VECTORAL PUBLISHING

Date of publication 11/01/2029, date of current version 11/01/2020.

VECTORAL manuscript Identifier 11.01-5/VECTORAL.2020.PUB

COMPARATIVE ANALYSIS OF HEURISTIC AND AI-BASED TASK SCHEDULING ALGORITHMS IN FOG COMPUTING: EVALUATING LATENCY, ENERGY EFFICIENCY, AND SCALABILITY IN DYNAMIC, HETEROGENEOUS ENVIRONMENTS

KAUSHIK SATHUPADI[1]

[1]Staff Engineer, Google LLC, Sunnyvale, CA

Corresponding author: Sathupadi, K.

ABSTRACT Fog computing extends cloud capabilities to the edge of the network, offering a distributed infrastructure to handle latency-sensitive, bandwidth-intensive applications. Task scheduling process in fog computing determines how computational tasks are allocated to heterogeneous, resource-constrained fog nodes. The dynamic and unpredictable nature of fog environments, characterized by stochastic task arrivals and fluctuating resource availability, adds complexity to the scheduling process. This paper presents a comparative study of heuristic and AI-based scheduling algorithms, focusing on their effectiveness in managing task allocation under dynamic and heterogeneous conditions. Heuristic algorithms are known for their computational efficiency and low complexity, making them suitable for resource-limited environments. However, they often lack the adaptability required to handle the stochasticity of real-world fog networks. AI-based scheduling algorithms using machine learning and optimization techniques can provide flexibility and adaptability by learning from system dynamics and predicting future states. This study evaluates these approaches using performance metrics such as latency, energy consumption, resource utilization, and scalability. The findings reveal trade-offs between the computational overhead associated with AI-based methods and their superior performance in dynamic, heterogeneous environments.

INDEX TERMS AI-based Scheduling, Fog Computing, Heuristic Algorithms, Resource Utilization, Scalability, Task Scheduling

I. INTRODUCTION

Fog computing, positioned as an intermediary between centralized cloud systems and the rapidly expanding ecosystem of Internet of Things (IoT) devices, is engineered to mitigate the inherent deficiencies of cloud-based models (Bonomi et al., 2012) (Chen et al., 2017). It achieves this by relocating computational, storage, and networking resources closer to the data's origin—on fog nodes—thus facilitating more efficient and responsive data processing. This is crucial in scenarios demanding real-time interaction and decision-making, such as in autonomous driving systems, industrial automation, or healthcare applications, where delays in processing could lead to catastrophic consequences (Dastjerdi et al., 2016) (Iorga et al., 2018). The issue of task scheduling in fog environments is exacerbated by the sheer diversity of fog nodes, which may range from high-performance servers to small embedded systems with limited computational power and energy reserves. Unlike the relatively homogeneous environment of a cloud data center, fog nodes vary dramatically in terms of processing capabilities, storage capacity, network bandwidth, and power availability. This variability introduces substantial complexity into the task scheduling process. Moreover, the dynamic nature of fog networks—where devices might enter and leave the network intermittently—only adds to the uncertainty, rendering traditional task scheduling paradigms largely ineffective. The challenge lies in determining which fog node should execute a given task, while optimizing performance metrics such as latency, energy consumption, and resource utilization (Kraemer et al., 2017) (Mahmud et al., 2018).

Latency is a critical factor in fog computing, as one of the primary motivations for shifting computational resources

closer to the edge is to reduce the time delay between data generation and task execution. However, latency is not a simple metric to optimize in a fog environment. Each node's physical proximity to the source of the data is only one variable in a complex equation that includes network congestion, node workload, and the inherent processing speed of the fog node itself (Yi et al., 2015). Some nodes might be closer geographically but are bottlenecked by limited bandwidth or are already overwhelmed by existing tasks. Furthermore, latency is impacted by the stochastic nature of task arrivals, which makes it nearly impossible to predict how many tasks will arrive at a given time and where. This unpredictability ensures that any static scheduling model will inevitably fall short, as it cannot account for the fluid and fluctuating nature of fog networks.

Energy consumption further complicates the scheduling process. Many fog nodes, especially those at the edge of the network, are severely constrained in terms of energy availability. These nodes may rely on battery power or other limited energy sources, making it essential to conserve energy wherever possible. However, energy efficiency comes at a price. Running tasks on lower-powered devices might conserve energy, but it also extends the time required for task execution, thus increasing latency. Conversely, offloading tasks to higher-powered nodes might reduce latency but at the cost of rapidly depleting energy reserves. This tension between conserving energy and minimizing latency creates a scheduling dilemma with no simple or universally optimal solution. The variability in energy consumption between nodes, due to differences in hardware architecture and operational modes (e.g., low-power states versus full-power states), only serves to complicate this already delicate balance.

Resource utilization, yet another pressing issue, becomes a critical concern in a fog environment where resources are finite and often statically allocated. Unlike cloud environments, where resources can be dynamically provisioned and scaled, fog nodes have hard limits on their computational and storage capacities. This necessitates a careful allocation of tasks to avoid resource contention, which can lead to task delays or failures. Overloading a fog node with too many tasks can degrade its performance significantly, while under-utilization of nodes results in wasted resources. The problem is further amplified by the unpredictable nature of task arrivals, as sudden bursts of tasks can overwhelm nodes, leading to bottlenecks and a degradation of overall network performance. The heterogeneity of the nodes only makes this balancing act more difficult, as each node's capabilities must be considered individually, rather than relying on a one-size-fits-all scheduling strategy.

Moreover, the fog environment is inherently decentralized, which presents profound challenges in coordinating task scheduling. Unlike the cloud, where a central authority can oversee and manage resource allocation, fog nodes often operate autonomously, without centralized control. This decentralization leads to inefficiencies (Dastjerdi et al., 2016), as fog nodes might make independent scheduling decisions without awareness of the global state of the network. A node might accept a task based on its local conditions, unaware that a nearby node is better suited to handle the task. The lack of coordination can also result in suboptimal resource allocation, with some nodes becoming overloaded while others remain idle. The decentralized nature of fog computing introduces a layer of complexity that simply does not exist in cloud environments, as there is no single point of control to manage and optimize the distribution of tasks.

The scheduling problem in fog computing is further complicated by the security and privacy concerns that arise when distributing tasks across a decentralized network of nodes. In applications where sensitive data is involved, such as healthcare or financial services, the task scheduling algorithm must account not only for computational and energy constraints but also for the security of the nodes. Different fog nodes may have varying levels of security, and scheduling sensitive tasks on nodes with inadequate security measures could expose the data to unauthorized access or manipulation. Additionally, the decentralized nature of fog computing makes it difficult to enforce uniform security policies across all nodes, as each node might be subject to different regulatory requirements or operate under different trust levels. The need to balance security considerations with performance and energy efficiency adds yet another layer of complexity to the scheduling problem, creating trade-offs that are difficult, if not impossible, to resolve in a satisfactory manner.

Furthermore, task scheduling in fog computing is not just a static problem but one that evolves continuously as the network changes over time (Arunarani et al., 2019). Fog nodes can enter and leave the network, either due to mobility, failure, or energy depletion, and the network topology itself may shift as nodes connect and disconnect. This dynamic nature means that any scheduling solution must be highly adaptable and capable of making real-time decisions under conditions of uncertainty. However, developing such adaptable scheduling algorithms is a daunting task, as it requires real-time monitoring of network conditions, predictive modeling of task arrivals, and the ability to make decisions on incomplete or rapidly changing information. The unpredictability of both task arrivals and network conditions makes it extremely difficult to develop a scheduling algorithm that can consistently optimize for all performance metrics. The inherent volatility of the fog network, combined with the diversity of nodes and the stochastic nature of tasks, creates an intractable problem with no clear solution.

Despite the growing interest in fog computing, the task scheduling problem remains a significant barrier to its widespread adoption. The decentralized, heterogeneous, and dynamic nature of fog networks creates a perfect storm of challenges that cannot be easily addressed by traditional scheduling methods. The variability in node capabilities, the unpredictability of task arrivals, the tension between latency and energy consumption, and the need for secure task allocation all contribute to a problem that is both complex and multidimensional. Furthermore, the decentralized nature of

fog networks makes it difficult to coordinate scheduling decisions across nodes, leading to inefficiencies and suboptimal performance. As fog computing continues to evolve, the task scheduling problem will remain a critical challenge that must be addressed before the full potential of this paradigm can be realized. Traditional heuristic algorithms, while computationally efficient, are often unable to adapt to these changing conditions, making AI-based approaches, which can learn from and respond to environmental changes, a promising alternative (Atabakhsh, 1991).

This paper conducts a comparative study of heuristic and AI-based scheduling algorithms in fog computing. We focus on the effectiveness of these algorithms in dynamic, heterogeneous environments with stochastic task arrivals. The main objective is to evaluate the performance of these algorithms in terms of key metrics, such as task completion time (latency), energy efficiency, load balancing, and scalability.

A. BACKGROUND ON FOG COMPUTING AND SCHEDULING CHALLENGES

Fog Computing Architecture Fog computing introduces an intermediate layer between cloud services and IoT devices, distributing computational resources across a wide array of devices, including routers, gateways, and micro-data centers located near the network's edge. This architecture is inherently heterogeneous, with fog nodes differing significantly in computational power, memory, network bandwidth, and energy resources. Tasks in fog computing environments can range from lightweight operations, such as data filtering and aggregation, to more complex computations like real-time analytics.

The dynamic nature of fog environments is another critical challenge. Fog nodes may experience varying workloads over time, with task demands fluctuating due to factors such as user activity, sensor data influx, and environmental conditions. Moreover, stochastic task arrivals mean that tasks arrive unpredictably, with no prior knowledge of their arrival time, size, or resource requirements. These characteristics necessitate sophisticated scheduling algorithms capable of making real-time decisions under uncertainty.

