An Intelligent Nature-Inspired Load Balancing Framework for Fog-Cloud Environments

1st Aishwarya Nayak
Department of Computer Science & Engineering
DRIEMS University
Cuttack, 754022, India
aishwaryanayak22@gmail.com

2nd Subhranshu Sekhar Tripathy
School of Computer Engineering
KIIT Deemed to be University
Bhubaneswar, 751024, India
subhranshu.008@gmail.com

3rd Sujit Bebortta
Department of Computer Science
Ravenshaw University
Cuttack, 753003, India
sujitbebortta1@gmail.com

4th Biswajit Tripathy
Departmenet of Computer Science
Ravenshaw University
Cuttack, 75303, India
biswajittripathy1998@gmail.com

***Abstract*—In this work, a Particle Swarm Optimization (PSO)-based method for load balancing in cloud-fog systems is presented, which tackles the dynamic and distributed settings' resource management issues. Effective load balancing techniques are required due to the complex interactions between fog and cloud computing in order to maximize resource usage and improve system performance. In order to reduce processing delays and boost overall system performance, the suggested PSO-based load balancing architecture makes use of the swarm intelligence concept to dynamically distribute jobs among fog nodes and the cloud. The efficiency of the PSO algorithm in reaching load equilibrium is shown by comprehensive simulations and performance assessments, highlighting its flexibility to changing workloads. Comparing the PSO-based load balancing strategy to conventional methods, the results show a considerable improvement in response times and resource utilization. Moreover, the PSO algorithm's distributed structure and scalability make it ideal for cloud-fog systems, where centralized management might not be feasible. This study adds to the current conversation on maximizing the benefits of fog and cloud computing in concert and provides a workable answer to load balancing problems in dynamic, heterogeneous environments.**

Keywords—Load balancing, Cloud Computing, Fog Computing, Nature-Inspired Optimization, Optimal Resource Utilization

I. INTRODUCTION

Several intelligent systems have emerged as a result of the Internet of Things (IoT), which is powered by the smooth gathering of environmental data via sensors and actuators. IoT-enabled devices include everything from watches, computers, and cellphones to bigger equipment like refrigerators and washing machines, as well as whole houses and vehicles. These gadgets produce data on the energy they use and consume, separating private information that is only available to the user from public information that controls the amount of electricity used overall. For example, a homeowner can monitor the energy use of particular appliances, such as TVs, refrigerators, fans, and air conditioners, by accessing private data.

Smart devices that rely on the IoT face difficulties with storage, network capacity, and latency—complexities that cloud computing may not be able to resolve on its own. Therefore, fog computing has been introduced to supplement cloud servers in order to overcome these issues. This becomes more important when it comes to load balancing in cloud-fog environments, because the way computational activities are divided across cloud and fog nodes is critical to maximizing system responsiveness and efficiency.

A. Cloud Computing

The cloud environment is a dynamic computing environment made up of physical machines (PM) and virtual machines (VM), providing customers with optimal use of computational and storage resources. In today's world, the smart grid appears as a driver of both economic and environmental advantages, using digital marketing and monitoring to reduce power costs in smart cities and communities. The rising demand for energy has fueled cloud computing's rise in popularity, especially in major industries where it effectively satisfies customer needs and requests.
In order to optimize the complexity of cloud operations, fog computing deliberately interjects services between end nodes and cloud servers. This becomes especially important when it comes to load balancing in Cloud-Fog situations, where efficiency is improved by the inclusion of game theory. As a model for providing internet-based services that include servers, databases, storage, and software, cloud computing is a fundamental component of modern technology. Even with its quick uptake, problems like traffic jams occur when several clients seek resources at once. 'CISCO's' introduction of the fog concept in 2014 is an attempt to address these constraints and enable more effective load balancing within the Cloud-Fog environment.

