IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 15, No. 6, December 2026, pp. 5329~5342
ISSN: 2252-8938, DOI: 10.11591/ijai.v15.i6.pp5329-5342

Resource allocation with load balancing in internet of things-fog-cloud computing environment

Srikanta Murthy Rajini[1], Chaithanya Dasarakoppal Jayaprakash[2]
[1]Department of Information Science and Engineering, Vidyavardhaka College of Engineering, Mysuru, India
[2]Department of Electronics and Communication Engineering, Vidyavardhaka College of Engineering, Mysuru, India

Article Info

Article history:

Received Dec 16, 2025
Revised Aug 21, 2026
Accepted Sep 20, 2026

Keywords:

Community
Fog computing
Internet of things
Load balancing
Resource allocation
Strongly connected components

ABSTRACT

Fog computing is an extension of cloud services to the network edge and internet of things (IoT) that decreases latency and the usage of bandwidth for time-sensitive applications. Existing fog-based systems lack dynamic and efficient workload distribution mechanisms, leading to poor network performance and bottlenecks. Each decision is based on the system's global view, which can become a bottleneck under a distributed fog environment and high-load conditions. To solve these problems, this research proposes a Louvain for community detection with strongly connected components (SCC), which helps monitor the capacity of each load. It adaptively assigns services to least-loaded within the selected partitions and prevents overload bottlenecks while enhancing the overall network's responsiveness. This decreases the hyperparameter sensitivity and also ensures robust performance when a demand pattern fluctuates dynamically, and it divides the infrastructure graph into communities using Louvain clustering. The community-based SCC primarily focuses on intra-layer balancing within the fog layers, improving services that are placed intelligently between fog nodes by involving both intra and inter-partition balance. The proposed community-SCC method attains 79.8% node utilization and 112.3 ms average latency only for load balancing.

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

Corresponding Author:

Srikanta Murthy Rajini
Department of Information Science and Engineering, Vidyavardhaka College of Engineering
Mysuru, Karnataka, India
Email: rajinis@vvce.ac.in

1. INTRODUCTION

Internet of things (IoT) has emerged as a transformative paradigm within information and communication technology (ICT) and enables the interconnection of large devices via the exchange of data and internet from their physical environment [1]. Various IoT applications demand substantial computation power and low latency for real-time processing that leads to excess energy consumption in resource-constrained devices [2]. For these problems, fog computing has a significant paradigm which is capable of detecting resource and computational demands that are associated with IoT systems [3]. The fog computing carries cloud services to the edge network to enhance IoT application performance in terms of workload balancing, energy usage, resource utilization, and service delay [4]. However, for efficient resource management strategies within the fog computing environment, fog nodes have limited resources [5]. The cloud structure provides increased latency and more data transmission loads [6].

In healthcare, fog computing has been extensively adopted to improve data management and reliability [7]. The cloud applications' resource-intensive migration process is used to identify the problem for data management [8]. The allocation of virtual machines and the migration process support efficient

management and handle the challenges that occurred in cloud computing [9]. The resource-constrained and distributed nature of fog computing optimizes bandwidth usage, decreases latency, and improves decision-making in real time [10]. The fog-cloud layer is responsible for processing and delivering the results as well as incoming tasks to the cloud for decision-making and monitoring [11]. Similarly, link failure, bandwidth limitations, and energy constraints remain as major difficulties which limit the reliability and scalability of fog networks [12]. Therefore, the resource allocation mechanism is important for both user and service providers regarding objectives because of inefficiency [13]. Resource allocation in heterogeneous fog platforms burden huge costs on provider's low quality and cash flow of experience from the user's viewpoint [14].

Although fog computing is a promising solution to enhance IoT applications' performance and reduce power consumption [15]. These tasks have varied storage demands and computational needs such as memory utilization, CPU usage, and time constraints for completion [16]. It is a procedure to possess the ability to identify a most ideal solution on a worldwide scale [17]. Load balancing is significant for workload distribution between edge and fog nodes to avoid any single node from becoming overloaded and enhancing seamless execution of IoT tasks [18]. Many single-fog computing systems cannot efficiently handle resource-intensive tasks due to a lack of inadequate processing capabilities and available resources and the dynamic nature of task demands [19]. Additionally, workload distribution is done across various heterogeneous fog devices that contain different computational capabilities and resources. The increasing number of requests exacerbates this issue, resulting in longer task queues for more powerful fog nodes [20].

The contributions of this research are like Louvain algorithm automatically groups highly connected fog nodes into communities, decreasing communication overhead among supporting distributed workload management and distant nodes. The strongly connected component (SCC) identifies groups which are strongly interdependent, ensuring that related tasks remain with the same community, which decreases execution time and latency. The integration of the community-SCC approach that unifies intra-partition load distribution, dependency-aware service grouping into a single model decision and inter-layer balancing by load balancing across partitions, thereby preserving service dependencies that helps the model to enhance scalability and stability of the fog-cloud performance.

The remainder of this paper is organized as follows. Section 2 presents a literature review. Section 3 describes the workflow of the proposed method. The experimental results and discussion are provided in section 4. Lastly, section 5 concludes the paper.

2. RELATED WORKS

The related works are provided as follows by considering existing approaches along with their advantages and disadvantages. Deepak *et al.* [21] implemented a deadline-constrained load balancing (DCLB), which was designed to handle IoT task batches and to ensure that each task meets its deadline constraint. Task batching was a process where IoT tasks were grouped into batches according to resource requirements and arrival time. Deadline-aware scheduling had each task, which was assigned as a priority based on its deadline and computational demand. A load balancing strategy distributed tasks across nodes to minimize makespan. The proposed method improved deadline satisfaction by preferring nodes with higher availability and lower current load. However, the overhead in profiling continuous monitoring of node status and task deadlines led to computational overhead. Srichandan *et al.* [22] developed an enhanced arithmetic optimization algorithm (AOA), which was utilized to explore and exploit the search space for optimal task allocation. The pattern search local search technique refined the solution that was generated by AOA to enhance precision and convergence. A task offloading strategy dynamically offloaded tasks to fog nodes based on deadline constraints and resource availability. The pattern search enhanced the AOA accurately and led to better task performance. However, the proposed approach increased the complexity, which affected the scalability in ultra-large networks.