Scheduling Objectives and Challenges in Fog Computing The primary objectives of scheduling in fog computing are to minimize task completion time (latency), reduce energy consumption, maximize resource utilization, and ensure scalability. Achieving these objectives is challenging due to the following factors:

Heterogeneous Resources: Fog nodes differ significantly in their processing power, memory, and communication capabilities, making it difficult to develop a unified scheduling strategy that can effectively utilize all nodes.

Latency Sensitivity: Many fog applications, such as autonomous vehicles and industrial IoT systems, require ultra-low latency to function effectively. Scheduling algorithms must prioritize tasks based on their latency sensitivity while ensuring that other system performance metrics are not degraded.

Energy Constraints: Many fog nodes, those deployed on IoT devices or edge routers, operate under limited energy budgets. Scheduling algorithms must optimize for energy consumption to extend the lifetime of these nodes while still meeting performance requirements.

Stochastic Task Arrivals: Task arrivals in fog environments are typically random, with unpredictable sizes and deadlines. This stochasticity increases the complexity of scheduling, as the system must make decisions without prior knowledge of future tasks or system states.

Scalability: As the number of devices and applications utilizing the fog network grows, the scheduling algorithm must scale efficiently, handling larger workloads and more fog nodes without a significant increase in scheduling overhead.

II. COMPARATIVE ANALYSIS OF HEURISTIC AND AI-BASED SCHEDULING ALGORITHMS

A. HEURISTIC SCHEDULING ALGORITHMS

The First-Come-First-Served (FCFS) scheduling algorithm is one of the simplest and most straightforward methods for allocating resources in fog and cloud computing environments. In FCFS, tasks are processed strictly in the order they arrive. This approach has the distinct advantage of simplicity and ease of implementation, requiring minimal computational overhead to track the sequence of task arrivals. The algorithm's intuitive design allows for quick task assignments without the need for complex calculations, which can be beneficial in environments where computational resources are scarce or where the goal is to minimize latency in task assignment.

Task	Arrival Time	Node Assigned	Execution Time
T1	0	Node 1	5
T2	1	Node 2	3
T3	2	Node 1	6
T4	3	Node 2	4
T5	4	Node 1	7
T6	5	Node 2	5

TABLE 1. Illustration of FCFS scheduling algorithm. Tasks are assigned to nodes based on their arrival times, without consideration of task priority, system load, or node heterogeneity.

However, FCFS suffers from several critical drawbacks, in dynamic and heterogeneous environments such as fog computing. One major limitation is the algorithm's failure to account for task priorities or the heterogeneity of the system's nodes. Tasks with higher priority, such as time-sensitive tasks in real-time applications, may be delayed significantly if they are placed behind lower-priority tasks in the queue. This can lead to inefficiencies in systems where certain tasks require immediate attention, such as in healthcare or emergency response applications, where a delay could be catastrophic.

Moreover, the FCFS method does not account for node heterogeneity, a common characteristic of fog computing environments. Fog nodes can vary significantly in terms of processing power, memory capacity, and network bandwidth. By scheduling tasks in the order they arrive, FCFS may

assign resource-intensive tasks to weaker nodes, while more capable nodes remain idle or underutilized. This leads to inefficient resource utilization, with some nodes becoming overloaded while others remain underloaded. The overall system performance degrades as a result, with tasks taking longer to execute, leading to higher task response times and decreased throughput.

Another significant issue with FCFS is its lack of adaptability to system load. In dynamic environments where the system load fluctuates rapidly, such as in fog computing, FCFS is unable to make intelligent scheduling decisions based on current load conditions. This often results in scenarios where the system becomes overloaded, as the algorithm continues to assign tasks without considering the current resource availability or load distribution across nodes. Consequently, tasks may experience longer wait times in the queue, and the overall system throughput can decrease substantially.

Figure **??** illustrates the operation of the FCFS scheduling algorithm. As shown, tasks are scheduled in the order they arrive, with no consideration given to task priority, system load, or node heterogeneity. While computationally inexpensive, this method can lead to significant performance issues in dynamic and heterogeneous fog computing environments.

The Round Robin (RR) scheduling algorithm represents an alternative to FCFS that attempts to distribute tasks more evenly across available nodes. In RR, tasks are assigned to fog nodes in a cyclical manner, ensuring that each node is allocated a roughly equal number of tasks. This method promotes fairness in resource allocation by preventing any single node from becoming overwhelmed with tasks, as each node is given its turn in a round-robin fashion. The simplicity of RR also makes it attractive in environments where computational overhead needs to be minimized, and its deterministic nature ensures predictability in task scheduling (El-Rewini et al., 1994).

Task	Arrival Time	Node Assigned (Round 1)	Node Assigned (Round 2)
T1	0	Node 1	Node 2
T2	1	Node 2	Node 3
T3	2	Node 3	Node 1
T4	3	Node 1	Node 2
T5	4	Node 2	Node 3
T6	5	Node 3	Node 1

TABLE 2. Illustration of Round Robin scheduling algorithm. Tasks are assigned cyclically to nodes without considering node capabilities or task priorities.

Despite its advantages, RR also suffers from several shortcomings, in environments where nodes have varying capabilities. One of the main weaknesses of RR is its failure to account for the heterogeneity of fog nodes. Since RR blindly assigns tasks to nodes in a circular manner, it ignores the fact that different nodes may have different processing capacities. More powerful nodes may end up being underutilized, as they receive the same number of tasks as weaker nodes. Similarly, weaker nodes may become overloaded, as they are assigned tasks that they are ill-equipped to handle. This lack of awareness of node capabilities can lead to inefficient resource utilization, with some tasks taking longer to complete than they would under a more adaptive scheduling algorithm.

Another drawback of RR is its lack of consideration for task priority and resource requirements. Tasks with urgent deadlines or higher resource demands are treated the same as less urgent tasks, potentially leading to delays in the execution of time-critical applications. This can be especially problematic in fog computing environments, where tasks may have varying degrees of importance and urgency.

Figure **??** demonstrates the operation of the RR algorithm. Tasks are distributed evenly across nodes in a circular manner, without consideration for node capabilities or task priorities. While this method ensures a balanced load distribution in homogeneous environments, it may result in underutilization or overloading of nodes in heterogeneous fog environments, ultimately affecting system performance.

Greedy algorithms represent another class of scheduling strategies, which aim to make locally optimal decisions at each step of the scheduling process. For instance, a task may be assigned to the node with the lowest current load or the highest available processing power. Greedy algorithms are attractive due to their simplicity and speed, as they make decisions based solely on the current system state without needing to account for future tasks or system conditions. This makes them effective in static environments, where task arrival patterns and resource availability remain relatively constant.

However, in dynamic fog computing environments, greedy algorithms often struggle to achieve globally optimal performance. Since these algorithms focus only on immediate gains, they may fail to account for long-term system states or future task arrivals. This can lead to suboptimal scheduling decisions, where a task is assigned to a node that appears to be the best option at the moment but may not be able to handle future workloads efficiently. Over time, this can result in system-wide inefficiencies, as tasks pile up on certain nodes while others remain underutilized.

Furthermore, greedy algorithms often struggle with the stochastic nature of task arrivals in fog computing. In such environments, task arrival rates can fluctuate significantly, and resources may become available or unavailable at any time. Since greedy algorithms base their decisions solely on the current system state, they are unable to anticipate these changes, leading to poor performance in dynamic environments.

Figure **??** illustrates how greedy algorithms allocate tasks to the least-loaded resource. While this strategy can lead to fast scheduling decisions in static environments, it may result in suboptimal long-term performance in dynamic systems where future tasks and demands are unpredictable.

In addition to FCFS, RR, and greedy algorithms, Min-Min and Max-Min heuristics are also commonly used for scheduling tasks in fog computing environments. The Min-Min heuristic operates by assigning the smallest task to

Task	Arrival Time	Node Load	Node Assigned (Greedy)	Execution Time
T1	0	Node 1: 10%, Node 2: 20%, Node 3: 15%	Node 1	5
T2	1	Node 1: 30%, Node 2: 20%, Node 3: 15%	Node 3	4
T3	2	Node 1: 35%, Node 2: 20%, Node 3: 40%	Node 2	6
T4	3	Node 1: 35%, Node 2: 45%, Node 3: 40%	Node 1	3
T5	4	Node 1: 60%, Node 2: 45%, Node 3: 40%	Node 3	7

TABLE 3. Illustration of Greedy scheduling algorithm. Tasks are assigned to the least-loaded node at the time of arrival, without considering future tasks or long-term system load distribution.

the node that can complete it the fastest, with the goal of minimizing the overall execution time for smaller tasks. This method can be effective in environments where smaller tasks are more common, as it reduces their waiting time and ensures that they are processed quickly. However, one of the main drawbacks of Min-Min is that it can lead to significant delays for larger tasks, as they are placed behind smaller tasks in the queue.

The Max-Min heuristic, on the other hand, prioritizes larger tasks. By assigning the largest task to the node that can complete it the fastest, Max-Min ensures that larger tasks are not delayed indefinitely by smaller tasks. This method is suited for heterogeneous environments, where nodes may have varying capabilities and tasks may have differing resource requirements. However, Max-Min can also lead to inefficiencies in cases where the system is dominated by smaller tasks, as these tasks may end up waiting longer than necessary.

Figure **??** compares the Min-Min and Max-Min algorithms. Min-Min schedules smaller tasks first, reducing their waiting time at the cost of delaying larger tasks. In contrast, Max-Min prioritizes larger tasks, ensuring that they do not dominate system resources and that smaller tasks are not delayed unnecessarily. Heuristic algorithms are typically based on static rules, which means they cannot adapt to changes in task arrival rates, node availability, or system load. This lack of adaptability makes them less suitable for fog computing environments, where conditions are constantly changing.

