Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

Resource Optimization In Fog Computing With Shift-Invariant Deep Convolutive Load Balancing

S. V. Nethaji[1] , Dr. M. Chidambaram[2]

[1]Research Scholar, PG & Reasearch Department of Computer Science, Rajah Serfoji Government College(Autonomous), (Affiliated to Bharathidasan University, Tiruchirappalli), Thanjavur, India.

[2]Assistant Professor, PG & Reasearch Department of Computer Science, Rajah Serfoji Government College(Autonomous), (Affiliated to Bharathidasan University, Tiruchirappalli), Thanjavur, India.

ABSTRACT

Fog computing is a well-known computing paradigm that uses fog environments to deliver computation and storage services, effectively managing decentralised resources. Internets of Things (IoT) applications choose fog computing nodes to meet end-user needs. Fog computing is a notion that lies somewhere between IoT devices and the cloud paradigm, with the purpose of lowering job scheduling and load balancing latency. Load balancing is what determines the effectiveness of resource allocation and management solutions. Various solutions have been proposed to resolve issues about cloud load balancing. Using population-based methods, on the other hand, did not improve load balancing efficiency or resource use. Multi objective chaotic salp swarm resource optimised shift-invariant deep convolutive load balancing (MCSSROSIDCLB) is a new technique for effective load balancing with a short makes pan. The MCSSROSIDCLB approach uses numerous processing layers to achieve correct results, including input, more than one hidden layer, and output layer. The MCSSROSIDCLB technique connects user-requested tasks to Fog nodes according to resource availability. The input layer collects the number of tasks before passing it on to the hidden layer. The load balancer evaluates fog node resource availability by taking end-user requests. The load balancer finds the resource-optimized fog node based on CPU, memory, and bandwidth in the second hidden layer by using Multi objective chaotic salp swarm optimization. Finally, the load balancer distributes incoming workloads to the resource-optimized fog node in the third hidden layer. At the output layer, this technique allows for more resource-optimized load balancing. Experiments on characteristics including load balancing efficiency, make span, and memory consumption are carried out for a variety of user tasks. The observed data and discussion reveal that the MCSSROSIDCLB technique improves load-balancing efficiency and reduces make span and memory utilisation when compared to state-of-the-art alternatives.

Keywords: Fog computing, load balancing, resource allocation, shift-invariant convolutive deep learning, Mult iobjective chaotic salp swarm optimization

1. INTRODUCTION

The Internet of Things (IoT) devices benefit from a fog computing network with well-organized fog nodes (FNs) at the network's edge, which enables higher communication performance. Data processing using Fog computing reduces data transfer latency and latency needs significantly. Many real-time applications, such as healthcare, industrial systems, and intelligent traffic signs, rely on fog computing. Fog, on the other hand, is a potentially promising computing model that still requires load balancing standardisation. In fog computing, load balancing is a process that evenly distributes many workloads among resources to improve performance. There have been several meta-heuristic load balancing techniques created. However, resource-optimized load balancing is a difficult problem to solve.

To handle the complete fog nodes and choose the best suited node for the current job assignment, [1] presented a particle swarm optimization-based Enhanced Dynamic Resource Allocation Method (EDRAM). Fog node computing, on the other hand, failed to apply an efficient load-balancing algorithm for real-time applications in order to reduce reaction time. [2] developed the Load Balancing and Optimization Strategy (LBOS), which uses dynamic resource allocation based on reinforcement learning and a genetic algorithm. LBOS was a useful tool for determining resource consumption and ensuring continuous operation with a shorter make span. The efficiency of load balancing, on the other hand, did not improve. [3] explored various load balancing strategies, but no performance study of various metrics was done. [4] created a novel Effective Load Balancing Strategy (ELBS) for fog computing that is applicable for a variety of real-time applications. The designed technique, on the other hand, proved ineffective in reducing the resource utilisation of the multiple task scheduling. [5] presented a virtual machine management strategy for assigning various service requests to active fog nodes chosen using a genetic algorithm. However, when it came to picking the active fog nodes, it failed to take into account the different objective functions. [6] proposed two nature-inspired metaheuristic algorithms, ACO and PSO, to efficiently conduct load balancing over fog nodes. Although the designed approach reduces reaction time considerations, load balancing efficiency was not enhanced. [7] developed the multi-objective Workflow Offloading (MOWO) technique for workflow scheduling and job offloading with the shortest reaction time. However, the MOWO approach's efficiency did not improve. For allocating fog and cloud resources, the Energy Make span Multi-Objective Optimization approach was created in [8]. In fog–cloud paradigms, however, it failed to investigate meta-heuristics for solving the multi-objective optimization problem. [9] proposed a prospective fog node-based energy-efficient resource allocation (CF-EE) algorithm. In fog computing networks, however, the dynamic positions of IoT devices were not taken into account. [10] proposed an Opposition-based Chaotic Whale Optimization Algorithm (Oppo CWOA) to improve task scheduling performance. However, the suggested algorithm did not take into account multi-objective optimization.

1.1 Major contributions of the paper

A novel MCSSROSIDCLB approach is created with the following contribution to tackle the existing issues:

The MCSSROSIDCLB technique is based on Multi objective chaotic salp swarm optimization applied to a shift-invariant deep convolutional neural learning to increase load balancing efficiency in fog computing.

The load balancer examines the fog node's hidden layer resource availability, such as CPU, bandwidth, and memory. The load balancer uses the Multi objective chaotic salp swarm optimization to a deep learning model to locate the resource-optimized fog node based on the highest resource availability, as determined by the study.