Sharma and Thangaraj [23] introduced long short-term memory (LSTM), which was utilized to forecast the movement of users and devices. The LSTM approach helped anticipate where services were deployed to minimize latency. A lightweight placement algorithm was employed to evaluate cloud and fog nodes according to resource availability, proximity, and predicted trajectory. The quality of service (QoS)-aware decision-making placement decision contained service continuity, bandwidth, and latency, thereby handling a high service quality as the users move. However, fog nodes struggled with high-demand applications despite optimal placement. Ghafari and Mansouri [24] implemented a task scheduling algorithm based on fuzzy reinforcement learning (TSFRL) to optimize resource allocation in a fog-cloud environment for IoT workloads. Fuzzy logic was utilized to manage uncertainty and imprecision in task parameters such as task size, deadline urgency and resource availability. The TSFRL helped classify tasks into priority based on fuzzy rules. Reinforcement learning (RL) framework agents learned optimal scheduling policies by interacting with the environment and receiving feedback according to task completion success, energy usage

and latency. Adaptive scheduling was learned and enhanced over time, adapting to changing task patterns and system conditions. However, the RL demanded extensive training and exploration, which led to initial inefficiencies and delays.

Karthikeyan and Brindha [25] developed a trust-driven hybrid fragmentation framework for optimizing data transmission in an IoT-enabled edge-fog-cloud system. The data was fragmented into smaller units based on contextual parameters like sensitivity and size. The vertical fragmentation was integrated with horizontal fragmentation to improve transmission efficiency. The smart data flow controller was dynamically providing a data flow based on network conditions, trust levels, and node availability. Security aware routing integrated trust metrics into outing decisions to avoid low-trust nodes and compromise to increase resilience against data tampering and unauthorized access. However, continuous trust monitoring and scoring led to communication and computational overhead. Deldari and Holghinezhad [26] presented an IoT-based bag of tasks (BoTs) scheduling framework for deadline-sensitive applications in fog-cloud environment for task classification. Tasks were grouped into BoTs based on application urgency and types. Subsequently, each task was annotated with metadata resource demand, deadline and priority. A deadline-aware scheduling algorithm assigned tasks to fog nodes based on their met deadline and ability. However, limited inter-task coordination model assumed task independence, which did not suit applications with complex dependencies.

Tomar *et al.* [27] implemented a deep reinforcement learning (DRL) to optimize task offloading decisions in a fog-enabled IoT architecture, where device have limited resources and was required to delegate computation to nearby fog nodes. The IoT devices generated tasks with varying latency sensitivity and computational demands. The agent utilized a deep Q-network (DQN) to enhance its decision-making over time. The optimal offloading was learned from adaptive decision-making under network condition fluctuation. However, the integration of DRL with live fog systems was challenging due to resource constraints and latency. Qi *et al.* [28] implemented a multi-objective global optimal and time-aware task scheduling algorithm (MOGO-TATSA) was utilized for task scheduling formulation as a problem of multi-objective that concentrated on deadline compliance, minimal latency, and balanced load distribution. The global optimization helped to increase the overall model performance by avoiding local minima. The time-sensitive tasks were handled by proposed method, thereby decreasing node congestion and increasing the distribution of tasks. However, its limited adaptability to sudden fluctuations struggled without additional reactive mechanisms.

3. METHOD

An energy efficient model for load balancing in a fog environment that utilizes dynamic load balancing techniques. This approach explores algorithms to dynamically allocate and handle tasks, optimize resource utilization and minimize energy consumption. The reinforcement model continuously learns from its environment, thereby adapting scheduling decisions to improve overall efficiency and performance in cloud computing systems. Figure 1 represents the proposed framework. Fog computing extends cloud services toward the network edge to serve latency-sensitive IoT workloads [29], and the proposed model follows the layered IoT-fog-cloud design described in comprehensive fog computing surveys [30].

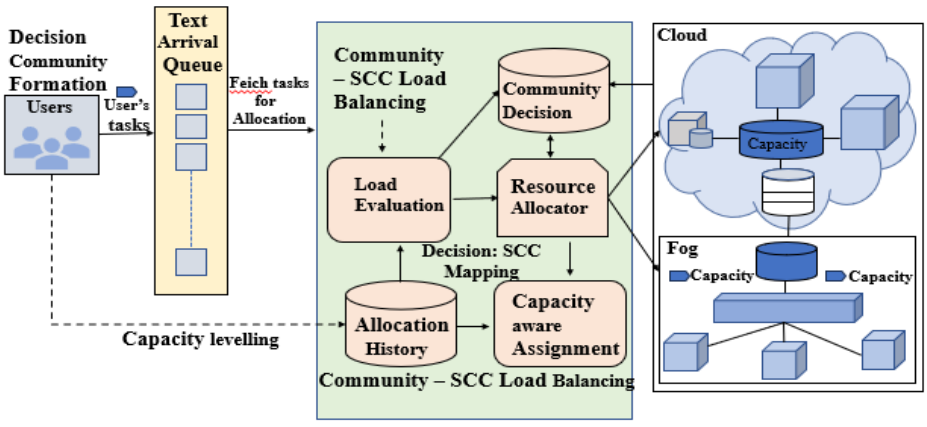

Figure 1. IoT-fog-cloud for load balancing and resource allocation framework

3.1. System model

The system model contains fog, edge and cloud layers and a base station (BS), the devices in the edge devices are generated by the BS, and then the number of tasks is assigned to the cloud and fog environment. The fog devices contained in fog environment is closer to the BS. The network is formalized as a $G = (V, E)$ directed fog-cloud graph, where each node $\mathcal{V} \epsilon V$ represents a cloud, fog and IoT devices that are characterized by memory limit, CPU capacity and support maximum concurrent task. The processing rate, resource availability, and load constraints provide clarity in load balancing decisions and task execution feasibility. Each link $e \in E$ is defined by the d_e propagation delay, B_e bandwidth, enhancing consistent communication modeling. Figure 2 illustrates the system model for resource allocation and load balancing. Representing the fog-cloud infrastructure as a graph with node capacities and link delays as attributes is consistent with the infrastructure models used for fog application placement [31], [32].

