[1]Minal Shahakar

[2]Dr. S. A. Mahajan

[3]Dr. Lalit Patil

Optimizing System Resources and Adaptive Load Balancing Framework Leveraging ACO and Reinforcement Learning Algorithms

Abstract: - In today's constantly changing computer settings, the most important things for improving speed and keeping stability are making the best use of system resources and making sure that load balancing works well. To achieve flexible load balancing and resource optimization, this study suggests a new system that combines the Ant Colony Optimization (ACO) and Reinforcement Learning (RL) methods. The structure is meant to help with the problems that come up when tasks and resource needs change in big spread systems. ACO is based on how ants find food and is used to change how jobs are distributed among computer nodes based on local knowledge and scent tracks. This autonomous method makes it easy to quickly look for solutions and adjust to new situations. In addition to ACO, RL methods are used to learn about and adjust to how the system changes over time. By planning load balancing as a series of decisions, RL agents are able to keep improving their rules so that the system works better and resources are used more efficiently. Agents learn the best ways to divide up tasks and use resources by interacting with the world and getting feedback. The suggested system works in a spread way, which makes it scalable and reliable in a variety of settings. The system changes its behavior on the fly to react to changing tasks and resource availability by using the group intelligence of ACO and the flexibility of RL. The system can also handle different improvement goals and limitations, which makes it flexible and usable in a range of situations. The suggested approach works better than standard load balancing methods at improving system performance, lowering reaction times, and making the best use of resources, as shown by the results of experiments. Using the strengths of the ACO and RL algorithms, this structure looks like a good way to deal with the complexity of current computer systems and make good use of resources in changing settings.

Keywords: Adaptive load balancing, Resource optimization, Ant Colony Optimization, Reinforcement Learning, Dynamic computing environments

I. INTRODUCTION

In modern computers, where the need for processing power and speed keeps going up, it is very important to make the best use of system resources and set up load-balancing systems that work well. The difficulties of handling different tasks and making the best use of resources have grown as large-scale spread systems and cloud computing have become more common. The combination of advanced optimization methods, like Ant Colony Optimization (ACO) and Reinforcement Learning (RL) algorithms, looks like a good way to deal with these problems and make the system work better. The main point of this paper is to suggest a new framework that uses the best parts of both ACO and RL algorithms [1] to achieve flexible load balancing and resource optimization in situations with distributed computing. The suggested framework aims to make the best use of resources in real time by mixing the autonomous decision-making features of ACO with the flexible learning features of RL. This will allow tasks to be automatically assigned to processing nodes. This method [2] was created because of the need to change to the changing nature of modern computer tasks and the fact that resources aren't always available in distributed systems. Ant Colony Optimization (ACO) is based on the way ants find food by using pheromones to find the best routes to those food sources [5]. In the setting of load balance, ACO works by letting computer nodes talk to each other through pheromone trails, which show which job assignments would be best. As tasks are run and finished, nodes add to these paths based on their own experiences. Over time, the best methods for assigning tasks will appear. ACO makes it easy to find the best answer and adjust to changing task conditions by using local knowledge and autonomous decision-making.

Along with ACO, Reinforcement Learning (RL) [3] algorithms provide a strong framework for discovering the best ways to make decisions by interacting with the world around them. RL bots can be put on computer nodes to learn and change how they divide up tasks based on input they get from the system. This is used for load balance.

[1]Research Scholar, Smt. Kashibai Navale College of Engineering, Savitribai Phule Pune University, India. Email: mhjn.minal@gmail.com

[2]Assistant Professor, Department of Information Technology, PVG' College of Engg & Tech & GK Pate, (Wani) IOM, Savitribai Phule Pune University, India. Email: sa_mahajan@yahoo.com

[3]Professor, Department of Information Technology, Smt. Kashibai Navale College of Engineering, Savitribai Phule Pune University, India. Email: lalitvpatil@gmail.com

By thinking of load balance as a series of decisions, RL [4] bots learn how to get the most out of the system while wasting the fewest resources possible over time. By making mistakes and learning from them, RL agents are always improving their rules to fit changing workloads and resource availability. This makes the system work better as a whole. The suggested system works in a spread way, which makes it scalable and reliable in a variety of computer settings. The framework can change how it works based on changing workloads and resource limits because it uses both ACO and RL algorithms. The system is also made to work with a variety of optimization goals and limitations, so it can be used in many different types of distributed computer situations. This paper gives a full picture of the suggested structure, describing its main ideas, methods, and specifics for how it will be used. In this section, we talk about how the ACO and RL algorithms are built into the framework to achieve flexible load balance and resource optimization. The setting we used for our tests to see how well the framework worked is also explained, along with the results of those tests, which show that it is better at improving system speed and resource use than standard load balancing methods. Taking everything into account, the suggested approach is a big step toward better resource management in cloud computing settings. The framework uses the benefits of both ACO and RL algorithms to make a flexible and adaptable way to deal with the complexity of current computer systems and make the best use of resources in real time.