The load balancer then sends the tasks to the fog nodes that are resource-optimized. This improves efficiency and reduces the time it takes to make a decision as well as the amount of memory used.

Finally, an exhaustive experimental evaluation using multiple performance indicators is carried out to demonstrate the superiority of the proposed MCSSROSIDCLB technique over traditional load balancing techniques.

1.2 Paper organisation

The remainder of this work is divided into the following sections. The linked work in Section 2 provides some existing studies in the area of fog computing load balancing. The suggested MCSSROSIDCLB approach is described in Section 3 with a clear architecture diagram. The experimental is described in Section 4 along with the dataset description. The performance results and comparison of the suggested technique and the standard load balancing technique are given in section 5. The conclusion is found in Section 6.

2. **WORKS IN CONNECTION**

[11] proposed an improved elitism-based genetic algorithm (IEGA) for job scheduling. The designed algorithm, however, was unable to handle the large-scale challenge. [12] created a technique for resource allocation and management (TRAM) to ensure resource use. However, the proposed strategy was unable to extend various hyper-heuristic and deep learning-based resource scheduling algorithms.

[13] proposed two distributed load balancing methods for resource management. However, there was no consistent performance in terms of low response time and low loss rate. In [14], a

new optimization technique was used to introduce cooperative three-layer fog-cloud computing. However, it was unable to implement large-scale real-time applications with minimal latency.

In [15], an improved firework algorithm (I-FA) for job scheduling was introduced. The suggested approach, however, was unable to reduce the task processing time. To balance the load of fog, [16] presented an Improved Particle Swarm Optimization with Levy Walk (IPSOLW). Despite the fact that the model reduces response and processing times, load balancing efficiency has not increased. [17] presented an energy-aware load balancing technique for low-cost scientific process scheduling. However, the nature-inspired load balancing alternatives were not implemented.

Load balancing is a term used to describe the process of balancing In [18], a technique for tackling task mapping with a shorter average reaction time was given. To increase load balancing efficiency, however, resource-aware scheduling was not used. To reduce latency and network use, a fog-based health monitoring system structural design was introduced in [19]. However, it was unable to increase the resource-optimized load balancing efficiency using the metaheuristic technique. When assigning requests, the Load Balanced Service Scheduling Approach (LBSSA) was introduced in [20]for load balancing amongst the resources. However, the strategy failed to adequately balance the workload while allocating resources efficiently.

3. **METHODOLOGY**

Fog computing has the potential to be a wonderful and widely used computing paradigm for a variety of IoT (Internet of Things) applications. The fog node has been widely used to analyse data collected from IoT edge devices in a fog computing environment. Fog computing's key uses are reducing latency and increasing resource utilisation for end-user queries coming from IoT devices. The more IoT devices there are, the more end-user demands there are and the more resources are used. Fog computing has a number of advantages, but it also has a number of drawbacks, such as resource optimal load balancing, which is a difficult problem to solve in a fog computing environment. Load balancing is a technique for enhancing throughput while reducing response time by optimising resource utilisation. As a result, an efficient algorithm is necessary to handle end-user requests in the shortest time possible. A unique MCSSROSIDCLB technique for distributed load balancing in a fog computing environment is introduced based on the motivation.

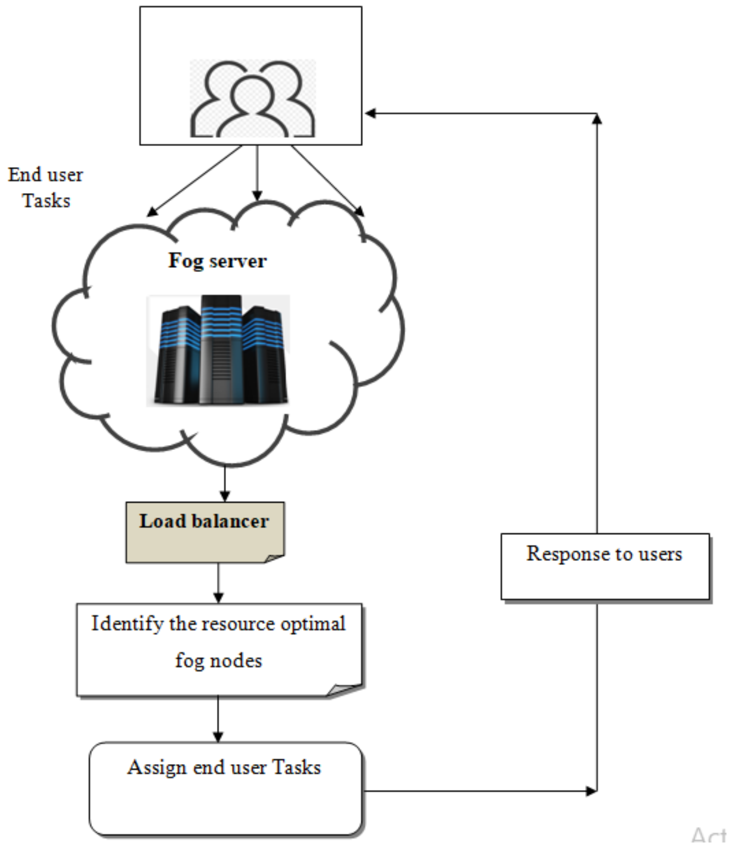

Figure 1 architecture of proposed MCSSROSIDCLB technique

The suggested MCSSROSIDCLB technique to schedule numerous end-user tasks (i.e. services) in the Fog computing network is depicted in Figure 1. Healthcare, industrial systems, smart city, smart transportation, and other real-time applications will be supported by the fog computing network. Multiple end users communicate their tasks to a fog server in fog computing. The load balancer in the fog server examines the end-user-requested tasks after receiving user requests. Following that, the load balancer locates the server's resource-optimized fog node for job scheduling. The technique's steps are outlined in the following sections.