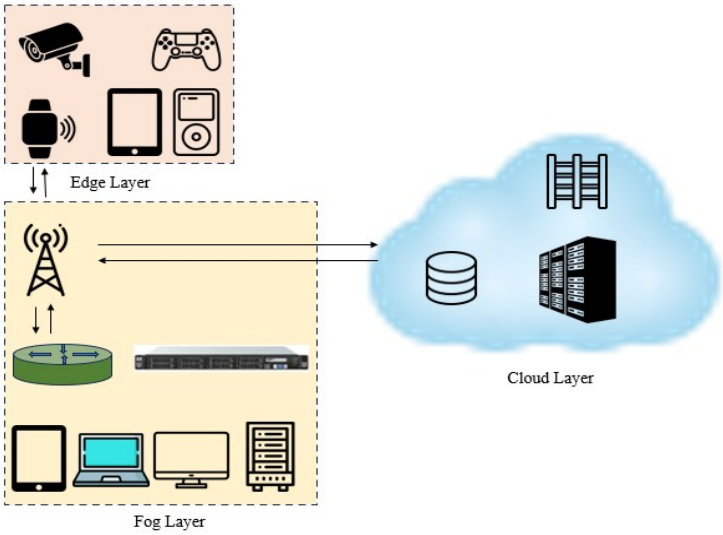

Figure 2. System model for fog environment

3.2. Layer selection and allocation

Layer allocation and selection for an incoming task are according to priority; each incoming task is computed by BS, as given in (1). Where, maximum tolerable time of task $Del^T_{\max}$ deadline of task $\mathrm{T_i}, D^T_l$ $\mathrm{t_i}$ and t^T_i is the current time as given in (2). Where delay in transmitting the task on the network is $\mathrm{Del}^T_{\mathrm{trans}}$, congestion intensity and number of tasks are submitted h_t, which is given in (3). Where processing delay is $\mathrm{Del}^T_{\mathrm{proc}}$, amount of active fog resources is $\mathrm{n(t)}$, each resource processing capacity is k, fog layer current workloads U^T_c given in (4) and (5). Each given mathematical equation represents task delay, network congestion, processing delay, workload computation, and resource consumption, which helps improve every tasks' arrival rate, variable intensity, and processing time. This allocation logic makes the delay computation transparent. The delay-based priority test in (5) reflects the delay-constrained workload allocation studied in [33], and the processing delay in (3) has the standard single-server queueing form described in [34].

$$Del^T_{max} = D^T_l - t^T_i \tag{1}$$

$$Del^T_{\mathrm{trans}} = h_t * m \tag{2}$$

$$Del^T_{proc} = \frac{U^T_c}{n(t) \cdot k - U^T_c} \tag{3}$$

$$T_{\mathrm{total}} = Del^T_{\mathrm{trans}} + Del^T_{\mathrm{proc}} \tag{4}$$

$$Priority = \begin{cases} Priority, \ if Del^T_{max} \leq T_{total} \\ Non - Priority, \ Otherwise \end{cases} \tag{5}$$

3.3. Girvan-Newman community detection algorithm

In fog computing, the proposed method optimizes the way that the applications and services are assigned to interconnected devices by identifying the node communities that communicate closely with each other. The fog network is treated as a graph where each node represents a device and each edge represents a communication link. Using the Girvan–Newman community detection algorithm [35], the method progressively eliminates the most critical connections, which are known as edges with high between centrality, to divide the network into smaller, tightly connected groups called communities. Each community represents a cluster of fog devices that have strong internal communication and fewer external connections, ensuring faster data exchange and lower latency within the group. With the service deployment within communities, the reliability of the system is increased, if another device fails and the similar community helps continue the communication operation without breaking. This method enhances the availability of the services and decreases network congestion, thereby improving resource utilization, which makes it more significant for fog computing environment performance. In the implementation, the Louvain method, which optimizes the modularity measure [36] and scales to large networks [37].

3.4. Strongly connected components-based application partition

The given map application once does mapping into a candidate community of devices, which is performed in the initial phase of the algorithm. The allocation of the services of the application to the devices in the community is addressed in the second phase. Primarily, application partition into sets of services. Second, suitable devices search for the placement of subsets of services. Ideally, each service set is placed only on a single device, if not, consider the smaller sets. For the initial step, the service partition of an application, the transitive closure use of the service in the application graph. The previous method is applied to the services in a decentralized service placement policy. The transitive closure of a directed graph highlights all the nodes which are reachable from one to another within the original graph architecture. If a j vertex is reachable by a I vertex, then a path exists from I to the matrix reachability of a graph is named as the transitive closure of the graph. The transitive closure of a directed graph can be computed with Warshall's algorithm [38].

Under the main problem conditions, the node transitive closure can be understood as the set of services which are requested for the given service execution. These are the outgoing requests in messages where a service is generated when it is requested. If decreasing the response time is of the application execution, the transitive closure should be allocated to the same device to minimize the communication delay among them. Assuming a 0.0 network delay for request messages within the same device. In addition, the best case is when each service of an application is placed on the same device. However, the resource constraint is limited in this condition. In the second stage, the concentration is on the most appropriate device determining which grouped services should be deployed. To attain this, a first-fit algorithm [39] is employed to reduce the graph influence partitioning on the results of final allocation. Tarjan's algorithm [40] initiates by organizing a service set based on their transitive closure, starting from the largest group that contains every service and progressing towards smaller ones having individual nodes or simple loops in the service flow. The aim of this algorithm is to improve every set if the services are effectively placed on the suitable device. Tarjan's approach does not require various traversals of the graph, which is one of its key flaws. Tarjan's method can keep the stack separate while computing SCC in a single depth-first search (DFS) pass. Because of its speed acceleration, this technique is the most sued sequential technique for locating SCCs. Tarjan's approach is purely sequential, and it is DFS-dependent on how the load is currently configured. As running in a mutual interconnected nodes for that SCC is used for strong connections.

The integration of SCC and community method consists of SCC-based dependency grouping for placement of intra partition and Louvain structural community detection for balancing inter-partition, and the service placement adapts simultaneously to real-time node capacity and dependency patterns using unified load-aware assignment mechanism. The proposed community method is combined with the SCC method for load balancing of two-level intra and inter in fog computing. The SCC used for grouping dependency performing adaptive load scoring, is SCC-aware placement and joint community selection to improve balanced distribution, dependency-preserving service placement, and balanced distribution across communities during fog execution, thereby reducing the cross-community communication costs. The SCC approach completely concentrates on mutually dependent service groups identified by using Tarjan's method that detect maximal subgraphs where each service reaches each other service. This process enhances dependency-preserving placement. The transitive closure is used to determine the extended dependency influence. The algorithm workflow is a layer selection outlining, community detection, grouping SCC, and final placement, which narrate the overall proposed methodology.

3.4.1. Feasibility model and placement cost

The explicit feasibility conditions and cost functions, like communication, computation cost, device capacity checks, and latency threshold, are defined as (6) to (8). The cost function is the integration of communication and computation cost and objective latency. Each component control influence are α, β, γ that allows the efficient balance local processing. Feasibility helps to improve s_i service placed on the node n_j, which is having sufficient sources remaining. The total node available resources capacity is denoted as feasibility. Latency constraint: s_i is the total delay experienced because of consisting communication and computation cost which does not exceed the maximum permissible $D_{\max}$ delay. The cost function in (6) combines computation, communication and latency terms, and the capacity and latency constraints in (7) and (8) correspond to the resource-aware feasibility conditions used for fog application placement [41].

$$C = \alpha \cdot comp_cost + \beta \cdot comm_cost + \gamma \cdot latency \quad (6)$$

$$demand(s_i) \leq capacity(n_j) - load(n_j) \quad (7)$$

$$\text{End-to-end delay } (s_i) \leq D_{max} \quad (8)$$

The overall procedure of the proposed method is summarized in Pseudocode 1. The infrastructure graph is first loaded and partitioned into communities using the Louvain method; the application graph is then loaded, and its service groups are obtained as SCC using Tarjan's algorithm. Finally, each service group is placed in the community with the lowest average load, and its services are assigned to the least-loaded node that satisfies the capacity constraint in (7).