B. Fog Computing

By serving as an intermediary between customers and the cloud, fog computing intentionally lowers latency and storage needs in comparison to more extensive cloud infrastructure. Because of this feature, fog is a preferred alternative for a lot of customers, which encourages individual fog adoption within societies to improve performance and dependability. Clients use cloud and fog environments to accomplish numerous tasks using virtual and physical devices. Fog computing is used to minimize latency and speed up customer requests.

The extant literature utilizes diverse service broker policies and load balancing methods to maximize cloud and fog system performance while concurrently reducing computational expenses. Importantly, load balancing algorithms heavily rely on service broker policies like RDL (Reconfigure Dynamically with Load), ORT (Optimize RT), and CDC (Closet Data Center). This framework's main goals are to speed up response times and make sure the cloud server is under a balanced load. This focus on load balancing is especially important in the developing Cloud-Fog environment, where the best possible system performance depends on the effective distribution of computational activities across fog and cloud resources.

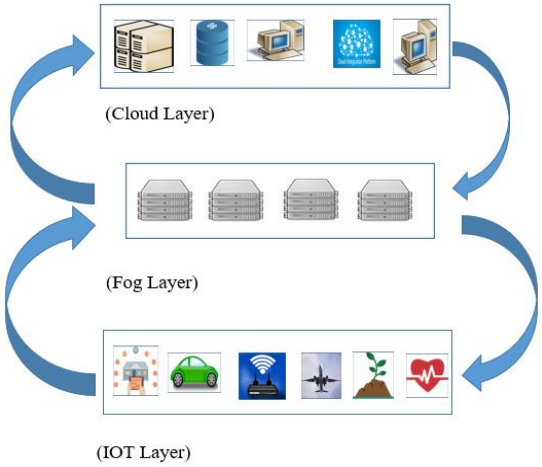

Figure 1. The overview architecture for IoT-Fog-Cloud infrastructure.

C. Motivations

Fog functions as an intermediary stratum synergizing with the cloud to ameliorate load and processing time dynamics. The intricate Fog environment necessitates the integration of sophisticated load balancing algorithms, including nature-inspired techniques like Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Genetic Algorithms (GA), along with diverse service broker policies. This research is distinctly focused on deploying fog computing within community networks, emphasizing its pivotal role in addressing load and processing time challenges. The core motivation lies in the exploration and implementation of nature-inspired load balancing algorithms, such as PSO, ACO, and GA, to adeptly manage and optimize the load on the cloud server within the Cloud-Fog environment.

D. Contributions

The fundamental contributions of this article are outlined as below:

- In order to effectively distribute computational workloads and maximize resource usage inside Cloud-Fog systems, the study presents and implements the PSO-based load balancing algorithm, a nature-inspired load balancing technique.
- The research endeavors to optimize system efficiency by employing the PSO algorithm to dynamically manage workloads between fog nodes and the cloud, ultimately reducing processing delays and boosting overall performance.
- The study makes a contribution by concentrating on the implementation of fog computing in community networks, demonstrating the usefulness of PSO-based load balancing in actual situations, especially in settings where community-based cloud-fog systems are essential.
- The main contributions include particular issues with load, processing time, response time, and costs in Cloud-Fog systems by utilizing the PSO-based load balancing algorithm in conjunction with some benchmark load balancing algorithms.

II. Related Work

This section deals with a comprehensive survey of some state-of-the-art studies which have implemented various load balancing algorithms to enhance the QoS of Cloud-Fog system. The Cloud Load Balancing (CLB) technique, which is intended to efficiently balance the load over the Cloud, is introduced by the authors in [1]. The CLB algorithm achieves successful load balancing and higher performance as compared to other methods. The results of a unique dynamic service broker strategy for fog selection are compared with those of the RR and Throttled load balancing algorithms in [1]. A new Service Broker Policy for load balancing is introduced in [2], and because of the policy's effective resource selection (PT), response times (RT) and processing times are shortened. Different approaches to load-balancing are examined in [3], where RR, throttled, and evenly dispersed current execution are contrasted. According to the study's findings, RR produces better results in their particular situation.