3.1 Fog Computing System Network

Fog computing is better suited to IoT applications that include many fog services. All data extraction, data transmission, and service execution events from IoT applications are covered by fog services.

Let us consider the fog computing services (i.e task) requested by the IoT applications are denoted as $T = \{T_1, T_2, \dots, T_n\}$, where n is the number of tasks generated by the IoT applications and the number of fog computing nodes in the fog server are denoted by $F = \{F_1, F_2, \dots, F_m\}$. The load balancer at each fog node schedules the end-user tasks to balance the workloads. For each fog computing node in the fog server, their resource capacities are measured including CPU, memory, and bandwidth. By applying the binary variable, find resource-optimized fog computing nodes to assign the tasks.

$$Z = \begin{cases} 1; & \text{if } \arg\max \ \text{cpu}\ (F_i), Bw\ (F_i), M\ (F_i) \\ 0; & \text{otherwise} \end{cases} \quad (1)$$

In (1), Z is a binary variable that determines whether user requests are routed to fog server resource-optimized processing nodes. The request is assigned to the resource optimal node if the variable function returns '1'. Otherwise, '0' is returned.

To accomplish load balancing and resource utilisation in fog computing, the proposed MCSSROSIDCLB technique employs a shift-invariant convolutive deep learning technique. The shift-invariant convolutive deep learning architecture is made up of a series of processing layers, including input, hidden, and output layers. The neurons in one layer are totally coupled to the neurons in another layer, forming a network. The processed input from the previous layer is passed on to the next layer.

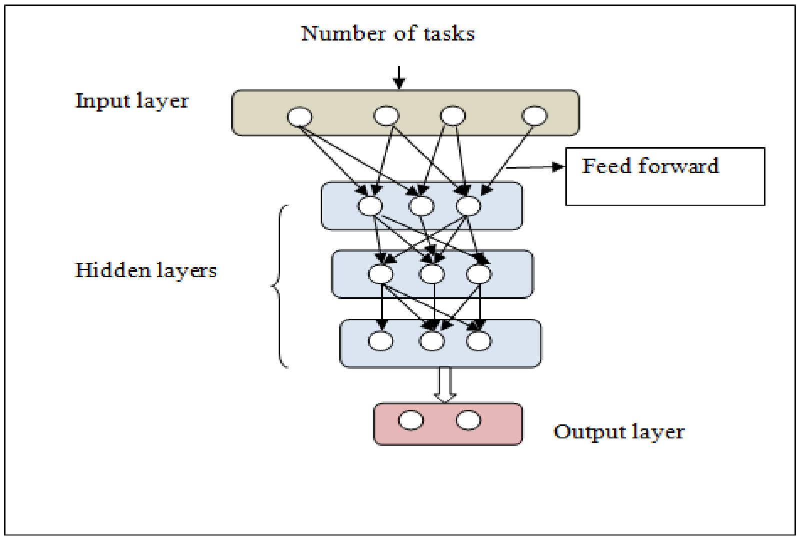

Figure 2 formation of the shift-invariant convolutive deep learning

The creation of a shift-invariant convolutive deep learning framework is shown in Figure 2. The convolutive deep learning system's shift-invariant system is shown below.

$$z_t = f\,[i(t)] \quad (2)$$

Where, z_t signifies the deep neural network's time-dependent output function, i(t) denotes the time-dependent input function, and f denotes the transfer function used to transfer the input from one layer to the next. The neuron's activity at the input layer is expressed as follows:

$$\delta(t) = \sum_{i=1}^{m} i_i(t) * \beta_0 + \alpha \quad (3)$$

Where, $i_i(t)$ specifies a neuron's activity at the input layer, β_0 denotes regulating weights, and means a bias that is stored as '1'. The received data is then transferred to the first hidden layer, which performs resource estimate.

The input layer considers the number of tasks $T = \{T_1, T_2, \ldots, T_n\}$ as input. . In the first hidden layer, the load balancer assesses the fog nodes' resource availability, such as CPU, $M(F_i)$, $Bw(F_i)$. The fog nodes' CPU time is computed as follows:

$cpu\ (F_i) = [\rho_{cpu_t} - \rho_{cpu_c}]$ (4)

Where, $cpu\ (F_i)$denotes CPU time available, $\rho_{cpu_t}))$indicates total time $\rho_{cpu_{cc}}]$ represents utilised time to complete the task.

The bandwidth availability of fog nodes is calculated as given below,

$Bw\ (F_i) = [\omega_{Bw\ (t)} - \omega_{Bw\ (c)}]$ (5)

Where, $Bw\ (F_i)$ implies bandwidth availability, $\omega_{Bw\ (t)}$ means total band width,, $\omega_{Bw\ (c)}$ denotes bandwidth consumption. The availability of memory is another important resource that aids in determining the amount of storage space required to complete the tasks. As a result, memory availability is determined as follows:

$M\ (F_i) = [\varphi_{M\ (t)} - \varphi_{M\ (c)}]$ (6)

Where, $M\ (F_i)$ represents the memory availability of the fog node, $\varphi_{M\ (t)}$symbolizes a total memory space of fog node and $\varphi_{M\ (c)}$denotes a consumed memory space of the fog node.

The estimated resources are transferred to a second hidden layer, where resource efficient nodes for assigning the provided input tasks are calculated. A multi objective chaotic salp swarm optimization algorithm is used to find the best node.