Pseudocode 1. Community-SCC-based load-balanced placement

```
FUNCTION Load Infrastructure_ Graph (filepath):
    - Load JSON data describing fog nodes and links
    - Create an empty undirected graph G ()
    - For each data.node in JSON:
        * Add node to G with attributes:
            - capacity
            - initial load
    - For each link in JSON:
        * Add edge to G with latency
    - RETURN G

FUNCTION Partition Infra Graph(G):
    - Apply Louvain community detection algorithm
    - Assign each node a community ID
    - Group nodes into a dictionary:
        {community_ id: list of node IDs}
    - RETURN community's dictionary

FUNCTION Load App Graph (file path):
    - Load JSON data describing services and dependencies
    - Create a directed graph App G
    - For each service in JSON:
        * Add node to App G with attribute: demand
    - For each dependency in JSON:
        * Add directed edge to App G
    - RETURN App G

FUNCTION Compute Service Groups (App G):
    - Compute strongly connected components (COMMUNITY-SCCs) of App G
    - Each COMMUNITY-SCC is a service group
    - RETURN list of service groups (each as a set of service IDs)

FUNCTION Calc Load Score(node):
    - RETURN node. load / node. capacity

FUNCTION Place Service Groups (G, Communities, App G, Service Groups):
    - Initialize empty list Placement Result
    - For each Group in Service Groups:
        * Initialize Best Community ID = None
        * Initialize Lowest Avg Load = Infinity

        * For each CommunityID in Communities:
            - Compute Avg Load = average Calc Load Score of all nodes in the community
            - If Avg Load < Lowest AvgLoad:
```

```
                · Update Lowest AvgLoad and Best Community ID

        * Get Candidate Nodes = nodes in Best Community ID

        * For each Service ID in Group:
            - Get ServiceDemand = App G node attribute
            - Sort Candidate Nodes by Calc Load Score ascending
            - For each Node in sorted Candidate Nodes:
                · If Node. load + Service Demand <= Node. capacity:
                    - Assign Service to Node
                    - Increment Node. load by Service Demand
                    - Add record to Placement Result:
                        {
                            service_ id, assigned_ node, community, demand, new_ load
                        }
                    - Break inner loop
                · Else:
                    - Mark Service as rejected if no node can fit
                    - Add record to Placement Result:
                        {
                            service_   id,   assigned_   node=None,   community,   demand,
status='rejected'
                        }
    - RETURN Placement Result

FUNCTION Save _Placement Result (Result, filename):
    - Write Result to JSON file with indentation
```

4. RESULTS AND DISCUSSION

The efficiency of the proposed community-SCC is trained and tested to evaluate the model overall performance. The developed is community-SCC experienced on the environment of Python 3.8, processor is 12th Gen Intel (R) Core(TM) i7-12700K (3.60 GHz), installed RAM 64.0 GB (63.8 GB usable), system type 64-bit operating system, x64-based processor, operating-OS: Windows 11 Pro, graphics card (GPU): 8 GB, SSD: 1.86 TB, GPU: Intel(R) UHD graphics 770, GPU: NVIDIA GeForce RTX 3050(40 GB). The developed community-SCC technique is validated by using different analysis metrics like resource allocation, makespan, throughput, and energy consumption. The performance metrics, namely latency, utilization, rejection rate, load imbalance, makespan, and energy consumption, are standard indicators for evaluating resource management in fog environments [42].

4.1. Performance analysis

The qualitative and quantitative analyses of the developed community-SCC is validated in this section. The efficiency of the designed community-SCC approach is validated in this section by using the performance of various metrics. Figure 3 represents the average latency comparison. This community-based method balances workload efficiently, reduces communication delay, and improves overall system responsiveness in fog computing environments. Latency-oriented evaluations of fog application placement are also reported in [43].

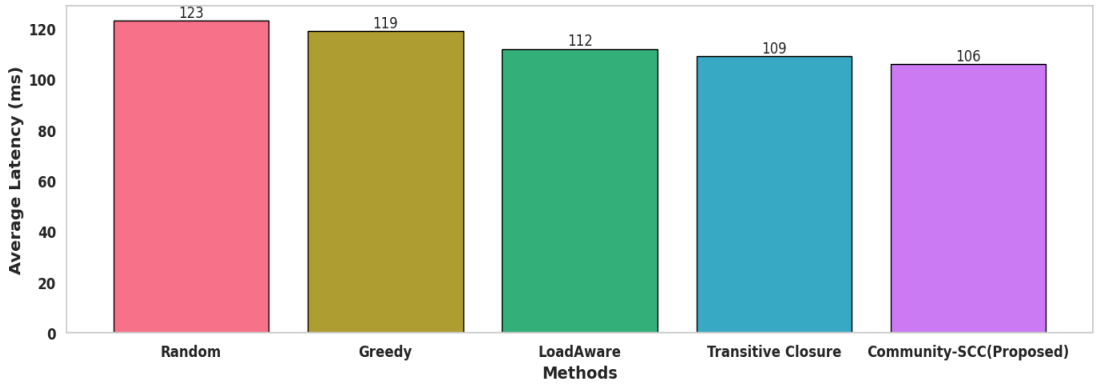

Figure 3. Comparison of average latency

Figure 3 represents the comparison of average latency for approaches to resource allocation in load balancing. The random records of the method having highest latency of 123 ms because of distributed tasks without considering node conditions. The load and greedy-aware methods show a moderate increase by accounting partially for load and node capacity, decreasing latency to 119 ms and 112 ms, respectively. The transitive closure method further lowers latency to 109 ms by enhancing coordination among interconnected nodes. The developed community-SCC approach attains less latency of 106 ms by clustering highly connected fog nodes and tasks allocation within these groups.

Better resource utilization is achieved by load-aware approach by involving the capacity of load for better performance. The utilization is enhanced highly using transitive closure method to increase the proposed method overall efficiency. The developed community-SCC approach records the node utilization most consistent and highest, demonstrating significant task allocation across connected fog nodes, idle time decreased, and enhanced overall efficiency of the resource.

