d_1) = g_1$; $f_P(d_2) = g_1$; $f_P(d_3) = g_2$; $f_P(d_4) = g_2$; $f_P(d_5) = g_1,g_2$; $f_P(d_6) = g_1$; $f_P(d_7) = g_3$; $f_P(d_8) = g_4$; $f_P(d_9) = g_3$; $f_P(d_{10}) = g_3$; $f_P(d_{11}) = g_4$; $f_P(d_{12}) = g_1,g_4$; fP(d13) = g2,g3; fP(d14) = g3,g4; fP(d15) = g4; fP(d16) = g1; fP(e1) = g1; fP(e2) = g2; fP(e3) = g3; $f_P(e_4) = g_4$; $f_P(c_1) = g_1$; $f_P(c_2) = g_2$; $f_P(c_3) = g_2$; $f_P(c_4) = g_3$; $f_P(c_5) = g_3,g_4$; $f_P(c_6) = g_1$. The PCNs for g1, g_2, g_3, g_4 are:

$$\begin{aligned} P_{C_N}(g_1) &= \{d_1, d_2, d_5, d_6, d_{12}, d_{16}\} \cup \{e_1\} \cup \{c_1, c_6\} \\ &= \{d_1, d_2, d_5, d_6, d_{12}, d_{16}, e_1, c_1, c_6\}; \end{aligned}$$

$$\begin{aligned} P_{C_N}(g_2) &= \{d_3, d_4, d_5, d_{13}\} \cup \{e_2\} \cup \{c_2, c_3\} \\ &= \{d_3, d_4, d_5, d_{13}, e_2, c_2, c_3\}; \end{aligned}$$

$$\begin{aligned} P_{C_N}(g_3) &= \{d_7, d_9, d_{10}, d_{13}, d_{14}\} \cup \{e_3\} \cup \{c_4, c_5\} \\ &= \{d_7, d_9, d_{10}, d_{13}, d_{14}, e_3, c_4, c_5\}; \end{aligned}$$

$$\begin{aligned} P_{C_N}(g_4) &= \{d_8, d_{11}, d_{12}, d_{14}, d_{15}\} \cup \{e_4\} \cup \{c_5\} \\ &= \{d_8, d_{11}, d_{12}, d_{14}, d_{15}, e_4, c_5\} \end{aligned}$$

4.2. Functional-Context Network

Functional-context network or logical-context network (LCN) represents the clustering of nodes considering their behavioral proximity. We regard a set of client nodes providing/consuming similar services as exhibiting behavioral proximity. We observe that requests generated by a client are repetitive and frequently request a transaction of services from a set of services.

RequestSet(R_{set}): is a binary relation $R \subseteq N \times S : \forall(n,s) \in R$, $n \in N$ and $s \in S$ denotes a service requested/provided by node n.

Transaction(T_i): A Transaction $T_i \subseteq S$ is a non-empty finite set of services. Consider a transaction Ti = {sj, sk, sl}, also written tjkl, representing the set of services $\{s_j,s_k,s_l\}$

Support($supp(T_i)$): is the frequency of occurrence of T_i, i.e., the proportion of nodes $n \in N$ that request/provide Ti:

$$supp(T_i) = \frac{|\{n \in N | T_i \subseteq R(n, T_i)\}|}{|N|} \tag{10}$$

Frequent Transactions: A transaction T_i with support $supp(T_i) \geq T^{th}$ is called a frequent transaction where T^{th} is the *minsupp*. Moreover, to be a frequent transaction, all subsets of the transaction have to be frequent [39].

$$T_i \in Freq_n : \forall T_i \subseteq S, supp(T_i) \geq minsupp \\ and\ \forall T_j \subset T_i, T_j \in Freq_{n-1}, |T_j| = n-1. \tag{11}$$

$Freq_n$ and $Freq_{n-1}$ are the sets of frequent transactions of length n and $n-1$.

Maximal Frequent Transactions (MFT): A frequent transaction $\forall T_i \subseteq S$ and $T_i \in Freq$ is said to be an MFT if none of its proper supersets $T_j \supset T_i$ is frequent.

$$MFT = \bigcup_{i=1}^{n} Freq_i \\ T_i \in MFT : \nexists T_j \in MFT, T_j \supset T_i \tag{12}$$

Logical transactions are calculated and maintained in each edge gateway for every PCN. Each node in a PCN provides the transaction of the frequently used services (FUS) to its respective edge gateway. A *RequestSet* (R_{set}) generated by an edge-gateway (g_i) depicts the relation of nodes and FUS of nodes in a PCN. More specifically, the R_{set} of g_1 is:

$$R_{set}(g_1) = \{ \begin{matrix} \{d_1 : FUS(d_1), d_2 : FUS(d_2), d_5 : FUS(d_5), \\ e_1 : FUS(e_1), c_1 : FUS(c_1)\} \end{matrix}$$

Next, an initial transaction $T_{initial}$ is generated in iteration I1 with all the participating services in *Rset* as : $T_{initial} = \{t_i : t_i = s_i; \forall s_i \in R_{set}\}$. Frequent transaction of cardinality 1 is generated using Eq.(11) as:

$$Freq_{\mathbb{I}_1} = \{t_i : supp(t_i) \geq minsupp_1; \forall t_i \in T_{initial}\} \tag{13}$$

Now, in iteration I2, the closeness of two services is measured through their occurrence in an individual user by calculating frequent transaction of cardinality 2 using Eq.(11):

$$Freq_{\mathbb{I}_2} = \{t_{ij} : \forall t_i; t_j \in Freq_1 \wedge supp(t_{ij}) \geq minsupp_2\} \tag{14}$$

Similarly, in iteration I3, the closeness of three services is measured by calculating frequent transaction of cardinality 3 using Eq.(11) as:

$$Freq_{\mathbb{I}_3} = \{t_{ijk} : t_{ij} \cup t_{ik}; \forall t_{ij}, t_{ik}, t_{jk} \in Freq_{\mathbb{I}_2} \\ \wedge supp(t_{ijk}) \geq minsupp_3\} \tag{15}$$

Next, the frequent transaction sets of higher cardinalities are calculated in the same manner until no service is left or the next iteration produces an empty set. Logical transactions (LTN) in each PCN are obtained by computing MFT from frequent transactions of all iterations with Eq. (12):

$$LTN(g_i) = \bigcup_{j=1}^{n} Freq_{\mathbb{I}_j} \\ \forall T_k \in LTN(g_i) : \nexists T_l \in LTN(g_i); T_l \supset T_k \tag{16}$$

Now, these logical transactions of each PCN are shared with agents of all edge gateways. LCN is calculated by taking a union of the transactions from all (m) edge gateways and eliminating subset logical transactions.

$$.^L C = \bigcup_{i=1}^{m} LTN(g_i) \\ where\ each\ T_j \in .^L C : \nexists T_l \in .^L C; T_j \subset T_l \tag{17}$$

After the creation of contexts, the nodes are clustered according to LCN. Now, the relation between nodes and the contexts is defined as:

$$.^f L : x \rightarrow .^L C_{T_i} : x \in (N_A \cup N_E \cup N_c)\ and\ T_i \in .^L C \\ where\ FUS(x) = T_i$$

If the FUS of a node doesn't match any context, then the node is considered to be part of the default context ${}^{L}C_{Tdef}$.

$$.^{f}\, L: x \rightarrow .^{L}\, C_{T_{def}}: x \in (N_A \cup N_E \cup N_c)\ and\ T_i \in .^{L}\, C$$
$$if \nexists\, T_i: FUS(x) = T_i$$

Consider the network described in Section 4.1 with ${}^{L}C = \{T_1, T_2, T_3, T_{def}\}$. The relation function is established as : ${}^{f}L(d_1) = T1$; ${}^{f}L(d_2) = T_2$; ${}^{f}L(d_3) = T_2$; ${}^{f}L(d_4) = T_3$; ${}^{f}L(d_5) = T_2$; ${}^{f}L(d_6) = T_1$; ${}^{f}L(d_7) = T_1$; ${}^{f}L(d_8) = T_3$; fL(d9) = T2; fL(d10) = T1; fL(d11) = T3; fL(d12) = Tdef; fL(d13) = T2; fL(d14) = Tdef; fL(d15) = T3; ${}^{f}L(d_{16}) = T_2$; ${}^{f}L(e_1) = T_1$; ${}^{f}L(e_2) = T_2$; ${}^{f}L(e_3) = T3$; ${}^{f}L(e_4) = T_{def}$; ${}^{f}L(c_1) = T_3$; ${}^{f}L(c_2) = T_2$ and ${}^{f}L(c_3) = T1$; fL(c4) = T_{def}; ${}^{f}L(c_5) = T_1$; ${}^{f}L(c_6) = T_2$. Next, the LCN for T_1, T_2, T_3, T_{def} are calculated as:

$$.^{L}\, C_N(T_1) = \{d_1, d_6, d_7, d_{10}, e_1, c_3, c_5\};$$
$$.^{L}\, C_N(T_2) = \{d_2, d_3, d_5, d_9, d_{13}, d_{16}, e_2, c_2, c_6\};$$
$$.^{L}\, C_N(T_3) = \{d_4, d_8, d_{11}, d_{15}, e_3, c_1\};$$
$$.^{L}\, C_N(T_{def}) = \{d_{12}, d_{14}, e_4, c_4\}$$

Regeneration: When network parameters deteriorate, it triggers the regeneration of the Logical Context Network (LCN). A node is treated as a candidate for regenerating the Functional Unit Set (FUS) if there is a significant decline in Quality of Service (QoS). Such degradation may result from changes in device request patterns or due to service replication or replacement on any server. Upon detecting these variations, the node forwards the relevant information to its associated edge gateway. If the gateway receives multiple such inputs, it performs an analysis to identify shifts in the logical transaction pattern. Detected changes are then broadcast to all gateways, prompting an update in the logical context. In scenarios where a gateway experiences minimal request activity, the connected devices and servers revert to the default logical context ${}^{L}C_{Tdef}$. After persisting in ${}^{L}C_{Tdef}$ for a predefined duration, a complete regeneration of the logical context is initiated across all edge nodes.