The multi objective chaotic salp swarm optimization algorithm is a population-based algorithm that starts by randomly initialising a predetermined number of individuals, such as the population of fog nodesF 1,F 2,...,F n. In the swarm of the salps optimizations, there are two categories of people: leaders and followers. The leader is the first salp in the chain, directing the movement of the followers. Multiple resources, such as CPU, memory, and bandwidth, are represented by the Multi objective function in this case. Set the population of fog nodes F 1, F 2,...,F n to zero. Multiple objective functions such as CPU availability, memory availability, and bandwidth availability are used to determine the fitness of each node in the server. The fitness level is determined as follows:

$ff = \arg\ \max\ [cpu\ (F_i), Bw\ (F_i), M\ (F_i)]$ (7)

Where arg max signifies an argument of a maximum function to calculate the fog node's maximum resource availability, and ff denotes fitness. The salp position p i (L) is determined at random.

$p_i(L) = R\ (N, D) * |U_b - L_b| + L_b$ (8)

Where p i (L) signifies a salp location, R denotes a random, N denotes the salp's population size, U b means the search space's upper bound, and L b denotes the search space's lower bound. Then, using the Gaussian chaotic map function, the leader position is updated.

$$p_{i+1}(L) = \begin{cases} F_{Si} + b_1\left(\exp\left(-\frac{1}{2} * \left(\frac{\|U_b - L_b\|}{\sigma}\right)^2\right) b_2 + L_b\right), b_3 \geq 0.5 \\ Fs_i - b_1\left(\exp\left(-\frac{1}{2} * \left(\frac{\|U_b - L_b\|}{\sigma}\right)^2\right)\right) b_2 + L_b), \; b_3 < 0.5 \end{cases} \quad (9)$$

Where p (i+1) (L) represents an updated location, Fs i represents a food source position, and U b and L b represent an upper and lower bound. The control parameters are b 1, b 2, and b 3, where b 2 and b 3 are random numbers in the range [0,1], the parameter b 2 controls the step size, the parameter b 3 controls the direction, and the parameter b 1 is calculated using the method below.

$$b_1 = 2e^{-\left(\frac{4Q}{Max-Iter}\right)^2} \quad (10)$$

Where Q stands for the current iteration and Max-Iter stands for the maximum iteration. As a result of being followed by, the position of the follower is updated as shown below.

$$p_{i+1}(F) = p_i(F) + D \quad (11)$$

Where, $D = \frac{1}{2}\,(p_{i-1}(F) - p_i(F))$ is substituted for the foregoing (11) as follows:

$$p_{i+1}(F) = \frac{1}{2}\left(p_i(F) + p_{i-1}(F)\right) \quad (12)$$

Where $p^{(i+1)}$ (F) signifies the follower's updated position, p i (F) denotes the follower's current location, and p (i-1) (F) denotes the preceding salp's movement. At the second hidden layer, the best solution is found in this fashion. The resource optimal fog nodes, on the other hand, have been identified. The Multiobjective chaotic salp swarm optimization flow process is shown below.

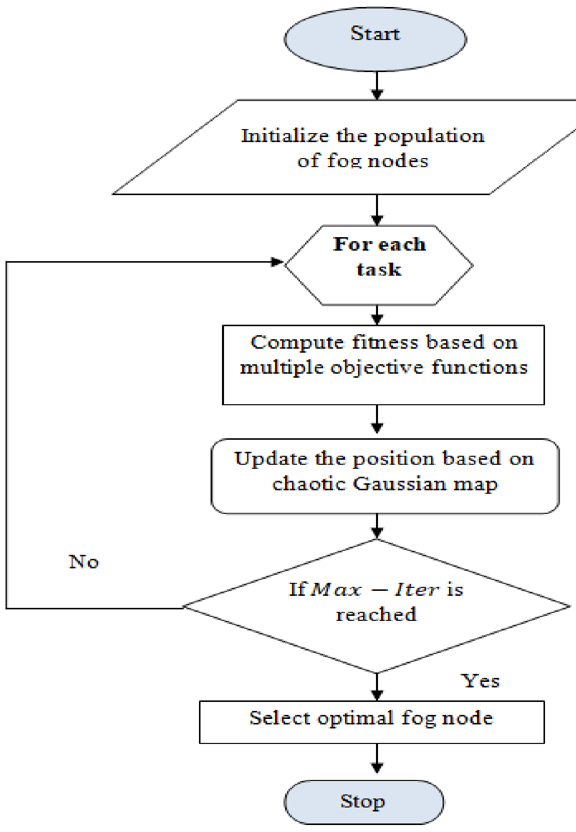
Start
Initialize the population of fog nodes
For each task
Compute fitness based on multiple objective functions
Update the position based on chaotic Gaussian map
If Max − Iter is reached
No
Yes
Select optimal fog node
Stop

Figure 3 flow process of multi objective chaotic salp swarm optimization algorithm.