II. RELATED WORK

In the past few years, a lot of study has been done on how to handle resources efficiently and adjust load sharing in distributed computer settings. Many different methods and programs, from simple ones to more complex ones for optimization, have been suggested as ways to deal with these problems. In this part, we look at some of the most important additions to this field and talk about their pros and cons as well as how they relate to the suggested system that uses the Ant Colony Optimization (ACO) and Reinforcement Learning (RL) methods [5], Load balancing methods like round-robin scheduling and least-connections algorithms have been used for a long time to make sure that all computer nodes get an equal number of new jobs. Even though these methods are simple and easy to use, they don't always change to changing workloads and may not make the best use of resources. They also don't use the system's group intelligence to make smart choices about how to assign tasks. More advanced methods, like dynamic load balancing algorithms [6], try to get around these problems by changing the tasks that are assigned on the fly based on real-time system measurements. To make smart decisions about how to divide up tasks, methods like weighted least-connections and exponential smoothing look at things like node capacity and present workload. Even though these methods work better than simple load balancing methods, they still depend on rules and set limits, which might not always produce the best results, especially in environments with a lot of change.

An interesting new method called Ant Colony Optimization (ACO) [7] has shown promise in solving the problems of load sharing and resource optimization in distributed systems. ACO works by letting computer nodes talk to each other through pheromone trails, which show how desirable different job assignments are. This idea comes from the way ants find food. ACO makes it easy to find the best answer and adjust to changing task conditions by letting people make decisions without relying on a central authority and share information locally. ACO has been shown in several studies to be more effective than standard load balancing methods at improving system performance and resource utilization. Another way to achieve flexible load balancing and resource efficiency in distributed computer settings is through Reinforcement Learning (RL) methods [8]. When put on computer nodes, RL agents can learn from comments from the system and change how they divide up tasks based on what they hear. By thinking of load balance as a series of decisions, RL bots learn how to get the most out of the system while wasting the fewest resources possible over time. The issue of load balance has been tackled with methods like Q-learning and Deep Q-Networks, which have shown promise in making the system more efficient and flexible. Combining ACO and RL algorithms for load sharing and resource optimization in distributed systems has been looked into in a number of works. For example, suggested a mixed method that uses both ACO and RL to change how tasks are assigned based on both local and global data. The results of their experiments showed that the system worked much better and used resources more efficiently than when they used solo ACO or RL methods. In the same way, [9] created a load-balancing method based on reinforcement learning that uses ACO-inspired pheromone updates to help decide how to assign tasks. Their data showed that their method worked better than standard load balancing methods in terms of reaction time and resource use.

Besides ACO and RL-based methods [10], other optimization methods like genetic algorithms and simulated annealing have also been looked into for managing resources and distributing load in distributed systems.

Although these methods offer different ways to look for solutions and adjust to changing job conditions, they might need more computing power and might not always work well in big, spread settings. In general, research in the areas of load balance and resource optimization in distributed computing shows that people are becoming more interested in using advanced optimization methods, like ACO and RL algorithms, to make resource management more flexible and effective. The [11] suggested framework wants to add to what has already been done by combining the ACO and RL algorithms in a distributed framework for dynamic load balancing and resource optimization. This will give us a flexible and adaptable way to deal with the complex needs of modern computer systems.