4.3. Context Creation

Both PCN and LCN help to reduce the *search space* or reduce the network size considering various proximity. Moreover, during the replication decision, SOM considers the combined context of the related clients and servers and then chooses the most suitable (promising) nodes. Let us assume a replication request is triggered by d_2 (Considering the example discussed in Sections 4.1 and 4.2) and the corresponding PCN for replication is g_2. Next, LCN of d_2 is calculated as $f_L(d_2) = T_2$. The nodes associated with g2-PCN are:

$$P_{C_N}(g_2) = \{d_3, d_4, d_5, d_{13}, e_2, c_2, c_3\}$$

The set of nodes associated with the T2-LCN includes:

$$.^{L}\, C_N(T_2) = \{d_2, d_3, d_5, d_9, d_{13}, d_{16}, e_2, c_2, c_6\}$$

The nodes that belong to T2-LCN under g_2-PCN are:

$$I_{i,j} = .^{P}\, C_N(g_i) \cap .^{L}\, C_N(T_j)$$
$$I_{2,2} = .^{P}\, C_N(g_2) \cap .^{L}\, C_N(T_2)$$
$$= \{d_3, d_5, d_{13}, e_2, c_2\}$$

Next, the selected nodes are considered potential nodes that can participate in DCDP (discussed in the next section).

5. Batch-wise Replica Allocation Through DCDP

A DCDP [39] is defined over a set of agents $A = \{a_1, a_2, \ldots, a_{|A|}\}$, each capable of executing certain computational tasks from a global task set $T = \{t_1, t_2, \ldots, t_{|T|}\}$. For each agent $a_i \in A$, let $T_i \subseteq T$ denote the subset of tasks it can perform. For every task $t_k \in T$, we define $A_k \subseteq A$ as the subset of agents eligible to execute it. Each agent a_i is associated with a task-specific cost function $\Lambda_i : T_i \rightarrow N^{+}$, where $\Lambda_i(t_k)$ represents the total computational and communication cost for executing task t_k. In the context of service replication, T represents service instances to be deployed across the network and $\Lambda_i(t_k)$ is inversely proportional to the utility u_i of agent a_i, indicating that more capable agents incur lower costs. The goal is to assign tasks to agents such that the overall system load is balanced while satisfying functional constraints. Formally, we define an allocation mapping $\phi : A \rightarrow 2^{T}$, assigning tasks to agents under the following constraints:

$$\phi(a_i) \cap \phi(a_j) = \emptyset, \forall a_i, a_j \in A, i \neq j, \tag{18}$$

$$\bigcup_{a_i \in A} \phi(a_i) = T. \tag{19}$$

Table 2. Notation.

Notation	Meaning
u_i	Utility of agent a_i (inversely proportional to Λ_i)
ϕ	Allocation function mapping agents to subsets of tasks ($\phi : A \rightarrow 2^T$)
ϕ^*	Optimal allocation that minimizes the maximum agent cost
Φ	Set of all possible valid allocations
S_i	Subset of tasks allocated to agent a_i
$W_i(S_i)$	Potential function: total cost for agent a_i over S_i
S	Global task assignment configuration
S^*	Optimal task assignment minimizing the worst-case agent cost
$v_i \rightarrow j(S_{ij})$	Message passed from agent a_i to a_j over shared variables S_{ij}
S_{ij}	Shared task variables between agents a_i and a_j
$S_{i/j}$	Task variables in S_i excluding those in S_j
$N(a_i)$	Set of neighboring agents of a_i
$\theta_i(S_i)$	Marginal contribution estimate for agent a_i
λ_r	Number of requests generated by a node
fP, fL	Physical and logical context mapping

The optimization objective is to minimize the worst-case (maximum) cumulative cost incurred by any agent:

$$\phi^* = \underset{\phi \in \Phi}{argmin} \max_{a_i \in A} \sum_{t_k \in \phi(a_i)} \Lambda_i(t_k) \tag{20}$$

To model the cost incurred by each agent, we define a potential function $\omega_i(S_i)$, where S_i denotes the subset of tasks assigned to agent a_i: $\omega_i(S_i) = {}^{X}\Lambda_i(t_k)$.

$$\omega_i(S_i) = \sum_{t_k \in S_i} \Lambda_i(t_k) \tag{21}$$

Thus, the global objective becomes:

$$S^* = \underset{S}{argmin} \max_{a_i \in A} \omega_i(S_i) \tag{22}$$

To solve this optimization problem, we apply the Min-Max inference algorithm based on the Generalized Distributive Law (GDL) [40]. The algorithm proceeds via message passing among neighboring agents. Let $S_{ij} = S_i \cap S_j$ denote the shared task variables between agents a_i and a_j. Each agent a_i transmits a message $v_{i \rightarrow j}(S_{ij})$ to its neighbor a_j, initialized to zero. The update rule for messages is defined as:

$$v_{i \rightarrow j}(S_{ij}) = \underset{S_{i \setminus j}}{min} \max \left[\omega_i(S_i), \max_{\substack{a_k \in \mathcal{N}(a_i) \\ k \neq j}} v_{k \rightarrow i}(S_{ki}) \right], \tag{23}$$

where $N(a_i)$ denotes the set of neighbours of agent a_i. Next, each agent estimates its marginal contribution through:

$$\theta_i(S_i) = max \left[\omega_i(S_i), \max_{a_j \in \mathcal{N}(a_i)} v_{j \rightarrow i}(S_{ji}) \right] \tag{24}$$

This formulation guarantees exact marginal inference [41], ensuring that the computed configuration is optimal under the Min-Max criterion. To further enhance system responsiveness and Quality of Experience (QoE), we adopt a proactive, batch-wise replica allocation strategy, designed to improve resource utilization across the distributed network.

5.1. Proactive Sensing and Batch-Wise Replica Allocation

Instead of making replication decisions independently for each event, we consider the set of replication events triggered within a time window of duration Δt. During the n^{th} sensing window, i.e., the time interval $\{t_0 + n\Delta t\}$ to $\{t_0 + (n + 1)\Delta t\}$, suppose the network nodes generate |K| replication events. Let R denote the set of services that need to be replicated as a consequence of the event set K. The replication events are sent only to the concerned gateway. For each such request k_i, the SOM of the corresponding gateway approaches the SOMs of other gateways to find if the number of requests for the

requested services is biased to a few PCs or has a scattered distribution. If a bias is found, then nodes from all three layers (cloud, edge, access) are considered from each of the biased PCs ($^{P}C_{Nrepi}$). The search space for eligible servers gets reduced as the search is confined within the combined context of selected gateways ($g_{repi} \in {}^{P}C_{Nrepi}$) and requested logical context (T_{reqi}) computed as $I_{reqi} = {}^{P}C_N(g_{repi}) \cap {}^{L}C_N(T_{reqi})$ (Algorithm 1) (discussed in Section 4.2.C). If the number of replication requests for the requested replica is independent of the PCs, this reflects the uniformity of demand across the physical contexts. Then DCDP considerss all the cloud nodes within the requested logical context ($I_{reqi} = N_{Cs} \cap {}^{L}C_N(T_{reqi})$). If any replication request (req_i) is generated from any node in $g_1\,{}^{P}C_N$ (example from Section 4), then the replication request is forwarded to the g_1 gateway. SOM then checks if the demand for the requested replica is biased or likely from each gateway. Suppose the demand is higher from g_1 and g_4 gateways, then the search is confined within the combined context of ${}^{P}C_N(g_{repi})$ = $\{g_1, g_2\}$ and the requested logical context. For instance, if the requested service belongs to $T_2\,{}^{L}C_N\,(T_{reqi} = T_2)$, then the eligible nodes for that replication request are computed as $I_{reqi} = ({}^{P}C_N(g_2) \cup {}^{P}C_N(g_4)) \cap {}^{L}C_N(T_2)$. DCDP considers only $d_2, d_3, d_5, e_2, c_2, c_6$ nodes for req_i. If the demand is reckoned to be uniform, then only the c_2, c_6 are considered. After analyzing all the generated requests (req) within the n^{th} time window, each service provider undergoes a threshold evaluation based on the resource requirements of the corresponding tasks. A filtered set of eligible providers is then computed as $I'_{req_i} = \text{trim}(I_{reqi})$. Subsequently, the DCDP is invoked using $DCDP(\mathrm{R}, I')$, where R is the set of services to be replicated and I' is the union of all filtered provider sets. The algorithm determines the optimal mapping $m^* : \mathrm{R} \rightarrow I'$ for replica allocation. Once the assignment is finalized, a response message $RepReq(m^*)$ is broadcast to all replication requesters. Upon receiving the *RepReq* message, each requesting node verifies whether the corresponding triggering parameters exceed predefined upper thresholds. If the condition is satisfied, an explicit CONFIRM message is sent to the designated provider node, initiating the replication process between the current and newly assigned servers. It is important to note that the set of requests *Req* captured in the n^{th} time slot is processed by the DCDP mechanism during the $(n+1)^{\text{th}}$ slot to determine the service-provider assignments. The actual replication process commences immediately after this decision epoch. The time required for each replication may vary depending on the specific service characteristics. During the n^{th} sensing window, the DCDP-based decision-making for services identified in the $(n-1)^{\text{th}}$ slot occurs in parallel. Moreover, all individual replication processes can proceed concurrently with both past and current decision phases, independent of their respective decision epochs. In our proposed scheme, the environment is segregated into different physical context networks according to network decisions regarding edge gateway and logical context networks according to their requesting/providing behavior. We considered a batch of replication requests sensed in the previous window to participate in DCDP to find a server for each request.