In [7], the authors use the Ant Colony System (MORA-ACS) to address the problem of effective resource allocation. Analytical comparisons with the RR algorithm for Energy Consumption, Processing Time (PT), and standard deviation show that the suggested algorithm achieves two goals: reducing energy use and guaranteeing a balanced load in the cloud. The work conducted by the authors in [4] uses algorithms like honeybee, throttled, RR, and equally dispersed current execution to offer a quantitative evaluation of load balancing strategies. According to simulation results, through processing time reduction and resource optimization, the equally dispersed current execution load balancing technique improves customer satisfaction. In order to reduce the workload for consumers, the article [5] presents a three-layer architecture that uses techniques such round-robin, throttled load balancing, and shortest remaining time first, which are selected in accordance with the broker policy and Closest Data Center Service (CDCS). In order to mitigate VM overloads brought on by intense IoT queries, load balancing at the fog layer is crucial, as discussed in [6], which tackles the growing user demand. By distributing the workload equally among all VMs, load balancing solutions at the fog layer effectively shorten response times.

To optimize load balancing in Cloud-Fog systems, a number of techniques have been put forth and evaluated. Especially in terms of allocating resources, accomplishing the twin goals of reducing energy usage and guaranteeing a steady load on the cloud. But among these options, the Particle Swarm Optimization (PSO) method is the most promising one for Cloud-Fog systems load balancing. Although the aforementioned algorithms have their advantages, the PSO

algorithm stands out for its exceptional ability to effectively handle computational tasks, maximize resource utilization, and guarantee a fair workload distribution. As such, it is a worthy candidate to handle the complexities of load balancing in the Cloud-Fog environment.

III. Proposed Framework

A number of Load Balancing (LB) algorithms that are used to control energy consumption and maximize resource usage are covered in this section. The three LB algorithms in Cloud Analyst—Round Robin (RR), Throttled, and Particle Swarm Optimization (PSO)—are thoroughly described in this article. To reduce Response Time (RT) for requests, Processing Time (PT) for data and transmission, and energy costs related to Cloud-Fog offloading and data transfer from service providers to customers, these algorithms are strategically used in the context of Cloud-Fog systems.

A. Round Robin Algorithm

The Round Robin (RR) algorithm is implemented for resource utilization on every host within the Cloud-Fog system. This algorithm is specifically employed for load balancing virtual machines, where, in the RR model, the data center controller selects a virtual machine randomly from the system panel after the initiation of the first request to fulfill the request. Subsequently, the DC controller continues to choose virtual machines randomly to address pending requests, with the selected virtual machine automatically being assigned to the top of the system panel. However, in scenarios with high traffic requests, the system experiences the drawback of nodes becoming overloaded, leading some nodes to go out of service and negatively impacting the overall system functionality [8].

Algorithm-1: Workflow for RR algorithm

```
1. def initialize_states():
2. % Set all VM states to VACANT
3. vm_states = {'VM1': 'VACANT', 'VM2': 'VACANT', 'VM3': 'VACANT', ...}
4. %Initialize the history matrix
5. history_matrix = []
6. return vm_states, history_matrix
7. def round_robin_load_distribution(vm_states, history_matrix, user_request):
8. vacant_vm = None
9. for vm, state in vm_states.items():
10. if state == 'VACANT' and is_comparable(vm, user_request):
        a. vacant_vm = vm
        b. break
11. if vacant_vm is None
12.     vacant_vm = round_robin_selection(vm_states)
13.  update_history_matrix(history_matrix, user_request, vacant_vm)
14.  update_vm_state(vm_states, vacant_vm)
15. def is_comparable(vm, user_request):
16.  return True
17. def round_robin_selection(vm_states):
18.     for vm in vm_states:
19.     if vm_states[vm] == 'VACANT':
20.         return vm
21. def update_history_matrix(history_matrix, user_request, vm):
22.  history_matrix.append({'user_request': user_request, 'allocated_vm': vm})
23. def update_vm_state(vm_states, allocated_vm):
24.     vm_states[allocated_vm] = 'BUSY'
25.     vm_states, history_matrix = initialize_states()
26. user_requests = ['Request1', 'Request2', 'Request3', ...]
27. for request in user_requests:
28.  round_robin_load_distribution(vm_states, history_matrix, request)
```