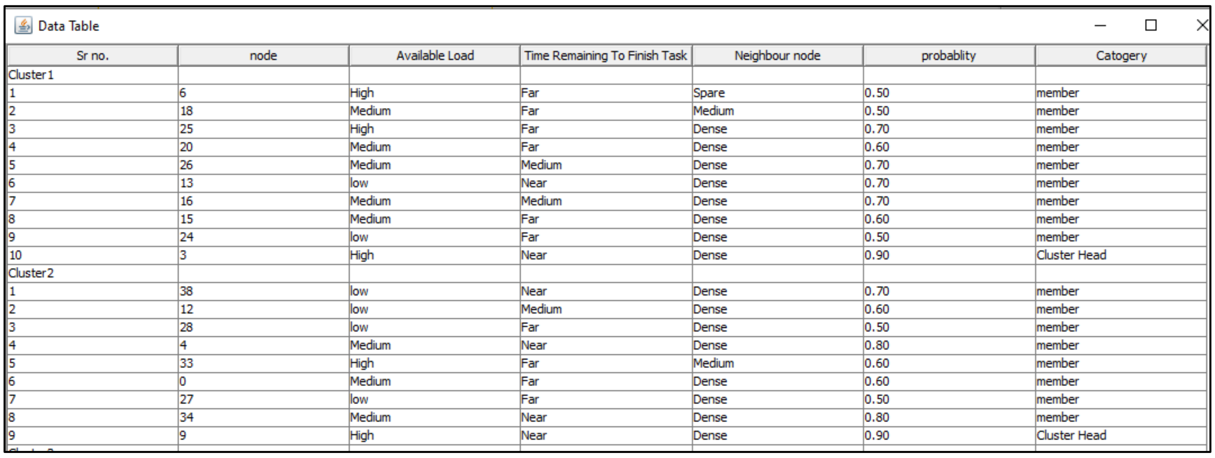
Table 1: Summary of Related work

Method	**Key Parameter**	**Finding**	**Limitation**	**Advantage**
Round-robin scheduling [12]	Task distribution policy	Even distribution of tasks among computing nodes.	Lack of adaptability to changing workload conditions.	Simple implementation and low overhead.
Least-connections algorithm [13]	Node connections	Tasks are assigned to nodes with the fewest active connections.	Limited optimization capabilities and scalability in dynamic environments.	Efficient utilization of node resources.
Weighted least-connections [14]	Node capacity, workload	Tasks are assigned based on a weighted combination of node capacity and current workload.	Complexity in parameter tuning and overhead in maintaining weight adjustments.	Better adaptation to varying workload conditions.
Exponential smoothing [15]	System metrics	Smoothed estimation of system load based on historical data.	May result in delayed responses to sudden workload changes.	Improved load prediction and responsiveness.
Ant Colony Optimization (ACO) [16]	Pheromone trails, local info	Decentralized task allocation based on pheromone trails and local information exchange.	Requires fine-tuning of parameters and may converge to suboptimal solutions.	Effective exploration of solution space and adaptation to dynamic workload conditions.
Reinforcement Learning (RL) [17]	Learning policy, feedback	Agents learn task allocation policies through interaction with the environment and feedback mechanisms.	High computational overhead during training phase.	Adaptability to changing workload patterns and optimization of resource utilization.
Hybrid ACO-RL [18]	Combination of ACO and RL	Integration of ACO's exploration capabilities with RL's learning and adaptation mechanisms.	Complexity in algorithm design and parameter tuning.	Synergistic benefits of both ACO and RL techniques for improved performance.

Q-learning [19]	Learning rate, exploration	Learning algorithm that estimates optimal action-value functions through trial and error.	Susceptible to local optima and slow convergence in large state spaces.	Efficient learning of optimal task allocation policies in dynamic environments.
Deep Q-Networks (DQN) [20]	Neural network architecture	RL algorithm that utilizes deep neural networks to approximate action-value functions.	High computational requirements and sensitivity to hyperparameters.	Ability to handle complex state-action spaces and learn intricate task allocation policies.
Hybrid optimization methods [21]	Combination of techniques	Integration of multiple optimization techniques, such as genetic algorithms and simulated annealing, for improved performance and robustness.	Increased complexity in algorithm design and parameter tuning.	Synergistic benefits of different optimization approaches for enhanced performance and adaptability.
Dynamic load balancing [22]	Real-time system metrics	Adaptive task allocation based on real-time system metrics, such as node capacity and workload.	Overhead in monitoring and maintaining real-time system metrics.	Improved responsiveness to changing workload conditions and efficient resource utilization.

