electronics

MDPI

Article

A Whale Optimization Algorithm Based Resource Allocation Scheme for Cloud-Fog Based IoT Applications

Ranumayee Sing [1], Sourav Kumar Bhoi [2], Niranjan Panigrahi [2], Kshira Sagar Sahoo [3,4], Nz Jhanjhi [5,*] and Mohammed A. AlZain [6]

1 Faculty of Engineering (Computer Science and Engineering), BPUT, Rourkela 769015, Odisha, India
2 Department of Computer Science and Engineering, Parala Maharaja Engineering College (Govt.), Berhampur 761003, Odisha, India
3 Department of Computer Science and Engineering, SRM University, Amaravati 522502, AP, India
4 Department of Computing Science, Umeå University, 901 87 Umeaå, Sweden
5 School of Computer Science, SCS Taylor's University, Subang Jaya 47500, Malaysia
6 Department of Information Technology, College of Computers and Information Technology, Taif University, Taif 21944, Saudi Arabia
* Correspondence: noorzaman.jhanjhi@taylors.edu.my

Abstract: Fog computing has been prioritized over cloud computing in terms of latency-sensitive Internet of Things (IoT) based services. We consider a limited resource-based fog system where real-time tasks with heterogeneous resource configurations are required to allocate within the execution deadline. Two modules are designed to handle the real-time continuous streaming tasks. The first module is task classification and buffering (TCB), which classifies the task heterogeneity using dynamic fuzzy c-means clustering and buffers into parallel virtual queues according to enhanced least laxity time. The second module is task offloading and optimal resource allocation (TOORA), which decides to offload the task either to cloud or fog and also optimally assigns the resources of fog nodes using the whale optimization algorithm, which provides high throughput. The simulation results of our proposed algorithm, called whale optimized resource allocation (WORA), is compared with results of other models, such as shortest job first (SJF), multi-objective monotone increasing sorting-based (MOMIS) algorithm, and Fuzzy Logic based Real-time Task Scheduling (FLRTS) algorithm. When 100 to 700 tasks are executed in 15 fog nodes, the results show that the WORA algorithm saves 10.3% of the average cost of MOMIS and 21.9% of the average cost of FLRTS. When comparing the energy consumption, WORA consumes 18.5% less than MOMIS and 30.8% less than FLRTS. The WORA also performed 6.4% better than MOMIS and 12.9% better than FLRTS in terms of makespan and 2.6% better than MOMIS and 4.3% better than FLRTS in terms of successful completion of tasks.

Keywords: resource allocation; cloud fog based IoT; whale optimization; WORA; TCB; TOORA

check for updates

Citation: Sing, R.; Bhoi, S.K.; Panigrahi, N.; Sahoo, K.S.; Jhanjhi, N.; AlZain, M.A. A Whale Optimization Algorithm Based Resource Allocation Scheme for Cloud-Fog Based IoT Applications. *Electronics* **2022**, *11*, 3207. https://doi.org/10.3390/electronics11193207

Academic Editor: Yu-Chen Hu

Received: 26 August 2022
Accepted: 26 September 2022
Published: 6 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Internet of Things (IoT) has expanded very quickly, providing many services in different domains, such as traffic management, vehicle networks, energy management, healthcare, smart homes, among others. [1–3]. Addressing diverse requirements means connecting end devices, such as sensors, smart mobile phones, actuators, advanced vehicles, advanced appliance, smart meters, etc. Although real-time tasks demand heterogeneous resource requirements for processing, the processing of tasks at end devices with limited resources run down the performance, which forces them to switch to other computing environments. Cloud computing with a large resource center can compute these tasks of end devices with on-demand resource requirements.

Cloud servers are usually located remotely from the end devices. With increasing of end devices the task offloading is also increased. This excessive data transfer most likely will create network congestion and degrade the performance of the network. Most of the

applications cannot afford the delay in processing the tasks in the cloud [4,5], as this is detrimental to the application sensitivity.

The above paradox is addressed with fog computing [6], which works in the middle tier between cloud and end devices. The fog computing being closer to end devices provides high-quality services by satisfying the requirements of delay-sensitive tasks and reducing the workload of the cloud server.

The devices (routers, gateways, embedded servers, controllers, etc.) that have the capability of computation, storage, and communication are treated as fog nodes. These nodes with limited resources and computation capability may not satisfy the requirement of heterogeneous resources for multiple tasks execution at a time [7,8]. The improper resource allocation may change the order of execution of tasks, which may lead to low throughput and failure in achieving deadlines of tasks.

The majority of the research work concerning IoT applications concentrates on exploration of fog computing or cloud computing environments individually. A relatively unexplored dimension in this research arena is a hybrid environment that can handle both delay-sensitive data and non-sensitive data with equal efficacy. This hybrid environment, termed as the cloud–fog model, is formed by combining both the cloud environment and the fog environment. There has not been a very significant number of studies carried out on the cloud–fog model. Therefore, the intricacies in handling the real-time heterogeneous tasks with different features such as deadline, data size, arrival time, and execution time, etc., are another challenge in the cloud–fog model. In this present work, the first task at hand is to process the heterogeneous tasks by multiple queues. As the fog node is limited in resources and resource allocation is a NP-hard problem [9,10], it motivates us to use meta-heuristic techniques for optimally allocating resources. The recent optimization technique named whale optimization algorithm (WOA) gives more optimal results under many complex situations. Therefore, the second motivation is to employ the whale optimization and explore the optimal solution for allocating resources. Energy consumption is another issue that leads to worldwide carbon emissions problem; thus, the third motivation is necessitating minimization of energy consumption in the cloud–fog model [11–13].

The primary objective is resource allocation for heterogeneous real-time tasks in the cloud–fog model within the deadline requirement of tasks, which can improve the makespan, task completion ratio, cost function, and energy consumption. In this paper, a three-tier cloud–fog model with parallel virtual queues architecture is considered.

The significant contributions of this work are as follows:

1. The task classification and buffering (TCB) module is designed for classifying tasks into different types using dynamic fuzzy c-means clustering, and these classified tasks are buffered in parallel virtual queues based on enhanced least laxity time scheduling.
2. Another module, named task offloading and optimal resource allocation (TOORA), is modeled for deciding on offloading the task in cloud or fog and uses WOA to allocate the resources of the fog node.
3. The approach is evaluating the metrics, such as makespan, cost, energy consumption, and the successful completed tasks within the deadline and comparing them with other algorithms such as SJF, MOMIS, and FLRTS for performance evaluation.
4. When 100 to 700 tasks are executed in 15 fog nodes, the results show that the WORA algorithm saves 10.3% of the average cost of MOMIS and 21.9% of the average cost of FLRTS. When comparing the energy consumption, WORA consumes 18.5% less than MOMIS and 30.8% less than FLRTS. The WORA is also performed 6.4% better than MOMIS and 12.9% better than FLRTS in terms of makespan and 2.6% better than MOMIS and 4.3% better than FLRTS in terms of successful completion of tasks.

The structure of the remaining sections are organized as follows. The survey on resource allocation in different environments is presented in Section 2. In Section 3 system model is described and problem is formulated. Section 4 describes the optimal resource allocation algorithm in fog nodes. Section 5 presents the performance evaluation of our proposed algorithm. Finally, the conclusion and future work are presented in Section 6.

2. Related Work

Currently, fog computing is the most popular research area in terms of service management. Many researchers are focused on the concept, architecture, and resource management issues of fog computing. The fog computing paradigm as a virtual platform was introduced by Bonomi et al. [14]. Refs. [15–17] highlighted the issues and challenges related to fog computing that need to be solved.

The cloud node contains massive storage and processors with high-speed network connectivity and various application services [18–20]. For assigning services to suitable service nodes with appropriate distribution of workload in every node, Chen et al. [21] proposed RQCSA and FSQSM, which improved the efficiency and minimized queue waiting time and makespan. Behzad et al. [22] proposed the queue based hybrid scheduling algorithm for storing jobs in the queue according to the order of priority. The job with lower quantum time is allocated with the CPU and executed. Venkataramanan et al. [23] studied the problem due to overflow of queue in wireless scheduling algorithm. In [24], the stability of the queue was achieved by applying a reinforcement learning approach to Lyapunov optimization for resource allocation of edge computing. Similarly, Eryilmaz and Srikant [25] stated that the length of the queue is bounded with the setting of the Lyapunov function drift. Hence, the Lyapunov function is important to control the virtual queue length. Some researchers have also used this queuing theory in fog computing. Iyapparaja et al. [26] designed a model based on queueing theory-based cuckoo search (QTCS) to improve QoS of resource allocation. Li et al. [4] considered heterogeneous tasks to be placed in parallel virtual queues. The task offloading is decided on the basis of the urgency of the task based on laxity time.

In the real world, continuous streaming data are generated that are required for online analysis. Most adaptive clustering is application-specific, so Sandhir and Kumar [27,28] proposed a modified fuzzy c-means clustering technique called dynamic fuzzy c-means (dFCM) clustering with the aid of a synthetic dataset. Most of the researchers in [1,4,29] considered laxity time for prioritizing the tasks: the lower laxity time task will be executed first. The laxity time is also estimated on account of the deadline, execution time, and current time, which also decided the task offloading. Ali et al. [30] proposed a fuzzy logic task scheduling algorithm for deciding tasks to be executed either in the fog node or cloud center. The tasks with constraints such as deadline and data size exploited the heterogeneous resources of fog nodes, which improved makespan, average turnaround time, delay rate, and successful task completion ratio. According to Pham et al. [10], resource allocation is a non-linear programming problem and an NP-hard problem. Such types of problem can be solved using three methods, including heuristic, meta heuristics, and hybrid. As there is no optimal performance guarantee of the heuristic method, one is forced to adopt a meta-heuristic method, such as a whale optimization algorithm, as a recent efficient optimization method. Here, WOA was used for solving allocation of power, secure throughput, and offloading in mobile edge computing. Hosseini et al. [31] used WOA for optimal resource allocation and minimized the total run-time of requested services in cloud center. Several optimization techniques in different platforms were studied [32–34].

Several studies have proposed solving resource allocation problems in different networks. Table 1 summarizes the methodology, policies, and limitations of the resource allocation problem. Some research work is discussed here. Li et al. [4] combined the method of fuzzy c-means clustering and particle swarm optimization to design a new resource scheduling algorithm that improved user satisfaction. Rafique et al. [9] proposed a novel bio-inspired hybrid algorithm (NBIHA) for task scheduling and resource allocation

of fog node, which reduced the average response time. Sun et al. [35] designed a resource scheduling model using improved non-dominated sorting genetic algorithm (NSGA-II) for the same fog clusters by which improved task execution and reduced service latency. In [36], Taneja and Devy handled the modules of fog-cloud model and mapped the modules to the mapping algorithm, that gave better performance to energy consumption, network usage, and end-to-end latency than that of traditional cloud infrastructure. In [11], Mao et al. designed separate energy-aware algorithm and time-aware algorithm for handling the task in a heterogeneous environment and developed a combined algorithms ETMCTSA that managed and controlled the performance of the cloud on the basis of parameter α of the algorithm. Bharti and Mavi [37] adopted ETMCTSA and discovered that underutilized resources of the cloud can increase the usage of resources. Anu and Singhrova [38] modeled P-GA-PSO algorithm that allocate resources efficiently in fog computing that reduced delay, waiting time, and energy consumption compared to round-robin and genetic algorithms. In a three-layer computing network, Jia et al. [39] presented an extension of the deferred acceptance algorithm called double-matching strategy (DA-DMS) that was a cost-efficient resource allocation in which a paired partner cannot change unilaterally for more cost-efficiency. In [40], an algorithm based on a Pareto-domination mechanism to particle swarm optimization algorithm searched for a multi-objective optimal solution. Ni et al. [41] modeled a dynamic algorithm based on PTPN, where the user can use appropriate resources from the available group of resources autonomously. Both price and cost are considered for the completion of the task. Many such resource allocation algorithms in different systems are found in [42–48].

Table 1. Related work on resource allocation in different systems.