In the above Algorithm-1, the workflow for RR Algorithm is presented. This pseudocode illustrates a load balancing mechanism in a cloud-fog system that uses Round Robin to assign user requests to virtual machines (VMs). The round robin load distribution function looks for any open virtual machine (VM) entries that match the current user request. If a VM is found, it assigns it to the user's request; if not, a VM is chosen using round robin. The history matrix and the VM state list are updated appropriately by the algorithm.

B. Throttled Algorithm

The first step in the Throttled algorithm is to see if the index of the virtual machines table is available. The Data Center (DC) controller gets the call when a new request comes in and handles it to determine which virtual machine should be allocated next. Virtual machine allocations are updated dynamically in response to recognized requests. The VM IDs are communicated to the DC controller for verification, and the load balancer receives the data for VM allocation. To improve operational efficiency, the load balancer takes on the duty of keeping an exhaustive index of all virtual machines (VMs) and their corresponding statuses [9]. When a new request comes in, the DC controller looks to the balancer for advice on which virtual machine (VM) would be most suited to fulfill it. At first, all VMs are in a state of readiness. After finding the right virtual machine (VM), the balancer assigns the VM ID to the DC controller by searching the pre-established index [10]. The DC controller then sends the request to the assigned virtual machine (VM), notifies the balancer to make the necessary updates to the index table, and waits for the VM's response. The DC controller receives a response from the VM when it has finished processing the request, and the Throttled VM load balancer is informed of the modifications to update the index values [10].

Algorithm-2: Workflow of the Throttled Algorithm

```
1. def initialize_vm_states():
2. % Set VM state list's VM table index and status to VACANT
3.  vm_states = {'VM1': {'index': -1, 'status': 'VACANT'},
        'VM2': {'index': -1, 'status': 'VACANT'},
        'VM3': {'index': -1, 'status': 'VACANT'}, ...}
4. return vm_states
5. def throttled_load_balancing(vm_states, dcc_request):
6.  fresh_call = dcc_request
7.  % Check VM availability and start load balancing if VM equals 1
8.  if len([vm for vm in vm_states.values() if vm['status'] == 'VACANT']) == 1:
```

```
        available_vm = next(vm for vm, state in vm_states.items() if state['status'] == 'VACANT')
        vm_states[available_vm]['index'] = fresh_call
        vm_states[available_vm]['status'] = 'BUSY'
        vm_id = available_vm
        dcc_notify_allocated_vm(vm_id, dcc_request)
        dcc_notify_load_balancer(vm_states, dcc_request)
    else:
        vm_id = -1
        dcc_notify_load_balancer(vm_states, dcc_request)
    return vm_id
def dcc_notify_load_balancer(vm_states, dcc_request):
    pass
def dcc_notify_allocated_vm(vm_id, dcc_request):
    pass
vm_states = initialize_vm_states()
dcc_requests = ['Request1', 'Request2', 'Request3', ...]
for request in dcc_requests:
    vm_id = throttled_load_balancing(vm_states, request)
    print(f'Allocated VM for {request}: {vm_id}')
```

In Algorithm-2, the pseudocode for the TA algorithm is presented. This pseudocode illustrates how a cloud-fog system's load balancing algorithm uses throttled to assign user requests to virtual machines (VMs). If just one virtual machine (VM) is available, the throttled_load_balancing function initiates load balancing. It allocates the virtual machine (VM) to the user request if it is accessible; if not, it returns -1. The VM state list is updated appropriately by the method.