B. AI-BASED SCHEDULING ALGORITHMS

AI-based scheduling algorithms apply machine learning (ML) techniques to model the complex, dynamic nature of fog computing environments. These algorithms can learn from historical data, predict future system states, and adapt their scheduling decisions accordingly. Common AI-based scheduling approaches are discussed in this section.

1) Reinforcement Learning (RL)

Reinforcement learning (RL) is increasingly becoming a critical approach in optimizing task scheduling in fog computing environments, where computation, storage, and network resources are distributed across multiple layers. Fog computing, which is characterized by its ability to bring cloud services closer to end users, necessitates efficient task scheduling to meet the demands of low-latency, high-throughput applications, in the context of Internet of Things (IoT) devices. Traditional scheduling algorithms, which rely on predefined heuristics or static optimization strategies, often struggle to cope with the dynamic and unpredictable nature of fog environments. This is where RL offers a more robust and adaptive solution by learning from the environment and continuously refining scheduling decisions over time (Noronha & Sarma, 1991).

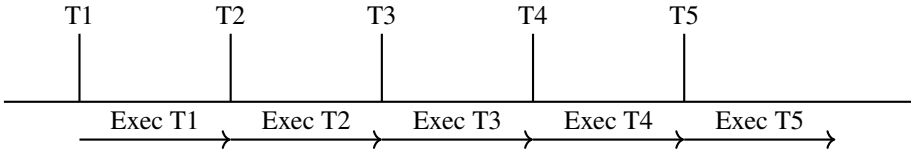
In reinforcement learning, an agent interacts with the environment, learning to take actions that maximize cumulative rewards over time. This process is typically modeled as a Markov Decision Process (MDP), where the environment's state changes in response to the agent's actions, and the agent receives feedback in the form of rewards. The goal of the agent is to learn a policy that maps states to actions, such that the expected long-term reward is maximized. In the context of fog computing, the environment consists of a distributed network of fog nodes with varying resource capacities, network conditions, and workload demands. The state of the environment includes information about the available resources, such as the load on each fog node, their energy consumption levels, and the incoming tasks that need to be scheduled. The actions represent different task scheduling decisions, which may involve allocating a specific task to a particular fog node, delaying a task, or offloading tasks to the cloud when local resources are insufficient.

One of the most prominent algorithms used in RL for task scheduling is Q-learning, a model-free method that aims to learn the optimal action-value function. The action-value function, denoted as Q(s, a), estimates the expected cumulative reward for taking action a in state s and following the optimal policy thereafter. The Q-learning algorithm updates the Q-values based on the Bellman equation, which expresses the value of a state-action pair in terms of the immediate reward and the discounted value of the best future action. In fog computing, the Q-learning process begins by initializing a Q-table, where each entry corresponds to a state-action pair. The agent then explores the environment by selecting actions according to an exploration-exploitation strategy. Initially, the agent may choose random actions to explore the space of possible task scheduling decisions, but as it gathers more information about the system's behavior, it begins to favor actions that yield higher Q-values.

Each time the agent takes an action, such as assigning a task to a fog node, the environment transitions to a new state based on the system's response to that decision. The agent then receives a reward, which is designed to reflect the quality

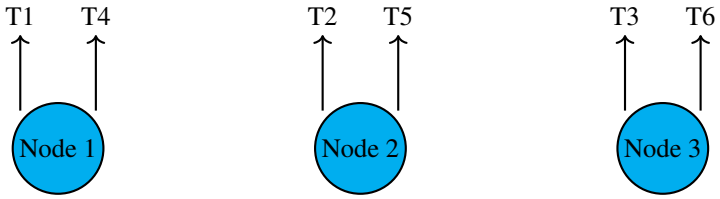
Task	Arrival Time	Task Size	Node Assigned (Min-Min)	Node Assigned (Max-Min)	Execution Time
T1	0	Small	Node 2	Node 1	3
T2	1	Medium	Node 3	Node 2	6
T3	2	Large	Node 1	Node 3	8
T4	3	Small	Node 2	Node 1	2
T5	4	Large	Node 3	Node 3	7

TABLE 4. Comparison of Min-Min and Max-Min heuristics. Min-Min assigns smaller tasks to nodes that can complete them the fastest, prioritizing fast execution of small tasks. Max-Min, on the other hand, assigns larger tasks to the fastest nodes, ensuring that they are not delayed by smaller tasks.



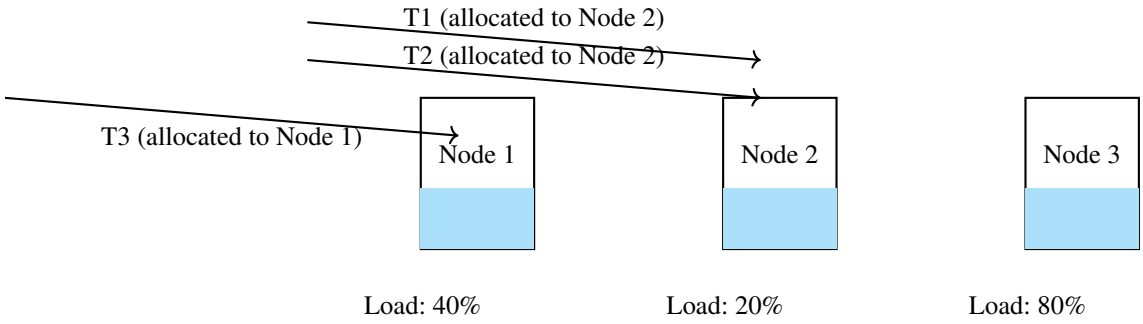
Tasks executed in First-Come-First-Served (FCFS) order

FIGURE 1. First-Come-First-Served (FCFS): Tasks are scheduled in the order they arrive. This method does not consider task priority, system state, or resource availability. While computationally inexpensive, it may lead to poor performance in dynamic environments.



Tasks distributed evenly across nodes in a circular (Round Robin) manner.

FIGURE 2. Round Robin (RR): Tasks are evenly distributed across available nodes in a circular manner. This scheduling method balances load but may result in underutilization or overloading due to the disregard of node capabilities and resource requirements.



Tasks are allocated to the least-loaded resource, following a greedy scheduling strategy.

FIGURE 3. Greedy Algorithms: Tasks are allocated based on the immediate best available resource, such as the least-loaded node. While fast, greedy algorithms fail to account for future tasks or demands, leading to potential suboptimal long-term performance.

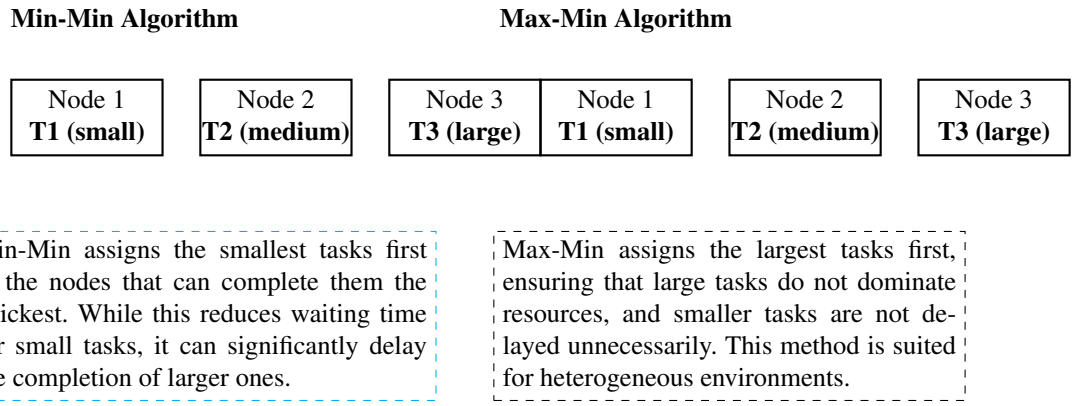


FIGURE 4. Min-Min and Max-Min Algorithms: Min-Min schedules smaller tasks first, reducing waiting time for small tasks at the cost of delaying larger tasks. Max-Min prioritizes larger tasks, ensuring they are handled without significantly delaying smaller tasks, which is better suited for heterogeneous environments.

TABLE 5. FCFS Scheduling Algorithm

Aspect	Details
Advantages	Simple to implement, minimal computational overhead, no complex calculations required
Disadvantages	Ignores task priority, node heterogeneity, and system load; leads to inefficient resource utilization
Environment Suitability	Suitable for static, non-real-time environments with homogeneous nodes

TABLE 6. Round Robin (RR) Scheduling Algorithm

Aspect	Details
Advantages	Ensures fair distribution of tasks across nodes, simple and predictable
Disadvantages	Ignores node capabilities, task priority, and resource demands; can overload weaker nodes
Environment Suitability	Suitable for homogeneous environments where fairness in task distribution is prioritized

TABLE 7. Greedy Scheduling Algorithm

Aspect	Details
Advantages	Makes fast, locally optimal decisions based on current system state
Disadvantages	Poor long-term performance, especially in dynamic environments; does not account for future system conditions
Environment Suitability	Effective in static systems with stable task arrivals and node capabilities

TABLE 8. Min-Min Heuristic Scheduling Algorithm

Aspect	Details
Advantages	Minimizes waiting time for smaller tasks, reducing overall task execution time for short tasks
Disadvantages	Larger tasks may experience significant delays; not adaptable to fluctuating system loads
Environment Suitability	Best suited for environments dominated by small tasks

TABLE 9. Max-Min Heuristic Scheduling Algorithm

Aspect	Details
Advantages	Prioritizes larger tasks, preventing them from dominating system resources
Disadvantages	Smaller tasks may experience unnecessary delays if system is not balanced
Environment Suitability	Ideal for heterogeneous environments with tasks of varying sizes and resource demands

of the scheduling decision in terms of system performance metrics. In fog computing, rewards are typically based on key objectives like minimizing task execution latency, reducing energy consumption, and balancing the load across fog nodes. For example, an action that results in low-latency task execution may receive a high reward, while an action that leads to task delays or node overloads is penalized. The Q-values are updated after each action, gradually improving the agent's understanding of which scheduling decisions lead to the best overall performance.