Figure 3 depicts the flow of a multi objective chaotic salp swarm optimization algorithm for assigning jobs to discover resource-optimized fog nodes for effective load balancing. Using a binary variable, the load balancer sends incoming tasks to the appropriate fog nodes in the server for optimal resource usage in the third hidden layer (1). The output layer is transformed from the hidden layer's output.

$$Q(t) = \sum_{i=1}^{m} i_i(t) * \beta_0 + \beta_1 * q(t-1) \qquad (13)$$

From (13),$Q(t)$represents the output of the hidden layer at a time 't',β_0 denotes a weight between the input and hidden layer, β_1 denotes a weight of hidden layers, $i_i(t)$represents the input, $q(t-1)$ denotes the previous hidden layer's output. A convolution is indicated by the operator '$*$' .The shift-invariant convolutive deep learning output is written as follows:

$$Y(t) = \beta_2 * Q(t) \qquad (14)$$

Where, $Y(t)$) denotes the output, β_2 denotes a regulating weight between the hidden and an output layer, $Q(t)$ is the hidden layer's output. . As a result, in Fog computing, the optimization of dynamic load balancing with predictive resource allocation. The suggested MCSSROSIDCLB technique's algorithmic procedure is outlined as follows:

Algorithm 1: Multi objective choatic salp swarm resource optimized shift invariant deep convolutive load balancing

Input: Number of user tasks$T_1, T_2, T_3, \ldots. T_n$, fog nodes $F_1, F_2, \ldots, F_n$
Output: Improve the load balancing efficiency

```
Begin
    1. Collect a number of end-user tasks$T_1, T_2, T_3, \ldots. T_n$ at input layer
    2. Send user tasks $T_1, T_2, T_3, \ldots. T_n$to fog server
    3. The load balancer receives the tasks $T_1, T_2, T_3, \ldots. T_n$ ad find fog nodes -—first hidden layer
    4. LB finds the resource optimized fog node -—second hidden layer
\\ Apply multi objective chaotic salp swarm optimization technique
    5.   Initialize the population of fog nodes $F_1, F_2, \ldots, F_n$
    6.  for each fog node
    7.   Initialize the population of fog nodes $F_1, F_2, \ldots, F_n$
    8.  for each fog node
    9. calculate the fitness 'ff'
    10. While ($t <Max - Iter$)
    11.         for each fog node
    12.            If $\left(ff(F_i) > ff(F_j)\right)$ then
    13. Update the position of leader using $p_{i+1}(L)$
    14. else
    15. Update the position of follower $p_{i+1}(K)$
    16.         end if
    17. end for
    18. t= t+1
    19. end while
    20. end for
    21. Find a leader in population
    22. Obtain best optimal solution 'best salp' i.e. fog node
    23. LB schedule task to optimal fog node at the output layer
End
```

In a distributed fog computing environment, Algorithm 1 defines the step-by-step process of resource-optimized load balancing. The number of ends user-requested tasks is received by the input layer. The load balancer receives user-requested tasks in the first hidden layer and assesses the fog computing nodes in the server's resource availability. The load balancer then searches for resource-optimized fog nodes using a multi objective chaotic salp swarm optimization technique. In the search space, the number of salp swarm (fog node) populations and their placements are randomly initialised. Based on resource availability such as CPU, bandwidth, and memory, the proposed optimization technique evaluates the fitness of all fog nodes (salp swarm). When one individual salp's fitness exceeds that of another, the leader's position is changed. Otherwise, the leader's position is updated. Finally, the leader's (best salp) dimensions are updated. All of the preceding steps are repeated until the maximum number of iterations is reached. Finally, the load balancer delivers the incoming jobs to the fog node with the best resource allocation. As a result, the optimization method improves load balancing efficiency while lowering make span.

4. EXPERIMENTAL EVALUATION

The proposed MCSSROSIDCLB method and the existing EDRAM[1] LBOS [2] are compared using Java and the I Fog Sim simulator, which allows for the modelling and assessment of resource management and scheduling rules across edge device and cloud resources in a variety of situations. Fog computing is a layer that sits between the faraway cloud and the end user, and it boosts the network's overall performance. A Personal Cloud Datasets is collected from http://cloudspaces.eu/results/datasets in order to run the experiment. User interface structures, IoT services, resources, and network applications are all included in the I Fog Sim simulator. A number of user-requested edge layer jobs are delivered to the best fog nodes in an I Fog Sim simulator for task scheduling in a distributed fog environment. Based on resource optimization, the best fog nodes are chosen. There are 17 attributes (columns) and 66,245 instances in the Datasets (i.e., user-requested tasks). The dataset's main goal is to do load and transfer tests between the edge user and the server. Row id, account id, file size (task size), operation time start, operation time end, time zone, operation id, operation type, bandwidth trace, node ip, node name, quoto start, quoto end, quoto total (storage capacity), capped, failed, and failure details are among the 17 attributes. Two attributes, time zone and capped, are not used among the 17 properties; the remaining attributes are used for load balancing by allocating activities to multiple fog nods according on resource availability.

5. PERFORMANCE ANALYSIS

The proposed MCSSROSIDCLB approach and the existing EDRAM [1] LBOS [2] results and debate are detailed using various factors such as load balancing efficiency, make span, throughput, and reaction time. Tables and graphical representations are used to discuss the results of three strategies.

5.1 Load balancing efficiency performance analysis

The ratios of a number of end-user jobs dispersed to fog computing nodes to the total number of end-user tasks are used to calculate load balancing efficiency. The load balancing efficiency is expressed as follows:

$$\mathrm{Eff_{LB}} = \left[\frac{\mathrm{NTD}}{\mathrm{n}}\right] * 100 \quad (15)$$

From (15),$\mathrm{Eff_{LB}}$ denotes a load balancing efficiency and ‘n’ denotes the number of end user tasks. Load balancing efficiency is measured in terms of percentage (%).