C. Particle Swarm Optimization Algorithm

To improve system performance, a load balancing technique for Cloud-Fog systems based on Particle Swarm Optimization (PSO) is suggested. The algorithm's goal is to optimize the distribution of computational workloads among fog nodes. This program iteratively creates a population of particles, each of which represents a possible load balancing setup. Each particle's position indicates a solution in the solution space, which corresponds to the characteristics of load and processing time. Particles dynamically modify their placements through iterative optimization, taking into account both the collective knowledge acquired from the swarm's optimal solution and their own past performance. The equilibrium between individual experience and collective behavior is governed by the cognitive and social coefficients, or c1 and c2, which make it easier to explore and utilize the solution space. The system load and processing time are reduced as the algorithm converges to an ideal load balancing choice. This method effectively solves load balancing in Cloud-Fog situations by utilizing PSO's built-in capacity to find global optima and adjust to dynamic changes.

Algorithm-3: Workflow of proposed PSO-based load balancing

```
PSO_Load_Balancing()
num_particles = 100
num_dimensions = 2
c1 = 1.5
c2 = 1.5
max_iterations = 50
particles = initialize_particles(num_particles, num_dimensions)
global_best_position = None
global_best_fitness = float('inf')
for iteration in range(max_iterations):
    for particle in particles:
        fitness = evaluate_fitness(particle)
        if fitness < particle['personal_best_fitness']:
            particle['personal_best_fitness'] = fitness
            particle['personal_best_position'] = particle['position']
        if fitness < global_best_fitness:
            global_best_fitness = fitness
            global_best_position = particle['position']
    update_particles(particles, global_best_position, c1, c2)
final_load_balancing_decision = global_best_position
def initialize_particles(num_particles, num_dimensions):
    particles = []
    for _ in range(num_particles):
        particle = {position': generate_random_position(num_dimensions),
  'velocity':
  generate_random_velocity(num_dimensions),
   'personal_best_position': None,
   'personal_best_fitness': float('inf')
}
        particles.append(particle)
    return particles
def generate_random_position(num_dimensions):
  pass
def generate_random_velocity(num_dimensions):
    pass
def evaluate_fitness(particle):
    pass
def update_particles(particles, global_best_position, c1, c2):
    pass
exit
```

IV. Performance Evaluation

A. Average Response Time

In order to manage the computing resources in resource constrained Fog-based computing platforms and for facilitating load balancing, the performance evaluation strategy for the proposed simulation environment has been presented [18-21]. The tasks corresponding to the fog layer correspond to some service demand made by a fog service-user viz., a remote mobile user, Internet users, or some web-based client at any given point in time. Here, the incoming requests to the fog layer i.e., the tasks are represented as t_i where $i = 1, 2, \cdots, n$ defines the tasks to be scheduled at the fog node by exploring the cloud resources. The present study investigates some well-known techniques like the RR algorithm, Throttled algorithm, and ESCE algorithm to

facilitate scheduling of the tasks generated at the fog nodes and for load balancing in the resource constrained fog computing devices. After completion of the task scheduling at fog nodes, the performance corresponding to different fog nodes considered for the analysis is performed. Thus, here the average response time of the fog nodes can be obtained by using the following formulation as [20],

$$T_{avg\_response} = \gamma \frac{t_i}{N} \quad (1)$$

where:

- γ = the binary variable which denotes whether a request is processed by the fog node.
- t_i = the time delay incurred by input services over the considered set of computing nodes.
- N =the total number of VMs involved for the simulation set-up.

B. Average Service Time

The fog nodes are capable of processing the requests generated by the users locally instead of pushing it to the cloud data centres and hence incur substantially lower response time as compared to cloud computing platforms. Thus, the average service time for a fog node can be given as [20,21],

$$T_{avg\_service} = p_i^{fog} \times \left(\delta_{ij} + X_{ij}^{fog} + Y_{ij}^{fog} \right) \quad (2)$$

where:

- p_i^{fog} = probability of the fog-user sending the requests to the fog layer for processing.
- δ_{ij} = delay incurred for processing and handling the requests generated between migration from fog node *i* to *j*.
- X_{ij}^{fog} = the propagation delay between nodes *i* to *j*.
- Y_{ij}^{fog} = sum of delay incurred at transmission links between nodes *i* and *j*.