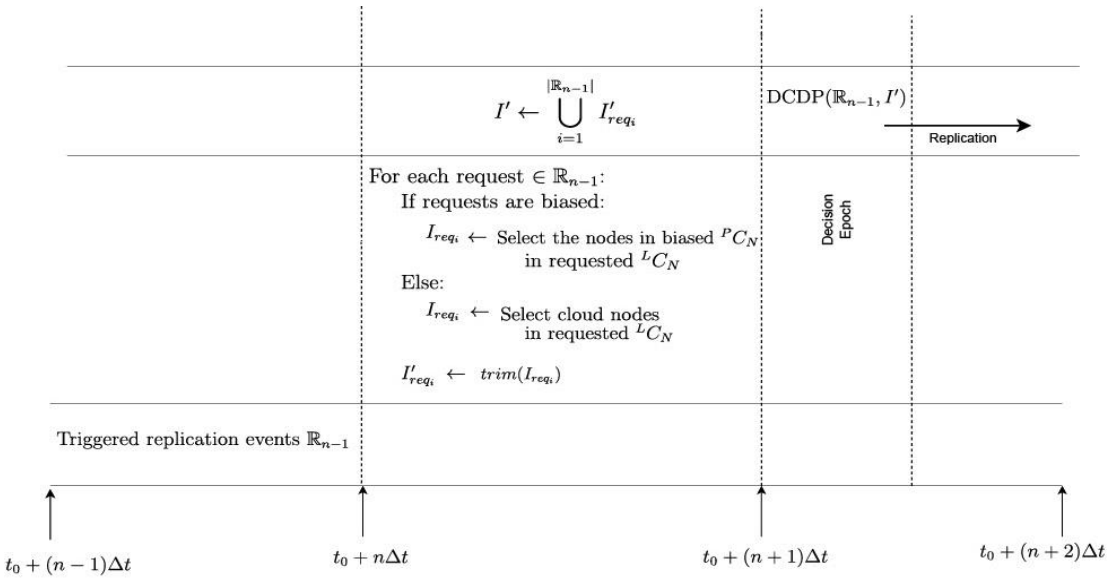

Fig. 2. Proactive Sensing and Replication Allocation in Batches.

Algorithm 1 Eligible Server Selection

procedure *Replication_Decision* ($\mathbb{R}$)
  for *Each* $req_i : s_{req_i} \in \mathbb{R}$ **do** // s_{req_i} service of each replication request
    ${}^{P}C_{N_{rep_i}} \leftarrow Choose\_PCN(req_i)$ // Region selection
  end for
  for *Each* $req_i \in \mathbb{R}$ **do**
    $T_{req_i} \leftarrow findT(s_{req_i})$ // Getting requested transaction
    if ${}^{P}C_{N_{rep_i}}$ *is not empty* **then**
      for *each* $g_{rep_j} \in {}^{P}C_{N_{rep_i}}$ **do**
        $I_{req_i} = {}^{P}C_N(g_{rep_j}) \cap {}^{L}C_N(T_{req_i})$
      end for
    else
      $I_{req_i} = N_{C_s} \cap {}^{L}C_N(T_{req_i})$
    end if
    $I'_{req_i} = trim(I_{req_i})$ // Based on u_i of server
    $I' \leftarrow I'_{req_i}$
  end for
  DCDP($\mathbb{R}, I'$)
end procedure

Fig. 3. Eligible Server Selection.

i. In Phase-I, the middleware searches for the bias in each requesting pattern for each requested service and chooses the PCN accordingly.
ii. In Phase-II, the optimal server is identified within a reduced search space, specifically located at the intersection of the LCN and the selected PCN. The proposed scheme incorporates a lightweight replication mechanism. In cases where sufficient resources are not available to replicate the requested service, the DCDP framework selects a minimally utilized service, already hosted by a server within the same LCN-PCN intersection, for replacement.

6. Overhead

The main source of overhead in our approach comes from the message exchanges and the convergence time required by the distributed algorithm.

6.1. Logical Context

In the computation of the communication and computation cost of logical context, the major contribution comes from LTN. To compute LTN, each node must share its FUS with the respective edge gateway. This exchange involves O(|*N*|) messages, where *N* denotes the set of nodes. Notably, this communication is only triggered during the reconstruction of the logical context, an infrequent event typically initiated by a noticeable degradation in network-wide QoS. Since each edge gateway computes the LTN independently and in parallel, the associated data volume and computation time remain minimal relative to the scale of the network. Once computed, the dissemination of LTNs among edge gateways necessitates O($|N_{Es}|^2$) messages, where *NEs represent* the set of edge nodes.

To calculate frequent sets at each level $k \geq 1$, candidates are generated from the lower cardinality to compare with *minsupp*. To generate candidates for $Freq_1k$ from $Freq_1k_{-1}$, all members of $Freq_1k_{-1}$ are joined to each other, incurring a cost of O($|Freq_1k_{-1}|^2$). The total frequent set generation cost up to $Freq_1k_{max}$ is:

$$T_{freq} = \sum_{k=1}^{k_{max}} \mathcal{O}\left(. \left|Freq_{\mathbb{I}_{k-1}}\right|^2\right) \tag{25}$$

After the generation of frequent sets (F_{freq}), each $Freq_1 \in F_{freq}$ is checked for maximality verification. Each frequent set is compared with other frequent sets to verify whether there is a superset. Since superset verification requires exploration of elements of each set, the comparison cost is proportional to the cardinality of the set. Each member of F_{freq} requires a pairwise comparison at an average cardinality cost *s*:

$$T_{maximal} = \mathcal{O}\left|(F_{freq}\right|^2 \cdot s) \tag{26}$$

The total complexity to calculate LTN is:

$$T_{LTN} = T_{freq} + T_{maximal} = \sum_{k=1}^{k_{max}} \mathcal{O}\left(. \left|Freq_{\mathbb{I}_{k-1}}\right|^2\right) + \mathcal{O}\left(|F_{freq}|^2 \cdot s\right) \tag{27}$$

6.2. DCDP Overhead

The communication complexity of solving the DCDP using the Min-Max approach is $O\left(.\left|\mathbb{R}\right|^{I'_{req}}\right)$, where R represents the batch of services to be replicated and $I_{req}^{'}$ denotes the maximum number of eligible servers for any requested service. In real-world applications, both the number of servers that offer a particular service and the number of services supported by each server are typically small. As a result, the overhead can be managed effectively by either limiting the size of R (through a smaller batch interval Δt) or restricting $I_{req}^{'}$. If convergence time becomes too long, as manifested by higher response times or increased drop rates, the control parameters (Δt and $I_{req}^{'}$) are dynamically adjusted to accelerate convergence.

7. Performance Evaluation

We derive the performance evaluation parameters for three distinct system types: a system without replication, a system with reactive replication and a system with context-aware proactive replication. Service requests are generated by users, each with specific Quality of Service (QoS) requirements, as discussed in Section 8. Some requests prioritize response time, completion time, or both, while others exhibit flexibility in their QoS demands. We assume that the request arrival follows a Poisson process.

Let R denote the total system resources, S the total number of services in the system, C the number of contexts in the system and α_m the total number of devices. N_A refers to the total number of access servers at any given instant, where $NA \leq \alpha m$; N_E represents the total number of edge nodes and N_C is the total number of cloud nodes. p is the probability that a node N_i generates a replication request for a specific service. We assume a uniform distribution for the allocation of resources, such that requests in the system without replication can access $\frac{R}{S}$ resources. For systems employing replication, the number of replicas varies according to service demand over time.

7.1. Service Drop Rate

7.1.1. System without Replication

Assume that an adaptive and elastic cloud environment is operational and it has created the necessary number of replicas within the edge and access networks. Let λ_r represent the number of requests generated per node and α_m be the total number of nodes (devices) in the access network. Consequently, the total number of requests generated across the network is given by:

$$i_a = \lambda_r \cdot \alpha_m$$

Let the probability of getting a service executed in the access network be Pa, then the probability of requests not served in the access network will be $(1 - P_a)$. The number of requests considered for allocation in the edge network is:

$$i_e = (1 - P_a)\lambda_r \alpha_m$$

Let the probability of getting a service executed in the edge network be Pe, then the probability of requests not served in the edge network will be $(1 - P_e)$. The number of requests considered for allocation in the core network is:

$$i_c = (1 - P_e)(1 - P_a)\lambda_r \alpha_m$$

Let Pc be the probability of a service getting executed in the cloud layer.

$$P_a = \binom{\alpha_m - 1}{i_a} p^{i_a}(1-p)^{\alpha_m - 1 - i_a} \tag{28}$$

$$P_e = (1 - P_a)\binom{N_E - 1}{i_e} p^{i_e}(1-p)^{N_E - 1 - i_e}$$

$$P_c = (1 - P_e)(1 - P_a)\binom{N_C - 1}{i_c} p^{i_c}(1-p)^{N_C - 1 - i_c}$$

Let $\frac{R_a}{S}, \frac{R_e}{S}$ and $\frac{R_c}{S}$ can afford k_a, k_e and k_c number of requests for service in access edge and cloud respectively. Hence, a new request for a service will get dropped if all the resources are allocated by k_a, k_e and k_c number of requests in a system without replication. Let the probability of a request being dropped be denoted by PDWR, given by:

$$P_{DWR} = 1 - \sum_{i=0}^{k_a} P_a + \sum_{i=0}^{k_e} P_e + \sum_{i=0}^{k_c} P_c \tag{29}$$

7.1.2. System with Reactive Replication

Let P_{DRR} denote the drop rate observed in the reactive replication system. In this model, replication is initiated only when an incoming request cannot be fulfilled due to insufficient available resources. If the replication process completes quickly enough to satisfy the response time constraint, the request is successfully served and not dropped.