The ability of RL to adapt over time is one of its main strengths in fog computing environments, which are inherently dynamic. Task arrival rates, resource availability, and network conditions can fluctuate unpredictably, making static scheduling approaches inefficient. RL agents, on the other hand, continuously learn from real-time feedback and adjust their scheduling strategies accordingly. This makes RL well-suited for environments with stochastic task arrivals and varying resource demands, as it allows the system to optimize scheduling decisions based on the current state of the system rather than relying on predetermined rules or assumptions.

In addition to its adaptive nature, reinforcement learning does not require a model of the environment. This is a significant advantage in fog computing, where the complexity of the system makes it difficult to construct accurate models of resource availability, network conditions, and workload patterns. Unlike model-based approaches, which depend on precise system models to predict future states, RL operates in a model-free manner, learning directly from interaction with the environment. This allows RL-based task scheduling algorithms to be applied in a wide range of fog computing scenarios, even when the underlying system dynamics are not fully understood or are too complex to be modeled explicitly.

RL allows for the incorporation of multiple performance metrics into the reward function, enabling multi-objective optimization in task scheduling. Fog computing systems often need to balance conflicting objectives, such as minimizing task completion time while reducing energy consumption or maintaining high throughput while avoiding overloading specific nodes. By carefully designing the reward function, it is possible to guide the RL agent towards policies that make trade-offs between these different objectives. For instance, the reward function could penalize actions that lead to high energy consumption while rewarding those that improve load balancing across nodes, thus encouraging the agent to find a balance between energy efficiency and system performance.

In fog computing, task scheduling decisions often need to be made on the fly as tasks arrive unpredictably, and the system's state changes rapidly. RL-based scheduling algorithms can make these real-time decisions by relying on the learned Q-values, which provide an estimate of the long-term benefit of each possible action. This allows the RL agent to make informed scheduling decisions that account not only for the immediate effects of an action but also for its impact on future system performance.

The flexibility of reinforcement learning also extends to the ability to scale across various fog computing architectures, from small-scale local networks to large-scale distributed systems with numerous fog nodes and heterogeneous resources. As fog computing systems grow in size and complexity, the number of possible states and actions increases, making it difficult for traditional scheduling approaches to cope with the expanded decision space. RL, however, is well-suited to handle such complexity through exploration of the state-action space and the gradual improvement of its scheduling policy over time. This scalability makes RL an attractive option for task scheduling in large, geographically dispersed fog environments, where resource heterogeneity and varying network conditions add layers of complexity to the scheduling problem.

Algorithm 1: Reinforcement Learning for Task Scheduling in Fog Computing

Input: Task set $T = \{t_1, t_2, \ldots, t_n\}$, fog environment state S, reward R, discount factor γ
Output: Optimal task scheduling policy π^*

Initialize Q-table $Q(S, A)$, learning rate α, exploration rate ϵ
while *not converged* **do**
  Observe the current state S
  if *Random number* $< \epsilon$ **then**
    Select a random action (task allocation) A
  else
    Select action $A = \arg\max_a Q(S, a)$
  end
  Execute action A, observe reward R and next state S'
  Update Q-value:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a'} Q(S', a') \right.$$

  - Q(S, A) Set $S \leftarrow S'$
  end while
end
Return optimal policy $\pi^*(S) = \arg\max_a Q(S, a)$

2) Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) represents a significant advancement in the field of reinforcement learning, in its ability to handle large-scale, complex environments. By integrating deep neural networks with traditional reinforcement learning techniques, DRL enables the agent to approximate optimal policies even when the state and action spaces are large and high-dimensional. This makes DRL highly suitable for fog computing environments, where task scheduling involves multiple interconnected nodes, varying workloads, and dynamic resource availability. Traditional reinforcement learning algorithms, such as Q-learning, struggle in such environments due to the "curse of dimensionality," where the number of possible states and actions becomes too large to handle with simple tabular methods. DRL addresses this limitation by using neural networks to approximate the action-value function, allowing the agent to make decisions based on high-level representations of the system's state.

In fog computing, the need for efficient task scheduling is critical. The system must make decisions regarding where to allocate computational tasks based on factors like task priority, node load, energy consumption, and network conditions. The state of the environment in this context is highly complex, as it includes the current status of all available fog nodes, the characteristics of incoming tasks, and the overall system load. Additionally, the action space is large, since the agent must decide on the optimal assignment of each task to a specific fog node or determine whether tasks should be delayed or offloaded to the cloud. DRL excels in this scenario by leveraging the power of deep neural networks to generalize across similar states and actions, thereby enabling more sophisticated and efficient scheduling policies.

A typical DRL framework for task scheduling in fog computing begins with the initialization of two neural networks: the primary network $Q(S, A; \theta)$, which estimates the action-value function for the current state-action pairs, and the target network $Q'(S, A; \theta^-)$, which is a periodically updated copy of the primary network. The target network is used to stabilize the training process by providing more consistent target values during updates. Additionally, the algorithm initializes a replay memory D, which stores past experiences, or transitions, in the form of tuples (S, A, R, S'). This replay memory plays a crucial role in the training process by breaking the temporal correlations between consecutive experiences and enabling the agent to learn from a diverse set of experiences through minibatch training.

The agent interacts with the fog environment by observing

the current state S, which includes detailed information about the fog nodes, their resource capacities, and the incoming tasks. Based on this state, the agent selects an action A, which represents a specific task scheduling decision. During the early stages of training, the agent follows an exploration-exploitation strategy, where it occasionally selects random actions with probability ϵ to explore the environment and gather information. As training progresses and the agent becomes more confident in its estimates of the Q-values, it shifts towards exploitation, selecting actions that maximize the Q-values based on the learned policy. Once the agent selects an action, the fog environment transitions to a new state S', and the agent receives a reward R, which quantifies the quality of the scheduling decision. The reward function is designed to capture key performance metrics in fog computing, such as minimizing task execution latency, reducing energy consumption, and balancing the load across nodes.

After each action, the agent stores the transition (S, A, R, S') in the replay memory. Periodically, when the memory reaches a sufficient size, the agent samples a minibatch of transitions from the replay memory and performs updates to the neural network. For each transition in the minibatch, the agent computes the target value y_j, which is derived from the Bellman equation. The target value depends on whether the episode has ended or not. If the episode has ended, meaning the task scheduling decision has been fully executed (Sinnen, 2007), the target value is simply the immediate reward R_j. Otherwise, the target value includes both the immediate reward and the discounted value of the best future action, as predicted by the target network $Q'(S', A'; \theta^-)$. The agent then performs a gradient descent step on the loss function, which measures the difference between the predicted Q-value $Q(S_j, A_j; \theta)$ and the target value y_j. This process updates the weights θ of the primary neural network, improving its ability to estimate Q-values for future state-action pairs.

One of the key innovations in DRL is the use of experience replay, which allows the agent to learn from past experiences in a more efficient and stable manner. By sampling a minibatch of transitions from the replay memory, the agent can avoid overfitting to recent experiences and learn from a broader range of state-action pairs. This helps to reduce the variance in the learning process and leads to more reliable convergence towards the optimal policy. Furthermore, the use of the target network provides additional stability to the training process by decoupling the target values from the current Q-value estimates. The target network is updated less frequently than the primary network, which prevents the Q-values from oscillating too much during training.

As the training progresses, the agent's neural network gradually converges to an optimal task scheduling policy. This policy maps the current state of the fog environment to the best possible scheduling action, taking into account various factors like task priority, the available resources on each fog node, and energy consumption. The agent learns to balance these competing objectives by maximizing the cumulative reward over time. For example, it may prioritize tasks with higher urgency, allocate tasks to underutilized nodes to prevent overload, or consider energy-efficient scheduling strategies that extend the lifetime of fog nodes. The learned policy is inherently adaptive, meaning that it can respond to changes in the fog environment, such as fluctuating task arrival rates or dynamic changes in resource availability.

Traditional scheduling algorithms are often limited by their reliance on predefined rules or static heuristics, which may not scale well in large, distributed systems with heterogeneous resources. In contrast, DRL can learn directly from the environment, allowing it to discover novel scheduling strategies that may not be apparent through manual design. The neural network architecture used in DRL enables the agent to capture complex relationships between different system variables, such as the trade-offs between latency, energy consumption, and load balancing. This results in more efficient and effective task scheduling policies that are better suited to the diverse and dynamic nature of fog computing environments (Topcuoglu et al., 2002).

3) Genetic Algorithms (GA)

Genetic algorithms (GA) represent a class of evolutionary optimization techniques that have gained significant traction in solving complex, nonlinear optimization problems. Inspired by the process of natural selection and genetics, GAs iteratively evolve a population of candidate solutions towards an optimal or near-optimal solution over successive generations. In the context of fog computing, where task scheduling is a critical and complex problem due to the distributed nature of resources, GAs provide a flexible and robust method for searching across vast solution spaces. Fog computing environments are characterized by heterogeneous resources, dynamic workloads, and the need to balance multiple objectives, such as minimizing task completion time, balancing load across nodes, and reducing energy consumption. These characteristics make traditional task scheduling algorithms less effective as they are often static and cannot easily adapt to fluctuating conditions (Omara & Arafa, 2010). GAs, with their population-based search and ability to explore a wide range of solutions, are well-suited to such dynamic environments.