V. Cost Model

The efficiency of a fog-based platform can be analysed by estimated the costs incurred at each level from the time of generation of a request at the fog device to its execution. The cost here refers to the financial cost imposed by the service provides for processing the tasks at the fog node [18-20].

A. VM Migration Cost

For achieving energy efficiency in constrained fog-based environments, an important technique is the consolidation of workloads through appropriate scheduling algorithms for making optimal usage of the fog-based computing resources. In this view, the migration of VMs is important for managing the workloads and for optimal utilization of servers. Thus, the VM migration cost can be obtained as [21],

$$\mathrm{Cost}_{VM\_migration} = \sum_{i=1}^{n} t_i \quad (3)$$

where:

- t_i = the tasks at i^{th} fog nodes.

B. Data Transfer Cost

The data transmission cost at the fog nodes can be given as [21],

$$\mathrm{Cost}_{data\_transmission} = \frac{S(t_i)}{I_i^{fog}} + T_{waiting}\left(t_{ij}\right) \quad (4)$$

where:

- $S(t_i)$ = the size of i^{th} tasks at the fog nodes.
- I_i^{fog} = the index of fog nodes computing the requests.
- $T_{waiting}\left(t_{ij}\right)$ = the waiting time of the tasks to be processed by the fog node.

C. Total Cost

The total cost for executing the requests by the fog layer is given as the sum of the data transfer cost and the VM migration cost and can be obtained as below [21],

$$\mathrm{Cost}_{total} = \sum_{i=1}^{n} t_i + \frac{S(t_i)}{I_i^{fog}} + T_{waiting}\left(t_{ij}\right) \quad (5)$$

Now by using Eqs.(3) and (4), the expression in Eq.(5) can be simplified to,

$$\mathrm{Cost}_{total} = \mathrm{Cost}_{VM\_migration} + \mathrm{Cost}_{data\_transmission} \quad (6)$$

VI. Simulation Results and Discussions

CloudSim and CloudAnalyst were used to build the experimental framework for the simulations, which were used to evaluate the performance of the studied technique in the context of fog computing [18, 25]. The simulations ran for sixty minutes and included a range of load balancing and work scheduling strategies. It is noteworthy that various service broker policies, including CDC, ORT, and RDL, were utilized to investigate in detail the effects of various workloads in the fog computing environment. The method used the Xen Virtual Machine Monitor (VMM) to obtain performance data from the CloudAnalyst platform, which has a processor running at 10000 GHz with 204 GB of RAM and can execute complicated instruction sets on an x86 architecture. A set pace of 60 requests per user were made; each request matched a 100-byte piece of data. A request grouping factor of 10 was defined on the fog nodes, indicating the maximum number of simultaneous requests that each fog server could handle at any given time.

A detailed depiction of the comparison study of average reaction times, expressed in milliseconds, for different quantities of fog nodes is given in Figure 2. Three different algorithms are compared in this illustration: the RR, TA, and the proposed PSO. It has been discovered that in a variety of fog node modification circumstances, the Proposed PSO consistently performs better than the benchmark techniques, RR and TA. The graph effectively demonstrates the PSO algorithm's superior efficiency in reducing average reaction times, confirming its effectiveness in improving the Cloud-Fog system's responsiveness. The PSO algorithm is a good option for load balancing in this distributed computing environment because of its strong flexibility and optimization skills, as evidenced by the declining trend in response times for the algorithm with an increase in fog nodes.

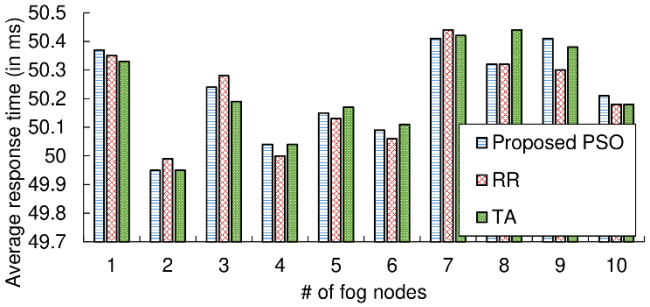

Figure 2. Comparison of average response time in milliseconds for varying number of fog nodes corresponding to Proposed PSO, RR, and TA algorithms.