Table I Load balancing efficiency

Number of tasks	Load balancing efficiency (%)		
	MCSSROSIDCLB	**EDRAM**	**LBOS**
5000	90	88	86
10000	92	89	85
15000	91	88	84
20000	92	87	85
25000	94	88	86
30000	93	90	88
35000	92	89	86
40000	93	90	88
45000	92	88	86
50000	91	87	85

Table I shows the load balancing efficiency as a function of the number of users' requested tasks generated by various IoT apps, with numbers ranging from 10000 to 50000. With varied counts of input tasks created from the dataset, ten distinct results are seen for each technique. Compared to the other two current approaches, the suggested MCSSROSIDCLB delivers greater performance in terms of increased load balancing. Consider 1000 activities generated by an IoT user, 90 of which are successfully scheduled to a resource-efficient virtual machine, and the MCSSROSIDCLB approach is 90% efficient. Using EDRAM [1] LBOS [2,] 88 and 86 tasks are accurately scheduled to resource-efficient fog nods, resulting in an efficiency of 88 and 86 percent, respectively. Similarly, three load balancing approaches yielded the remaining efficiency results. The MCSSROSIDCLB technique's observed outcomes are compared to the performance of existing approaches. The average is then used to compare the performance of the MCSSROSIDCLB methodology to other methods. The total comparison results show that the MCSSROSIDCLB approach improves load balancing efficiency by 4% and 7%, respectively, when compared to [1] and [2].

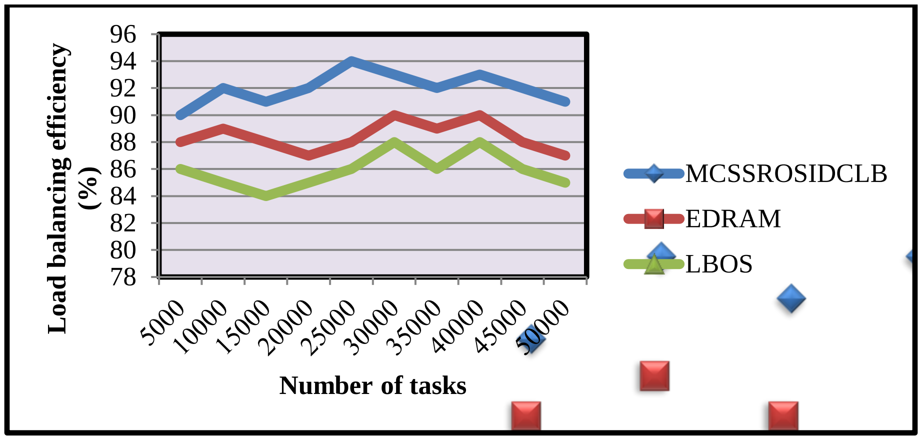

Figure 5 Load balancing efficiency with varying number of tasks

Figure 5 shows a graphical illustration of load balancing efficiency utilising three different methods: MCSSROSIDCLB, EDRAM [1], and LBOS [2.] The load balancing efficiency of the three different strategies is shown by three different colours of lines, blue, red, and green, respectively, in the graphical depiction. The number of users is entered on the 'x' axis, while efficiency is measured on the 'y' axis, as seen in the graphical display. The plot shows that the MCSSROSIDCLB approach outperforms the others in terms of efficiency gains. This is because the resource-optimized fog nodes were identified using Multi objective chaotic salp swarm optimization in a shift-invariant deep convolutional neural learning. The load balancer routes incoming jobs to a fog node with the most available resources. This method improves the efficiency of fog computing load balancing.

5.1 Performance analysis of Makes pan

It is calculated as the time it takes to assign jobs to fog nodes that are resource-optimized. The formula for the Makes pan is as follows:

$$\mathrm{Ms} = \mathrm{n} * \mathrm{t}\,[\mathrm{SOT}] \quad (16)$$

Where Ms represents a makes pan, n represents the number of end-user tasks, and t represents the time for scheduling one task ('SOT'). The makes pan is expressed in milliseconds (ms).

Table II Makes pan

Number of tasks	**Makes pan** (ms)		
	MCSSROSIDCLB	**EDRAM**	**LBOS**
5000	23	25	30
10000	28	32	35
15000	33	38	42
20000	38	42	46
25000	43	45	50
30000	45	48	53
35000	49	53	56
40000	52	56	60
45000	54	58	63
50000	58	60	65

Table II shows the results of makes pan's performance utilising three distinct strategies. With respect to a number of tasks ranging from 5000 to 50000, MCSSROSIDCLB, EDRAM [1] LBOS [2] were used. The results show that when compared to the other two ways, the performance of makes pan utilising the MCSSROSIDCLB methodology is significantly worse. The sample computation demonstrates this. Similarly, with varying numbers of input, different results are obtained for each method. For each procedure, ten results are observed. The MCSSROSIDCLB technique's performance is then compared to that of existing approaches. The performance of makes pan is significantly lowered by 8% when compared to [1] and by 16% when compared to [2], according to the average of ten comparison results.

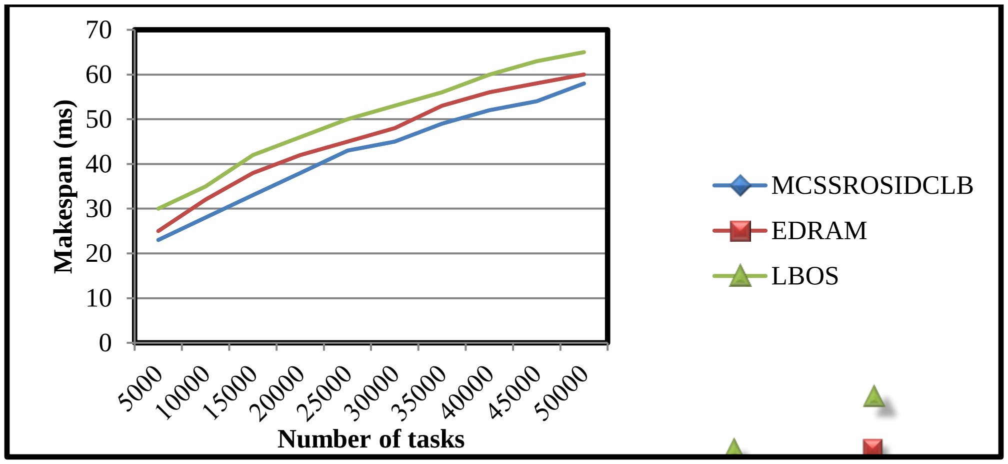

Figure 6 Makes pan with varying number of tasks

Figure 6 shows the makes pan experimental result for a variety of user-requested tasks. The number of tasks completed in the range of 5000 to 50000 is taken into account. When demonstrated in