A genetic algorithm begins by initializing a population of potential solutions, where each individual in the population represents a possible task scheduling strategy. The initial population is often generated randomly to ensure diversity, which is essential for the algorithm to explore different regions of the solution space. Each individual is evaluated using a fitness function that measures the quality of the scheduling solution based on specific performance criteria relevant to fog computing. The fitness function can be designed to optimize multiple objectives simultaneously, such as minimizing task execution latency, maximizing resource utilization, or reducing energy consumption. The fitness value assigned to each individual provides a basis for selecting the best candidates to serve as parents for the next generation.

Algorithm 2: Deep Reinforcement Learning for Task Scheduling in Fog Computing

Input: Task set $T = \{t_1, t_2, \ldots, t_n\}$, fog environment state S, reward R, discount factor γ, neural network parameters θ
Output: Optimal task scheduling policy π^*

Initialize replay memory D, neural network $Q(S, A; \theta)$, target network $Q'(S, A; \theta^-)$
Initialize learning rate α, exploration rate ϵ
while *not converged* **do**
  Observe the current state S
  if *Random number* $< \epsilon$ **then**
    Select a random action (task allocation) A
  else
    Select action $A = \arg\max_a Q(S, a; \theta)$
  end
  Execute action A, observe reward R and next state S'
  Store transition (S, A, R, S') in replay memory D
  if *Replay memory D is large enough* **then**
    Sample a minibatch of transitions (S_j, A_j, R_j, S'_j) from D
    Compute the target value for each transition:

$$y_j = \begin{cases} R_j & \text{if the episode ends at step } j+1 \\ R_j + \gamma \max_{a'} Q'(S'_j, a'; \theta^-) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on the loss:

$$L(\theta) = \frac{1}{|D|} \sum_j \left(y_j - Q(S_j, A_j; \theta)\right)^2$$

    Update the target network $Q'(S, A; \theta^-) \leftarrow Q(S, A; \theta)$
  end
  Set $S \leftarrow S'$
end
Return optimal policy $\pi^*(S) = \arg\max_a Q(S, a; \theta)$

Selection is the process by which individuals with higher fitness values are more likely to be chosen to contribute their genetic material (i.e., their task scheduling strategies) to the next generation. Several selection methods can be employed, such as roulette wheel selection, where individuals are chosen probabilistically based on their fitness, or tournament selection, where a subset of individuals competes, and the best among them is selected. The idea behind selection is to preserve and propagate good solutions while discarding poor ones, thus guiding the search towards optimal task scheduling strategies.

Once the parents are selected, genetic operations such as crossover and mutation are applied to create offspring for the next generation. Crossover is the process of combining the genetic material (i.e., the task scheduling decisions) of two parent individuals to produce new offspring. This operation is analogous to biological reproduction, where the offspring inherit traits from both parents. In the context of fog computing, crossover might involve exchanging parts of the task allocation strategies between parents, thereby creating new schedules that combine different aspects of both parents' solutions. The crossover probability p_c controls how often crossover is performed. If a random number is less than p_c, crossover is applied; otherwise, the offspring are copies of the parents.

Mutation is another crucial genetic operation that introduces diversity into the population by making random changes to an individual's solution. In task scheduling, mutation might involve changing the assignment of a task to a different fog node or altering the scheduling order of tasks. The mutation probability p_m determines how frequently mutation occurs, and its role is to prevent the population from converging prematurely to suboptimal solutions by exploring new parts of the solution space. Mutation ensures that the algorithm does not get stuck in local optima, thereby enhancing the overall search process's robustness.

After crossover and mutation, the offspring population is evaluated using the same fitness function as before. The fitness of each new individual is computed, and the best solutions are selected to form the next generation. This process of selection, crossover, mutation, and evaluation is repeated over multiple generations, gradually evolving the population towards better task scheduling solutions. The number of generations G determines the termination condition for the algorithm, though it can also be based on convergence criteria, such as when the improvement in the best fitness value becomes negligible over successive generations.

The power of genetic algorithms lies in their ability to explore a broad solution space while retaining the most promising solutions through selection. This makes GAs effective in dynamic environments like fog computing, where workloads and resource availability fluctuate over time. In such environments, the optimal task scheduling solution can change as new tasks arrive, resources are consumed, or network conditions vary. GAs are well-suited to adapt to these changes because they continually evolve the population of solutions, allowing the system to respond to new conditions without the need for manual intervention or reconfiguration. This adaptability is especially beneficial in fog computing, where the decentralized and heterogeneous nature of the infrastructure makes it difficult to predict future states of the system accurately.

Moreover, GAs can optimize multiple objectives simultaneously, making them ideal for the multi-objective nature of task scheduling in fog computing. For instance, minimizing task completion time may conflict with minimizing energy consumption, as faster task execution often requires more intensive use of resources. By using a carefully designed fitness function that balances these objectives, GAs can find solutions that provide an acceptable trade-off between conflicting goals. This is done by assigning different weights to each objective in the fitness function or by using multi-

objective optimization techniques such as Pareto-based approaches, where a set of non-dominated solutions is evolved over time.

Fog computing environments often involve a large number of tasks and nodes, making the task scheduling problem combinatorially complex. The number of possible task-to-node assignments grows exponentially with the number of tasks and nodes, making exhaustive search methods impractical. GAs, however, use population-based search techniques that allow them to explore large solution spaces without needing to evaluate every possible solution. By focusing on the most promising regions of the solution space through selection, GAs can find high-quality solutions in a reasonable amount of time, even for large-scale fog computing systems.

Since the fitness of each individual in the population can be evaluated independently, GAs can easily be parallelized to take advantage of modern multi-core processors or distributed computing environments. This parallelism is beneficial in fog computing, where the distributed nature of resources can be leveraged to evaluate multiple task scheduling solutions concurrently, further speeding up the optimization process (Parsa & Entezari-Maleki, 2009).

Algorithm 3: Genetic Algorithm for Task Scheduling in Fog Computing

Input: Task set $T = \{t_1, t_2, \ldots, t_n\}$, population size P, crossover probability p_c, mutation probability p_m, maximum generations G
Output: Optimal or near-optimal task scheduling solution

Initialize population P_0 with random task scheduling solutions
for $g = 0$ ***to*** G **do**
  Evaluate the fitness of each individual in population P_g
  Select parents from P_g based on fitness
  Generate offspring population by applying crossover and mutation:
  for *each pair of parents* **do**
    if *Random number* $< p_c$ **then**
      Perform crossover to create offspring
    else
      Copy parents without modification
    end
    for *each offspring* **do**
      if *Random number* $< p_m$ **then**
        Perform mutation on the offspring
      end
    end
  end
  Evaluate the fitness of the offspring
  Select the next generation population P_{g+1} by combining parents and offspring
end
Return the best solution from the final population P_G

4) Fuzzy Logic-Based Scheduling

Fuzzy logic-based scheduling offers a unique and adaptable approach for handling task allocation in fog computing environments, when dealing with uncertainty or imprecise information. Unlike traditional scheduling methods, which require precise numerical inputs and outputs, fuzzy logic systems operate using degrees of truth rather than binary logic. This makes them highly suitable for environments like fog computing, where system states such as node load, energy levels, or task priorities are not always clearly defined or may fluctuate over time. By representing these factors as fuzzy variables, fuzzy logic enables more flexible and context-aware decision-making, providing a robust solution for dynamic task scheduling in fog environments (Zhang & Zhou, 2017).

In fog computing, task scheduling involves distributing computational tasks among available fog nodes, each with varying capabilities and resource availability. The challenge lies in the fact that the state of the system is constantly changing, with incoming tasks of different sizes, resource constraints, and performance requirements. Traditional scheduling algorithms may struggle to accommodate such variability, especially when faced with incomplete or uncertain information about the current system state. Fuzzy logic, however, excels in these situations by allowing for a graded response to changing conditions rather than relying on rigid thresholds or exact measurements.

The core concept behind fuzzy logic is the use of fuzzy sets and membership functions to represent system state variables. For example, a fog node's load might be described using fuzzy terms such as "low load," "medium load," or "high load." Each of these terms is associated with a membership function that defines the degree to which the current load level belongs to each category. This contrasts with classical logic, where the load would be either low or not low, with no intermediate values. In fuzzy logic, the current load might partially belong to both the "medium load" and "high load" categories, with membership degrees representing the extent to which the load satisfies each condition.

Fuzzy logic-based scheduling begins by initializing fuzzy sets for the system state variables that are relevant to task scheduling decisions (Singh et al., 2017). These variables might include factors such as node load, energy consumption, network latency, and task priority. For each of these variables, a corresponding fuzzy membership function is defined to describe the different possible states. For instance, a fuzzy set for energy levels might include terms like "low energy," "medium energy," and "high energy," each with a membership function that assigns a degree of membership based on the current energy level of a fog node.

Once the fuzzy sets and membership functions are defined, the scheduling process starts by evaluating the current state of the fog environment for each task that needs to be scheduled. The system gathers information about the current

load on each fog node, their available energy levels, and the characteristics of the incoming tasks, such as their priority or computational requirements. For each fuzzy variable, the degree of membership is computed using the membership functions. For example, if a fog node is currently operating at 70% capacity, it might have a membership degree of 0.7 in the "high load" fuzzy set and a membership degree of 0.3 in the "medium load" fuzzy set. Similarly, if the node's energy level is at 40%, it might belong to the "medium energy" set with a membership degree of 0.6 and to the "low energy" set with a degree of 0.4.