Article	Ideas	Target System	Improved Criteria	Limitations
Li et al. [4]	Laxity time and Lyapunov optimization	Fog computing	Throughput and task completion ratio	No other parameters are considered
Bae et al. [24]	Reinforcement learning and Lyapunov optimization	Edge computing	Time-average penalty cost cost	Operates with general non-convex and discontinuous penalty functions
Iyapparaja et al. [26]	Queueing theory-based cuckoo search	Fog computing	Response time and energy consumption	Resource allocation to the edge node is challenging
Ali et al. [30]	Fuzzy logic	Cloud–fog environment	Makespan, average turnaround time, success ratio of the tasks, and delay rate	Large-scale network

Table 1. *Cont.*

Article	Ideas	Target System	Improved Criteria	Limitations
Pham et al. [10]	Whale optimization algorithm	Wireless network	System utility, overhead	Small dataset of user
Li et al. [4]	Fuzzy clustering with particle swarm optimization	Fog computing	User satisfaction	Small dataset of tasks
Rafique et al. [9]	Novel bio-inspired hybrid algorithm (NBIHA)	Fog computing	Average response time	Small dataset of tasks
Sun et al. [35]	Non-dominated sorting genetic algorithm (NSGA-II)	Fog computing	Reduced service latency and improved stability of task execution	Other parameters such as cost is not considered
Taneja and Devy [36]	Module mapping algorithm	Fog–cloud Infrastructure	Energy consumption, network usage, and end-to-end latency	Only compared with traditional cloud infrastructure
Mao et al. [11]	Energy-performance trade-off multi-resource cloud task scheduling algorithm (ETMCTSA)	Green cloud computing	Energy consumption, execution time, overhead	Small task dataset
Bharti and Mavi [37]	ETMCTSA for underutilized resources	Cloud computing	Energy consumption, overhead	Used 100 cloudlets
Anu and Singhrova [38]	Hybridization of priority, genetic algorithm, and PSO	Fog computing	Reduced energy consumption, waiting time, execution delay, and resource wastage	Considered end devices
Jia et al. [39]	Double-matching strategy based on deferred acceptance (DA-DMS)	Three-tier architecture (cloud data center, fog node, and users)	High-cost efficiency	Large-scale network
Feng et al. [40]	Particle swarm optimization with Pareto-dominant	Cloud computing	Large-scaled instances, middle-scaled instances, small-scaled instances	Did not use complex tasks and resources
Ni et al. [41]	Priced timed Petri nets strategy	Fog computing	Makespan, cost	Did not consider average completion time and fairness

Most of the research discussed different resource allocation methods in the fog environment, the cloud environment, and wireless networks. These studies also tried to improve metrics (i.e., response time, makespan, consumption of energy, overhead, etc.). This paper adopts WOA which can allocate resources optimally. The metrics such as cost, makespan, task completion ratio, and energy consumption are improved and compared with recent studies. The abbreviation table presented in Table 2 lists all abbreviations that are used in this paper.

Table 2. Abbreviations and description.

Abbreviation	Description
TCB	Task classification and buffering
TOORA	Task offloading and optimal resource allocation
WORA	Whale optimized resource allocation
SJF	Shortest job first
MOMIS	Multi-objective monotone increasing sorting-based
FLRTS	Fuzzy logic-based real-time task scheduling
FCM	Fuzzy c-means
dFCM	Dynamic fuzzy c-means
EDF	Earliest deadline first
WOA	Whale Optimization Algorithm
WOASU	Whale optimization algorithm spiral updating
WOAEP	Whale optimization algorithm encircling prey

3. System Model

Considering end devices, fog layer and cloud layer a three-tier cloud-fog model is designed as shown in Figure 1.

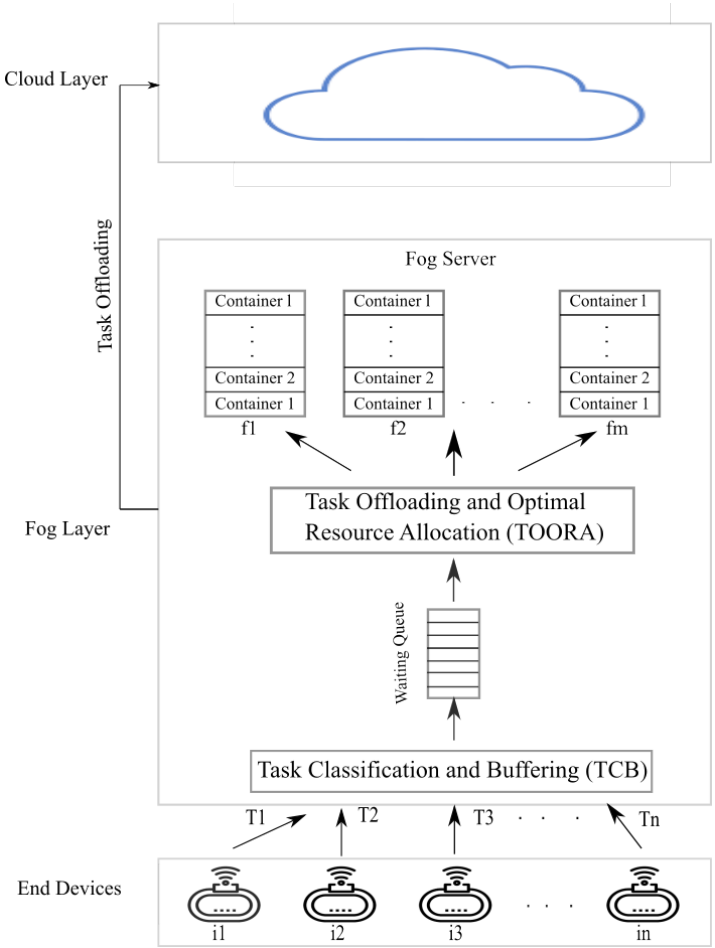

Figure 1. System architecture.

End devices: The end devices $\{i_1, i_2, i_3, \ldots.., i_n\}$ include sensors, actuators, mobile vehicles, smart cameras, etc. The end devices generate tasks $\{T_1, T_2, T_3, \ldots.., T_n\}$ with different resource requirements. These tasks are classified and buffered in fog node for further execution.

Fog layer: Fog nodes $\{f_1, f_2, f_3, \ldots.., f_m\}$ are the network devices (e.g., controller, router, gateways, embedded server). Every fog node consists of a set of containers $\{c_{i1}, c_{i2}, c_{i3}, \ldots.., c_{ik}\}$. The tasks require different resource requirements (e.g., CPU, bandwidth, memory, and storage configuration) to process the data. Therefore, each container contains a set of resources $\left\{r_{ij}^1, r_{ij}^2, r_{ij}^3, \ldots.., r_{ij}^l\right\}$ where $r_{ij}^l = \{CPU, bandwidth, memory\}$. Due to the limited resource of fog nodes, all tasks cannot process at fog nodes simultaneously, thus necessitating buffering of tasks in the queue.

Cloud layer: This layer has a cloud server that includes unlimited resources. The cloud is placed far from fog nodes, thus causing data transmission latency. Even if there is data transmission latency for transferring tasks to the cloud, it completes its processing without waiting for resources, because of unlimited resources.

In the fog layer, two modules are designed as follows:

- Task classification and buffering (TCB): On the arrival of tasks at the fog node, the similar type of tasks are gathered and buffered in parallel virtual queues according to their execution order.
- Task offloading and optimal resource allocation (TOORA): All the tasks may not be assigned with fog resources by their deadline. The tasks may wait long time in queue which may lead failure of execution. These tasks can be transferred to cloud layer and achieved the deadline. The transmission of tasks may increase the transmission cost, thus, TOORA try to assign maximum tasks with fog resources. Table 3 represents all notations of this paper.

Table 3. Notations and description.

Sl. No.	Notation	Description
1	i_n	Represents end devices
2	f_m	Represents fog nodes
3	c_{ik}	Containers of fog node
4	r_{ij}^l	Resources of a container
5	T_i	Individual task where $i \epsilon [1, n]$
6	$arrt_i$	Arrival time of ith task
7	$etlow_i$	Execution lower bound time of ith task
8	$etup_i$	Execution upper bound time of ith task
9	$dsize_i$	Data size of ith task
10	$respt_i$	Response time of ith task
11	dt_i	Deadline time of ith task
12	len_i	Number of instructions of ith task
13	μ_{ij}	Membership of ith task to jth cluster center
14	v_j	Cluster center
15	α	Error threshold
16	V_{XB}	Xie–Beni index
17	β	Membership threshold
18	lf_i	Laxity time of ith task
19	EDF_i	Earliest deadline first of ith task
20	Q_c	$type - c$ queue
21	lf^{max}	Maximum laxity time of head task of the queue
22	lf_i^j	Laxity time of ith task of jth queue

Table 3. *Cont.*

Sl. No.	Notation	Description
23	$\overrightarrow{W_b(t)}$	Best agent
24	$\overrightarrow{A}, \overrightarrow{C}$	Coefficient vectors
25	$\overrightarrow{r}$	Random vector value lies in $[0,1]$
26	$\overrightarrow{a}$	Parameter controller
27	b	Constant used for logarithmic spiral shape
28	l	Random value in $[-1,1]$
29	W_i	Represents whale
30	c_1	Processing cost per time unit for cloud
31	cc	Communication cost per time unit for cloud
32	cf	Communication cost per time unit for fog
33	ef	Energy per unit for execution of the task in fog
34	e_{idle}	Energy used when fog node is idle
35	ec	Energy per unit for execution of task cloud
36	e_{comm}	Energy per unit for transmission of data

3.1. Process Flow Model

The process flow model shows how the tasks are executed in the cloud–fog model by assigning limited resources of fog nodes. The following are presented and shown in Figure 2.

1. Step-1: The end devices collect data and send task requests to the nearest fog node.
2. Step-2: The task requests transfer from fog node to the TCB.
3. Step-3: The resource usage, data size, arrival time, deadline, etc., are estimated.
4. Step-4: Tasks are classified into different types in the TCB, which can be buffered in the waiting queue by running an algorithm for ordering the task.
5. Step-5: Tasks are transferred to the waiting queue for buffering.
6. Step-6: A set of tasks of the queues are transferred to the TOORA for further processing.
7. Step-7: TOORA makes a decision of task offloading so that task may execute in cloud server or fog node.
8. Step-8: The tasks meant for offloading to the cloud are transferred to the cloud server. The tasks are sent back to the end devices that are not achieved the deadline.
9. Step-9: An optimal resource allocation scheduler is run in the TOORA module to optimally assign resources of the fog node to the task.
10. Step-10: As the result of the algorithm, the tasks are assigned to the fog nodes.
11. Step-11: Each task is processed in the respective node.
12. Step-12: After completion of task execution, the result is sent back to the end devices through the fog node.

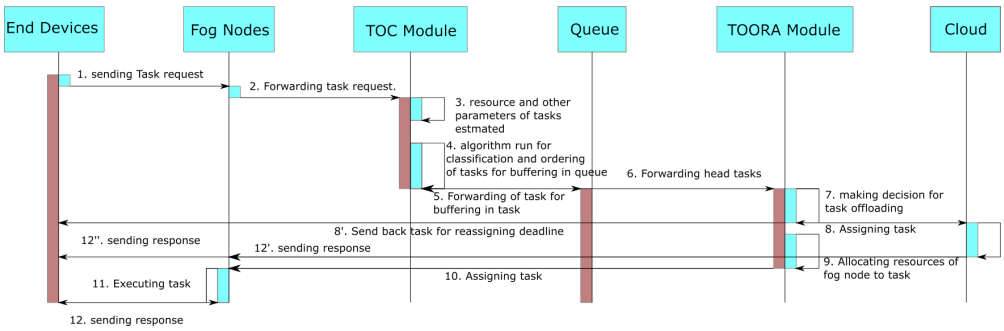

Figure 2. Process flow model of the architecture.

3.2. Problem Formulation

We are considering a set of fog nodes $F = \{f_1, f_2, f_3, \ldots.., f_m\}$, where every fog node consists of set of containers $f_i = \{c_{i1}, c_{i2}, c_{i3}, \ldots.., c_{ik}\}$, and each container contains set of resource blocks $c_{ij} = \left\{r_{ij}^1, r_{ij}^2, r_{ij}^3, \ldots.., r_{ij}^l\right\}$. The resource can be represented as the collection of $r_{ij}^l = \{CPU, bandwidth, memory\}$. The fog node has limited resource capacity. The total resource of a fog node is

$$R_f = \sum_{i=1}^{m} \sum_{j=1}^{k} \sum_{l=1}^{l} r_{ij}^l \tag{1}$$

The allocated resource of a fog node cannot exceed than total resource of the fog node. Let $P_f i(t)$ be total tasks that process at t time in fog node f_i where each task has different resource requirement configuration (i.e, $r_T i$). The total resource requirement is

$$R(T_P fi(t)) = \sum_{i=1}^{P_{fi}(t)} r_T i \tag{2}$$

The constraint of resource allocation can be represented as follows:

$$R(T_P fi(t)) \leq R_f \tag{3}$$

Example: Suppose a fog node f_1 has three containers c_1, c_2, and c_3, and each container has three resource requirement configurations r_{ij}^1, r_{ij}^2, and r_{ij}^3. All the resource requirements with different configuration of $\{CPU, bandwidth, memory\}$ are represented as follows:

$$f_1 = \begin{bmatrix} \{800, 1000, 1200\} & \{1100, 1400, 800\} & \{1600, 1600, 1540\} \\ \{880, 980, 1090\} & \{650, 200, 580\} & \{1800, 1400, 1620\} \\ \{1600, 1100, 1520\} & \{1040, 1500, 1040\} & \{1300, 950, 1150\} \end{bmatrix}$$

Let one task with resource requirement of $\{800, 900, 1150\}$ try to allocate the resource of fog node f_1. There are several solutions to allocate the required resource of fog node f_1 (e.g., $\{800, 1000, 1200\}$,$\{1600, 1600, 1540\}$, $\{1800, 1400, 1620\}$, $\{1600, 1100, 1520\}$, $\{1040, 1500, 1040\}$, and $\{1300, 950, 1150\}$). By considering higher numbers of fog nodes, the resource availability will be increased. If we consider another task with resource requirements (e.g., $\{1900, 1800, 1200\}$), it may not be allocated in fog nodes and offloaded to cloud server. Therefore, our task is making the decision that the task will be executed either in the cloud server or a fog node and tasks will be optimally allocated the resources of fog nodes that processed at time t.