Figure 6, as the number of user tasks grows, the time consumption of all methods increases. The MCSSROSIDCLB method, on the other hand, reduces the performance of makes pan. This is because the MCSSROSIDCLB technique employs deep learning to examine the fog node's resource availability and pick the best fog node in the shortest amount of time for accurately assigning numerous tasks. The average makes pan time for task scheduling in fog is reduced as a result of this.

5.2Performance analysis of memory consumption

It is measured as the amount of memory space consumed to store the multiple tasks. The formula for calculating the overall memory space is expressed as given below,

$$\mathrm{Con}_{\mathrm{Mem}} = \mathrm{n} * \ \mathrm{M}\ (\text{storing one task}) \quad (17)$$

Where,$\mathrm{Con}_{\mathrm{Mem}}$ denotes a memory consumption, n represents the number of tasks and M denotes memory consumption. The memory consumption is measured in terms of megabytes (MB).

Number of tasks	**Memory consumption** (MB)		
	MCSSROSIDCLB	EDRAM	LBOS
5000	**28**	**31**	**35**
10000	**33**	**36**	**39**
15000	**36**	**42**	**45**
20000	**40**	**44**	**48**
25000	**45**	**50**	**55**
30000	**48**	**54**	**60**
35000	**53**	**56**	**63**
40000	**56**	**60**	**64**
45000	**61**	**65**	**70**
50000	**67**	**70**	**75**

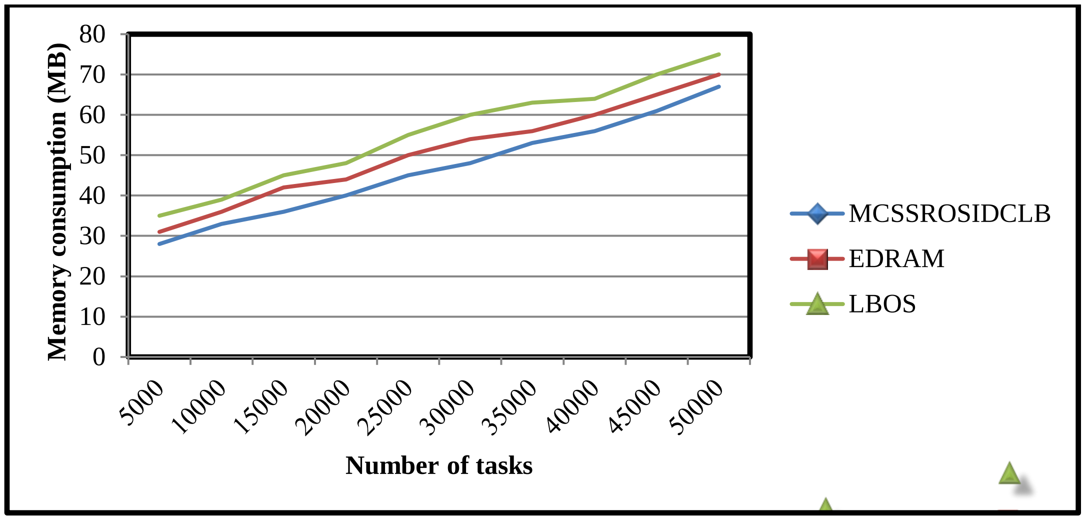

Figure 7Memory consumption with varying number of tasks

Table III and Figure 7 show the memory consumption findings for scheduling multiple jobs using three different techniques: MCSSROSIDCLB, EDRAM [1] and LBOS [2]. While raising the number of tasks for each iteration, the memory usage for all methods increases. The performance of memory usage utilising the MCSSROSIDCLB methodology is significantly better than the other two ways. In the first iteration, we'll use the 5000 tasks to calculate memory use. To begin with, the MCSSROSIDCLB method used 28MB of memory to schedule the 5000 tasks. Furthermore, EDRAM [1] and LBOS [2] consume 31MB and 35MB of memory, respectively. As a result, for each technique with a variable number of inputs, different results are observed. When comparing the three strategies, the MCSSROSIDCLB technique reduces memory consumption by 8% when compared to [1] and by 16% when compared to [2] while distributing numerous workloads into the fog node.

6. CONCLUSION

In Fog computing, load balancing is the process of spreading computational jobs over a group of resources, reducing response time while maintaining high-quality computation. In fog computing settings, load balancing is critical, and it must schedule user jobs to the available fog nodes in order to reduce the makes pan. A unique technique called MCSSROSIDCLB is developed in this research to improve load balancing efficiency. For deep resource allocation and Multi objective chaotic salp swarm resource optimization based task scheduling in fog, the proposed MCSSROSIDCLB approach leverages shift-invariant deep convolution neural learning. The load management then uses the Multi objective chaotic salp swarm optimization technique to find the server's resource-optimized fog node, which improves scheduling performance over a single objective function. Different performance criteria, such as load balancing efficiency, makes pan, and memory usage, are used to evaluate the MCSSROSIDCLB technique's performance. As a consequence of the quantitative results and discussion, the given MCSSROSIDCLB technique appears to be highly promising in terms of providing higher load balancing efficiency with less makes pan and memory consumption than traditional load balancing systems.