Figure 4 represents the node utilization distribution for various resource allocation methods fog computing. The random method shows uneven and lower utilization, showing that some nodes remain underused while others are overloaded. Slightly enhancing balance still lacks uniform distribution by using greedy approach. It contains capacity and node load leading to more stable performance. The developed community-SCC approaches record the most consistent and highest node utilization, which shows efficient task allocation across connected nodes, thereby enhancing overall resource efficiency and reduce idle time.

Figure 5 demonstrates the relationship among offered load and rejection rate for various resource allocation approaches in fog computing. As the provided load increases, each approach shows an increase in injection rate, but the increasing rate varies. The greedy and random methods faced more rejection rate due to the failure of resource allocation under heavy load conditions. The transitive closure and load-aware methods perform better, thereby managing moderate level of rejection via enhanced load distribution. The developed community-SCC approach attains the lowest injection rate across every level of load by effectively interconnecting grouping figure nodes and workload balancing within these communities. This results in fewer rejection service requests and better resource utilization under high network demand.

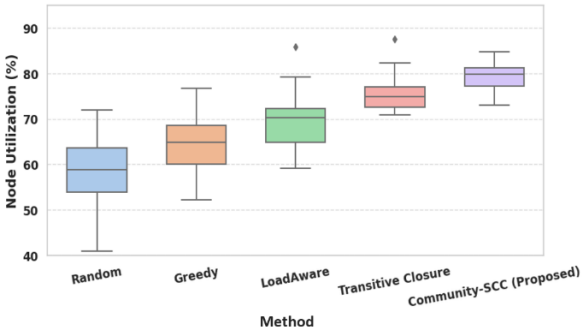

Figure 4. Node utilization distribution

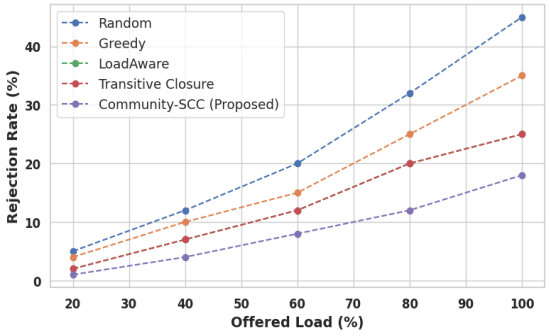

Figure 5. Offered load vs rejection rate

Figure 6 shows the comparison of load imbalance between various resource allocation approaches in fog computing. The load imbalance is presented as standard deviation, measuring how workload is evenly distributed across fog nodes. The highest imbalance of 0.32 indicates that the workload is distributed poorly using the random method records. The load-aware and greedy approach decreases to imbalance to 0.29 and 0.24 respectively, thereby containing node capacity during task allocation. The transitive closure approach performing a better 0.21 by reflecting enhanced coordination among nodes. The developed community-SCC method attains the lowest imbalance of 0.19, representing the uniform distribution of a task across nodes. In fog computing, the balance distribution enhances resource utilization and also the overall system stability.

The highest runtime exhibited by transitive closure indicated a larger network with slower scalability for load balancing. The load-aware approaches moderately enhance and additionally reflect the load evaluation process. The greedy and random having lower allocation efficiency but perform faster compared to the proposed method. The developed community-SCC method manages runtime even if increased the nodes representing scalability better. The efficient community-based grouping minimizes redundant computations, allowing faster execution while supporting larger fog environments effectively.

Figure 7 demonstrates the analysis on scalability of different resource allocation approaches in fog computing while comparing run time in algorithms with the rise in the number of nodes. Each method shows

a rise in runtime because of higher computational complexity, as the number of nodes grows. The transitive closure approach exhibits slower scalability and the highest runtime with larger networks. The greedy and random approaches perform faster but at lower efficiency when allocation is lower. The developed community-SCC approach handles a balanced runtime as the nodes enhanced; it shows better scalability. Its significant community-based grouping decreases redundant computation which allows faster execution while supporting larger fog environments effectively. Community-detection methods of this kind scale to large graphs and do not require the number of communities to be fixed in advance [44].

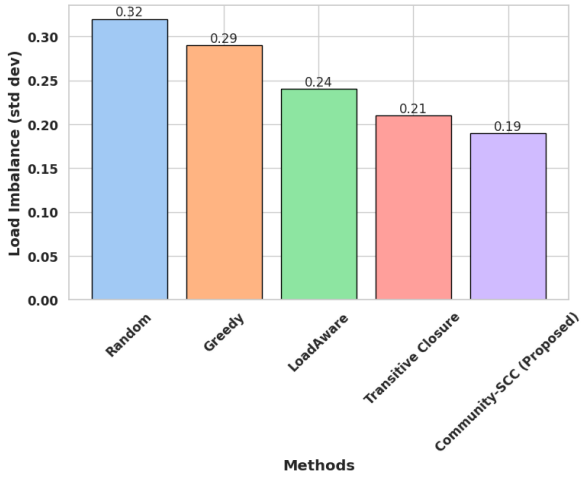

Figure 6. Load balancing comparison with existing approaches

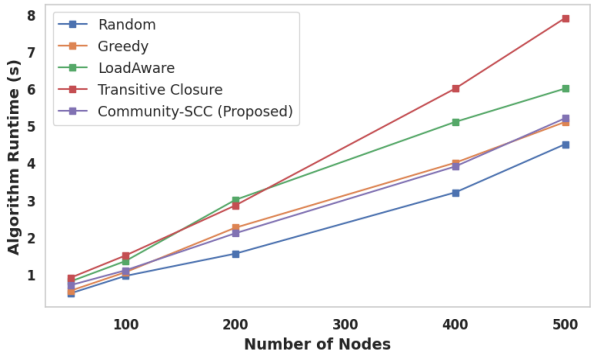

Figure 7. Scalability analysis

Hyper-parameter sensitivity analysis: the community-SCC method decreases hyper-parameter sensitivity because it uses SCC-based deterministic grouping and modularity-driven community which eliminates manual tuning of thresholds. The proposed model stabilizes the performance inherently under fluctuating parameter values. The hyperparameter analysis increases the proposed model overall performance in distributed environment.

The proposed community-based SCC method improves the efficiency of detection across large-scale directed networks. The SCC approach used to pre-process the network, efficiently decreasing the initial node-search space compared to greedy method. The proposed model concentrates on strongly connected sub-structures that avoids transitive closure high computational overhead and load-aware models' redundant load-balancing. This results in higher modularity and faster convergence when compared with the other models, and the proposed method achieves superior performance in complex networks. SCC can be found in linear time, whereas computing the full transitive closure is substantially more expensive [45].