4. Proposed Work

To solve the above problem, two modules—task clasification and buffering (TCB) and task offloading and optimal resource allocation (TOORA)—are modeled. The working process of these modules is given below.

4.1. Task Classification and Buffering (TCB)

Due to computation incapability of end devices, the tasks are transferred to nearest fog. The latency-sensitive tasks need to be processed first, thus these tasks are transferred to fog node. As noted above, the fog nodes are limited in resources and prediction of resource allocation is not immediately possible, that forced for buffering of tasks in the queue. If the queue length is long, then the time complexity is high. Similar to [4], parallel virtual queues are considered for buffering the same type of tasks into separate virtual queues, which helps to reduce the time complexity, as shown in Figure 3.

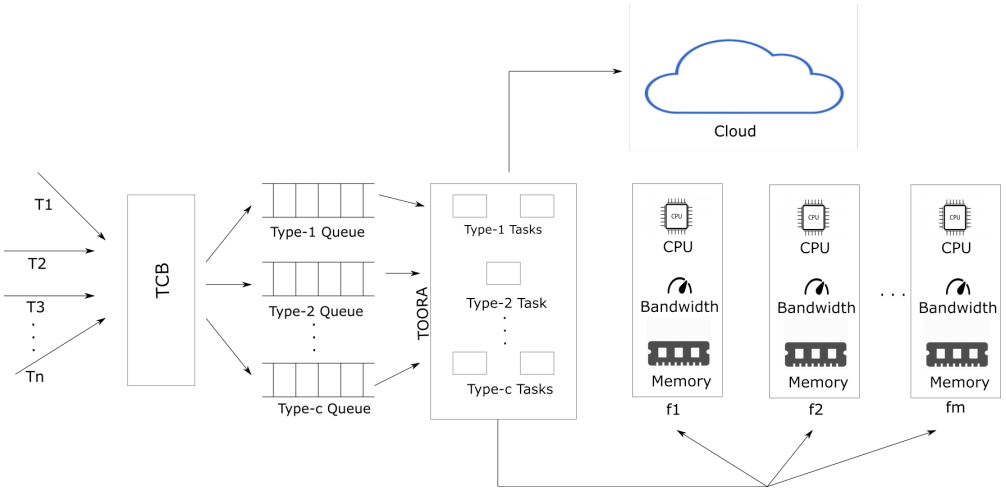

Figure 3. Queuing model.

Theorem 1. *Parallel virtual queues reduce the time complexity.*

Proof. If a single queue with length M is considered for buffering tasks, then the time complexity for buffering all tasks is $O(M)$. If we consider four types of tasks that can be buffered in four separate virtual queues, then each queue length is $O(M/4)$. So, the time complexity is also decreased to $O(M/4)$. □

The real-time tasks $T = \{T_1, T_2, T_3, \ldots.., T_n\}$ are streaming continuously from end devices and transferred to fog nodes. Each Task T_i can be represented with $\{arrt_i, etlow_i, etup_i, dsize_i, len_i, respt_i, dt_i\}$, where $arrt_i$, $etlow_i$, $etup_i$, $dsize_i$, len_i, $respt_i$, dt_i present arrival time, execution lower bound time, execution upper bound time, data size, number of instructions, response time, and deadline of the ith task, respectively. Assume that the tasks arrive at fog nodes in equal time intervals. The $arrt_i$, $dsize_i$, and dt_i of a task cannot be predicted before the task's arrival. The execution time of the task is also not predicted before completion of the task. However, the upper and lower bounds of execution time (i.e., $etlow_i$ and $etup_i$) can be estimated using machine learning algorithms proposed in [49]. As per estimation, $etup_i$ should not exceed $dt_i - arrt_i$. Here, we set $etup_i = dt_i - arrt_i$. Taking the above parameters of the task, the tasks can be classified into different types. The similar tasks can be grouped using a clustering algorithm. The tasks are overlapping, hence the FCM clustering algorithm is applied so that each task has a strong or weak association to the clusters. For the set of tasks T, the association to each cluster can be calculated as follows:

$$J_m(U, V; T) = \sum_{i=1}^{n} \sum_{j=1}^{c} (\mu_{ij})^m \left\| T_i - v_j \right\|^2 \tag{4}$$

where n is the total tasks, m is the fuzziness index $m\epsilon[1, \infty]$, and μ_{ij} represents the membership of the ith task to jth cluster center. (U, V) can minimize J_m when $m > 1$ and $\left\| T_i - v_j \right\| > 2$ for all i and j. Then μ_{ij} is

$$\mu_{ij} = 1 / \sum_{k=1}^{c} \left(\frac{\left\| T_i - v_j \right\|}{\left\| T_k - v_j \right\|} \right)^{\frac{2}{m-1}} \tag{5}$$

The cluster center can be calculated as

$$v_j = \frac{\sum_{i=1}^{n} (\mu_{ij})^m T_i}{\sum_{i=1}^{n} (\mu_{ij})^m} \tag{6}$$

An iteration technique is applied until the minimum of J_m or minimum error criteria are satisfied. An error threshold α can satisfy the condition, $\|V_{t-1} - V_t\|_{err} \leq \alpha$. The tasks are selected into a cluster using validity index. The Xie–Beni index [27,28] V_{XB} is one of widely used validity index is used here and can be defined as

$$V_{XB}(U, V; T) = \frac{\sum_{j=1}^{c} \sum_{i=1}^{n} \mu_{ij}^2 \|T_i - v_j\|^2}{n \left(\underbrace{\min}_{i \neq k} \{ \|v_j - v_k\| \} \right)} \tag{7}$$

Fuzzy c-means cluster can classify the tasks for a given time interval t. As the tasks are streaming continuously, dFCM [27] is used adaptively to update cluster centers. A new cluster center is generated automatically with new cluster generation. Initially, *cmin* number of clusters are generated, where $cmin \geq 2$. Upon arrival of new tasks, the membership of the present cluster is calculated. If the maximum membership value of the task exceeds or equals to membership threshold value (β), then it takes a new cluster center and generates a new cluster. Membership threshold (β) can avoid evaluation of cluster validity every time the tasks arrive. If the tasks satisfy the cluster membership, then there is no need to check for other, better clusters. The validity index is also evaluated when new centers are dissimilar to old ones. Then β can have the condition

$$\|V_{old} - V_{new}\| > \beta \tag{8}$$

Let C number of clusters are there in time t and maximum membership value of a task is lower than β then validity of clusters C is compared with $C - 2$ to $C + 2$. The clusters are generated using FCM and evaluated the validity index. The new cluster centers are generated for deviated tasks. This process is repeated to get the cluster center of best validity index until the arrival of tasks stop. The algorithm of task classification is as follows.

Algorithm 1 presents task classification using dFCM, which is discussed as follows. In this paper, an algorithm is presented using number of lines, and here, we consider the line number as a step. The parameters, such as threshold error α, membership threshold β, and range of c (i.e., number of clusters) are initialized in step-1. Here we are considering that tasks are coming in the same interval of time, hence *tmax* is considered as the last interval. Time interval t is initialized with 0 in step-2, and the initial number of cluster c is *cmin* in step-3. The following steps are computed until t reaches *tmax*:

- In step-5, take all the arrival tasks T in time t.
- Calculate c number of cluster centers, i.e., v_j and μ_{ij}, using Equations (6) and (5); in steps 6 and 7.
- In steps 8–20, check if the maximum membership value (i.e., μ_{ij}) of a task is more than or equal to membership threshold value (i.e., β). If true, then update μ_{ij} and v_j until $\|V_{t-1} - V_t\|_{err} \leq \alpha$, otherwise do steps 11–18 for $c - 2$ to $c + 2$ cluster centers. If no changes in clusters generated before then, store values of v_j, otherwise generate new cluster for deviated tasks and update c. Then, update μ_{ij} and v_j until $\|V_{t-1} - V_t\|_{err} \leq \alpha$ in step-20.
- In step-21, compute validity index using Equation (7) and select best clusters with best validity and assign to C' in step-22.
- Update the time interval t with t+1 in step-23.
- Finally, return clusters of tasks in step-25.

Algorithm 1 dFCM for task classification.

Input: Continuous streaming tasks $T = \{T_1, T_2, T_3, \ldots.., T_n\}$
Output: Cluster of tasks C'

1: Initialize μ, β, $cmin$, $cmax$, $tmax$, α;
2: $t \leftarrow 0$;
3: $c \leftarrow cmin$;
4: **while** $t < tmax$ **do**
5: Take a list of arrival tasks T at time t;
6: Compute c number of v_j using Equation (6);
7: Compute μ_{ij} using Equation (5);
8: **if** $max(\mu_{ij}) \geq \beta$ **then**
9: Update μ_{ij} and v_j until $\|V_{t-1} - V_t\|_{err} \leq \alpha$;
10: **else**
11: **for** number of clusters from c $-$ 2 to c + 2 **do**
12: **if** same number of clusters generated before **then**
13: Store values of v_j;
14: **else**
15: Generate new cluster center for deviated tasks;
16: $c \leftarrow c + 1$;
17: **end if**
18: **end for**
19: Update μ_{ij} and v_j until $\|V_{t-1} - V_t\|_{err} \leq \alpha$;
20: **end if**
21: Compute validity index using Equation (7);
22: $C' \leftarrow$ *Select clusters with best validity*;
23: $t \leftarrow t + 1$;
24: **end while**
25: Return clusters of task C';

On basis of the number of clusters, that number of virtual queues are modeled for buffering the tasks. The task can be buffered in the queue by comparing the level of urgency that presents how much time a task can wait. The level of urgency of the task can be determined multiple ways. Here, we are considering deadline and laxity time, which are most useful for finding maximum waiting time from current time. The upper bound execution time is considered as actual execution time of a task cannot predict before completion of task. The waiting time of a task is calculated using laxity time as follows:

$$lf_i = dt_i - (t + etup_i) \tag{9}$$

According to the lowest laxity time, the tasks can be buffered in different queues. However, some tasks may have the same lf_i; those tasks are then grouped, and the earliest deadline first (EDF) time is considered for determining the waiting time. EDF of task i is calculated as follows:

$$EDF_i = dt_i - etup_i \tag{10}$$

The algorithm for buffering the tasks in different queues is given below.

Algorithm 2 presents the task buffering in the queue that is discussed here. The results of Algorithm 1 are fed as the input of this algorithm (i.e., clusters of tasks). According to the number of clusters C', that number of queues are created (i.e., Q) in step-1. The following steps are computed for each cluster C'.

- Compute lf_i using Equation (9); for each task T_i in step-4.
- Sort all the tasks T_i according to lf_i in ascending order in step-6.
- If any tasks have similar lf_i, then group them and store them in LT in step-8.
- For each task of LT, compute EDF_i using Equation (10), and sort the tasks according to EDF_i in ascending order in steps 10–13.
- Insert all the tasks T_i in queue Q_i according to their lf_i and EDF_i in step-14.

- Finally, return the queues Q in step-16.

Algorithm 2 Buffering task in queues.

Input:Cluster of tasks C'
Output:Tasks buffered in queues Q

1: Take number of queues with number of clusters C';
2: **for** each cluster c in C' **do**
3: **for** each tasks T_i in c **do**
4: Compute lf_i using Equation (9);
5: **end for**
6: Sort all the tasks T_i according to lf_i in ascending order;
7: **if** some tasks having same lf_i **then**
8: $LT \leftarrow$ tasks of same lf_i;
9: **end if**
10: **for** each task LT_i in LT **do**
11: Compute EDF_i using Equation (10);
12: Sort the task LT_i in LT according to EDF_i in ascending order;
13: **end for**
14: Insert tasks of T_i in queue Q_i according to their lf_i and EDF_i;
15: **end for**
16: Return Queues Q;

4.2. Task Offloading and Optimal Resource Allocation (TOORA)

The buffered tasks in virtual queues are going to be executed in either the cloud or fog node. The head tasks of each virtual queue are checked in parallel as to whether they will be executed in the cloud server or fog node or there may be a failure to achieve the deadline. The laxity time (lf) of the task is used to determine the participation of the number of tasks of each queue for further operations.