$$P_{DRR|during\_replication} = P_{DWR} - (P_{saved_a} + P_{saved_e} + P_{saved_c}) \tag{30}$$

where $P_{saved_a}, P_{saved_e} and\ P_{saved_c}$ are the probabilities of a request being saved in access, edge and cloud, given as:

$$\begin{aligned} P_{saved_a} &= P_a probability(\bar{t}_{rep} < t_{rrt})(i_a - k_a) \\ P_{saved_e} &= P_e probability(\bar{t}_{rep} < t_{rrt})(i_e - k_e) \\ P_{saved_c} &= P_c probability(\bar{t}_{rep} < t_{rrt})(i_c - k_c) \end{aligned} \tag{31}$$

considering $\bar{t}_{rep}$ as the time required to complete replication of a service and t_{rrt} as the required response time constraint. Hence

$$\begin{aligned} P_{DRR|during\_replication} = P_{DWR} - (probability(\bar{t}_{rep} < t_{rrt}) \\ (P_a(i_a - k_a) + P_e(i_e - k_e) \\ + P_c(i_c - k_c))) \end{aligned} \tag{32}$$

After completion of the replication process, $k' = k'_a + k'_e + k'_c$ additional resources are added to the access, edge and cloud networks. Hence

$$P_{DRR|after\_replication} = \quad P_{DWR} - \sum_{i=0}^{k'_a} P_a + \sum_{i=0}^{k'_e} P_e + \sum_{i=0}^{k'_c} P_c \tag{33}$$

Assuming that the probability of the system operating in a stable state is p_s, the probability of the system entering a replication state can be expressed as $(1 - p_s)$.

$$P_{DRR} \quad = p_s P_{DRR|after\_replication} + (1 - p_s) P_{DRR|during\_replication} \tag{34}$$

The above derivations do not account for the revocation of replicas. In practice, when the demand for a service declines, existing replicas are withdrawn to free up system resources.

7.1.3. Context-Aware Proactive Replication

Let t_s represent the time taken to search for eligible nodes for service replication, t_{decision} denote the time required to decide which node should replicate the service and $t_{\text{replication}}$ be the duration needed to complete the replication process.

$$\bar{t}_{rep} = t_s + t_{decision} + t_{replication} \tag{35}$$

In a context-aware system, the search space for identifying eligible nodes is significantly reduced, leading to an approximate reduction of t_s by a factor of C. Assuming the decision-making algorithm has a time complexity of $O(n^2)$, the incorporation of contextual information reduces this complexity by a factor of $\frac{1}{C^2}$. Therefore, the total time required for replication in a context-aware system, denoted as $\bar{t}_{rep_{CA}}$, can be expressed as:

$$\bar{t}_{rep_{CA}} = \frac{1}{C} t_s + \frac{1}{C^2} t_{decision} + t_{replication} \tag{36}$$

From Eq. (36) it is evident that adding context to the system reduces the replication time, resulting in a reduction in drop rate. In proactive replication, the replication process gets started after sensing the system generates the replication request depending on sensing the increase in the number of requests. Let the replication process while serving $k = k_a + k_e + k_c$ number of requests and add resources for $k'' = k''_a + k''_e + k''_c$ new requests in three layers before incurring a loss. Let drop rate of the context aware proactive system is denoted as P_{DPR}.

$$P_{DPR|during_{replication}} = P_{DWR} - (probability(\overline{t}_{rep_{CA}} < t_{rrt}) \\ (P_a(i_a - k_a) + P_e(i_e - k_e) \\ + P_c(i_c - k_c))) \tag{37}$$

Now as $t_{repCA} < t_{rep}$ more requests are saved during replication.

$$P_{DPR|during\_replication} < P_{DRR|during\_replication} \tag{38}$$

Now after replicating resources, the drop rate in a stable state is given as:

$$P_{DRR|after\_replication} = P_{DWR} - \sum_{i=0}^{k_a''} P_a + \sum_{i=0}^{k_e''} P_e + \sum_{i=0}^{k_c''} P_c \tag{39}$$

Now as the replication process starts much earlier in the proactive system than in the reactive system, it provides more resources to allocate i_a, i_e and i_c requests, so $k + k'' > k + k'$ results in

$$P_{DPR|after\_replication} < P_{DRR|after\_replication} \tag{40}$$

Taking a similar assumption of the system being in a stable state and replicating state we get the total drop rate for the context-aware proactive system as:

$$P_{DPR} = p_s P_{DPR|after\_replication} + (1 - p_s) P_{DPR|during\_replication} \tag{41}$$

From Eq.38 and Eq.40 we can derive that

$$P_{DPR} < P_{DRR} \tag{42}$$

The revocation of replicas has a similar effect on proactive systems as on reactive systems.

7.2. System Throughput and Resource Utilization

A substantial reduction in the drop rate positively affects the throughput and results in better resource utilization. We assume that each node has an identical request probability p. The total request generation probability across all nodes contributes to the system's load, denoted as L. Therefore, the request throughput of the system can be derived as:

$$Q_{WR} = (1 - P_{DWR})L \\ Q_{RR} = (1 - P_{DRR})L \\ Q_{PR} = (1 - P_{DPR})L \tag{43}$$

Now, as R denotes the maximum available resources we get the utilization of the system as:

$$U_{WR} = \frac{1 - P_{DWR}}{R} \\ U_{RR} = \frac{1 - P_{DRR}}{R} \\ U_{PR} = \frac{1 - P_{DPR}}{R} \tag{44}$$

7.3. Response Time

i) The response time of the system without replication is given as:

$$t_r = t_s + t_{delay} \tag{45}$$

Under the assumption of Poisson network traffic, the delay t_{delay} is approximately given by:

$$t_{delay} \simeq 2\overline{h} \cdot \frac{1}{\mu - \lambda_r \alpha_m}$$

where h represents the average number of hops from the client node to the server node and μ is the service rate. If a distributed hash table (DHT)-based service registry is used, the search process can be completed in $O(\alpha_m)$.

$$t_{r_{WR}} = \mathcal{O}(log(\alpha_m)) + 2\overline{h}\frac{1}{\mu - \lambda_r \alpha_m} \tag{46}$$

ii)In a context-aware reactive replication system, the search space is significantly reduced. The number of hops between the client and server depends on the number of nodes in the requested set ${}^{L}C_N$ and the selected set ${}^{P}C_N$, which is bounded by the intersection $|{}^{P}C_N \cap \text{LCN}|$. The maximum number of hops is given by $|{}^{P}C_N|$, assuming that all nodes in the selected set ${}^{P}C_N$ are present within the requested set ${}^{L}C_N$.

$$t_{r_{WR}} = \mathcal{O}(log(\alpha_m)) + 2\overline{h}\frac{1}{\mu - \lambda_r \alpha_m} \tag{47}$$

The maximum possible value of α'_m for a network with average connectivity c and a maximum hop count of $|{}^{P}C_N|$ can be computed as:

$$\alpha'_m = \frac{(c^{|{}^{P}C_N|+1} - 1)}{(c-1)}$$

A ${}^{P}C_N$ with higher connectivity results in a smaller number of hops. The decision time t_{decision} refers to the time taken to choose the serving server, which in our work is determined by the DCDP. This is applied to each replication instance to identify the optimal node.

iii) In our proposed context-aware proactive selection, DCDP is applied to a batch of R requests within the combined physical and logical context. As a result, the response time can be expressed as follows:

$$\begin{aligned} t_{r_{PR}} \quad = \quad & (1 - P_{DPR})\{\mathcal{O}(\tfrac{log(\alpha'_m)}{c}) + \tfrac{|{}^{P}C_N|}{\mu - \lambda_r \alpha_m}\} \\ . \quad & + P_{DRR}\{\mathcal{O}(\tfrac{log(\alpha'_m)}{c}) + \tfrac{1}{\mathbb{R}} t_{decision} + \tfrac{\overline{h}}{\mu - \lambda_r \alpha_m}\} \end{aligned} \tag{48}$$

7.4. Completion Time

The service completion time t_c is typically calculated as:

$$t_c = t_r + t_e$$

where t_e represents the execution time required to fulfill the requested service. The workload of a server depends on the number of services it must handle concurrently. Let S_{con} be the number of services present in the system at any given time that need to be served simultaneously. In a system with uniform distribution, each node serves $\frac{S_{\text{con}}}{\alpha_m}$ services. If X represents the CPU slots required to execute a single service, the execution time for a system without replication is $\left(\frac{S_{\text{con}}}{\alpha_m} \cdot X\right)$. The value of S_{con} corresponds to the system's throughput. Therefore, the execution time for the system without replication, with reactive replication and with context-aware proactive replication can be expressed as:

$$\begin{aligned} t_{e_{WR}} &= \frac{Q_{WR}}{\alpha_m} X \\ t_{e_{RR}} &= \frac{Q_{RR}}{\alpha_m} X \\ t_{e_{PR}} &= \frac{Q_{PR}}{\alpha_m} X \end{aligned} \tag{49}$$

8. Result

We evaluate the performance of our Elastic Context-Aware Replication (ECAR) scheme in comparison with several existing approaches, including the Cloud-Intensive Replication (CIR) scheme, the Cloud-Edge-Intensive Replication (CEIR) scheme, the Cloud-Edge-Access-Intensive Replication (CEAIR) scheme, the Group-Delay-Aware Scalable Mobile Edge Computing with Service Replication (GDAR) scheme [16] and a Combined Context-Aware Replication (CCA) [22] along with analytical results obtained in an identical environment [43].

1. CIR: We evaluate our proposed scheme in a cloud-intensive scenario where only core or cloud is considered as the service provider.