After evaluating the degrees of membership for each fuzzy variable, the system applies a set of fuzzy inference rules to determine the appropriate scheduling decision. These rules are typically structured in an "if-then" format, where the antecedent defines the conditions that must be met, and the consequent specifies the resulting action. For instance, a fuzzy rule might state, "if the load is high and the energy is low, then delay the task." Another rule might state, "if the load is medium and the task priority is high, then assign the task to the node." The fuzzy inference process involves checking the antecedent conditions for each rule and combining the degrees of membership of the relevant fuzzy variables to determine the degree to which the rule is satisfied. This process is known as fuzzy reasoning, and it allows for a more nuanced decision-making process that can handle multiple, potentially conflicting factors simultaneously.

Once the fuzzy inference step is completed, the next phase is defuzzification, where the fuzzy output is converted into a crisp, actionable decision. Various defuzzification methods can be employed, with the centroid method being one of the most commonly used. In this method, the defuzzified output is calculated as the center of gravity of the fuzzy set produced by the inference step. This results in a single, crisp value that represents the final scheduling decision. For example, the defuzzified output might indicate the specific fog node to which the task should be allocated or suggest delaying the task for later execution if the system is currently overloaded (Radulescu & Van Gemund, 2000).

Unlike deterministic algorithms, which rely on precise thresholds and rigid decision boundaries, fuzzy logic-based schedulers can adapt to changing conditions by reasoning with imprecise information. This enables more context-aware scheduling decisions that account for the current state of the system in a more holistic manner. For example, a traditional scheduler might fail to allocate a task if a node's load is slightly above a predefined threshold, even if the node has sufficient resources to handle the task. In contrast, a fuzzy logic-based scheduler can consider the node's load as "moderately high" and assign the task if other conditions, such as energy levels and task priority, are favorable.

Moreover, fuzzy logic-based scheduling is inherently suitable for real-time decision-making, which is crucial in fog computing environments where tasks must be scheduled promptly to meet latency requirements. The fuzzy inference process can be executed quickly, allowing the system to evaluate the current state and make scheduling decisions in real time. This is important in fog computing, where the delay caused by scheduling decisions can directly impact the performance of time-sensitive applications.

Fog computing systems often need to balance competing objectives, such as minimizing task execution time, conserving energy, and avoiding overloading specific nodes. Fuzzy logic allows these objectives to be incorporated into the scheduling decision-making process without requiring complex mathematical formulations. By defining fuzzy sets and rules that account for each objective, the scheduler can make trade-offs between them in a way that reflects the current state of the system. For example, if a node's energy level is low but its load is moderate, the scheduler might prioritize energy conservation by delaying non-urgent tasks, while still allowing high-priority tasks to be executed.

Algorithm 4: Fuzzy Logic-Based Scheduling for Fog Computing

Input: Task set $T = \{t_1, t_2, \ldots, t_n\}$, fuzzy variables for system state (e.g., load, energy, task priority), fuzzy rule base
Output: Task scheduling decision

Initialize fuzzy sets for system state variables (e.g., "low load", "medium load", "high load") and corresponding membership functions
while *tasks remain in the system* **do**
  for *each task* t_i *in* T **do**
    Evaluate the current fog environment state (e.g., node load, energy level)
    for *each fuzzy variable (e.g., load, energy, priority)* **do**
      Compute the degree of membership for each fuzzy set (e.g., "low load", "high energy") using the membership functions
    end
    Apply fuzzy inference using the fuzzy rule base:
    for *each rule in the fuzzy rule base* **do**
      If the antecedent conditions are satisfied (e.g., "if load is high and energy is low") **then** infer a scheduling decision (e.g., delay task, allocate to a specific node)
    end
    Defuzzify the output using a defuzzification method (e.g., centroid method) to get a crisp scheduling decision
    Assign task t_i to the appropriate fog node or delay based on the defuzzified result
  end
end
Return final task schedule

Advantages and Limitations of AI-Based Scheduling in Fog Computing AI-based scheduling algorithms excel in

handling the complexity and uncertainty of dynamic fog environments. Their ability to learn from past experiences and adapt to future conditions makes them highly effective in managing stochastic task arrivals and fluctuating resource availability. These algorithms are also capable of multi-objective optimization, balancing multiple system metrics such as latency, energy consumption, and resource utilization.

However, AI-based algorithms come with significant computational overhead, during the training phase. Deep learning models, for instance, require substantial computational power and time to train, which may not be feasible on resource-constrained fog nodes. Additionally, the real-time decision-making requirements of fog computing systems can be challenging for AI-based algorithms that involve complex inference processes. The trade-off between adaptability and computational complexity must be carefully managed, especially in scenarios where fast, real-time scheduling is critical.

III. PERFORMANCE METRICS FOR COMPARATIVE EVALUATION

In the evaluation of scheduling algorithms, in the context of fog computing, performance metrics are critical for assessing the relative strengths of heuristic and AI-based approaches. This section provides a detailed discussion of key metrics used for comparative analysis: latency, energy consumption, resource utilization, and scalability.

Latency, or task completion time, is a fundamental metric in task scheduling systems, in environments with real-time or near-real-time constraints (Krause et al., 1975). Latency refers to the total time elapsed from the moment a task enters the system to the moment it is fully processed and its results are available. In fog computing, where resources are distributed and often heterogeneous, latency can vary significantly depending on how effectively tasks are scheduled and allocated across available nodes. Traditional heuristic algorithms typically follow predefined rules, which can be efficient in static or predictable environments. However, their inability to dynamically adjust to changing system conditions often results in suboptimal latency performance under fluctuating workloads or node failures. AI-based algorithms, especially those using reinforcement learning (RL) and deep reinforcement learning (DRL), are generally more effective in reducing latency in dynamic environments. By continuously learning from the system's state and adapting scheduling decisions based on real-time conditions, these algorithms can significantly reduce task waiting times and ensure quicker completion of tasks.

The latency L of a task can be expressed as:

$$L = W_q + W_s + E$$

where W_q is the queuing time (i.e., the time the task waits before being processed), W_s is the service time (i.e., the time taken by the node to execute the task), and E is any additional overhead incurred during task transmission between nodes. AI-based algorithms, by learning to minimize both W_q and W_s, can adapt to varying network conditions and load distributions, often outperforming heuristics in complex scenarios. Moreover, RL-based methods can further optimize latency by predicting the future state of the system and making proactive scheduling decisions, rather than merely reacting to current conditions.

Another crucial metric in fog computing is energy consumption. Since fog nodes often operate on limited power supplies, in edge environments, minimizing energy consumption is essential to prolong node lifetimes and ensure the sustainability of the system. Heuristic algorithms, due to their simple, rule-based structures, generally impose lower computational and energy overheads. This is because they typically do not involve complex decision-making processes or frequent state evaluations, allowing them to execute more efficiently in terms of energy usage. However, this simplicity often comes at the cost of suboptimal resource allocation, which can lead to higher cumulative energy consumption over time.

AI-based algorithms, on the other hand, offer the potential for more sophisticated energy optimization strategies. By learning to schedule tasks on nodes that exhibit higher energy efficiency or lower energy costs, these algorithms can significantly reduce the overall energy footprint of the system. For example, in an RL-based approach, the reward function can be designed to incentivize energy-efficient scheduling, where the agent learns to prioritize nodes with lower power consumption per task unit. The energy consumption E_c of a task can be modeled as:

$$E_c = P_t \cdot T$$

where P_t is the power consumed by the node during task execution and T is the execution time of the task. AI-based approaches can minimize both P_t and T by dynamically allocating tasks to nodes with better energy profiles and by optimizing the task execution order to minimize idle times.

Resource utilization is another critical metric, in fog computing environments where resources such as CPU, memory, and bandwidth are often limited. High resource utilization is generally desirable as it indicates that the system is effectively using its available resources to process tasks. Low resource utilization, on the other hand, suggests that resources are underutilized, leading to inefficiencies and potential bottlenecks. Heuristic algorithms typically have fixed strategies for task allocation that do not account for the real-time state of system resources. As a result, these algorithms may fail to fully utilize the available resources, especially under highly variable workloads.

In contrast, AI-based algorithms, those using dynamic optimization techniques such as RL or genetic algorithms (GAs), can adapt task allocation strategies in response to changing system conditions, leading to higher resource utilization. By continuously monitoring the state of CPU, memory, and bandwidth, these algorithms can distribute tasks

more effectively across nodes, ensuring that no single resource becomes a bottleneck. The resource utilization U for a particular resource (e.g., CPU) can be defined as the ratio of the used resource capacity to the total available capacity:

$$U = \frac{C_{\text{used}}}{C_{\text{total}}}$$

where C_{used} is the amount of resource capacity currently being used, and C_{total} is the total available capacity of the resource. AI-based approaches can increase U by dynamically allocating tasks to underutilized nodes, thus balancing the load more effectively across the system.

Scalability is a key concern in fog computing, as the number of devices and tasks in the network grows. An ideal scheduling algorithm should be able to handle increasing numbers of tasks and nodes without significant degradation in performance. Heuristic algorithms, due to their simplicity, are often easier to scale in terms of computational overhead. However, their performance may degrade as the system becomes more complex because they lack the flexibility to adapt to larger, more heterogeneous environments.

AI-based approaches, those employing GAs and RL, often exhibit better scalability in terms of their ability to handle increasing system complexity. GAs, for example, can evolve scheduling strategies over time, allowing them to explore a broader solution space as the network grows. RL-based methods can similarly scale by learning more complex policies that take into account the larger state space associated with more tasks and nodes. However, the computational overhead of AI-based algorithms tends to increase with the size of the network. For instance, in an RL-based approach, the state space and action space both grow exponentially with the number of nodes and tasks, which can lead to increased computation times and memory requirements.

The scalability S of a scheduling algorithm can be evaluated in terms of its ability to maintain performance as the size of the network N increases. For instance, a scalable algorithm would ideally have a performance degradation rate dS/dN close to zero. However, in practical scenarios, AI-based approaches often exhibit a non-zero degradation rate due to their increased computational complexity. To mitigate this, various optimization techniques, such as hierarchical RL or distributed learning approaches, can be employed to reduce the computational burden as the network scales.