The laxity time lf_i of tasks in each queue are compared with the maximum laxity time of the head task of the queues; if the laxity time lf_i of the task is below or equivalent to maximum laxity time of the head task, then those tasks are fetched for further processing, which can be represented as follows:

$$lf^{max} = max(lf^{j}_{HT}) \; where \, HT \leftarrow head(Q_j) \, and \, \forall \, j \epsilon [1, c] \tag{11}$$

$$lf^{j}_{i} \leq lf^{max} \; where \, i \epsilon T_i, \; j \epsilon Q \, and \, T_i \epsilon Q_j \tag{12}$$

The fetched tasks from queues are further processed in TOORA for deciding whether the task will be offloaded or failed due to longer waiting time with three conditions as follows:

- When $lf^{j}_{i} = 0$, the deadline and executable upper bound time are nearly the same, so the task cannot wait for longer time to execute in fog node. therefore, the task must be moved to the cloud server for successful completion.
- When $lf^{j}_{i} < 0$, the executable upper bound time is more than the deadline, thus, the task cannot complete before the deadline and is sent back to end devices requesting to increase the deadline.
- When $lf^{j}_{i} > 0$, the task has enough time for executing successfully at the fog node before the deadline.

Algorithm 3 can be represented as follows for task offloading:

Algorithm 3 Task offloading at fog node.

Input:Tasks in C'-type queues
Output:Tasks at fog node NF_c, cloud NC_c and Failure task $NFail_c$

1: **for** $j = 1$ to C' **do**
2: Compute lf^{max} using Equation (11);
3: **end for**
4: **for** $j = 1$ to C' **do**
5: **for** T_i of Q_j **do**
6: **if** $lf_i^j \leq lf^{max}$ **then**
7: Remove T_i from Q_j;
8: $TE \leftarrow T_i$;
9: **end if**
10: **end for**
11: **end for**
12: **for** i in TE **do**
13: **if** $lf_i^j == 0$ **then**
14: $NC_c \leftarrow i$;
15: **else if** $lf_i^j < 0$ **then**
16: $NFail_c \leftarrow i$;
17: **else**
18: $NF_c \leftarrow i$;
19: **end if**
20: **end for**
21: Return Queues NC_c, NF_c and $NFail_c$;

Algorithm 3 presents the task offloading at the fog node, which can distinguish the tasks of different types of queues. It takes all the tasks of C'-type queues and considers the tasks that are eligible for processing at that time. The number of tasks from each queue can be considered by computing steps 1–11. First, maximum laxity time of the head tasks of queues is computed in steps 1–3; next, the tasks from all C'-type queues whose laxity time is less than or equal to lf^{max} are selected and stored in TE list in steps 4–11. In steps 12–20, for each task in TE, check if laxity time of the ith task is equal to zero, then that task will send to the cloud server; if laxity time of the ith task is less than zero, then that task is marked as a failure and sent back to end devices for increasing the deadline; otherwise it will be executed in fog node. Finally, the tasks for fog node, the cloud, and failure are returned in step-21.

According to parallel virtual queues, let the number of tasks of type-c queue in time slot t be $Q_c(t)$ and $Q_c(0) = 0$. The tasks leave the queue when tasks are allocated resources in fog node or moved to the cloud server. The current length of type-c queue in a given time can be evaluated based on total tasks arrived and removed from the queue at the previous time slot. If N_c is total tasks of type-c that arrived, then the length of the queue can be evaluated as follows:

$$Q_c(t+1) = max[Q_c(t) + N_c(t) - NC_c(t) - NF_c(t) - NFail_c(t), 0] \tag{13}$$

where $NC_c(t)$, $NF_c(t)$, and $NFail_c(t)$ contain total tasks that are moved to the cloud, tasks allocated for resources at fog node, and tasks that are failed at time slot t.

To improve throughput and avoid starvation of tasks, the length of the $Q_c(t)$ can be controlled using a Lyapunov function as follows:

$$LD(t) = \frac{1}{2}\left[\sum_{c=1}^{C'} Q_c(t)\right]^2 \tag{14}$$

The Lyapunov drift, a difference of the Lyapunov function of two slots, can be defined as follows:

$$\Delta LD(t) = LD(t+1) - LD(t) \tag{15}$$

Applying Equations (13)–(15), we can rewrite as follows:
$\Delta LD(t) \leqslant B(t) - \sum_{c=1}^{C'} Q_c(t)[NF_c(t) + NC_c(t) + NFail_c(t) - N_c(t)]$
where $B(t) = \frac{1}{2}\sum_{c=1}^{C'}[NF_c(t) + NC_c(t) + NFail_c(t) - N_c(t)]^2$
The conditional expected Lyapunov drift can be represented as follows:

$$\begin{gathered} E\big[\Delta LD(t)\big|Q_u(t)\big] \leqslant B - \sum_{c=1}^{C'} Q_c(t) E\big[(NF_c(t) + NC_c(t) + NFail_c(t) - N_c(t))\big|Q_u(t)\big] \\ where\, Q_u(t) = \sum_{c=1}^{C'} Q_c(t),\ E\big[B(t)\big|Q_u(t)\big] \leq B \,and\, B > 0 \end{gathered} \tag{16}$$

On basis of Lyapunov drift theory, if $\Delta LD(t)$ is equivalent to zero or non-positive value, then the queue length is stable. The stability of queue depends on $\sum_{c=1}^{C'} Q_c(t)$. Although $NC_c(t)$, $NF_c(t)$, and $NFail_c(t)$ can influence the value of Equation (16), the number of tasks in $NC_c(t)$ and $NFail_c(t)$ are independent of tasks containing $NF_c(t)$. The tasks of $NF_c(t)$ are allocated to the available resources of fog nodes and satisfied the following:

$$\begin{gathered} maximize \sum_{i=1}^{C'} Q_c(t) NF_c(t), \\ s.t. \sum_{i=1}^{n} (NF_c(t) + NF_c^{'}(t)) r_{ij}^{l} \leq R_f \end{gathered} \tag{17}$$

where $NF_c^{'}(t)$ is the total ongoing tasks that cannot be released the resources in time t. The objective of our work is to satisfy Equation (17) and optimally allocate the resources. Most of the time meta-heuristic algorithms gives a near-optimal solution for the resource allocation problem [15,17]. Here, we are considering a meta-heuristic algorithm named whale optimization algorithm (WOA) [50]. The main strategy of WOA is the hunting behavior of one species of whale called Humpback. Humpback whales use the unique feeding method named bubble-net feeding to create circle around the prey and spread bubbles, so that the prey move to nearer surface of the ocean, as shown in Figure 4. The WOA get optimum solution using enclosing, bubble-net and explore methods.

In WOA, the random generated whale population are considered for optimization. These whales try to explore the location of prey and enclose them with bubble-net. During enclosing method, the whales upgrade their locations depending on best agent (i.e., target prey) as follows:

$$\overrightarrow{D} = |\overrightarrow{C} \otimes \overrightarrow{W_b(t)} - \overrightarrow{W(t)}| \tag{18}$$

$$\overrightarrow{W(t+1)} = \overrightarrow{W_b(t)} - \overrightarrow{A} \otimes \overrightarrow{D} \tag{19}$$

where $\overrightarrow{D}$ is the position vector difference of best agent ($W_b(t)$) and whales ($W(t)$), t is the present iteration, $\otimes$ is used for element-wise multiplication, and $\overrightarrow{A}$ and $\overrightarrow{C}$ are coefficient vectors and computed as

$$\overrightarrow{A} = 2\overrightarrow{a} \otimes \overrightarrow{r} - \overrightarrow{a} \tag{20}$$

$$\overrightarrow{C} = 2 \otimes \overrightarrow{r} \tag{21}$$

where every iteration decreases $\overrightarrow{a}$ from 2 to 0 linearly, and random vector $\overrightarrow{r}$ value lies in [0, 1]. The control parameter $\overrightarrow{a}$ can be improved as $\overrightarrow{a} = 2(1 - \frac{t}{T_{max}})$ (where T_{max} is the maximum iterations).

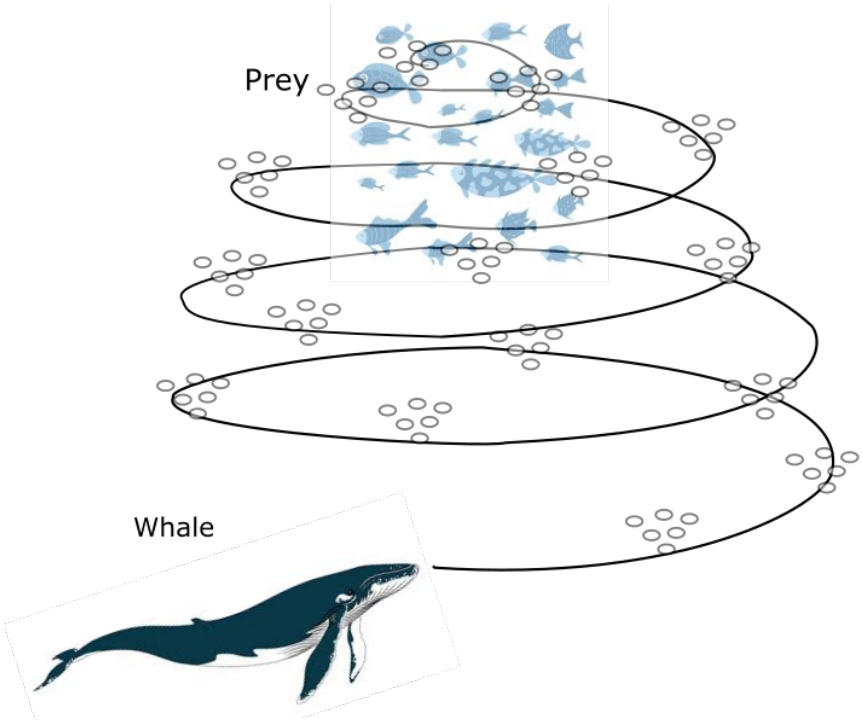

Figure 4. Whale hunting method.

Equations (20) and (21) balance the exploration and exploitation. When $\overrightarrow{A} \geq 1$, exploration occurs, and exploitation occurs when $\overrightarrow{A} < 1$. During exploitation, the probability of getting location solutions can be avoided by taking parameter $\overrightarrow{C}$ as a random value in [0, 2].

The bubble-net method has two approaches: shrinking enclosing and spiral updating. The shrinking enclosing can be achieved by taking $\overrightarrow{A}$ in $[-1, 1]$ with a linear decreasing value of $\overrightarrow{a}$ in each iteration. The spiral updating inspired with helix-shaped movement of Humpback whales is applied to update the position of the best agent and the whales as follows:

$$\overrightarrow{D'} = |\overrightarrow{W_b(t)} - \overrightarrow{W(t)}| \tag{22}$$

$$\overrightarrow{W(t+1)} = \overrightarrow{D'} \otimes e^{bl} \otimes \cos(2\pi l) + \overrightarrow{W_b(t)} \tag{23}$$

where a random generated l value lies in $[-1, 1]$ and b is a constant used for logarithmic spiral shape.

The shrinking enclosing and spiral updating are performed simultaneously as whales move around the prey using both approaches. This behavior can be modeled by taking each approach with 50% probability as follows:

$$\overrightarrow{W(t+1)} = \begin{cases} \overrightarrow{W_b(t)} - \overrightarrow{A} \otimes \overrightarrow{D} & if\ prob < 0.5 \\ \overrightarrow{D'} \otimes e^{bl} \otimes \cos(2\pi l) + \overrightarrow{W_b(t)} & if\ prob \geq 0.5 \end{cases} \tag{24}$$

where $prob \in [0, 1]$. When the coefficient vector $\overrightarrow{A}$ is greater than 1, the explore method is applied in which the whale location is replaced with a random whale rather than best agent. Thus, the algorithm can extend the search to a global search and can be represented as follows:

$$\overrightarrow{D} = |\overrightarrow{C} \otimes \overrightarrow{W_{rand}(t)} - \overrightarrow{W(t)}| \tag{25}$$

$$\overrightarrow{W(t+1)} = \overrightarrow{W_{rand}(t)} - \overrightarrow{A} \otimes \overrightarrow{D} \tag{26}$$

The bubble-net attach exploits the local solution from the current solution; whereas explore method tries to get a global solution from the population.

Here, we are considering WOA for allocating resources of fog nodes. Our whale optimized resource allocation (WORA) algorithm begins with generating a population of whales. Each whale denotes a random solution for a resource allocation problem. The fitness of each whale is calculated using a fitness function and selects a best solution with minimum fitness value as the current best agent. After this, the whales begin searching the global solution by updating each whale values of A, C, a, l,rob in each iteration. Where A and C are random coefficients, a is decreasing from 2 to 0 linearly, $prob$ is [0, 1], and l is [2, 0]. Distance function is the most important function in WOA, which is designed for a continuous problem. As resource allocation problem is a discrete problem, the distance function can be modified. Whale creation, fitness function, and distance function as per our model is discussed below.