2. CEIR: In a cloud-edge-intensive scenario, both the cloud and edge layers are considered service providers. Initially, replicas are present in the cloud layer, but due to on-demand replication, services are available in the cloud and edge.
3. CEAIR: This scenario considers the cloud, edge and access devices jointly as service providers. Initially, replicas are present in the cloud layer, but due to on-demand replication, services are available in the cloud, edge and access layers.
4. GDAR: This is an existing scheme [16] that includes cloud and edge layers as service providers. The replication strategy aims to replicate services on edge servers within selected regions. If the selected region's servers are overloaded, comparatively underloaded remote edge servers are considered.
5. CCA: This is also an existing scheme [22] that introduces a proactive context-aware replication strategy, specifically targeted for a quasi-ad hoc scenario in IoT. The scheme makes use of segregation of all services in both logical and physical dimensions and combines them to form clusters that help in reducing registry search space and message passing overhead.
6. ECAR: In our proposed scheme, Elastic Context-Aware Replication (ECAR), we consider that a replica can be present in the cloud, edge and access layers. Moreover, to reduce the overall burden, the whole network is segregated into PCN and LCN.

8.1. Experimental Setup

We create an IoT scenario using the *NetworkX* package in Python 3.7.0. The setup includes 35 cloud nodes, 90 edge nodes and 900 user-end nodes. The distance of inter-cloud nodes ranges from 15 to 350 hops and for inter-edge nodes, it ranges from 10 to 700 hops. Each cloud node connects to 4 to 5 edge nodes, with distances ranging from 20 to 50 hops. As a result, a single edge node is connected to multiple cloud nodes. Devices in a PCN are located within 5 to 10 hops from the edge gateway. We assume each hop introduces an average delay of 1 ms. In our experiment, we adopt a Smart City Scenario [42] that involves 15 distinct services with QoS pattern described in Table 4. We ensure a clear separation among cloud, edge and access nodes as shown in Table 3. Each cloud node can serve 5 services, with each service having 4 instances; each edge node can serve 3 services, with 3 instances per service; and each access server can serve 2 services, with 2 instances per service. The maximum load capacity is 16 instances for cloud nodes, 7 for edge nodes and 5 for access nodes.

8.2. QoS Types

Traditionally, the Quality of Service (QoS) for interactive services has been primarily measured through the service's response time (t_r). However, with the growing complexity of content delivery systems and the increasing demand for compute-intensive services, service completion time (t_c) has become a critical factor. In our experiment, we categorize services into four distinct types based on their specific QoS requirements:

a) Services that prioritize response time (t_r) over completion time (t_c), referred to as *Type1* services. These are typically end-user services that require interactive responses.
b) Services that emphasize minimizing completion time (t_c) over response time (t_r), known as *Type2* services. These services are usually programmatic, where the end user is a program that needs to update or reconfigure itself.
c) *Type3* services, which have stringent constraints on both response time (t_r) and completion time (t_c). Examples include interactive video or voice calls, where both timing parameters must be met.
d) Services with no strict QoS requirements, classified as *Type4* services. Examples include background file transfers like messages or emails, where response time and completion time are less critical.

Table 3. Types of QoS.

Services	QoS Pattern
*Type*1	200*ms*, 300*ms*, 400*ms*
*Type*2	400*ms*, 500*ms*, 600*ms*
*Type*3	(200*ms*, 400*ms*), (300*ms*, 500*ms*), (400*ms*, 600*ms*)
*Type*4	No QoS

8.3. Result

8.3.1 Drop Rate

We calculate the percentage of requests dropped in each scheme shown in Fig. 4 We observe that CEIR and GDAR exhibit lower drop rates compared to CIR. This improvement is due to the expansion of compatible servers when the edge layer is utilized as a resource-provider. Initially, the performance of CEAIR declined due to the increased search space (Fig.8.a). However, during increased load, CEAIR performs well because the inclusion of the access layer increases the number of compatible servers. CCA, with only the cloud and access layers, has a lower drop rate due to its better utilization of resources (Fig. 9(b)) through distributed allocation and context evaluation. On the other hand, ECAR, with

logical context, significantly reduces response time (Fig.8.b.), completion time (Fig. 7(a)) and enhances resource utilization (Fig. 9(b)), resulting in the lowest drop rate among all the evaluated schemes.

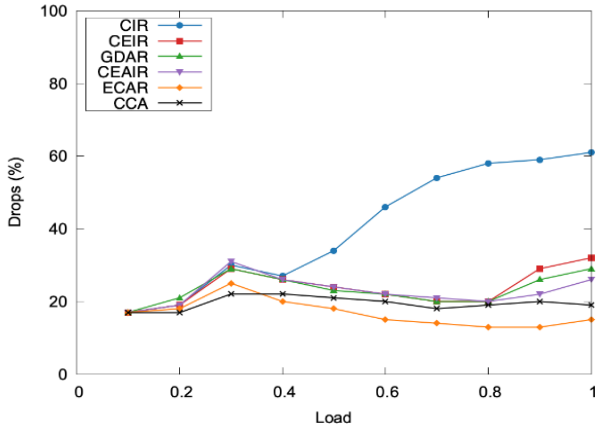

Fig. 4. Percentage of Drop Rate.

8.3.2. Replication

We evaluated the number of replications orchestrated by each scheme in Fig.5.a. Initially, only half of the cloud nodes were designated as service providers. The replication strategy is consistent for all schemes in deciding the PCN of replication. For CIR, replication was limited to cloud nodes; for CEIR, both cloud and edge nodes were considered; and for CEAIR and ECAR, cloud, edge and access-device nodes were included. We observed that CIR had the highest replication rate due to its higher drop rate, which compelled the SOM to perform more replication or replacement to maintain service availability. Due to the increased drop rate, the rate of replication increases significantly with increasing load, especially when the load is more than 60%. Since CIR had fewer resource-providing opportunities, it also experienced the highest service replacement rate (Fig.5.b). CEIR and GDAR exhibited similar replication rates throughout the experiment, as both schemes utilized the same number of resource-providing options. Replication in CCA uses both cloud and access nodes as potential target nodes. Including more devices as a resource-providing alternative helps to achieve higher allocation (Fig.6.a). CEAIR, with its broader range of resource-providing options, initially replicated more services than all other schemes until CIR's replication rate surpassed it at higher load. Due to the higher number of nodes available for replication, CEAIR had a lower percentage of replacements. In contrast, ECAR achieved the lowest replication rate while attaining the highest allocation rate (Fig.6.a). This depicts that ECAR's efficient resource-utilization (Fig.8.b) minimizes replication needs while maximizing resource allocation.

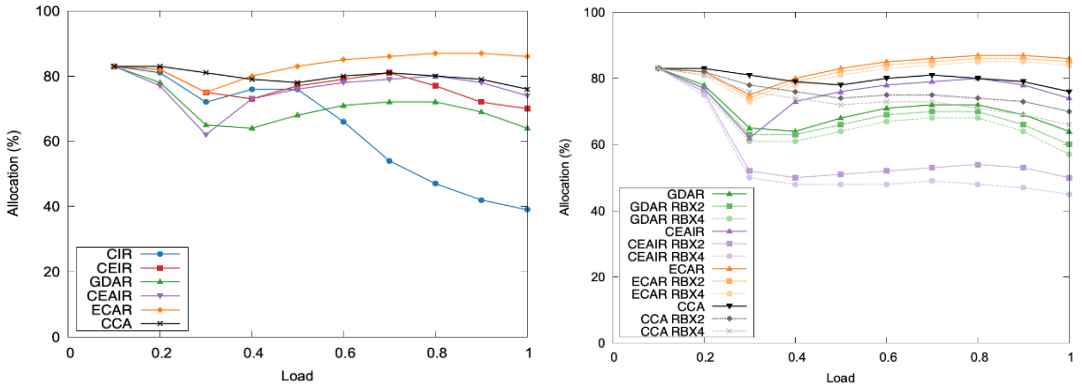

Fig. 5. (a) Number of Replication, (b) Percentage of Replacement.

8.3.3. Allocation

We evaluate the request-to-server allocation rates for all the schemes under identical environments. Initially, all schemes successfully allocated over 80% of requests. However, as the load increased, CIR's allocation rate dropped to around 35% at peak load. CEIR, which includes the edge layer as a service provider, maintained an allocation rate between 70% and 80%, eventually settling at 68% under maximum load. The allocation performance of GDAR was similar to CEIR, with a 4% to 5% higher allocation rate. The inclusion of devices as resource-providing options in CEAIR provided an opportunity for the SOM to replicate services at the access layer, thereby improving allocation rates. However, managing this increased number of resources proved time-consuming (Fig.8), which led to less optimal results.

Moreover, with only the cloud and access layers as resource providers, CCA with an efficient context management produces more replicas (Fig. 5(a)) and better resource utilization (Fig.8.b), resulting in a similar allocation (76%). On the other hand, ECAR, leveraging context-based management and three-tier resource providing alternatives, significantly improved allocation, maintaining a consistent 80% to 85% allocation rate throughout the experiment, even as the load increased. If we increase the overhead for each replication, the request-allocation decreases for all the schemes except ECAR (Fig.6.b).

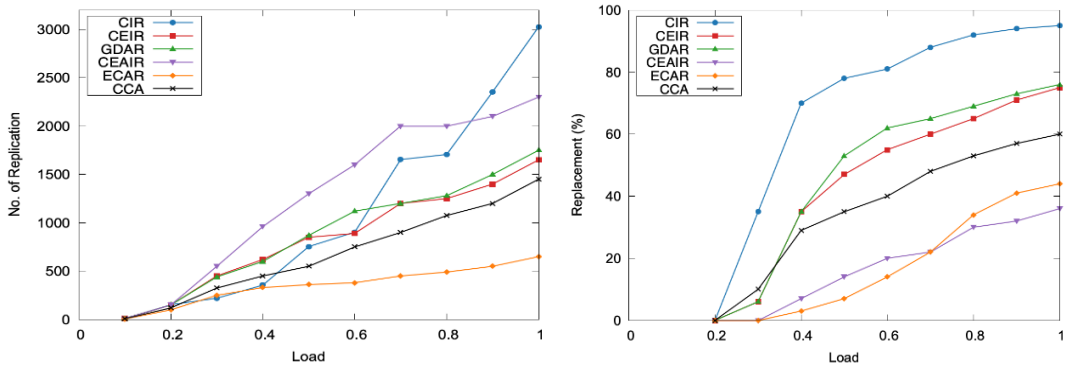

Fig. 6. (a) Percentage of Allocation, (b) Allocation with Increased Replication Overhead.