4.2. Comparative analysis

This setup represents a realistic fog computing environment, where 100 distributed fog nodes handle 1000 service requests of varying computational and memory needs. In fog nodes, the Louvain algorithm identifies communities, while the proposed method for grouping improves dependent services and are

processed together. For performance and consistency, multiple simulation runs are used, and also for load balance, latency, runtime efficiency and fairness. Table 1 illustrates the parameter settings. Scale-free graphs of this kind are commonly generated with the preferential-attachment model [46].

Table 1. Parameter settings

Parameter	Value/description
Number of fog nodes	100
Number of services	1000
Average service demand	CPU: 2–8 units, memory: 512–2048 MB
Network topology	Scale-free graph with 5 communities
Communication delay	1–20 ms between nodes
Partitioning algorithm	Louvain method
Dependency grouping	Community-SCCs
Workload type	Mixed latency-sensitive and compute-intensive
Random seed	42 (fixed for reproducibility across all runs)
Failure model	Link failure with probability 0.02 per simulation round
Simulation runs	30 independent runs
Evaluation metrics	Latency, utilization, rejection %, imbalance, runtime

The developed community-SCC approach is compared with traditional methods to attain 105.8 ms latency and 79.8% node utilization, where tasks are effectively used and executed faster. The 18.3% rejection rate is dropped that helps manage more incoming requests significantly. The 0.189 std dev low imbalance and 0.935 Jain's high fairness index ensure the distribution of the workload across each node is even. The 5.21s is the runtime, which remains viable and demonstrates the scalability and efficiency of the model. Table 2 represents the proposed method performance analysis. Jain's fairness index [47] ranges from 1/n to 1, and values closer to 1 indicate a more even distribution of load across nodes.

Table 2. Comparative analysis by considering various performance metrics

Metric	Random	Greedy	Load-aware	Transitive closure	Community-community-SCC (proposed)
Avg latency (ms)	123.4	118.7	111.9	109.2	105.8
95th latency (ms)	198.7	192.3	180.5	173.1	165.4
Rejection rate (%)	45.3	34.8	25.2	24.9	18.3
Node utilization (%)	59.7	65.3	70.1	74.6	79.8
Load imbalance (std dev)	0.321	0.287	0.238	0.212	0.189
Fairness index (Jain's)	0.703	0.779	0.847	0.891	0.935
Placement success (%)	84.7	88.1	91.8	93.7	96.9
Runtime (s)	4.48	5.12	5.98	7.92	5.21

Table 3 shows how every component denotes the efficiency of the system by eliminating rejection and increased latency. The dependency cluster is lost when only load balancing is applied, which leads to suboptimal performance. The complete configuration of community-SCC provides minimal latency and rejection, delivering better outcomes.

The developed community-SCC model achieves dependent service of 89.2% within the same partition, thereby reducing communication overhead among communities. This helps decrease 42.7% cross- community links while improving requests, which transfers shorter paths. The average hops per request is 2.91 decreases, that confirming which service communication becomes more localized and faster. Table 4 represents the dependent preservation.

Table 3. Ablation study by considering baseline models

Variant	Avg latency (ms)	Rejection rate (%)	Load imbalance	Notes
w/o community-SCC grouping	118.6	28.7	0.247	Loses dependency preservation
w/o community partition	115.9	24.8	0.223	Higher network cost
Only load balancing	112.3	21.5	0.205	Ignores dependency clustering
Full community-community-SCC (proposed)	105.8	18.3	0.189	Best overall

Table 4. Dependent preservation

Metric	Random	Greedy	Load-aware	Proposed
Community-SCCs placed in same partition (%)	62.3	68.7	75.1	89.2
Cross-community links reduced (%)	15.4	21.6	28.3	42.7
Average inter-node hops per request	4.82	4.15	3.74	2.91

The proposed community-SCC method attains 0.189 lowest standard deviation with most uniform resource usage and 0.935 highest fairness index, which helps manage the nearby even workloads. The minimum and maximum narrow gap load confirms that no nodes are overloaded and it results in balanced and stable system operations. The proposed community-based SCC approach helps to improve the load distribution effectively. Table 5 demonstrates the load and fairness distribution.

The two-level balancing method that combined both inter and intra partition for load management attains better performance. This helps decrease rejection rate and latency effectively while handling the highest fairness and lowest imbalance. The effectiveness of the two-level balancing method increases the load balancing significance. Table 6 illustrates node levels and community balance that improve consistent system stability and efficiency.

Table 5. Fairness and load distribution

Method	Max load (%)	Min load (%)	Std dev	Fairness index (Jain's)
Random	82.4	38.9	0.321	0.703
Greedy	78.1	42.7	0.287	0.779
Load-aware	73.5	48.9	0.238	0.847
Transitive closure	69.8	52.4	0.212	0.891
Proposed	65.2	55.3	0.189	0.935

Table 6. Two-level balancing effectiveness

Method variant	Avg latency (ms)	Rejection rate (%)	Imbalance (std dev)	Fairness index	Comments
Only inter-partition balancing	110.7	22.8	0.203	0.876	Community chosen well, but local overloads occur
Only intra-partition balancing	113.5	24.2	0.191	0.839	Good local load balance, but poor global fairness
Two-level (inter + intra)	105.8	18.3	0.189	0.935	Best results — balanced globally & locally
No balancing (static mapping)	123.4	45.3	0.321	0.703	Baseline poor balance

The proposed community-SCC method is compared with existing approaches to represent the better performance of the model. Table 7 comparative analysis of the proposed and traditional approaches which enhance the efficiency of the model. The community-SCC attains better load balancing by considering less resources with less energy consumption and makespan.

The community-SCC performs effectively when compared with the existing LSTM, AOA, and TSFRL approaches. The proposed approach removes high dimensionality and iterative training computational overhead. The TSFRL and LSTM require backpropagation to learn node dependencies and extensive time, the proposed community-SCC uses structural pruning to isolate communities in an initial transmission while AOA struggles with slow convergence. The community-SCC attains faster execution and superior scalability on large networks.

Table 7. Proposed method comparative analysis with traditional approaches

Method	Load balancing	Resource utilization	Energy consumption	Makespan
AOA [22]	56.9	70.5	300	250
LSTM [23]	58.8	89.7	90	379
TSFRL [24]	69.1	90.8	754	0.447
Proposed	79.8	112.3	250	240

4.3. Discussion

The proposed community-SCC method helps with resource allocation and efficient load balancing while sharing the data. The task loading and resource allocation are identified using community-SCC approach by grouping them into communities. This structure is used to group efficiency increases, which improve workload-balanced distribution across the system and decrease communication overhead. The SCC method arranges tasks that depend on each, thereby detecting interrelated tasks that increase local resources to decrease inter-node communication delays. The community method performs like every community is mapped with the suitable resource groups according to communication cost and computation capacity. The community approach allows the system to analyze the intensity of the workload within every SCC and distribute it evenly across nodes. The two-level adaptive load-balancing mechanism performs together: inter-community minimization, intra-community load equalization, and placement of dependency-aware task together build a unified balancing approach. The proposed community-SCC approach achieves node

utilization of 79.8% and average latency of 112.3 ms for only load balancing. Placing mutually dependent services together within load-aware communities is consistent with the resource-management strategies surveyed in [48], and the community-detection step can be strengthened with refinement-based methods such as Leiden, which guarantees well-connected communities [49].