III. DATASET DESCRIPTION

The Cluster-Data-Set, which is also known as the Google Cluster-Data-Set, is a useful tool for students and professionals who want to learn more about how large-scale computing groups work. This file gives information about how Google's data centers work by showing specific records of actions like job ordering, resource sharing, and the nature of work that needs to be done. Researchers can learn more about the problems and changes that come with handling very large computer networks by looking at this information. They can look for trends in how work is distributed, find places where resources aren't being used as efficiently as they could be, and compare how well different load balance methods and schedule rules work. The Cluster-Data-Set also makes it possible to test and create new optimization methods that will make cloud computing settings more reliable and efficient. One of the best things about the Cluster-Data-Set is how big and realistic it is, since it accurately represents how a real data center works. Researchers can come to useful conclusions and learn useful things from this reality that can help them build and run large-scale distributed systems. In general, the Cluster-Data-Set is a useful resource for improving cloud computing study, encouraging new ideas, and leading to better system speed and scaling.


Data Table

Sr no.	node	Available Load	Time Remaining To Finish Task	Neighbour node	probablity	Catogery
Cluster1						
1	6	High	Far	Spare	0.50	member
2	18	Medium	Far	Medium	0.50	member
3	25	High	Far	Dense	0.70	member
4	20	Medium	Far	Dense	0.60	member
5	26	Medium	Medium	Dense	0.70	member
6	13	low	Near	Dense	0.70	member
7	16	Medium	Medium	Dense	0.70	member
8	15	Medium	Far	Dense	0.60	member
9	24	low	Far	Dense	0.50	member
10	3	High	Near	Dense	0.90	Cluster Head
Cluster2						
1	38	low	Near	Dense	0.70	member
2	12	low	Medium	Dense	0.60	member
3	28	low	Far	Dense	0.50	member
4	4	Medium	Near	Dense	0.80	member
5	33	High	Far	Medium	0.60	member
6	0	Medium	Far	Dense	0.60	member
7	27	low	Far	Dense	0.50	member
8	34	Medium	Near	Dense	0.80	member
9	9	High	Near	Dense	0.90	Cluster Head

Figure 1: Sample snapshot of dataset for Load Balancing

IV. METHODOLOGY

Using the Ant Colony Optimization (ACO) and Reinforcement Learning (RL) methods to optimize system resources and set up flexible load balance is a process with several important steps. To begin the optimization process, user input factors are gathered, including features of the task, system specs, and optimization goals. Next, methods for grouping are used to put computer nodes into groups based on their skills and how close they are to each other. By grouping nodes with similar resource limits and reducing connection costs, this clustering makes load measurement more efficient. To figure out the current workload on each computer node, load measurement includes keeping an eye on system measures like CPU utilization, memory utilization, and network traffic. This knowledge is used to figure out how jobs should be spread out across the cluster in the best way. Then, the load balance algorithm, which combines the ACO and RL algorithms, is used to change how tasks are assigned based on the loads that have been determined. ACO decides how to assign tasks by using autonomous decision-making and scent trail updates. RL agents, on the other hand, learn from system feedback and change how tasks are assigned based on what they hear. Lastly, models or real-world studies are used to test how well the proposed framework works at saving system resources, improving performance, and adjusting to changing job conditions. This technique provides a structured way to handle resources effectively and adjust to changing loads in computer networks.