Specifically, when the replication overhead (two-times and four-times) increases, the allocation efficiency of the state-of-the-art GDAR framework declines to 63% and 58%, and CCA's to 70% and 60%, respectively. Similarly, CEAIR, operating with resource parity to ECAR, exhibited a reduction in allocation efficiency to 48% and subsequently to 44%. In contrast, ECAR demonstrated robustness by maintaining a stable allocation rate, demonstrating its advanced context-aware replication strategy that optimizes resource utilization even under escalated load conditions.

8.3.4. Completion Time

Fig. 7(a) illustrates the comparison of average completion time for all schemes, showing the cumulative time taken by non-dropped requests to complete their tasks. As the initial scenario contains only half of the cloud nodes as resource-provider, the replication improved resource availability, leading to a decrease in average completion time for all schemes.

As more resources were available, CIR's average completion time initially decreased. However, with increasing load, the completion time increases as replication is confined to the cloud layer. A similar increase was observed in CEIR and GDAR to a lesser extent, as they had more resource providers in both the cloud and edge layers. Servers in the edge layer reduce the distance between requests and resources, lowering the completion time. By consolidating the access layer in place of the edge layer as a service provider, CCA brings resources closer to requests, thus decreasing completion time. In contrast, CEAIR and ECAR did not experience an increase in completion time as both have more and nearer resource-providing options in edge and access. Notably, ECAR outperformed CEAIR by incorporating logical context, which reduced search time (Fig. 8(a)) and response time (Fig.7.b.).

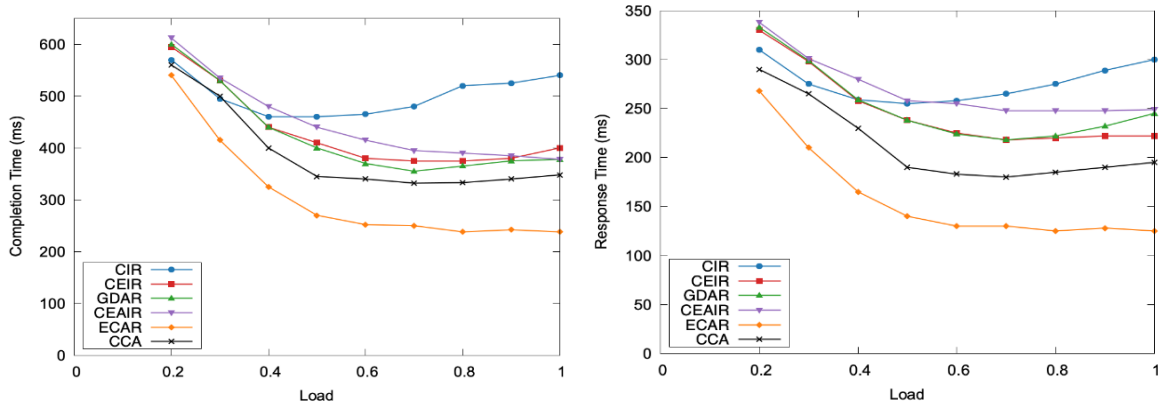

Fig. 7. (a) Average Completion Time, (b) Average Response Time.

8.3.5. Response Time

We calculate the average response time for each scheme and plot the results in Fig.7.b. Initially, the response time

decreases for all schemes as the replication process increases the resource availability. CIR exhibits the highest response time since the presence of servers is limited to only the cloud layer. Initially, the larger search space (Fig.8.a) in CEIR and GDAR results in a higher response time. However, the increase of available resources allows these schemes to perform better than CIR. CEAIR, despite having more and closer resource providers, shows a consistently higher response time. The expanded search space in CEAIR offsets the benefits of having additional resource-providing options. In contrast, CCA with similar resource providers across cloud and access layers, achieves lower response time by leveraging context. On the other hand, ECAR operates across all three layers, yet its effective context management significantly reduces the search space (Fig.8.a) and results in lower response time.

8.3.6. Average Number of Nodes Searched for Allocation

Next, we evaluate the average number of node searches needed to allocate a request by each of the schemes (Fig. 8(a)). CIR requires the lowest number of searches among the four schemes, as its search is limited to the cloud layer. In contrast, CEIR expands the search space with the inclusion of both cloud and edge layers, resulting in a higher average number of node searches needed for request allocation. Although GDAR has the same total resource-providing options as CEIR, it requires more searches due to its replication strategy, which replicates resources in either a local or remote edge server group without considering neighboring server groups in the selected region. As CEAIR includes access servers, the search space further increases. The growing load and gradual replication across three layers extend the search space, leading to an increase in the number of searches needed for request allocation. The context-awareness of CCA reduces the search space, although it has higher replication (Fig.5.a.) and extended resource provisioning alternatives. Despite having three layers of resource providers, ECAR achieves the lowest number of node searches by reducing the search space. The logical-context restricts the search of nodes for allocation within the servers of that requesting service's logical context across three layers, efficiently managing resources across all three layers.

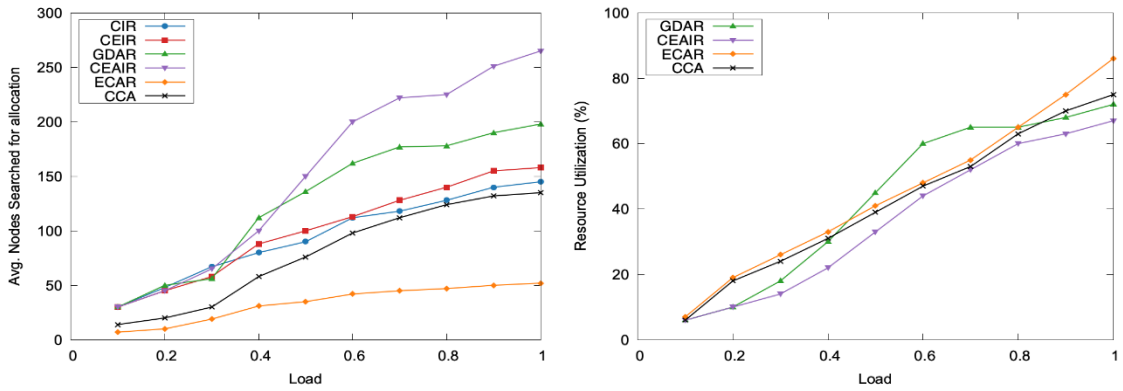

Fig. 8. (a) Average Node Search for Allocation, (b) Percentage of Resource Utilization.

8.3.7. Resource Utilization

We compare the utilization of resources as the ratio of utilized resources to total resources. The results shown in Fig. 8(b), compare our scheme ECAR with CEAIR (which has an equal number of resources) and with the existing GDAR and CCA. We observed that GDAR and CCA achieve better resource utilization than CEAIR. The larger number of alternatives in CEAIR allows it to allocate more requests than GDAR. CCA with better utilization of access layer achieves higher allocation. The added context in ECAR, enables it to effectively utilize more resources, resulting in a higher utilization rate.

8.3.8. QoS Based Performance Analysis

To evaluate the impact of our proposed ECAR scheme on four different QoS types, Fig.9.a. presents a multi-metric comparison of ECAR across four types in terms of allocation rate, drop rate, average response time and average completion time. Services of *Type*3 achieve the lowest allocation rate (62.7%), because of stringent response and completion time constraints. The stringent time constraints impose the most restrictive selection criteria on the allocation process and limit the selection within low latency servers. *Type*1 services manage to allocate 68.4% due to their relaxation on the completion time constraint. Though the stringent response time limits ECAR to low-latency servers in proximity, it helps in achieving the lowest average response and completion time. *Type*2 services achieve 87.6% allocation. The relaxed deadline constraints over response time allow ECAR to include high-latency servers. With the inclusion of high-latency servers, *Type2* services incur higher average response and completion time than *Type*1 and *Type*3 services. Services of *Type*4 achieve the highest allocation rate of 97.2%. The absence of any stringent time constraint permits ECAR to select from the complete pool of available servers. However, it experiences the highest average response time.

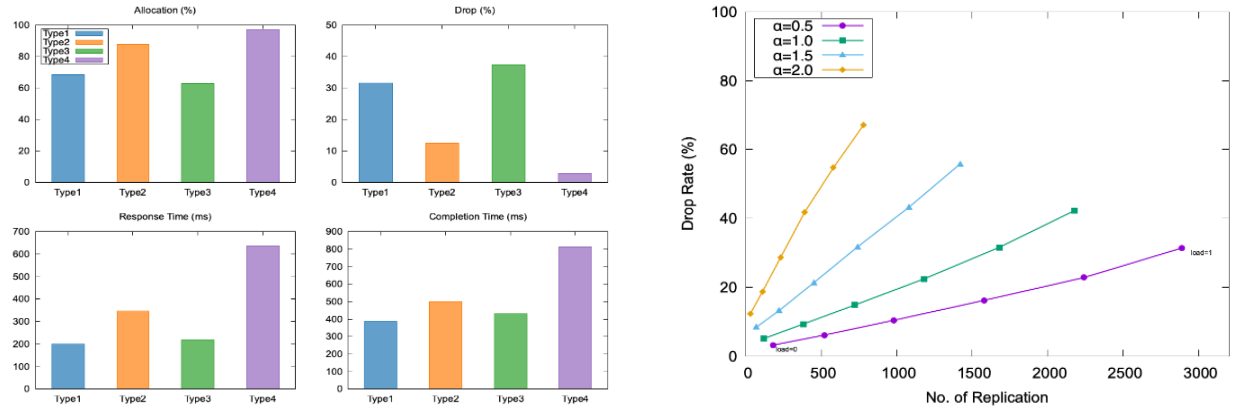

Fig. 9. (a) QoS based performance Evaluation, (b) Effects of Replication Threshold.