5. CONCLUSION

The proposed community-SCC approach is used for efficient, robust, and secure cloud access, which improves the overall system performance. Identifying significant workload balancing with respect to proper resource utilization helps for real-world distributed environment. The combination of SCC with the community detection approach is used for effective load balancing. The existing methods faced challenges like communication overhead that occurs due to resource consumption, more time and energy is consumed while exchanging information among different nodes. The traditional models limited inter-task coordination; each task independently ran on the outcomes of other tasks. The community-SCC method attains an average latency of 112.3 ms and node utilization of 79.8% for load balancing.

ACKNOWLEDGMENTS

The author expresses sincere gratitude to the Department of Information Science and Engineering, Vidyavardhaka College of Engineering, Mysuru, Karnataka, India, for their continuous support, guidance, and encouragement throughout the course of this research work.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Srikanta Murthy Rajini	✓			✓		✓	✓	✓	✓		✓	✓	✓	
Chaithanya Dasarakoppal Jayaprakash		✓	✓		✓	✓	✓		✓	✓	✓		✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] S. Azizi, M. Shojafar, P. Farzin, and J. Dogani, "DCSP: A delay and cost-aware service placement and load distribution algorithm for IoT-based fog networks," *Computer Communications*, vol. 215, pp. 9–20, Feb. 2024, doi: 10.1016/j.comcom.2023.12.016.

[2] D. H. Abdulazeez and S. K. Askar, "A novel offloading mechanism leveraging fuzzy logic and deep reinforcement learning to improve IoT application performance in a three-layer architecture within the fog-cloud environment," *IEEE Access*, vol. 12, pp. 39936–39952, 2024, doi: 10.1109/ACCESS.2024.3376670.

[3] D. V. Anh and V.-H. Nguyen, "Leveraging priority queueing in IoT-edge-fog-cloud infrastructures for efficient healthcare monitoring," *IEEE Access*, vol. 13, pp. 80461–80477, 2025, doi: 10.1109/ACCESS.2025.3565679.