- Whale creation: In our algorithm, each whale denotes a solution to the resource allocation problem. If we have a set of resources $R = r^1_{0,0}, r^2_{1,0}, r^0_{1,1}$ and a set of request tasks $T = T_0, T_1$, then the whale can be represented as a random combination of resource with task $W = [\{r^1_{0,0}, T_0\}, \{r^2_{1,0}, T_1\}]$. The resource is represented as $[f, c, r, CPU, bw, mem]$, where f, c, r, CPU, bw, and mem represent fog node, container of the fog node, resource block of the container, CPU usage, bandwidth, and available memory, respectively. The task can be represented as $[id, CPU, bw, mem]$, which denote task identification number, requirement of CPU usage, bandwidth, and memory. For example,
  $R = \{[0, 0, 1, 800, 1000, 1200], [1, 0, 2, 750, 1800, 1080], [1, 1, 0, 1800, 2400, 1620]\}$
  $T = \{[0, 600, 500, 500], [1, 700, 800, 1000]\}$
  Then a whale can be generated as follows:
  $$W1 = \begin{bmatrix} \{[0, 0, 1, 800, 1000, 1200], [0, 600, 500, 500]\} \\ \{[1, 0, 2, 750, 1800, 1080], [1, 700, 800, 1000]\} \end{bmatrix}$$
  $$W2 = \begin{bmatrix} \{[0, 0, 1, 800, 1000, 1200], [1, 700, 800, 1000]\} \\ \{[1, 0, 2, 750, 1800, 1080], [0, 600, 500, 500]\} \end{bmatrix}$$
  $$W3 = \begin{bmatrix} \{[1, 0, 2, 750, 1800, 1080], [0, 6, 150, 500]\}, \\ \{[1, 1, 0, 1800, 2400, 1620], [1, 700, 800, 1000]\} \end{bmatrix}$$
  In a similar fashion, all the whales are generated.
- Fitness function: For each whale, the fitness function is the optimal resource allocation to the task and can be calculated as

$$f = \sum_{k=1}^{len(W)} \frac{\left\{ (r^l_{ij}[cpu] - T_i[cpu]) + (r^l_{ij}[bw] - T_i[bw]) + (r^l_{ij}[mem] - T_i[mem]) \right\}}{len(W)} \tag{27}$$

  The whale with minimum fitness is the optimum solution. Hence, the goal of the algorithm is the minimization of the fitness function.
  The population can be generated by the collection of whales with their corresponding fitness.

$$pop = \{[w_1, w_1(f)], [w_2, w_2(f)], \ldots [w_p, w_p(f)]\} \tag{28}$$

- Distance function: The most important function of WOA is the distance function. As three parameters (i.e., CPU usage, bandwidth, and memory) are considered, the distance function can be redefined as follows:

$$\begin{array}{c} CPU_D = |\overrightarrow{C} \otimes W_i[CPU] - W_j[CPU]| \\ bw_D = |\overrightarrow{C} \otimes W_i[bw] - W_j[bw]| \\ mem_D = |\overrightarrow{C} \otimes W_i[mem] - W_j[mem]| \\ \overrightarrow{D} = [CPU_D, bw_D, mem_D] \end{array} \quad where\, i \neq j \tag{29}$$

$$\begin{array}{c} CPU_{D'} = |W_i[CPU] - W_j[CPU]| \\ bw_{D'} = |W_i[bw] - W_j[bw]| \\ mem_{D'} = |W_i[mem] - W_j[mem]| \\ \overrightarrow{D'} = [CPU_{D'}, bw_{D'}, mem_{D'}] \end{array} \quad where\, i \neq j \tag{30}$$

The WORA algorithm is given below.

Algorithm 4 presents assignment of fog resources to the tasks contained in NF. We initialize the whale population W_i where $i = 1, 2, 3, \ldots P$, time t is 0, and the maximum iteration is T_{max} in step-1. The best search agent $W_b(t)$ that has minimum fitness value is identified in step-2. While t is less than T_{max}, steps 3–21 are performed as follows:

- For each whale, steps 4–16 are performed. The value of A, C, a, l, and $prob$ are found in step 5.
- If $prob$ is less than 0.5, then check the absolute value of A in steps 6 and 7. If the absolute value of A is less than 1, then update D and W_i using Equations (18), (19), and (29) in step 8. Otherwise, select a random whale W_{rand} and update D and W_i using Equations (25), (26), and (29) in steps 10 and 11.
- If p is greater than 0.5, then update D' and W_i using Equations (22), (23), and (30) in step 14.
- After updating, amend W_i that goes beyond the search space in step 17. Then compute the fitness of all W_i and update the best search agent with minimum fitness in steps 18 and 19.
- Increment t by 1 in step 20.

Algorithm 4 Whale optimized resource allocation (WORA) algorithm.

Input: Set of resources R **and tasks for fog node** NF **where** $NF = \sum_{c=1}^{C'} NF_c$
Output: Best solution for resource allocation W_b

1: Initialize the whale population pop with W_i, where $i = 1, 2, \ldots P$, iteration $t = 0$, maximum iteration T_{max};
2: Identify the best search agent $W_b(t)$;
3: **while** $t < T_{max}$ **do**
4: **for** k = 1 to P **do**
5: Amend A, C, a, l and $prob$;
6: **if** $prob < 0.5$ **then**
7: **if** $|A| < 1$ **then**
8: Amend D and W_i by Equations (18), (19) and (29);
9: **else**
10: Choose a random whale W_{rand};
11: Amend D and W_i by Equations (25), (26) and (29);
12: **end if**
13: **else**
14: Amend D' and W_i by Equations (22), (23) and (30);
15: **end if**
16: **end for**
17: Amend W_i that goes beyond the search space;
18: Compute fitness of whale W_i;
19: Update W_b of best search agent;
20: $t \leftarrow t + 1$;
21: **end while**
22: Return W_b;

Finally, return the best search agent that has optimal resource allocation to the tasks in step 22.

The complexity of an algorithm measures both space and time complexity. The space complexity is the amount of space occupied by the algorithm. In the WORA algorithm, the space complexity is related to the population size and the dimension of the problem.

The population size is P and the dimension of the problem is D. Then, the space complexity is $O(P * D)$. In WORA, $D = 3$ for $\{CPU, bw, mem\}$, thus the space complexity is $O(P)$.

For time complexity, three major processes (i.e., initialization of the best whale, main loop for updating, and return of best solution) are considered. In WORA, T_{max} is the maximum iterations.

Initializing the best whale takes $O(P)$ times. The main loop updates the parameters, the whale that goes beyond the search space, and the optimum solution. The time complexity of these three stages are as follows:

Time required for updating the parameters is $O(P * D)$;

Time required for searching whales beyond the search space is $O(P)$;

Time for updating of optimal solution is $O(P)$;

Time required for main loop is the sum of above the operations $T_{max}(O(P) + O(P * D) + O(P)) \cong O(P)$ where $D = 3$ and ignored;

The time required for last step $O(1)$.

Therefore, total time complexity of WORA algorithm is $O(P)$.

The following lemmas [51] are required for optimal convergence of the algorithm:

Lemma 1. *The population* $\{W(t), t = 1, 2, ...\}$ *of WOA supports Markov chain which is finite and homogeneous.*

Lemma 2. *The population* $\{W(t), t = 1, 2, ...\}$ *of WOA absorb Markov process.*

Lemma 3. *If an individual of WOASU is stuck in local optima* $lp(t)$ *in the tth iteration, the transition probability of population* $\{W(t), t = 1, 2, ...\}$ *is*

$$P\big(W_i(t+1) = lp(t+1) \big| W_i(t) = lp(t)\big) = \begin{cases} 1 & lp(t) = lp(t+1) \\ 0 & lp(t) \neq lp(t+1) \end{cases} \tag{31}$$

Lemma 4. *The probability of convergence of WOASU algorithm towards the global optimal solution cannot possible.*

Lemma 5. *If an individual of WOAEP is stuck in the local optima* $lp(t)$ *in the tth iteration, the transition probability of population* $\{W(t), t = 1, 2, ...\}$ *is*

$$P\big(W_i(t+1) = lp(t+1) \big| W_i(t) = lp(t)\big) = \begin{cases} 1 & lp(t) = lp(t+1) \quad if\ A = 0\ or\ C = 1 \\ 0 & lp(t) = lp(t+1) \quad if\ A \neq 0\ and\ C \neq 1 \end{cases} \tag{32}$$

Lemma 6. *The WOAEP can converge in probability to the global optimum.*

Lemma 7. *WOA can converge in probability to the global optimum.*

In the WORA algorithm, each whale represents random combination of resource with task as $W = \left[\left\{r^1_{0,0}, T_0\right\}, \left\{r^2_{1,0}, T_1\right\}\right]$. The valid whales, where the amount of $[CPU, bw, mem]$ of resource is more than the requested task, can be considered for generating the populations. The fitness function, Equation (27), calculates the average minimum difference of requested resource to allocated resource. Thus, the best whale is the whale that has the minimum fitness value. Using Lemmas 1–7, it is proved that WOA with spiral updating or enclosing method with probability of 50% can converge to a global optimum. Even if WOA is trapped to local optima by executing spiral updating mechanism, it can be come out from local optima using the enclosing mechanism. The WORA algorithm also adopts both spiral updating and enclosing method with 50% probability. Hence, the WORA algorithm can converge to global optima with probability to a point in infinite iterations.

The whole process of Algorithms 1–4 of our work is shown in the flowchart in Figure 5.

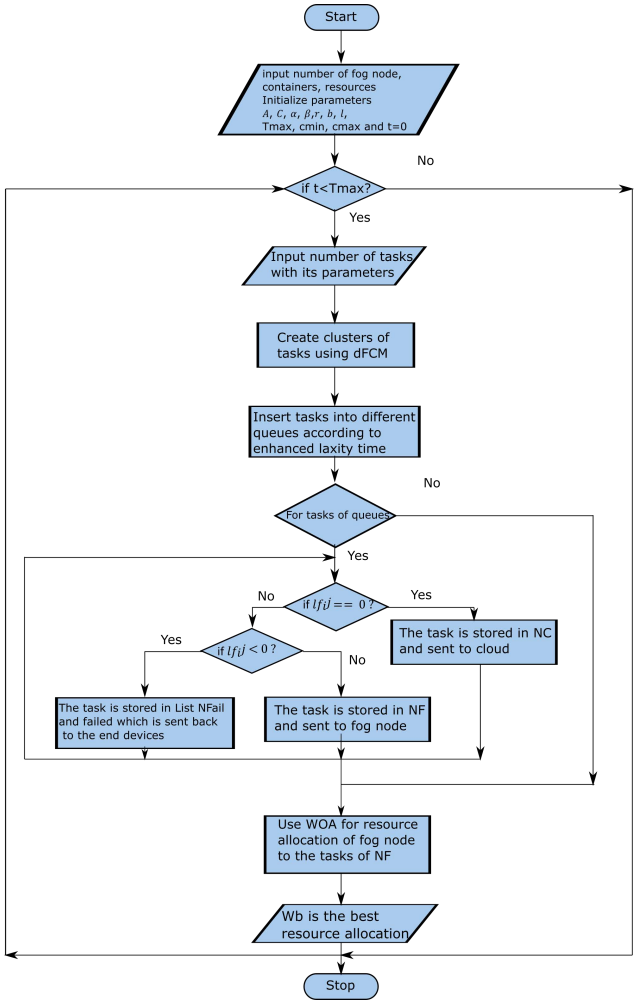

Figure 5. Flowchart of the algorithm.

5. Performance Evaluation

This section provides simulation setup, metrics performance, and evaluation of WORA compared with other algorithms.

5.1. Simulation Setup

We used python for implementing and evaluating our proposed algorithm. The hardware or software taken for the simulation is given in Table 4. We assumed different resource configurations for the different containers of the fog. Each fog has different resource configurations, hence each resource of the containers of fog is also different. The tasks are configured randomly. Table 5 gives a detailed configuration of cloud–fog infrastructure and tasks.

Table 4. Hardware/software specification.

Sl. No.	Hardware/Software	Configuration
1	System	Intel® Core ™ i5-4590 CPU @ 3.30 GHz
2	Memory (RAM)	4 GB
3	Operating System	Windows 8.1 Pro

We performed extended simulations with varied number of tasks and fog nodes in the system. The results of WORA are compared with SJF, FLRTS [30], and MOMIS [4]. We considered 3 to 20 fog nodes and 8 to 700 tasks.

Table 5. Resource configuration of cloud–fog infrastructure and task.