8.3.9. Effects of Replication Threshold

To examine the sensitivity of our proposed model to the replication threshold parameter α, we investigate the joint behavior of replication count and drop rate under varying network load. We present (Fig.9.b) a trade-off graph in the replication-drop rate space rather than analyzing these two metrics separately. Each line corresponds to a fixed value of α and increasing load across the individual line from the lower-left origin upward. Four different values are considered: α = 0.5 (strict), α = 1.0 (balanced), α = 1.5 (moderate) and α = 2.0 (relaxed). At low load, ECAR incurs minimal drops and triggers few replications irrespective of the threshold setting. With α = 0.5, ECAR produces the highest number of replications (975) at load 1 while sustaining the lowest drop rate of 11%. This confirms that a strict threshold causes ECAR to replicate proactively and frequently, investing substantially in replication overhead to suppress drops. In contrast, for α = 2.0, the relaxed threshold restricts the system to considerably fewer replications (228 at load 1.0) and thus has the highest drop rate of 22%. For α = 1.5, we observe a slight decrease in drop rate (18%) with more replication events. With α = 1.0, ECAR manages to confine the drop rate within 14% with significantly fewer replicas (650) than α = 0.5. Based on these observations, α = 1.0 emerges as the optimal threshold for ECAR as it maintains a lower drop rate comparable to that of α = 0.5 under moderate load while avoiding the excessive replication overhead.

9. Conclusion and Future Work

In this paper, we present Elastic Context-Aware Replication (ECAR), a new method that significantly improves service replication in dynamic cloud-edge-access computing environments. The unique advantage of ECAR lies in its integration of both physical and logical context and in the incorporating changes in request demands into the replication process, enabling it to adapt to changing network conditions. The consideration of disparity in demand from different regions amplifies the significance of the replica. The choice of servers through DCDP optimizes the resource utilization. This approach ensures efficient resource utilization and service availability by dynamically adjusting to the operational context of cloud, edge and access layers. We extensively evaluated our scheme's performance with Cloud-Intensive Replication (CIR), Cloud-Edge-Intensive Replication (CEIR), Cloud-Edge-Access-Intensive Replication (CEAIR) scenarios and existing Group-Delay-Aware Replication (GDAR) including a Combined Context-Aware Replication (CCA) in an identical IoT environment. The comparison result demonstrates reduced drop rates (18%), improved allocation efficiency (82%) and optimized resource utilization (82%). The results highlight ECAR's superior performance and adaptability to evolving conditions, managing high loads and minimizing response times, establishing it as a substantial advancement in service replication strategies. However, the unavailability of resources for fulfilling a replication request within its logical context initiates a shift in the context. Repeated occurrences of such instances may result in frequent regeneration of the logical context. However, with effective resource management using DCDP, such occurrences become exceptionally rare. Next, the proposed replication mechanism effectively identifies the most suitable potential node for service-replication. However, it does not explicitly address the selection of the optimal source node from which the service should be replicated. Additionally, the shift of request patterns and introduction of new services may lead to frequent regeneration of contexts. Incorporating source node optimization into the replication framework with an improved context-regeneration policy remains an important direction for future research to achieve a more comprehensive and efficient end-to-end service replication strategy. Further, to enable more adaptive, distributed and privacy-preserving decision-making across different IoT layers, federated learning (FL) can be incorporated. By integrating FL, the system could continuously learn from dynamic network conditions, user demands and service patterns, which helps to improve the global QoS and resource utilization many-fold.

All the Declarations and Statements

Author Contributions Statement
Mrinal Kanti Mahato – contributed to the conceptualization, methodology, formal analysis, investigation and writing of the original draft.
Bikash Choudhury – contributed to the conceptualization, supervision, validation, and writing, review and editing.
Tanushree Garai – contributed to validation, data curation, writing, review and editing.
Sudip Kumar Adhikari – contributed to validation, resources, writing, review and editing.
Himadri Nath Saha – contributed to supervision, project administration, writing, review and editing.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement
The authors declare no conflict of interest.

Funding Declaration
The authors received no financial support for this research.

Data Availability Statement
The dataset generated through simulation might be shared if needed.

Ethical Declarations
This research does not involve human subjects or animals.

Acknowledgments
The authors sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed to improving the quality of the experiment and the reliability of its results.

Declaration of Generative AI in Scholarly Writing
The authors confirm that no generative AI or AI-assisted technologies were used in the preparation of this manuscript.

Abbreviations
The following abbreviations are used in this manuscript:

IoT - Internet of Things
MFT - Maximal Frequent Transactions
PCN - Physical Context Network
LCN - Logical Context Network
ECAR - Elastic Context-Aware Replication
DCDP - Distributed Constraint Distribution Problem

Appendix
No appendices are needed for this research.

References

[1] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An Overview on Edge Computing Research," IEEE Access, vol. 8, pp. 85714–85728, 2020. doi: 10.1109/ACCESS.2020.2991734.
[2] R. Roman, J. Lopez, and M. Mambo, "Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges," Future Gener. Comput. Syst., vol. 78, pp. 680–698, Jan. 2018. doi: 10.1016/j.future.2016.11.009.
[3] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets," IEEE Internet Things J., vol. 4, no. 5, pp. 1216–1228, Oct. 2017. doi: 10.1109/JIOT.2017.2709814.
[4] J. J. López Escobar, R. Díaz Redondo, and F. Gil-Castiñeira, "In-depth Analysis and Open Challenges of Mist Computing," J. Cloud Comput., vol. 11, no. 1, pp. 1–26, Nov. 2022. doi: 10.1186/s13677-022-00354-x.
[5] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," IEEE Trans. Serv. Comput., vol. 3, no. 3, pp. 223–235, Jul./Sep. 2010. doi: 10.1109/TSC.2010.3.