IV. CONCLUSION

Fog computing represents a novel paradigm that expands traditional cloud computing by relocating computational tasks to network nodes positioned closer to the data source, such as edge devices, routers, and gateways. This decentralized approach is critical for applications that require low latency and high bandwidth, including autonomous vehicles, real-time analytics for the Internet of Things (IoT), industrial automation, and smart city infrastructures. In fog computing, the nodes involved typically operate in heterogeneous environments, characterized by significant disparities in computational power, memory, network bandwidth, and energy efficiency. Additionally, fog networks are inherently dynamic, with varying network conditions and fluctuating workloads due to the stochastic nature of task arrivals, which occur at unpredictable times and with diverse resource demands.

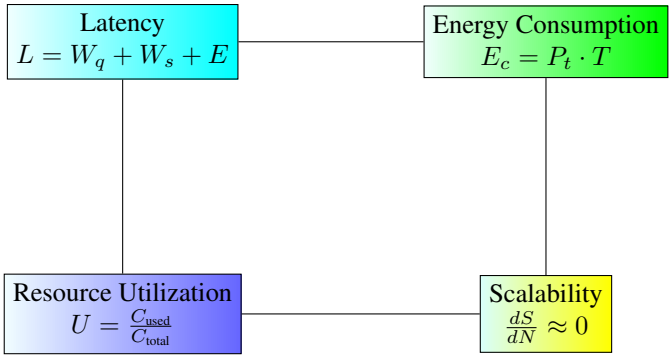


FIGURE 5. Performance Metrics for Comparative Evaluation in Scheduling Algorithms

Scheduling in fog computing refers to the process of assigning computational tasks to fog nodes in a manner that maximizes overall system performance. Key performance metrics include minimizing task completion time, reducing energy consumption, and ensuring efficient use of available resources. Compared to traditional cloud computing, scheduling in fog environments is more complex due to the heterogeneity of resources and the unpredictable, stochastic nature of task arrivals. The dynamic conditions of fog networks demand real-time, adaptive scheduling strategies capable of handling these variabilities while optimizing performance. Effective scheduling must address several objectives, such as minimizing latency by reducing the time from task arrival to completion, optimizing energy consumption—for energy-limited edge devices—and maximizing resource utilization to avoid overloading or underutilizing network nodes. Additionally, scheduling strategies must scale efficiently as the number of tasks and fog nodes increases without significant degradation in performance. This paper will focus on two primary categories of scheduling algorithms: heuristic-based approaches, which rely on rule-based decision-making, and AI-based approaches, which leverage learning techniques to optimize scheduling decisions in real time.

Fog computing infrastructures are fundamentally different from centralized cloud systems, in how they are structured and operate. One of the most significant differences is their decentralized nature, which requires scheduling algorithms to account for the proximity of nodes to the data source. Several characteristics of fog computing directly impact scheduling decisions. First, fog nodes are highly heterogeneous, meaning they vary greatly in terms of processing power, memory capacity, storage, and communication capabilities. Scheduling tasks without considering these differences can result in performance bottlenecks, such as assigning a high-complexity task to a low-capacity node, leading to delays

and reduced system efficiency. Additionally, the dynamic nature of fog environments further complicates scheduling. Nodes may unpredictably join or leave the network due to factors like mobility, hardware failures, or energy depletion. Resource availability also fluctuates as the network experiences varying loads, requiring scheduling algorithms to be adaptive and responsive to these changes in real time. Another challenge is the stochastic nature of task arrivals, where tasks arrive unpredictably, often without prior knowledge of their size, resource requirements, or deadlines. Scheduling decisions, therefore, need to be made in real time and must balance the resources available with future task demands.

Fog computing is also designed to support latency-sensitive applications, such as real-time video analytics, industrial IoT, and autonomous systems. These applications demand ultra-low latency, placing additional pressure on scheduling algorithms to prioritize tasks requiring minimal delay while managing resource contention and energy consumption effectively. Energy efficiency is another critical factor in fog scheduling, for edge devices with limited energy resources. Scheduling algorithms must balance the need for low latency with energy efficiency to ensure that the system remains sustainable over time. As a result, the main objectives for scheduling in fog computing revolve around minimizing task latency, optimizing energy consumption, maximizing resource utilization, and ensuring scalability in increasingly complex environments.

Heuristic-based scheduling algorithms offer a straightforward approach to task allocation, focusing on computational efficiency and ease of implementation. These algorithms are useful in fog environments where resources are constrained, and fast scheduling decisions are necessary. However, their simplicity comes at the cost of adaptability, as heuristic methods are generally less effective in handling the dynamic and stochastic conditions characteristic of fog networks. Common heuristic approaches include First-Come-First-Served (FCFS), Round Robin (RR), and Greedy algorithms. FCFS schedules tasks in the order they arrive, which is computationally inexpensive but often leads to poor performance in dynamic environments where task priorities and resource availability vary. Similarly, RR distributes tasks evenly across nodes but fails to consider the capabilities of each node or the specific resource requirements of the tasks, potentially overloading weaker nodes while underutilizing stronger ones. Greedy algorithms allocate tasks to the best available resource at the time, such as the least-loaded node, but this myopic approach often results in suboptimal long-term performance as future task arrivals and resource demands are not considered.

Other heuristic algorithms, such as Min-Min and Max-Min, attempt to balance task allocation more effectively. Min-Min schedules the smallest tasks first, assigning them to the nodes that can complete them most quickly, thereby reducing waiting times for small tasks but potentially delaying larger ones. Max-Min, on the other hand, prioritizes larger tasks, assigning them to nodes that can handle them without significantly delaying smaller tasks. This approach is useful in heterogeneous environments where large tasks might otherwise monopolize node resources. Despite their computational efficiency, heuristic algorithms are generally limited in their ability to handle the unpredictable and dynamic conditions of fog computing environments. They do not account for future task arrivals or resource availability fluctuations, which can lead to inefficiencies, especially in highly dynamic networks.

In contrast, AI-based scheduling algorithms offer a more sophisticated approach, in complex, dynamic, and stochastic environments like fog computing. These algorithms leverage machine learning (ML) techniques to predict future resource demands and optimize scheduling decisions based on real-time feedback. Reinforcement learning (RL) is one of the most common approaches used in AI-based scheduling, where the scheduling problem is modeled as a Markov Decision Process (MDP). Here, the system state, such as node load, task queue length, and resource availability, is continuously updated based on actions, like task allocations. The RL agent learns an optimal policy by interacting with the environment and receiving rewards, such as lower latency or reduced energy consumption. Over time, the agent improves its decision-making process to maximize cumulative rewards.

Q-learning and Deep Q-Networks (DQN) are popular RL algorithms applied to task scheduling. Q-learning updates a value function that estimates the expected reward for each action in a given state, while DQN extends this by using neural networks to approximate value functions in more complex, high-dimensional state spaces. Deep Reinforcement Learning (DRL) further advances RL by combining it with deep neural networks, enabling the scheduling algorithm to handle large-scale, complex environments with multiple nodes, tasks, and resource constraints. DRL-based schedulers are effective in fog environments where traditional heuristic methods fall short. By learning from past experiences and predicting future task arrivals, DRL-based schedulers can optimize task allocations in real time, reducing latency, energy consumption, and resource contention. However, the computational complexity of these algorithms, during the training phase, can be a limiting factor in resource-constrained fog environments.

Other AI-based scheduling techniques include genetic algorithms (GAs) and fuzzy logic-based systems. GAs are evolutionary algorithms that search for optimal scheduling solutions through selection, crossover, and mutation processes. These algorithms are useful in nonlinear and complex scheduling problems, where they can explore a wide range of potential solutions and adapt to changing system conditions over time. Fuzzy logic-based systems, on the other hand, handle uncertainty and imprecision in decision-making by reasoning with fuzzy sets and rules. In fog computing, fuzzy logic-based schedulers evaluate system states, such as node load or task priority, using fuzzy linguistic variables like "high load" or "low energy." This allows for more flexible and adaptive scheduling decisions in environments where precise

information about resource availability or task requirements is not readily available.

AI-based scheduling algorithms generally outperform heuristic methods in dynamic, stochastic, and heterogeneous environments due to their ability to learn from system behavior and adapt to changing conditions. By predicting future task arrivals and resource demands, AI-based algorithms can make real-time scheduling decisions that improve performance metrics such as latency, energy efficiency, and resource utilization. However, these benefits come at the cost of increased computational overhead. For instance, DRL models require extensive training data and significant computational resources, which may not be feasible for fog nodes with limited processing power. As a result, the trade-off between adaptability and computational complexity must be carefully considered, especially for time-critical applications.

Comparing heuristic and AI-based scheduling algorithms reveals several key performance differences. In terms of latency, AI-based algorithms, those using RL and DRL, generally outperform heuristic methods by dynamically optimizing task allocations and reducing queuing times. AI-based approaches also tend to be more energy-efficient in the long run, as they can learn to allocate tasks to nodes with higher energy efficiency, although heuristic methods consume less energy upfront due to their lower computational demands. Regarding resource utilization, AI-based algorithms offer more efficient use of available resources, dynamically adjusting task allocations based on real-time feedback. Finally, AI-based approaches, DRL and GA, scale more effectively in larger, more complex systems, although their computational overhead increases with system size, which may limit their applicability in certain resource-constrained environments. While the research presented provides insights into the comparative performance of heuristic and AI-based scheduling algorithms in fog computing, there are several limitations that should be acknowledged. These limitations stem from the inherent complexity of fog environments, the constraints of current methodologies, and the challenges of implementing these algorithms in practical, real-world settings. Addressing these limitations is crucial for advancing the understanding of task scheduling in fog computing and refining the approaches evaluated in this study (Guo et al., 2012).