REFERENCES

[1] D. Baburao, T. Pavan kumar & C. S. R. Prabhu, "Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method", Applied Nanoscience, Springer, 2021, Pages 1-10

[2]Fatma M. Talaat, Mohamed S. Saraya, Ahmed I. Saleh, Hesham A. Ali & Shereen H. Ali, "A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment", Journal of Ambient Intelligence and Humanized Computing, Springer, Volume 11, 2020, Pages4951–4966

[3] Simar Preet Singh, Rajesh Kumar, Anju Sharma, Anand Nayyar, "Leveraging energy-efficient load balancing algorithms in fog computing", Concurrency Computation and Practice Experience, Wiley, 2020, Pages 1-16

[4]Anees Ur Rehman, Zulfiqar Ahmad, Ali Imran Jehangiri, Mohammed Alaa Ala'anzy, Mohamed Othman, Arif Iqbal Umar, And Jamil Ahmad, "Dynamic Energy Efficient Resource Allocation Strategy for Load Balancing inFog Environment", IEEE Access , Volume 8, 2020, Pages 199829 - 199839

[5]K. Hemant Kumar Reddy, Ashish Kr. Luhach, Buddhadeb Pradhan, Jatindra Kumar Dash, Diptendu Sinha Roy, "A genetic algorithm for energy efficient fog layer resource management in context-aware smart cities", Sustainable Cities and Society, Elsevier, Volume 63, 2020, Pages 1-10

[6] Mohamed K. Hussein and Mohamed H. Mousa, "Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization", IEEE Access, Volume 8, 2020, Pages 37191 – 37201

[7]Vincenzo De Maio and Dragi Kimovski, "Multi-objective scheduling of extreme data scientific workflows in Fog", Future Generation Computer Systems, Elsevier, Volume106,2020, Pages 171–184

[8]Samia Ijaz, Ehsan Ullah Munir, Saima Gulzar Ahmad, M. Mustafa Rafique & Omer F. Rana, "Energy-makespan optimization of workflow scheduling in fog–cloud computing", Computing, Springer,Volume 103, 2021, Pages2033–2059

[9] Xiaoge Huang, Weiwei Fan, Qianbin Chen, Jie Zhang, "Energy-Efficient Resource Allocation in Fog Computing Networks With the Candidate Mechanism", IEEE Internet of Things Journal ,Volume7, Issue 9, 2020, Pages 8502 - 8512

[10]Zahra Movahedi, Bruno Defude & Amir mohammad Hosseininia, "An efficient population-based multi-objective task scheduling approach in fog computing systems", Journal of Cloud Computing, Springer, Volume 10, 2021, Pages 1-31

[11]Mohamed Abdel-Basset, Reda Mohamed, Ripon K. Chakrabortty, Michael J. Ryan, "IEGA: An improved elitism-based genetic algorithm for task scheduling problem in fog computing", International Journal of Intelligent System, 2021, Pages 1–40

[12]Heena Wadhwa & Rajni Aron, “TRAM: Technique for resource allocation and management in fog computing environment”, The Journal of Supercomputing, Springer, 2021, Pages 1-24
[13] Roberto Beraldi , Claudia Canali , Riccardo Lancellotti , Gabriele Proietti Mattia, “Distributed load balancing for heterogeneous fog computing infrastructures in smart cities”, Pervasive and Mobile Computing, Elsevier, Volume 67, 2020, Pages 1-16
[14]Mirza Mohd Shahriar Maswood, MD. Rahinur Rahman, Abdullah G. Alharbi, Deep Medhi, “A Novel Strategy to Achieve Bandwidth Cost Reduction and Load Balancing in a Cooperative Three-Layer Fog-Cloud Computing Environment”, IEEE Access, Volume 8, 2020, Pages 113737 – 113750
[15]Shudong Wang, Tianyu Zhao, Shanchen Pang, “Task Scheduling Algorithm Based on Improved Firework Algorithm in Fog Computing”, IEEE Access,Volume8, 2020, Pages 32385 - 32394
[16]Zahoor Ali Khan, Ayesha Anjum Butt, Turki Ali Alghamdi, Aisha Fatima, Mariam Akbar, Muhammad Ramzan, “Energy Management in Smart Sectors Using Fog Based Environment and Meta-Heuristic Algorithms”, IEEE Access , Volume 7,2019, Pages 157254 – 157267
[17]Mandeep Kaur and, Rajni Aron, “Energy-aware load balancing in fog cloud computing”, Materials Today: Proceedings, Elsevier, 2020, Pages 1-5
[18]Anil Singh Nitin Auluck, “Load balancing aware scheduling algorithms for fog networks”, Software: Practice and Experience, Wiley, Volume50, Issue11, 2019, Pages 1-19.
[19]Anam Asghar; Assad Abbas; Hasan Ali Khattak; Samee U. Khan, “Fog Based Architecture and Load Balancing Methodology for Health Monitoring Systems”, IEEE Access ,Volume 9, 2021, Pages 96189 – 96200
[20]Fayez Alqahtani, Mohammed Amoon & Aida A. Nasr, “Reliable scheduling and load balancing for requestsin cloud-fog computing”, Peer-to-Peer Networking and Applications, Springer**,** Volume 14, Issue 5, 2021, Pages 1-12