[6] J. Al-Jaroodi, N. Mohamed, and J. Aziz, "Service Oriented Middleware: Trends and Challenges," in Proc. 7th Int. Conf. Inf. Technol.: New Gener. (ITNG), 2010, pp. 974–979. doi: 10.1109/ITNG.2010.55.
[7] B. Cheng, D. Zhu, S. Zhao, and J. Chen, "Situation-Aware IoT Service Coordination Using the Event-Driven SOA Paradigm," IEEE Trans. Netw. Serv. Manag., vol. 13, no. 2, pp. 349–361, Jun. 2016. doi: 10.1109/TNSM.2016.2541171.
[8] S. Slimani, T. Hamrouni, and F. Ben Charrada, "Service-Oriented Replication Strategies for Improving Quality-of-Service in Cloud Computing: A Survey," Cluster Comput., vol. 24, no. 1, pp. 361–392, Mar. 2021. doi: 10.1007/s10586-020-03108-z.
[9] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative Cloud and Edge Computing for Latency Minimization," IEEE Trans. Veh. Technol., vol. 68, no. 5, pp. 5031–5044, May 2019. doi: 10.1109/TVT.2019.2904244.
[10] P. Rodriguez and E. W. Biersack, "Dynamic Parallel Access to Replicated Content in the Internet," IEEE/ACM Trans. Netw., vol. 10, no. 4, pp. 455–465, Aug. 2002. doi: 10.1109/TNET.2002.801413.
[11] H. Shen and G. Liu, "A Lightweight and Cooperative Multifactor Considered File Replication Method in Structured P2P Systems," IEEE Trans. Comput., vol. 62, no. 11, pp. 2115–2130, Nov. 2013. doi: 10.1109/TC.2012.104.
[12] N. B. Salah and N. Bellamine Ben Saoud, "IoT Data Placement in the Fog Infrastructure with Mobile Devices," in Proc. IEEE/ACM 21st Int. Symp. Cluster Cloud Internet Comput. (CCGrid), 2021, pp. 21–30. doi: 10.1109/CCGrid51090.2021.00012.
[13] S. He and X. H. Sun, "A Cost-Effective Distribution-Aware Data Replication Scheme for Parallel I/O Systems," IEEE Trans. Comput., vol. 67, no. 10, pp. 1374–1387, Oct. 2018. doi: 10.1109/TC.2018.2831689.
[14] D. M. Bui, S. Hussain, E.-N. Huh, and S. Lee, "Adaptive Replication Management in HDFS Based on Supervised Learning," IEEE Trans. Knowl. Data Eng., vol. 28, no. 6, pp. 1369–1382, Jun. 2016. doi: 10.1109/TKDE.2016.2523510.
[15] C. Berger, H. P. Reiser, J. Sousa, and A. Bessani, "AWARE: Adaptive Wide-Area Replication for Fast and Resilient Byzantine Consensus," IEEE Trans. Dependable Secure Comput., vol. 19, no. 3, pp. 1605–1620, May/Jun. 2022. doi: 10.1109/TDSC.2020.3030605.
[16] S. A. Mohamed, S. Sorour, and H. S. Hassanein, "Group Delay-Aware Scalable Mobile Edge Computing Using Service Replication," IEEE Trans. Veh. Technol., vol. 71, no. 11, pp. 11911–11920, Nov. 2022. doi: 10.1109/TVT.2022.3193887.
[17] H. Shen, G. Liu, and H. Chandler, "Swarm Intelligence Based File Replication and Consistency Maintenance in Structured P2P File Sharing Systems," IEEE Trans. Comput., vol. 64, no. 10, pp. 2953–2967, Oct. 2015. doi: 10.1109/TC.2015.2389845.
[18] P. T. Wojciechowski, T. Kobus, and M. Kokocinski, "State-Machine and Deferred-Update Replication: Analysis and Comparison," IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 3, pp. 891–904, Mar. 2017. doi: 10.1109/TPDS.2016.2590422.
[19] S. Verma, A. K. Yadav, D. Motwani, R. S. Raw, and H. K. Singh, "An Efficient Data Replication and Load Balancing Technique for Fog Computing Environment," in Proc. 3rd Int. Conf. Comput. Sustainable Global Dev. (INDIACom), 2016, pp. 2888–2895.
[20] Y. J. Yu, T.C. Chiu, A.-C. Pang, M.-F. Chen, and J. Liu, "Virtual Machine Placement for Backhaul Traffic Minimization in Fog Radio Access Networks," in Proc. IEEE Int. Conf. Commun. (ICC), 2017, pp. 1–7. doi: 10.1109/ICC.2017.7996500.
[21] L. Yang, A. Zhou, X. Ma, Y. Zhang, and S. Wang, "Reliability-Aware Task Replication for Mobile Edge Computing," IEEE Internet Things J., vol. 11, no. 14, pp. 24846–24857, Jul. 2024. doi: 10.1109/JIOT.2024.3388156.
[22] B. Choudhury, S. Choudhury, and A. Dutta, "A Proactive Context-Aware Service Replication Scheme for Ad Hoc IoT Scenarios," IEEE Trans. Netw. Serv. Manag., vol. 16, no. 4, pp. 1797–1811, Dec. 2019. doi: 10.1109/TNSM.2019.2928698.
[23] M. Polverini, J. Galan-Jimenez, F. G. Lavacca, A. Cianfrani, and V. Eramo, "Improving Dynamic Service Function Chaining Classification in NFV/SDN Networks Through the Offloading Concept," Comput. Netw., vol. 182, p. 107480, Dec. 2020. doi: 10.1016/j.comnet.2020.107480.
[24] M. A. Ahmed, M. H. Khafagy, M. E. Shaheen, and M. R. Kaseb, "Dynamic Replication Policy on HDFS Based on Machine Learning Clustering," IEEE Access, vol. 11, pp. 18551–18559, 2023. doi: 10.1109/ACCESS.2023.3247190.
[25] F. Arar, R. Mokadem, and D. E. Zegour, "Towards Using Reinforcement Learning for Scaling and Data Replication in Cloud Systems," in Proc. Doctoral Conf. Comput. Sci. (ADCCS), Algiers, Algeria, May 2024.
[26] A. Najjar et al., "Towards Designing an Energy Aware Data Replication Strategy for Cloud Systems Using Reinforcement Learning," arXiv preprint arXiv:2507.18459, 2025.
[27] A. Murimi, "How Machine Learning-Data Driven Replication Strategies Enhance Fault Tolerance in Large-Scale Distributed Systems," arXiv preprint arXiv:2511.11749, 2025.
[28] L. Zhai, Z. Lu, J. Sun, and X. Li, "Joint Task Offloading and Computing Resource Allocation with DQN for Task-Dependency in Multi-access Edge Computing," Comput. Netw., vol. 263, p. 111222, Mar. 2025. doi: 10.1016/j.comnet.2025.111222.
[29] L. Zhai, P. Zhao, K. Xue, Y. Li, and C. Cheng, "Task Offloading and Multi-Cache Placement in Multi-access Mobile Edge Computing," Comput. Netw., vol. 258, p. 111030, 2025. doi: 10.1016/j.comnet.2024.111030.
[30] J. Zhang, Y. Peng, F. Zhou, F. Lei, W. Li, and X. Qiu, "Resource and Delay Aware Fine-Grained Service Offloading in Collaborative Edge Computing," Comput. Netw., vol. 218, p. 109383, Dec. 2022. doi: 10.1016/j.comnet.2022.109383.
[31] Hanen Tlich et al, "A Clustering-Aware and Multi-Feature Replication Strategy for Efficient Storage in HDFS," Procedia Comput. Sci. (KES 2025), Amsterdam: Elsevier, 2025.
[32] Nour-Eddine Bakni and Ismail Assayad, "Smart Data Placement Strategy in Heterogeneous Hadoop," HighTech Innov. J., 2025. doi: 10.28991/HIJ-2025-06-01.
[33] G. Manogaran et al., "Machine Learning Assisted Information Management Scheme in Service Concentrated IoT," IEEE Trans. Ind. Informat., vol. 17, no. 4, pp. 2871–2879, Apr. 2021. doi: 10.1109/TII.2020.3012759.
[34] T. Bilen and B. Canberk, "Handover-Aware Content Replication for Mobile-CDN," IEEE Netw. Lett., vol. 1, no. 1, pp. 10–13, Mar. 2019. doi: 10.1109/LNET.2018.2873982.
[35] S. Zhou, Y. Sun, Z. Jiang, and Z. Niu, "Exploiting Moving Intelligence: Delay-Optimized Computation Offloading in Vehicular Fog Networks," IEEE Commun. Mag., vol. 57, no. 5, pp. 49–55, May 2019. doi: 10.1109/MCOM.2019.1800230.
[36] I. M. Amer, S. M. A. Oteafy, and H. S. Hassanein, "Task Replication in Unreliable Edge Networks," in Proc. IEEE 47th Conf. Local Comput. Netw. (LCN), 2022, pp. 173–180. doi: 10.1109/LCN53696.2022.9843613.
[37] J. Khan, C. Westphal, and Y. Ghamri-Doudane, "A Content-Based Centrality Metric for Collaborative Caching in Information-Centric Fogs," May 2017.

[38] K. C. Mondal, "Algorithms for Data Mining and Bio-informatics," Ph.D. dissertation, Université Nice Sophia Antipolis, Nice, France, Jul. 2013.
[39] K. Macarthur, "Multi-Agent Coordination for Dynamic Decentralised Task Allocation," Ph.D. dissertation, Univ. of Southampton, Southampton, UK, 2011.
[40] S. M. Aji and R. J. McEliece, "The Generalized Distributive Law," IEEE Trans. Inf. Theory, vol. 46, no. 2, pp. 325–343, Mar. 2000.
[41] S. Bistarelli, R. Gennari, and F. Rossi, "Constraint Propagation for Soft Constraints: Generalization and Termination Conditions," in Proc. 6th Int. Conf. Princ. Pract. Constraint Program. (CP), 2000, pp. 83–97.
[42] N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar, and S. Mahmoud, "SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services," IEEE Access, vol. 5, pp. 17576–17588, 2017. doi: 10.1109/ACCESS.2017.2731382.

Authors ' Profiles

Mrinal Kanti Mahato passed his B.Sc. in Computer Science from The University of Burdwan, India, and obtained his Master of Computer Applications (MCA) degree from Maulana Abul Kalam Azad University of Technology (MAKAUT), India. He is currently serving as a State Aided College Teacher (SACT) in the Department of Computer Science at Nistarini College, Purulia, India. He is also pursuing his Ph.D. in the Department of Computer Science and Engineering at Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India. His research interests include Adaptive Internet of Things, Resource Management in the Internet of Things, and Federated Learning.

Bikash Choudhury received the B.Tech. degree from WBUT, India, and the M.Tech. and Ph.D. degrees from the National Institute of Technology Durgapur, India. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Ramkrishna Mahato Government Engineering College, Purulia, India. His research interests include distributed service-oriented computing, Resource Management in the Internet of Things, and Tiny Machine Learning.

Tanushree Garai received the B.Tech. degree from Government College of Engineering and Leather Technology under Maulana Abul Kalam Azad University of Technology (formerly WBUT), India, and the M.E. degree from Jadavpur University, India. She is currently an Assistant Professor with the Department of Computer Science and Engineering, Government College of Engineering and Ceramic Technology, India. Her research interests include cryptography, information security, steganography, digital forensics, and data privacy.

Sudip Kumar Adhikari passed B. Tech in Computer Science & Engineering from Vidyasagar University. He obtained the M.E. and Ph.D. degree in Computer Science & Engineering from Jadavpur University. He has more than 20 years of teaching experience. He is currently an assistant professor in Computer Science & Engineering Department of Kalyani Government Engineering College, Kalyani, India. He has published many of research papers in reputed International Journals and Conferences. His research interests include Medical image processing, IoT, Machine Learning. Dr. Adhikari is a Senior Member of IEEE and member of Institute of Engineers. He served as a reviewer in several International conferences and also in several reputed international journals such as Applied Soft Computing, Elsevier, IET Computer Vision, IEEE Access, and IEEE Transactions on Fuzzy System etc. He has been the member of the organizing and technical program committees of several national and international conferences.

Himadri Nath Saha is a distinguished academician and researcher with over 24 years of experience in teaching, research and academic leadership. Recognized among Stanford University and Elsevier's world's top 2% scientists, he holds a Ph.D. (Engineering) from Jadavpur University in secure MANET routing protocols, along with degrees from Jadavpur University, IIEST Shibpur, and an MBA. His research spans AI/ML, IoT, cybersecurity, cryptography, UAVs, network security, and algorithms. Dr. Saha has authored over 130 publications with 2,500+ citations, holds multiple patents and copyrights, and has written more than 10 textbooks. A Senior Member of IEEE and Fellow of IETE and IEI, he has served as visiting professor/research scientist at institutions in the USA and Germany, and continues to contribute globally through research collaborations, invited talks, editorial roles, and academic governance.

How to cite this paper:

Mrinal Kanti Mahato, Bikash Choudhury, Tanushree Garai, Sudip Kumar Adhikari, Himadri Nath Saha, "Adaptive Context-Aware Replication Mechanism for Distributed IoT Environments", International Journal of Computer Network and Information Security(IJCNIS), Vol.18, No.4, pp. 106-129, 2026. DOI: 10.5815/ijcnis.2026.04.06