[4] J.-P. Yang, "Dynamic threshold-based resource management for fog computing environments," *IEEE Access*, vol. 13, pp. 87898–87908, 2025, doi: 10.1109/ACCESS.2025.3571394.
[5] Z. Shamsa, A. Rezaee, S. Adabi, and A. M. Rahmani, "A decentralized prediction-based workflow load balancing architecture for cloud/fog/IoT environments," *Computing*, vol. 106, no. 1, pp. 201–239, Jan. 2024, doi: 10.1007/s00607-023-01216-3.
[6] R. Kumar and N. Agrawal, "EDMA-RM: An event-driven and mobility-aware resource management framework for green IoT-edge-fog-cloud networks," *IEEE Sensors Journal*, vol. 24, no. 14, pp. 23004–23012, Jul. 2024, doi: 10.1109/JSEN.2024.3404470.
[7] R. Zhanuzak, M. A. Ala'Anzy, M. Othman, and A. Algarni, "Optimizing cloud computing performance with an enhanced dynamic load balancing algorithm for superior task allocation," *IEEE Access*, vol. 12, pp. 183117–183132, 2024, doi: 10.1109/ACCESS.2024.3508793.
[8] M. Zolghadri, P. Asghari, S. E. Dashti, and A. Hedayati, "Dynamic task offloading for IoT-fog-cloud systems: a network traffic-aware decision tree approach," *Computing*, vol. 107, no. 4, Apr. 2025, doi: 10.1007/s00607-025-01449-4.
[9] K. Abedpour, M. H. Shirvani, and E. Abedpour, "A genetic-based clustering algorithm for efficient resource allocating of IoT applications in layered fog heterogeneous platforms," *Cluster Computing*, vol. 27, no. 2, pp. 1313–1331, Apr. 2024, doi: 10.1007/s10586-023-04005-x.
[10] Z. Shamsa, A. Rezaee, S. Adabi, A. M. Rahimabadi, and A. M. Rahmani, "A distributed load balancing method for IoT/fog/cloud environments with volatile resource support," *Cluster Computing*, vol. 27, no. 4, pp. 4281–4320, Jul. 2024, doi: 10.1007/s10586-024-04403-9.
[11] P. B. Reddy and C. Sudhakar, "IOTD: intelligent offloading of tasks with deadlines in edge-fog-cloud computing environment using hybrid approach," *Cluster Computing*, vol. 27, no. 7, pp. 9873–9885, Oct. 2024, doi: 10.1007/s10586-024-04482-8.
[12] D. Yu and W. Zheng, "A hybrid evolutionary algorithm to improve task scheduling and load balancing in fog computing," *Cluster Computing*, vol. 28, no. 1, Feb. 2025, doi: 10.1007/s10586-024-04749-0.
[13] L. Zhao *et al.*, "Improved task scheduling and load balancing in fog computing infrastructure using hybrid evolutionary method," *Cluster Computing*, vol. 28, no. 6, Sep. 2025, doi: 10.1007/s10586-024-05019-9.
[14] F. Arvaneh, F. Zarafshan, and A. Karimi, "Using fog computing (FC) and optimization techniques for tasks migration and resource allocation in the internet of things (IoT)," *International Journal of Computers and Applications*, vol. 46, no. 2, pp. 113–121, Feb. 2024, doi: 10.1080/1206212X.2023.2287257.
[15] S. K. Srichandan, S. K. Majhi, S. Jena, K. Mishra, and R. Bhat, "A secure and distributed placement for quality of service-aware IoT requests in fog-cloud of things: a novel joint algorithmic approach," *IEEE Access*, vol. 12, pp. 56730–56748, 2024, doi: 10.1109/ACCESS.2024.3390723.
[16] M. Maashi *et al.*, "Elevating survivability in next-gen IoT-fog-cloud networks: scheduling optimization with the metaheuristic mountain gazelle algorithm," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 3802–3809, Feb. 2024, doi: 10.1109/TCE.2024.3371774.
[17] M. B. Shaik, K. S. Reddy, K. Chokkanathan, S. A. A. Biabani, P. Shanmugaraja, and D. R. D. Brabin, "A hybrid particle swarm optimization and simulated annealing with load balancing mechanism for resource allocation in fog-cloud environments," *IEEE Access*, vol. 12, pp. 172439–172450, 2024, doi: 10.1109/ACCESS.2024.3489960.
[18] V. C. Bharathi, S. S. Abuthahir, M. Ayyavaraiah, G. Arunkumar, U. Abdurrahman, and S. A. A. Biabani, "O2O-PLB: a one-to-one-based optimizer with priority and load balancing mechanism for resource allocation in fog-cloud environments," *IEEE Access*, vol. 13, pp. 22146–22155, 2025, doi: 10.1109/ACCESS.2025.3536210.
[19] A. Mahapatra, S. K. Majhi, K. Mishra, R. Pradhan, D. C. Rao, and S. K. Panda, "An energy-aware task offloading and load balancing for latency-sensitive IoT applications in the fog-cloud continuum," *IEEE Access*, vol. 12, pp. 14334–14349, 2024, doi: 10.1109/ACCESS.2024.3357122.
[20] A. A. Al-Atawi, "Enhancing data management and real-time decision making with IoT, cloud, and fog computing," *IET Wireless Sensor Systems*, vol. 14, no. 6, pp. 539–562, Dec. 2024, doi: 10.1049/wss2.12099.
[21] Deepak, M. K. Upadhyay, and M. Alam, "Deadline constrained based load balancing algorithm for batch of IoT tasks on fog and edge computing environments," *Iran Journal of Computer Science*, vol. 8, no. 3, pp. 1115–1136, Sep. 2025, doi: 10.1007/s42044-025-00263-7.
[22] S. K. Srichandan, S. K. Majhi, S. Jena, K. Mishra, and D. C. Rao, "Efficient latency-and-energy-aware IoT-fog-cloud task orchestration: novel algorithmic approach with enhanced arithmetic optimization and pattern search," *International Journal of Information Technology*, vol. 16, no. 5, pp. 3311–3324, Jun. 2024, doi: 10.1007/s41870-024-01807-z.
[23] A. Sharma and V. Thangaraj, "A lightweight trajectory aware application placement in IoT-fog-cloud environment," *Journal of Grid Computing*, vol. 23, no. 1, Mar. 2025, doi: 10.1007/s10723-025-09794-6.
[24] R. Ghafari and N. Mansouri, "Fuzzy reinforcement learning algorithm for efficient task scheduling in fog-cloud IoT-based systems," *Journal of Grid Computing*, vol. 22, no. 4, Dec. 2024, doi: 10.1007/s10723-024-09781-3.
[25] P. Karthikeyan and K. Brindha, "Smart data flow: a trust-driven hybrid fragmentation framework for optimizing data transmission in IoT-enabled edge-fog-cloud systems," *Computing*, vol. 107, no. 6, Jun. 2025, doi: 10.1007/s00607-025-01491-2.
[26] A. Deldari and A. Holghinezhad, "An IoT-based bag-of-tasks scheduling framework for deadline-sensitive applications in fog-cloud environment," *Computing*, vol. 107, no. 1, Jan. 2025, doi: 10.1007/s00607-024-01371-1.
[27] A. Tomar, M. Sharma, A. Agarwal, A. N. Jha, and J. Jaiswal, "Task offloading of IoT device in fog-enabled architecture using deep reinforcement learning approach," *Pervasive and Mobile Computing*, vol. 112, Aug. 2025, doi: 10.1016/j.pmcj.2025.102067.
[28] M. Qi, X. Wu, K. Li, and F. Yang, "IPAQ: a multi-objective global optimal and time-aware task scheduling algorithm for fog computing environments," *The Journal of Supercomputing*, vol. 81, no. 2, Jan. 2025, doi: 10.1007/s11227-024-06853-9.
[29] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16, doi: 10.1145/2342509.2342513.
[30] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2018, doi: 10.1109/COMST.2017.2771153.
[31] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec. 2017, doi: 10.1007/s11761-017-0219-8.
[32] A. Brogi and S. Forti, "QoS-aware deployment of IoT applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, Oct. 2017, doi: 10.1109/JIOT.2017.2701408.
[33] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016, doi: 10.1109/JIOT.2016.2565516.
[34] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. New York, United States: Wiley, 1975.
[35] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002, doi: 10.1073/pnas.122653799.

[36] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, Feb. 2004, doi: 10.1103/PhysRevE.69.026113.
[37] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, 2008, doi: 10.1088/1742-5468/2008/10/P10008.
[38] S. Warshall, "A theorem on Boolean matrices," *Journal of the ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962, doi: 10.1145/321105.321107.
[39] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, Jun. 1974, doi: 10.1016/S0022-0000(74)80026-7.
[40] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, Jun. 1972, doi: 10.1137/0201010.
[41] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 1222–1228, doi: 10.23919/INM.2017.7987464.
[42] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, Sep. 2017, doi: 10.1002/spe.2509.
[43] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, Oct. 2019, doi: 10.1016/j.jpdc.2018.03.004.
[44] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3–5, pp. 75–174, Feb. 2010, doi: 10.1016/j.physrep.2009.11.002.
[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, United States: MIT Press, 2009.
[46] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999, doi: 10.1126/science.286.5439.509.
[47] R. Jain, D. M. Chiu, and W. R. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer systems*, Hudson, United States: Digital Equipment Corporation, 1984.
[48] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys*, vol. 52, no. 5, Sep. 2019, doi: 10.1145/3326066.
[49] V. A. Traag, L. Waltman, and N. J. V. Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, Mar. 2019, doi: 10.1038/s41598-019-41695-z.

BIOGRAPHIES OF AUTHORS



Srikanta Murthy Rajini is currently an associate professor in Department of Information Science and Engineering at the Vidyavardhaka College of Engineering. She obtained her Ph.D. in Computer and Information Sciences under Visvesvaraya Technological University, Belagavi, Karnataka, India. Her area of interest includes wireless sensor networks, internet of things, big data analytics, and machine learning. She can be contacted at email: rajinis@vvce.ac.in.



Chaithanya Dasarakoppal Jayaprakash received a Bachelor of Engineering in Electronics and Communication Engineering from Visvesvaraya Technological University, Belgaum in 2012; an M.Tech. in VLSI Design and Embedded Systems in 2014; and a Ph.D. degree in 5G communication from Visvesvaraya Technological University, Belgaum in 2024 respectively. She is working as an assistant professor in Department of Electronics and Communication Engineering at the Vidyavardhaka College of Engineering, Mysuru. She has 8.5 years of teaching experience. She can be contacted at email: rcchaithudj@gmail.com.