Figure 3 compares the costs of virtual machine migration for varying numbers of fog nodes and shows the proposed PSO, RR, and TA. The PSO algorithm consistently performs better than RR and TA, demonstrating its efficient optimization skills in reducing the expenses associated with virtual machine migration as the number of fog nodes increases.

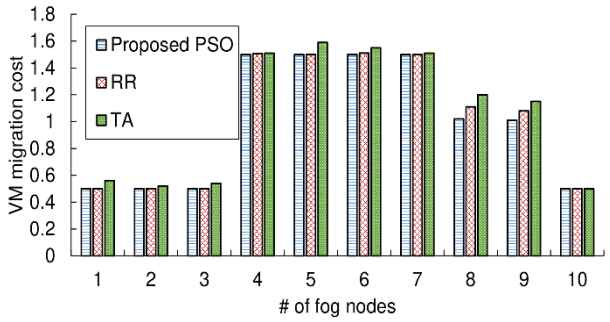

Figure 3. Comparison of VM migration cost for varying number of fog nodes corresponding to Proposed PSO, RR, and TA algorithms.

The comparison of data transfer costs with different numbers of fog nodes, including the proposed PSO, RR, and TA, is shown in Figure 4. As the number of fog nodes rises, the representation amply illustrates the PSO algorithm's superior performance and shows how effective it is in minimizing data transfer costs as compared to the benchmark techniques (RR and TA).

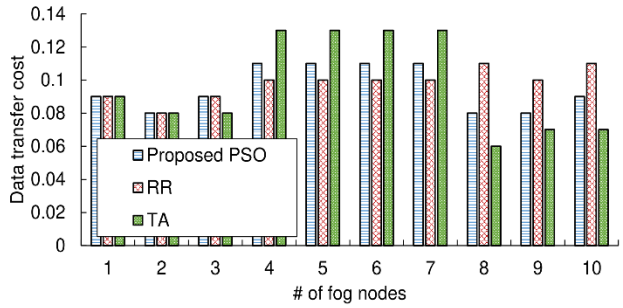

Figure 4. Comparison of data transfer cost for varying number of fog nodes corresponding to Proposed PSO, RR, and TA algorithms.

The comparison of total costs using the Proposed PSO, RR, and TA is shown in Figure 5 for different numbers of fog nodes. The graphic depiction emphasizes the PSO algorithm's constant superiority over the benchmark techniques (RR and TA), highlighting how well it minimizes total costs in a variety of fog node settings.

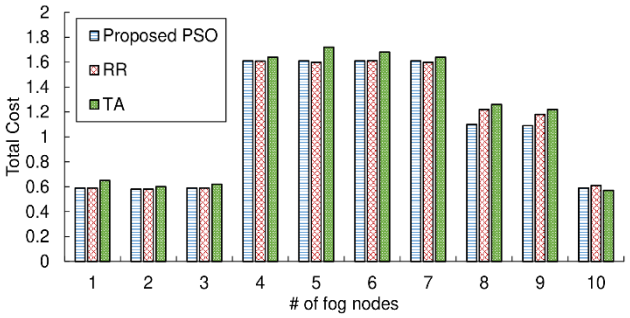

Figure 5. Comparison of total cost incurred over varying number of fog nodes corresponding to Proposed PSO, RR, and TA algorithms.

VII. Conclusions and Future Work

The present study for facilitating load balancing in Cloud-Fog systems with the PSO algorithm has produced interesting findings. The performance results were thoroughly simulated with different numbers of fog nodes, and important QoS metrics were examined, such as average response time, VM migration cost, data transfer cost, and overall cost. The PSO algorithm's higher effectiveness was brought to light through a comparative examination with two benchmark load balancing techniques: RR and the TA algorithm. Across a range of cases, the PSO continuously showed better optimization skills than the benchmark methods. The present study validates PSO as a resilient and flexible load balancing solution in Cloud-Fog systems, demonstrating its capacity to improve QoS metrics and make a substantial contribution to the general dependability and efficiency of distributed computing environments.