Name	Values
CPU rate of cloud	44,800 MIPS
Bandwidth of cloud	15,000 Mbps
Memory of cloud	40,000 MB
CPU rate of fog	22,800 MIPS
Bandwidth of fog	10,000 Mbps
Memory of fog	10,000 MB
Arrival time of tasks ($arrtime_i$)	[0, 10] ms
Execution lower bound of task ($etlow_i$)	[1, 6] ms
Execution upper bound of task ($etup_i$)	$[0, 6] + etlow$ ms
Execution time (et_i)	$([etlow, etup])$ ms
Data size of task	[10, 500] MB
deadline	$max(et, 20) + arrtime$
resptime	$arrtime + etlow$
No. of Instructions (len_i)	[10, 1700] MI
Bandwidth required for task	[10, 1800] Mbps
Memory required for task	[10, 1800] MB
CPU required for task	[10, 2200] MIPS

5.2. Performance Metrics

Here, the algorithm considered cost, energy consumption, makespan, and completion of task ratio as the performance metrics. All are defined below.

- Cost: *Cost* is the amount of monetary cost for processing the tasks in cloud and fog nodes. The cloud charges cost for both processing and communication, whereas the fog node only charges a cost for communication [1]. The cost of the system is defined as follows:

$$cost = \sum_{i=1}^{n} \begin{cases} c_1 * et_i + cc * (ds_i + \frac{len_i}{bw_c}) & i \in NC_c \\ cf * et_i & i \in NF_c \end{cases} \tag{33}$$

- Energy consumption: The total amount of energy consumed to execute all the tasks of a system is represented with *Energy consumption* metric. The total energy consumed in fog nodes is summed of the energy consumption for executing tasks and utilization of energy of the fog nodes being idle. When tasks are executed in the cloud, then total energy is summed of consumed energy for the execution of the task and also energy for transferring the task and data. The total consumed energy is as follows:

$$energy = \sum_{i=1}^{n} \begin{cases} ef * et_i + e_{idle} & i \in NF_c \\ ec * et_i + e_{comm} * (ds_i + \frac{len_i}{bw_c}) & i \in NC_c \end{cases} \tag{34}$$

- Makespan: The time required for completing all the tasks in the system is represented as *Makespan* [30]. It can be computed as

$$makespan = \underbrace{max}_{i}(resp_i + et_i)) \;\; i \in [NC_c, NF_c] \tag{35}$$

- Task completion ratio: *Task completion ratio* is the ratio of total tasks successfully completed within the deadlines.

$$Taskcompletionratio = \frac{\sum_{c=1}^{C'}(NC_c + NF_c)}{\sum_{c=1}^{C'}(NC_c + NF_c + NFail_c)} \tag{36}$$

The parameters for evaluating the metrics are given Table 6.

Table 6. Simulation parameters and values setup.

Parameters	Values
Processing cost per time unit for cloud (c_1)	0.5 G$/s
Communication cost per time unit for cloud (cc)	0.7 G$/s
Communication cost per time unit for fog (cf)	[0.3, 0.7] G$/s
Energy per unit for execution of the task in fog (ef)	[1, 5] w
Energy used when fog node is idle (e_{idle})	0.05 w
Energy per unit for execution of task cloud (ec)	10 w
Energy per unit for transmission of data (e_{comm})	2 w

5.3. Performance Analysis

Several experiments were carried out with different scenarios. When three fog nodes are considered where each fog node has three containers and each container has three resource blocks, Figure 6 shows the cost, energy consumption, makespan, and task completion ratio of varying tasks.

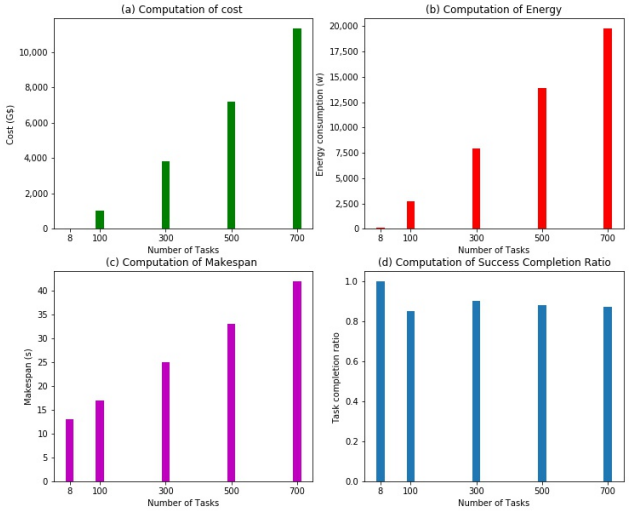

Figure 6. Cost, energy consumption, makespan, and task completion ratio in three fog nodes.

The proposed WORA algorithm is analyzed and compared with other three algorithms considering the metrics that we have taken. Figure 7 compares expenditure of cost of WORA with the other three algorithms with different numbers of fog nodes with 500 tasks. With an increase in fog nodes, the resource blocks are increased. Hence, a larger number of tasks are assigned to the fog nodes and a small number of tasks are transferred to the cloud for execution, which reduces the cost. The SJF algorithm forwards tasks to the cloud while the required resource is unavailable in the fog layer. The deadline as well as transmission delay of the task are considered in FLRTS. The tasks with a soft deadline or minimal latency are forwarded to the cloud. Therefore, less tasks are executed at the fog nodes in the FLRTS algorithm, which can increase the cost. Most of the tasks are assigned with resources of fog nodes in the MOMIS algorithm. Hence, the cost of the system is nearer of our WORA algorithm. The proposed WORA algorithm saves 23.89% of the average cost of FLRTS and 17.24% of the average cost of MOMIS algorithm.

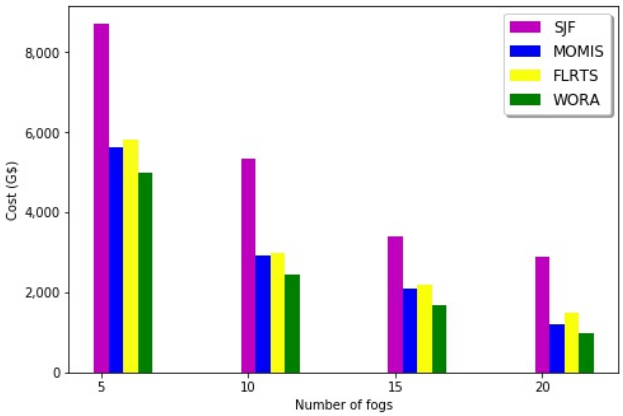

Figure 7. Computation of cost for fog with 500 tasks.

Figure 8 shows the computation of energy consumption with the number of fog nodes handling 500 tasks. It can be observed that increasing fog nodes can reduce energy consumption, because most of the tasks are executed in fog nodes where less tasks are moved to the cloud. When comparing the average energy consumption, it is observed that the WORA algorithm consumes 23.8% less energy than MOMIS and 30.76% less energy than FLRTS.

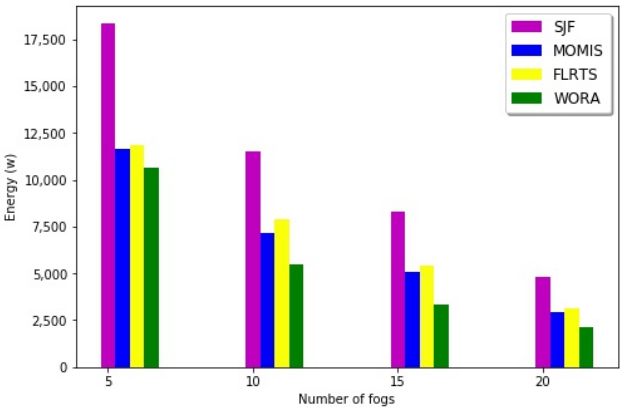

Figure 8. Computation of energy consumption for fog with 500 tasks.

When considering makespan with the number of fog nodes handling 500 tasks in Figure 9, it is observed that with an increase in fog nodes, the makespan is decreased. Instead of waiting for resources, the tasks are executed when fog nodes increases, which decreases the makespan. It is also observed that our WORA algorithm performed 6.8% better than MOMIS and 9% better than FLRTS in terms of makespan.

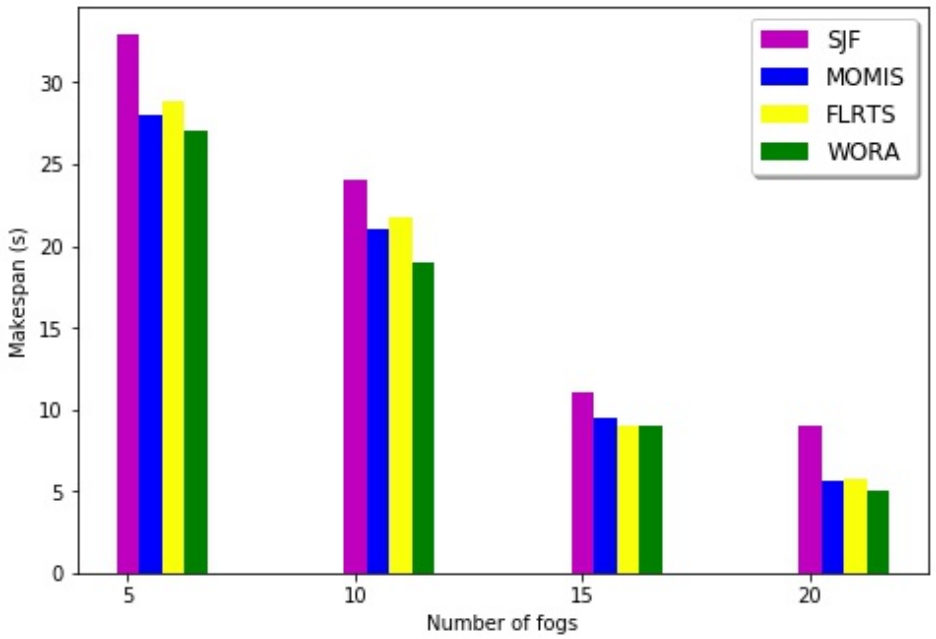

Figure 9. Computation of makespan for fog with 500 tasks.

When 500 tasks are executed in different fog nodes from 5 to 20, Figure 10 shows that our WORA algorithm performed 3.51% better than MOMIS and 5.4% better than FLRTS in terms of successful completion ratio of task.

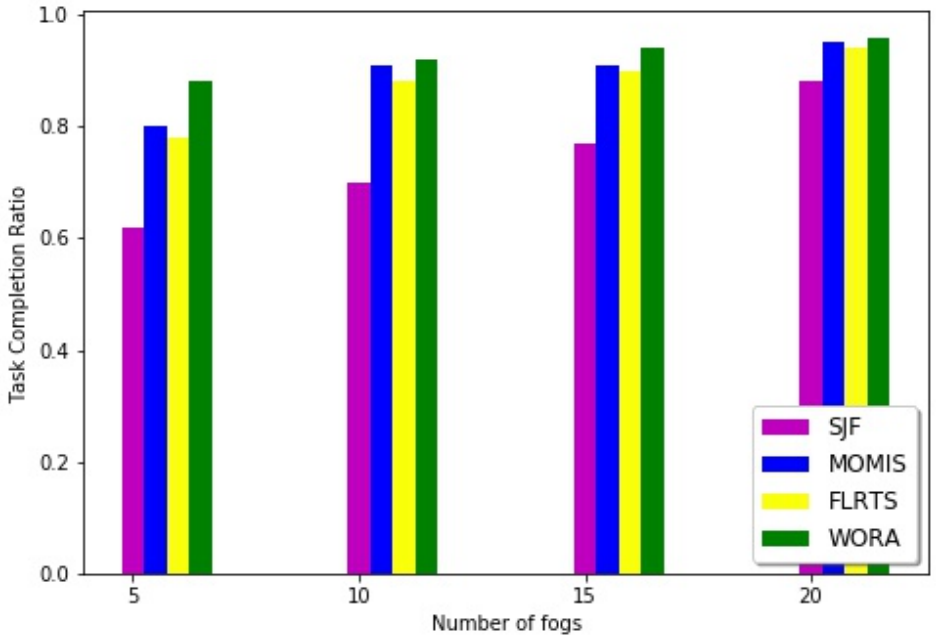

Figure 10. Compuatation of successful completion of task ratio for fog with 500 tasks.

When 15 fog nodes are considered with tasks varying from 100 to 700, Figure 11 shows that cost increased with increasing tasks. Our WORA algorithm saves 10.3% of the average cost of MOMIS and 21.9% of the average cost of FLRTS. Similarly, the WORA algorithm saves 18.57% of the average energy of MOMIS and 30.8% of the average energy of FLRTS, shown in Figure 12. Figure 13 shows that WORA performed 6.4% better than MOMIS and 12.9% better than FLRTS in terms of makespan. The successful completion of tasks within

the deadline is shown in Figure 14, where it is observed that WORA is 2.6% better than MOMIS and 4.3% better than FLRTS.