One of the primary limitations of this research is the focus on idealized or simplified models of fog computing environments, which may not fully capture the complexities and nuances of real-world scenarios. While the comparative study of heuristic and AI-based algorithms offers theoretical insights, fog networks in practical implementations are far more heterogeneous and unpredictable than what is modeled in many of the simulations and evaluations conducted. Real-world fog environments often involve a broader range of devices with vastly different capabilities, including low-power IoT sensors, mobile devices, and more powerful edge servers, each with varying levels of computational power, memory, bandwidth, and energy reserves. The research assumes that the fog nodes are relatively homogeneous within certain constraints, but this assumption may lead to an oversimplification of the scheduling problem. In reality, the disparity between the weakest and most powerful nodes in a fog network can be substantial, and this variability can significantly impact the performance of both heuristic and AI-based algorithms in ways that are difficult to predict from simplified models.

Additionally, real-world fog environments are subject to far more complex and unpredictable task arrival patterns than those typically modeled in research simulations. In many cases, task arrivals in fog networks are influenced by external factors such as user behavior, sensor activity, and network traffic, which introduce additional layers of stochasticity that are not fully captured by the probabilistic models used in most studies. As a result, the conclusions drawn from these models may not translate seamlessly to real-world applications, where the timing, size, and computational requirements of tasks can vary dramatically. This unpredictability poses a significant challenge for scheduling algorithms, those based on heuristics, which rely on fixed rules and may not be flexible enough to adapt to such dynamic conditions. Even AI-based algorithms, despite their capacity to learn from historical data and predict future states, may struggle to handle the full range of variability encountered in operational fog environments. Therefore, the results of this study, while in controlled scenarios, may not fully generalize to the complexities of real-world fog computing.

A second significant limitation of this research lies in the computational overhead associated with AI-based scheduling algorithms, which is problematic in resource-constrained fog environments. While AI techniques such as reinforcement learning (RL), deep reinforcement learning (DRL), and genetic algorithms (GA) offer superior performance in dynamic and heterogeneous conditions, their implementation comes at a substantial cost in terms of computational power, memory usage, and training time. These algorithms often require extensive training data and significant computational resources to learn optimal scheduling policies, which can be prohibitive in environments where the fog nodes themselves are resource-constrained. In many fog networks, edge devices have limited processing power and battery life, making it challenging to implement resource-intensive AI algorithms without negatively impacting the overall performance of the network.

The trade-off between the adaptability of AI-based methods and the computational demands they place on the system is a crucial issue that this study does not fully address. For instance, while DRL-based schedulers may outperform heuristic approaches in terms of latency reduction and resource utilization, the overhead involved in training and deploying these models can outweigh the benefits in smaller or less resource-rich fog networks. This issue is relevant for applications that require real-time or near-real-time task scheduling, where the latency introduced by complex AI algorithms could negate their advantages in terms of optimized resource allocation. Furthermore, the energy consumption

associated with running AI-based schedulers on resource-limited devices can also be a significant drawback. The study touches on this issue but does not explore it in sufficient depth, leaving open the question of how to balance the computational complexity of AI-based algorithms with the practical constraints of fog environments.

Another limitation concerns the scalability of the algorithms examined in this study, when applied to large-scale fog networks with a high number of nodes and tasks. While the paper highlights scalability as a key performance metric, it does not fully address the challenges associated with scaling AI-based methods, especially in highly distributed and resource-constrained fog environments. AI algorithms, those involving deep learning, tend to become more computationally expensive as the number of nodes and tasks increases (Sugomori et al., 2017). As the fog network grows, the complexity of the scheduling problem increases exponentially, and the computational burden of maintaining accurate models and making real-time scheduling decisions can become prohibitive. Although heuristic algorithms may scale more efficiently due to their simplicity, their lack of adaptability makes them less effective in large, heterogeneous networks where node capabilities and task requirements vary significantly.

Moreover, the research does not adequately account for the impact of network latency and communication overhead on the performance of scheduling algorithms. In real-world fog networks, task scheduling often requires communication between distributed nodes, and the delays introduced by this communication can significantly affect the overall system performance. AI-based algorithms, those that rely on centralized training or decision-making processes, may suffer from increased latency as the network size grows, leading to delays in task allocation and execution. Heuristic methods, while less computationally intensive, may also face challenges in large networks due to the increased communication required to gather and distribute task and resource information. The study does not fully explore these issues, which could have a significant impact on the real-world applicability of the algorithms being evaluated.

Finally, the study does not sufficiently consider the security and privacy challenges associated with fog computing, which are becoming increasingly important as fog networks are deployed in sensitive applications such as healthcare, autonomous driving, and industrial control systems. Task scheduling in fog computing often involves the transmission of sensitive data between nodes, and the algorithms used must ensure that data privacy and security are maintained throughout the scheduling process. AI-based algorithms, those involving machine learning, may require access to large amounts of data for training, raising concerns about data privacy and the potential for data breaches. While this research focuses on performance metrics such as latency, energy consumption, and resource utilization, it does not address the security implications of the scheduling approaches being evaluated. As fog computing continues to expand into critical applications, ensuring the security and privacy of task scheduling will be essential, and this is an area that requires further investigation.

VECTORAL PUBLISHING POLICY

VECTORAL maintains a strict policy requiring authors to submit only novel, original work that has not been published previously or concurrently submitted for publication elsewhere. When submitting a manuscript, authors must provide a comprehensive disclosure of all prior publications and ongoing submissions. VECTORAL prohibits the publication of preliminary or incomplete results. It is the responsibility of the submitting author to secure the agreement of all co-authors and obtain any necessary permissions from employers or sponsors prior to article submission. The VECTORAL takes a firm stance against honorary or courtesy authorship and strongly encourages authors to reference only directly relevant previous work. Proper citation practices are a fundamental obligation of the authors. VECTORAL does not publish conference records or proceedings.

VECOTORAL PUBLICATION PRINCIPLES

Authors should consider the following points:

1) To be considered for publication, technical papers must contribute to the advancement of knowledge in their field and acknowledge relevant existing research.
2) The length of a submitted paper should be proportionate to the significance or complexity of the research. For instance, a straightforward extension of previously published work may not warrant publication or could be adequately presented in a concise format.
3) Authors must demonstrate the scientific and technical value of their work to both peer reviewers and editors. The burden of proof is higher when presenting extraordinary or unexpected findings.
4) To facilitate scientific progress through replication, papers submitted for publication must provide sufficient information to enable readers to conduct similar experiments or calculations and reproduce the reported results. While not every detail needs to be disclosed, a paper must contain new, usable, and thoroughly described information.
5) Papers that discuss ongoing research or announce the most recent technical achievements may be suitable for presentation at a professional conference but may not be appropriate for publication.

References

Arunarani, A., Manjula, D., & Sugumaran, V. (2019). Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, *91*, 407–415.

Atabakhsh, H. (1991). A survey of constraint based scheduling systems using an artificial intelligence approach. *Artificial Intelligence in Engineering*, *6*(2), 58–73.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 13–16.

Chen, S., Zhang, T., & Shi, W. (2017). Fog computing. *IEEE Internet Computing*, *21*(2), 4–6.

Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., & Buyya, R. (2016). Fog computing: Principles, architectures, and applications. In *Internet of things* (pp. 61–75). Elsevier.

El-Rewini, H., Lewis, T. G., & Ali, H. H. (1994). *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc.

Guo, L., Zhao, S., Shen, S., & Jiang, C. (2012). Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of networks*, *7*(3), 547.

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N. S., & Mahmoudi, C. (2018). Fog computing conceptual model.

Kraemer, F. A., Braten, A. E., Tamkittikhun, N., & Palma, D. (2017). Fog computing in healthcare–a review and discussion. *IEEE Access*, *5*, 9206–9222.

Krause, K. L., Shen, V. Y., & Schwetman, H. D. (1975). Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *Journal of the ACM (JACM)*, *22*(4), 522–550.

Mahmud, R., Kotagiri, R., & Buyya, R. (2018). Fog computing: A taxonomy, survey and future directions. *Internet of everything: algorithms, methodologies, technologies and perspectives*, 103–130.

Noronha, S., & Sarma, V. (1991). Knowledge-based approaches for scheduling problems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *3*(2), 160–171.

Omara, F. A., & Arafa, M. M. (2010). Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed computing*, *70*(1), 13–22.

Parsa, S., & Entezari-Maleki, R. (2009). Rasa: A new grid task scheduling algorithm. *International Journal of Digital Content Technology and its Applications*, *3*(4), 91–99.

Radulescu, A., & Van Gemund, A. J. (2000). Fast and effective task scheduling in heterogeneous systems. *Proceedings 9th heterogeneous computing workshop (HCW 2000)(Cat. No. PR00556)*, 229–238.

Singh, P., Dutta, M., & Aggarwal, N. (2017). A review of task scheduling based on meta-heuristics approach in cloud computing. *Knowledge and Information Systems*, *52*, 1–51.

Sinnen, O. (2007). *Task scheduling for parallel systems* (Vol. 60). John Wiley & Sons.

Sugomori, Y., Kaluza, B., Soares, F. M., & Souza, A. M. (2017). *Deep learning: Practical neural networks with java*. Packt Publishing Ltd.

Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, *13*(3), 260–274.

Yi, S., Li, C., & Li, Q. (2015). A survey of fog computing: Concepts, applications and issues. *Proceedings of the 2015 workshop on mobile big data*, 37–42.

Zhang, P., & Zhou, M. (2017). Dynamic cloud task scheduling based on a two-stage strategy. *IEEE Transactions on Automation Science and Engineering*, *15*(2), 772–783.