Hybrid optimization techniques represent a viable direction for future research in the field of load balancing for Cloud-Fog systems. PSO's advantages can be combined with those of other nature-inspired algorithms or meta-heuristic techniques to potentially provide a synergistic solution that reduces the drawbacks of each method and improves system performance even more. Furthermore, it could be insightful to look into how well these hybrid algorithms adapt to changing and dynamic fog environments where workload patterns or the number of nodes fluctuate.

References

[1] Tripathy SS, Imoize AL, Rath M, Tripathy N, Bebortta S, Lee CC, Chen TY, Ojo S, Isabona J, Pani SK. A novel edge-computing-based framework for an intelligent smart healthcare system in smart cities. Sustainability. 2022 Dec 31;15(1):735.

[2] Ebneyousef S, Shirmarz A. A taxonomy of load balancing algorithms and approaches in fog computing: a survey. Cluster Computing. 2023 Feb 21:1-22.

[3] Sandhiya B, Canessane RA. An Extensive Study of Scheduling the Task using Load Balance in Fog Computing. In2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS) 2023 Mar 23 (pp. 1586-1593). IEEE.

[4] Tripathy SS, Rath M, Tripathy N, Roy DS, Francis JS, Bebortta S. An Intelligent Health Care System in Fog Platform with Optimized Performance. Sustainability. 2023 Jan 18;15(3):1862.

[5] Bebortta S, Tripathy SS, Basheer S, Chowdhary CL. DeepMist: Towards Deep Learning Assisted Mist Computing Framework for Managing Healthcare Big Data. IEEE Access. 2023 Apr 11.

[6] Bebortta S, Tripathy SS, Basheer S, Chowdhary CL. FedEHR: A Federated Learning Approach towards the Prediction of Heart Diseases in IoT-Based Electronic Health Records. Diagnostics. 2023 Oct 10;13(20):3166.

[7] Bebortta S, Tripathy SS, Modibbo UM, Ali I. An optimal fog-cloud offloading framework for big data optimization in heterogeneous IoT networks. Decision Analytics Journal. 2023 Sep 1;8:100295.

[8] Zahoor, S., Javaid, N., Khan, A., Ruqia, B., Muhammad, F.J., Zahid, M.: A cloudfog-based smart grid model for efficient resource utilization, 04 2018

[9] Fatima, I.,Javid, N.,Iqbal,M.N.,Shafi,I.,Anjum,A.,Memon,U.;, "Integration of cloud and fog based environment for effective resource distribution in smart buildings," In:14th IEEE International Wireless Communications and Mobile Computing Conference(IWCMC-2018), 2018.

[10] Lu, K., Yahyapour, R., Wieder, P., Yaqub, E., Abdullah, M., Schloer, B., Kotsokalis, C., " Fault-tolerant service level agreement lifecycle management in clouds using actor system," Future generation. Computer system, 2016.

[11] Kaur, Paramjeet., ""A Comparison of Popular Heuristics for Load Balancing in Cloud Computing."," 2018.

[12] Khalid, S.: Applied Computational Intelligence and Soft Computing in Engineering. IGI Global, Hershey (2017)

[13] Mandeep Kaur, Rajni Aron, "Equal Distribution Based Load balacing techniques for fog based cloud computing".

[14] Meftah A, Youssef AE, Zakariah M, " Effect of service broker policies and load balancing on the performance of large scale internet applications in cloud data centers," , 2018.

[15] Verma S, Yadav AK, Motwani D, Raw RS, Singh HK, "An efficient data replication and load balancing technique for fog computing environment.," 3rd international conference on computing for sustainable global development (INDIACom)., 2016.

[16] S. Pandey, "cloud load balancing a perspective study," "International journal of engineering and computer science", 2017.