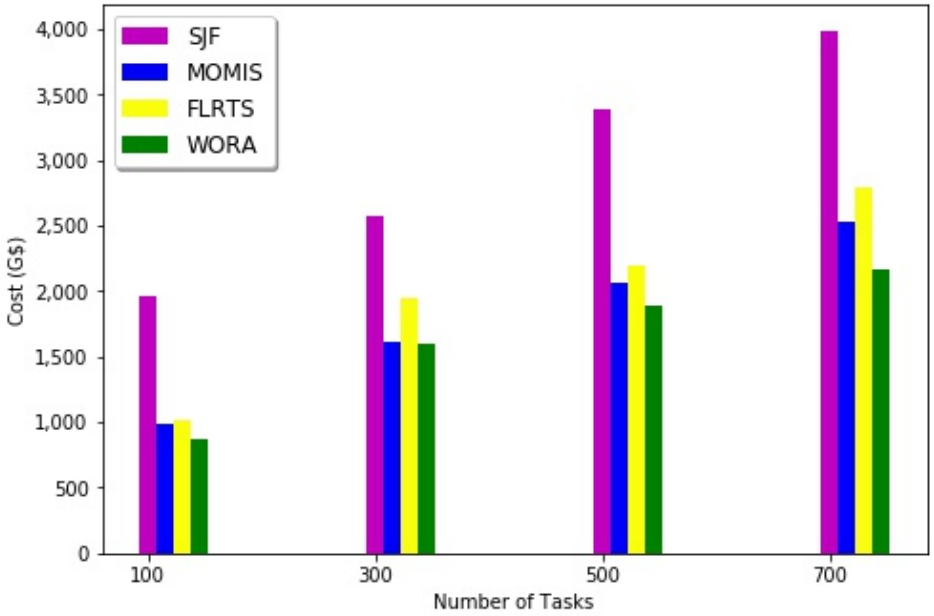

Figure 11. Computation of cost of tasks with 15 fog nodes.

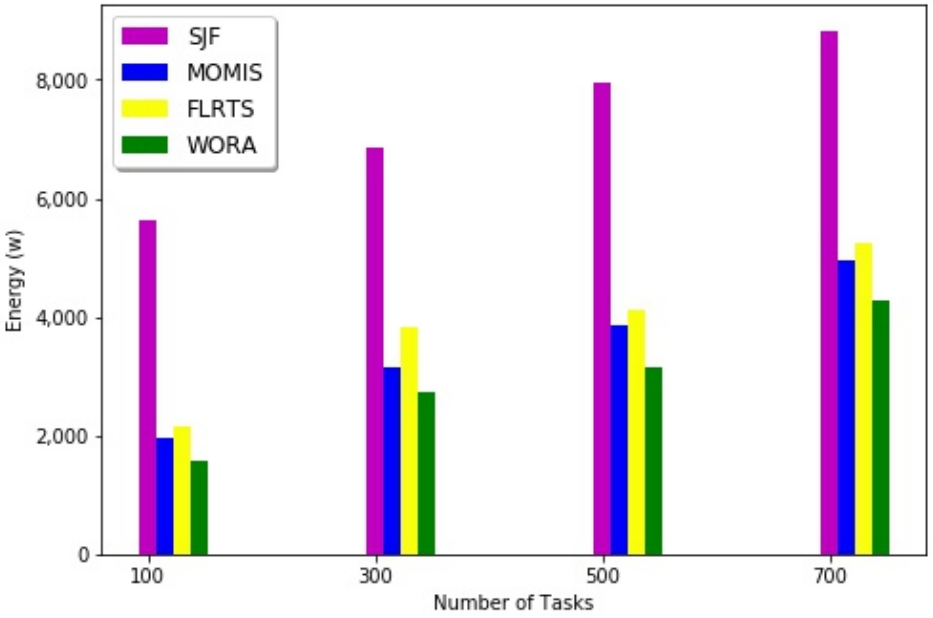

Figure 12. Computation of energy consumption of tasks with 15 fog nodes.

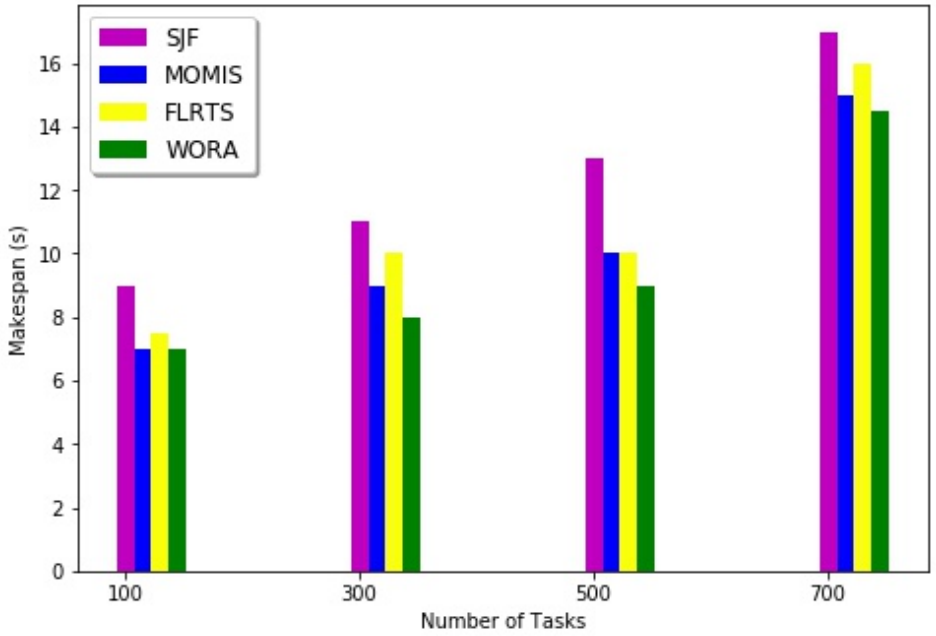

Figure 13. Computation of makespan of tasks with 15 fog nodes.

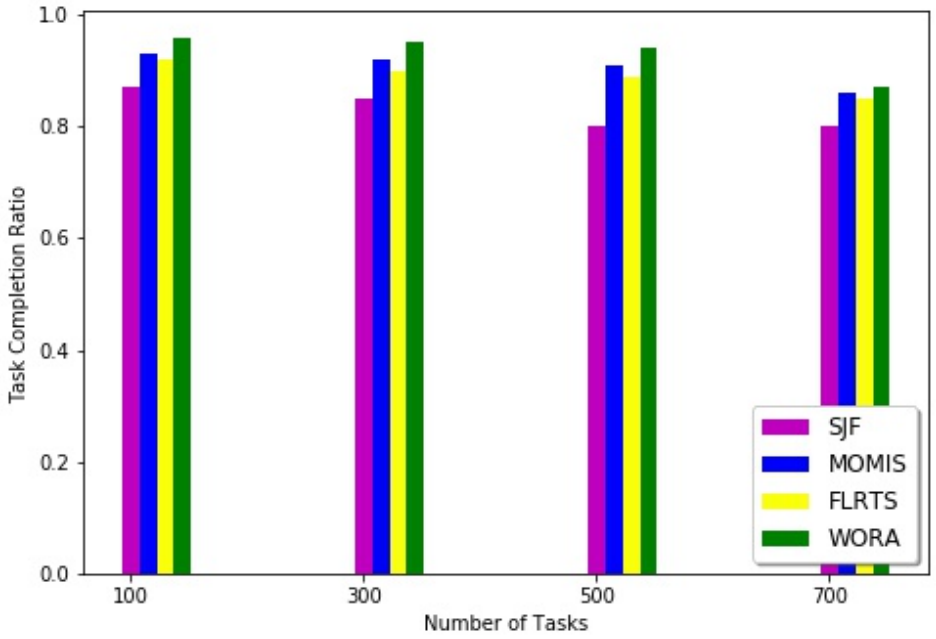

Figure 14. Computation of success completion ratio of tasks with 15 fog nodes.

In our WORA algorithm, whale optimization algorithm is used for resource allocation. The tasks have arrived at different time intervals. Figure 15 shows the minimum fitness value in different time intervals for numbers of tasks in three fog nodes.

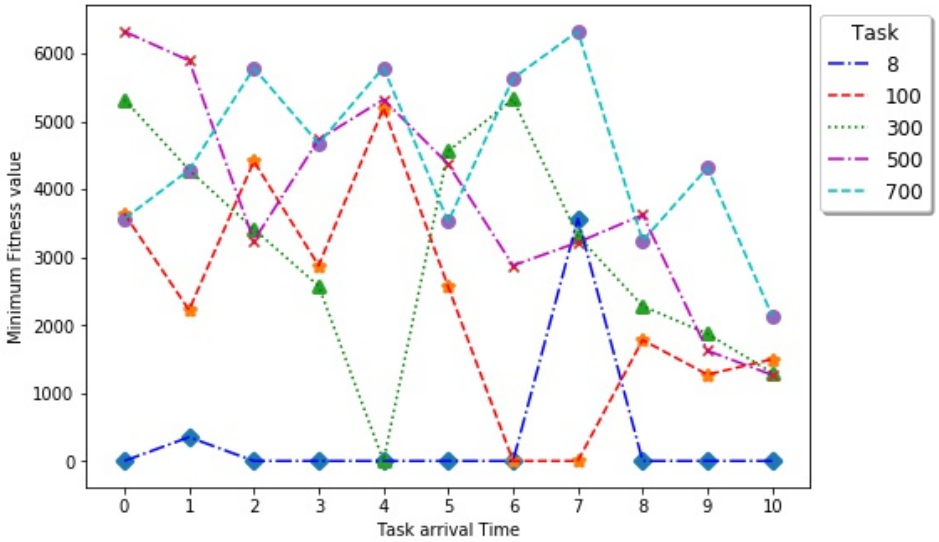

Figure 15. Minimum fitness value of tasks with arrival time intervals.

6. Conclusions

In this work, two modules—task classification and buffering (TCB) and task offloading and optimized resource allocation (TOORA)—are modeled for buffering the tasks in several queues according to their types, using the enhanced least laxity time the tasks are transferred to the cloud or fog. Considering the resource demand and deadline constraints of the tasks, a WOA is applied to assign the task to the optimal resource block of the fog node. The simulation results of our WORA algorithm evaluate metrics such as cost, energy consumption, makespan, and successful completion ratio of tasks and compare them with the standard SJF algorithm and existing algorithms such as MOMIS and FLRTS. When 500 tasks are executed in different fog nodes (e.g., 5 to 20), the results show that the WORA algorithm saves 23.89% of the average cost of FLRTS and 17.24% of the average cost of MOMIS. In terms of energy consumption, the WORA algorithm consumed 23.8% less energy than MOMIS and 30.76% less energy than FLRTS. Similarly, the WORA algorithm performed 6.8% better than MOMIS and 9% better than FLRTS in terms of makespan; and the WORA algorithm performed 3.51% better than MOMIS and 5.4% better than FLRTS in terms of successful completion ratio of the task. Similarly, when 100 to 700 tasks are executed in 15 fog nodes, it was observed that the WORA algorithm performed 3.51% better than MOMIS and 5.4% better than FLRTS in terms of successful completion ratio of the task, saving 18.57% of the average energy of MOMIS and 30.8% of the average energy of FLRTS. WORA performed 6.4% better than MOMIS and 12.9% better than FLRTS in terms of makespan and 2.6% better than MOMIS and 4.3% better than FLRTS in terms of successful completion ratio of the task. In the future, we will consider other metrics, such as throughput and delay rate for evaluating the performance of the algorithm. We also expand our research for virtual machine (VM) migration to balance the resource allocation.

Author Contributions: Conceptualization, R.S., S.K.B. and N.P.; methodology, R.S.; software, R.S.; validation, R.S., S.K.B. and N.P.; formal analysis, R.S.; investigation, S.K.B., N.P.; resources, R.S., S.K.B., N.P. and K.S.S.; data curation, R.S.; writing—original draft preparation, R.M.; writing—review and editing, S.K.B., N.P., K.S.S, N.J., M.A.A.; visualization, K.S.S., N.J. and M.A.A.; supervision, S.K.B., N.P.; project administration, N.J., M.A.A.; funding acquisition, N.J., M.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: Taif University Researchers Supporting Project number (TURSP-2020/98), Taif University, Taif, Saudi Arabia.

Data Availability Statement: Data and materials are available on request.