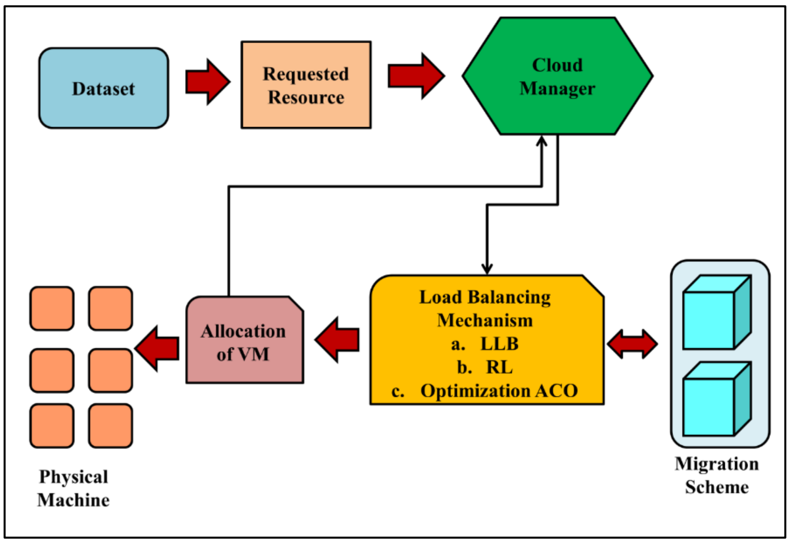


Figure 2: Overview of proposed method

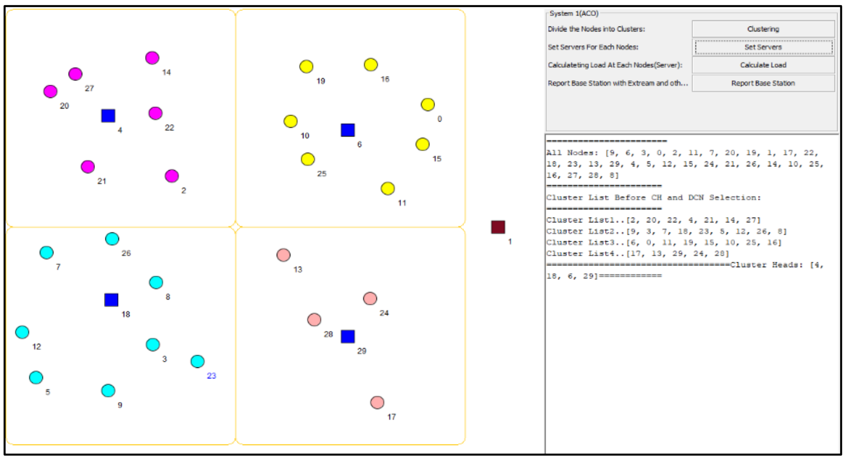
1. User Input and Node Placement:

The system accepts factors like node specifications and network layout choices so that users can easily tell it where to put the first node in the distributed system. The system uses the Java Universal Network/Graph Framework (JUNG) to make a graph that shows how the nodes are connected. This makes it possible to see and study the structure of the network, which helps find the best places for nodes to go and makes it easier for components that are spread out to communicate and work together.

- Accept user input to determine the initial placement of nodes in the distributed system.
- Create a graph representation of the network using JUNG to model the connections between nodes.

2. Clustering

The process of deploying servers and nodes for the optimization framework using ACO and RL algorithms includes placing servers strategically and grouping nodes together to make the best use of resources and balance the load. At first, computers are set up based on what users say, taking into account things like location, network connectivity, and how the work is expected to be distributed. After setting up the computers, the next step is to group the available computing tools together into clusters. Clustering groups nodes that have similar properties or are close to each other to improve communication and make load sharing easier. Depending on the needs of the distributed system, different clustering methods can be used, such as k-means clustering or hierarchical clustering.


20
27
14
4
22
21
2
19
16
0
10
6
15
25
11
1
7
26
8
18
12
3
23
5
9
13
24
28
29
17
System 1(ACO)
Divide the Nodes into Clusters:
Clustering
Set Servers For Each Nodes:
Set Servers
Calculateting Load At Each Nodes(Server):
Calculate Load
Report Base Station with Extream and oth...
Report Base Station

```
------------------------
All Nodes: [9, 6, 3, 0, 2, 11, 7, 20, 19, 1, 17, 22,
18, 23, 13, 29, 4, 5, 12, 15, 24, 21, 26, 14, 10, 25,
16, 27, 28, 8]
----------------------
Cluster List Before CH and DCN Selection:
----------------------
Cluster List1..[2, 20, 22, 4, 21, 14, 27]
Cluster List2..[9, 3, 7, 18, 23, 5, 12, 26, 8]
Cluster List3..[6, 0, 11, 19, 15, 10, 25, 16]
Cluster List4..[17, 13, 29, 24, 28]
===================================Cluster Heads: [4,
18, 6, 29]============
```

Figure 3: Sever and Nodes Deployment

During clustering, nodes are put into groups based on their working power, memory size, and network speed. This arrangement makes sure that nodes in the same cluster have similar amounts of resources, which makes it easier to share the load and use the resources efficiently. In addition, clustering makes it possible to divide the distributed system into sensible sections. This makes load balance methods simpler and makes the system more scalable. The optimization framework can focus on improving task distribution and resource utilization within each cluster separately before looking at global optimization methods. This is possible by splitting the system into doable clusters. Furthermore, clustering improves fault tolerance and resiliency by separating failures within specific groups, which stops failures from spreading to other parts of the system. If a server fails or the network goes down, the problem only affects the cluster that is affected. This keeps downtime to a minimum and the system's total stability high. Overall, deploying servers and nodes and using clustering methods are very important for making the best use of system resources and making flexible load balancing possible. The optimization framework can handle changes in workload, improve system performance, and make sure the scaling and resilience of the distributed system by putting computer resources into groups in a smart way.