Acknowledgments: Taif University Researchers Supporting Project number (TURSP-2020/98), Taif University, Taif, Saudi Arabia. We want to thank BPUT Rourkela (Govt.), Odisha, India for providing adequate facility and infrastructure for conducting this research work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pham, X.Q.; Man, N.D.; Tri, N.D.T.; Thai, N.Q.; Huh, E.N. A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. *Int. J. Distrib. Sens. Netw.* **2017**, *13*, 1–16. [CrossRef]
2. Sahoo, K.S.; Tiwary, M.; Luhach, A.K.; Nayyar, A.; Choo, K.K.R.; Bilal, M. Demand–Supply-Based Economic Model for Resource Provisioning in Industrial IoT Traffic. *IEEE Internet Things J.* **2021**, *9*, 10529–10538. [CrossRef]
3. Lin, Z.; Lin, M.; De Cola, T.; Wang, J.B.; Zhu, W.P.; Cheng, J. Supporting IoT with Rate-Splitting Multiple Access in Satellite and Aerial-Integrated Networks. *Internet Things J.* **2021**, *8*, 11123–11134. [CrossRef]
4. Li, L.; Guan, Q.; Jin, L.; Guo, M. Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system. *IEEE Access* **2019**, *7*, 9912–9925. [CrossRef]
5. Bhoi, S.K.; Panda, S.K.; Jena, K.K.; Sahoo, K.S.; Jhanjhi, N.; Masud, M.; Aljahdali, S. IoT-EMS: An Internet of Things Based Environment Monitoring System in Volunteer Computing Environment. *Intell. Autom. Soft Comput.* **2022**, *32*, 1493–1507. [CrossRef]
6. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Available online: https://studylib.net/doc/14477232/fog-computing-and-the-internet-of-things--extend (accessed on 2 September 2022).
7. Sahoo, K.S.; Sahoo, B. Sdn architecture on fog devices for realtime traffic management: A case study. In *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 323–329.
8. Nayak, R.P.; Sethi, S.; Bhoi, S.K.; Sahoo, K.S.; Nayyar, A. ML-MDS: Machine Learning based Misbehavior Detection System for Cognitive Software-defined Multimedia VANETs (CSDMV) in smart cities. *Multimed. Tools Appl.* **2022**, 1–21. [CrossRef]
9. Rafique, H.; Shah, M.A.; Islam, S.U.; Maqsood, T.; Khan, S.; Maple, C. A Novel Bio-Inspired Hybrid Algorithm (NBIHA) for Efficient Resource Management in Fog Computing. *IEEE Access* **2019**, *7*, 115760–115773. [CrossRef]
10. Pham, Q.V.; Mirjalili, S.; Kumar, N.; Alazab, M.; Hwang, W.J. Whale Optimization Algorithm with Applications to Resource Allocation in Wireless Networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 4285–4297. [CrossRef]
11. Mao, L.; Li, Y.; Peng, G.; Xu, X.; Lin, W. A multi-resource task scheduling algorithm for energy-performance trade-offs in green clouds. In *Sustainable Computing: Informatics and Systems*; Elsevier Inc.: Amsterdam, The Netherlands, 2018; Volume 19, pp. 233–241. [CrossRef]
12. Nayak, R.P.; Sethi, S.; Bhoi, S.K.; Sahoo, K.S.; Jhanjhi, N.; Tabbakh, T.A.; Almusaylim, Z.A. TBDDosa-MD: Trust-based DDoS misbehave detection approach in software-defined vehicular network (SDVN). *CMC-Comput. Mater. Contin.* **2021**, *69*, 3513–3529. [CrossRef]
13. Ravindranath, V.; Ramasamy, S.; Somula, R.; Sahoo, K.S.; Gandomi, A.H. Swarm intelligence based feature selection for intrusion and detection system in cloud infrastructure. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–6.
14. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things Characterization of Fog Computing. In Proceedings of the MCC' 12, Helsinki, Finland, 17 August 2012; pp. 13–15.
15. Lahmar, I.B.; Boukadi, K. Resource Allocation in Fog Computing: A Systematic Mapping Study. In Proceedings of the 2020 5th International Conference on Fog and Mobile Edge Computing, Paris, France, 20–23 April 2020; pp. 86–93. [CrossRef]
16. Ahmed, K.D.; Zeebaree, S.R.M. Resource Allocation in Fog Computing: A Review. *Int. J. Sci. Bus.* **2021**, *5*, 54–63. [CrossRef]
17. Ghobaei-Arani, M.; Souri, A.; Rahmanian, A.A. Resource Management Approaches in Fog Computing: A Comprehensive Review. *J. Grid Comput.* **2020**, *18*. [CrossRef]
18. Mishra, S.K.; Mishra, S.; Alsayat, A.; Jhanjhi, N.; Humayun, M.; Sahoo, K.S.; Luhach, A.K. Energy-aware task allocation for multi-cloud networks. *IEEE Access* **2020**, *8*, 178825–178834. [CrossRef]
19. Bhoi, A.; Nayak, R.P.; Bhoi, S.K.; Sethi, S.; Panda, S.K.; Sahoo, K.S.; Nayyar, A. IoT-IIRS: Internet of Things based intelligent-irrigation recommendation system using machine learning approach for efficient water usage. *PeerJ Comput. Sci.* **2021**, *7*, e578. [CrossRef] [PubMed]
20. Rout, S.; Sahoo, K.S.; Patra, S.S.; Sahoo, B.; Puthal, D. Energy efficiency in software defined networking: A survey. *SN Comput. Sci.* **2021**, *2*, 1–15.
21. Chen, C.L.; Chiang, M.L.; Lin, C.B. The high performance of a task scheduling algorithm using reference queues for cloud-computing data centers. *Electronics* **2020**, *9*, 371. [CrossRef]
22. Behzad, S.; Fotohi, R.; Effatparvar, M. Queue based Job Scheduling algorithm for Cloud computing. *Int. Res. J. Appl. Basic Sci.* **2013**, *4*, 3785–3790.
23. Venkataramanan, V.J.; Lin, X. On the queue-overflow probability of wireless systems: A new approach combining large deviations with lyapunov functions. *IEEE Trans. Inf. Theory* **2013**, *59*, 6367–6392. [CrossRef]
24. Bae, S.; Han, S.; Sung, Y. A Reinforcement Learning Formulation of the Lyapunov Optimization: Application to Edge Computing Systems with Queue Stability. *arXiv* **2020**, 1–14. arXiv:2012.07279.

25. Eryilmaz, A.; Srikant, R. Asymptotically tight steady-state queue length bounds implied by drift conditions. In *Queueing Systems*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 72, pp. 311–359. [CrossRef]
26. Iyapparaja, M.; Alshammari, N.K.; Kumar, M.S.; Krishnan, S.S.R.; Chowdhary, C.L. Efficient resource allocation in fog computing using QTCS model. In *Computers, Materials and Continua*; Tech Science Press: Henderson, NV, USA, 2022; Volume 70, pp. 2225–2239. [CrossRef]
27. Sandhir, R.P.; Kumar, S. Dynamic fuzzy c-means (dFCM) clustering for continuously varying data environments. In Proceedings of the 2010 IEEE World Congress on Computational Intelligence, Barcelona, Spain, 18–23 July 2010. [CrossRef]
28. Sandhir, R.P.; Muhuri, S.; Nayak, T.K. Dynamic fuzzy c-means (dFCM) clustering and its application to calorimetric data reconstruction in high-energy physics. In *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*; Elsevier: Amsterdam, The Netherlands, 2012; Volume 681, pp. 34–43. [CrossRef]
29. Xu, J.; Hao, Z.; Zhang, R.; Sun, X. A Method Based on the Combination of Laxity and Ant Colony System for Cloud-Fog Task Scheduling. *IEEE Access* **2019**, *7*, 116218–116226. [CrossRef]
30. Ali, H.S.; Rout, R.R.; Parimi, P.; Das, S.K. Real-Time Task Scheduling in Fog-Cloud Computing Framework for IoT Applications: A Fuzzy Logic based Approach. In Proceedings of the 2021 International Conference on COMmunication Systems and NETworkS, COMSNETS 2021, Bengaluru, India, 5–9 January 2021; Volume 2061, pp. 556–564. [CrossRef]
31. Hosseini, S.H.; Vahidi, J.; Tabbakh, S.R.K.; Shojaei, A.A. Resource allocation optimization in cloud computing using the whale optimization algorithm. *Int. J. Nonlinear Anal. Appl.* **2021**, *12*, 343–360. [CrossRef]
32. Lin, Z.; Niu, H.; An, K.; Wang, Y.; Zheng, G.; Chatzinotas, S.; Hu, Y. Refracting RIS-Aided Hybrid Satellite-Terrestrial Relay Networks: Joint Beamforming Design and Optimization. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 3717–3724. [CrossRef]
33. Lin, Z.; An, K.; Niu, H.; Hu, Y.; Chatzinotas, S.; Zheng, G.; Wang, J. SLNR-based Secure Energy Efficient Beamforming in Multibeam Satellite Systems. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, 1–4. [CrossRef]
34. Lin, Z.; Lin, M.; Wang, J.B.; De Cola, T.; Wang, J. Joint Beamforming and Power Allocation for Satellite-Terrestrial Integrated Networks with Non-Orthogonal Multiple Access. *Signal Process.* **2019**, *13*, 657–670. [CrossRef]
35. Sun, Y.; Lin, F.; Xu, H. Multi-objective Optimization of Resource Scheduling in Fog Computing Using an Improved NSGA-II. *Wirel. Pers. Commun.* **2018**, *102*, 1369–1385. [CrossRef]
36. Taneja, M.; Davy, A. Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. In Proceedings of the IM 2017—2017 IFIP/IEEE International Symposium on Integrated Network and Service Management, Lisbon, Portugal, 8–12 May 2017. [CrossRef]
37. Bharti, S.; Mavi, N.K. Energy efficient task scheduling in cloud using underutilized resources. *Int. J. Sci. Technol. Res.* **2019**, *8*, 1043–1048.
38. Anu; Singhrova, A. Prioritized GA-PSO algorithm for efficient resource allocation in fog computing. *Indian J. Comput. Sci. Eng.* **2020**, *11*, 907–916. [CrossRef]
39. Jia, B.; Hu, H.; Zeng, Y.; Xu, T.; Yang, Y. Double-matching resource allocation strategy in fog computing networks based on cost efficiency. *J. Commun. Netw.* **2018**, *20*, 237–246. [CrossRef]
40. Feng, M.; Wang, X.; Zhang, Y.; Li, J. Multi-objective particle swarm optimization for resource allocation in cloud computing. In Proceedings of the Proceedings—2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012, Hangzhou, China, 30 October–1 November 2013; Volume 3, pp. 1161–1165. [CrossRef]
41. Ni, L.; Zhang, J.; Yu, J. Priced timed petri nets based resource allocation strategy for fog computing. In Proceedings of the 2016 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI 2016, Beijing, China, 20–21 October 2016; Volume 2018, pp. 39–44. [CrossRef]
42. Wang, Z.; Deng, H.; Zhu, X.; Hu, L. Application of improved whale optimization algorithm in multi-resource allocation. *Int. J. Innov. Comput. Inf. Control.* **2019**, *15*, 1049–1066. [CrossRef]
43. Alsaffar, A.A.; Pham, H.P.; Hong, C.S.; Huh, E.N.; Aazam, M. An Architecture of IoT Service Delegation and Resource Allocation Based on Collaboration between Fog and Cloud Computing. *Mob. Inf. Syst.* **2016**, *2016*, 6123234. [CrossRef]
44. Talaat, F.M. Effective prediction and resource allocation method (EPRAM) in fog computing environment for smart healthcare system. *Multimed. Tools Appl.* **2022**, *81*, 8235–8258. [CrossRef]
45. De Vasconcelos, D.R.; Andrade, R.M.D.C.; De Souza, J.N. Smart shadow—An autonomous availability computation resource allocation platform for internet of things in the fog computing environment. In Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS 2015, Fortaleza, Brazil, 10–12 June 2015; pp. 216–217. [CrossRef]
46. Wu, C.G.; Wang, L. A Deadline-Aware Estimation of Distribution Algorithm for Resource Scheduling in Fog Computing Systems. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, 10–13 June 2019; pp. 660–666. [CrossRef]
47. Bian, S.; Huang, X.; Shao, Z. Online task scheduling for fog computing with multi-resource fairness. In Proceedings of the IEEE Vehicular Technology Conference 2019, Honolulu, HI, USA, 21–25 September 2019; Volume 2019. [CrossRef]
48. Zhang, H.; Xiao, Y.; Bu, S.; Niyato, D.; Yu, F.R.; Han, Z. Computing Resource Allocation in Three-Tier IoT Fog Networks: A Joint Optimization Approach Combining Stackelberg Game and Matching. *IEEE Internet Things J.* **2017**, *4*, 1204–1215. [CrossRef]
49. Pham, T.p.; Durillo, J.J.; Fahringer, T. Predicting Workflow Task Execution Time in the Cloud using A Two-Stage Machine Learning Approach. *IEEE Trans. Cloud Comput.* **2017**, *8*, 256–268. [CrossRef]

50. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. In *Advances in Engineering Software*; Elsevier Ltd.: Amsterdam, The Netherlands, 2016; Volume 95, pp. 51–67. [CrossRef]
51. Feng, W. Convergence Analysis of Whale Optimization Algorithm. *J. Phys. Conf. Ser.* **2021**, *1757*, 1–10. [CrossRef]