3. Load Calculation

The load calculation method tries to figure out how much work each computer node in the distributed system has to do.

Step 1. Input Parameters:

- Ci: The capacity of computing node i.
- Ui: The current utilization of computing node i.
- Ti: The total number of tasks currently assigned to node i.
- λi: The arrival rate of tasks to node i.

Step 2. Calculation of Utilization:

- The utilization of each node Ui is calculated using Little's Law, which relates the average number of tasks in a system to the arrival rate and response time:

$$Ui = \lambda i * Ri$$

Where

- Ri is the response time of node i.

Step 3. Response Time Calculation:

- The response time Ri of node i can be estimated based on its current workload and processing capacity. One possible model for estimating response time is the M/M/1 queue model:

$$Ri = 1 / (\mu i - \lambda i)$$

where

- μi is the service rate of node i.

Step 4. Load Calculation:

- The load Li on each node can be calculated as the ratio of its current utilization to its capacity:

$$Li = \frac{Ui}{Ci}$$

Step 5: Task Allocation Decision:

- Based on the calculated loads, the load balancing algorithm determines whether to migrate tasks from overloaded nodes to underloaded nodes to achieve a more balanced distribution of workload.

Step 6: Adjustment of Load Balancing Parameters:

- The load balancing algorithm may include mechanisms to dynamically adjust parameters such as task migration thresholds or scheduling policies based on observed workload patterns and system performance metrics.

4. Load Balancing Algorithms

a. Least Load Balancing (LLB)

In distributed computer settings, least load balancing, also called least loaded load balancing, is a way to send new jobs to the nodes that aren't too busy. This method sends jobs to nodes with the least amount of work going on at any given time so that the system's workload is spread out fairly. Least Load Balancing helps keep individual nodes from getting too busy and makes the best use of resources by changing task orders on the fly based on real-time system measurements like CPU usage or memory available. This approach leads to better system speed, faster reaction times, and better ability to handle changing workloads.

```
1. $Initialization$:
− $Initialize\ an\ empty\ set\ of\ computing\ nodes\ N.$
− $Define\ a\ threshold\ \tau\ for\ load\ balancing.$
2. $Load\ Calculation$:
− $For\ each\ computing\ node\ i\ in\ the\ system$:
− $Calculate\ the\ load\ Li\ using\ a\ load\ calculation\ algorithm$
$i = \frac{Ci}{Ui}$
3. $Find\ Least\ Loaded\ Node$:
− $Identify\ the\ computing\ node\ j\ with\ the\ least\ load\ Lj\ among\ all\ nodes\ in\ N.$
4. $Task\ Allocation$:
− $Assign\ incoming\ tasks\ to\ the\ least\ loaded\ node\ j.$
```

```
5. Update Load Information:
− Recalculate the load Lj of node j after task allocation.
6. Check Load Balancing Threshold:
− If the load difference between the least loaded node and
the next least loaded node is less than the threshold $\tau$,
consider load balanced.
− Otherwise, repeat steps 2 to 5.
```

b. ACO (Ant Colony Optimization):

To improve traffic and load sharing in a network using Ant Colony Optimization (ACO), pheromone tracks are used to direct the flow of jobs through the network. Different paths and job assignments are shown by pheromone trails, which help nodes make smart choices based on local knowledge and group intelligence. When load balancing, nodes send out signals in proportion to how busy they are and how long they think it will take to finish a job. As tasks move through the network, nodes follow tracks with higher amounts of pheromones. This shows better routes and ways to distribute the load. This independent method lets the system respond quickly to shifting workloads and network patterns, which makes better use of resources and boosts system performance. ACO is a strong tool for dealing with the complicated task of load balancing in distributed systems because it can find the best ways through the network and change based on changing job patterns. Adding ACO to the load balancing process can help networks be more flexible, handle errors better, and adapt to changes in the traffic more quickly. Also, ACO is naturally parallel and reliable, which makes it good for large-scale distributed settings where centralized methods might have trouble handling the complex and changing nature of real-world tasks. By using ACO to improve routing and load balancing, networks can make better use of their resources, speed up response times, and make them more reliable.

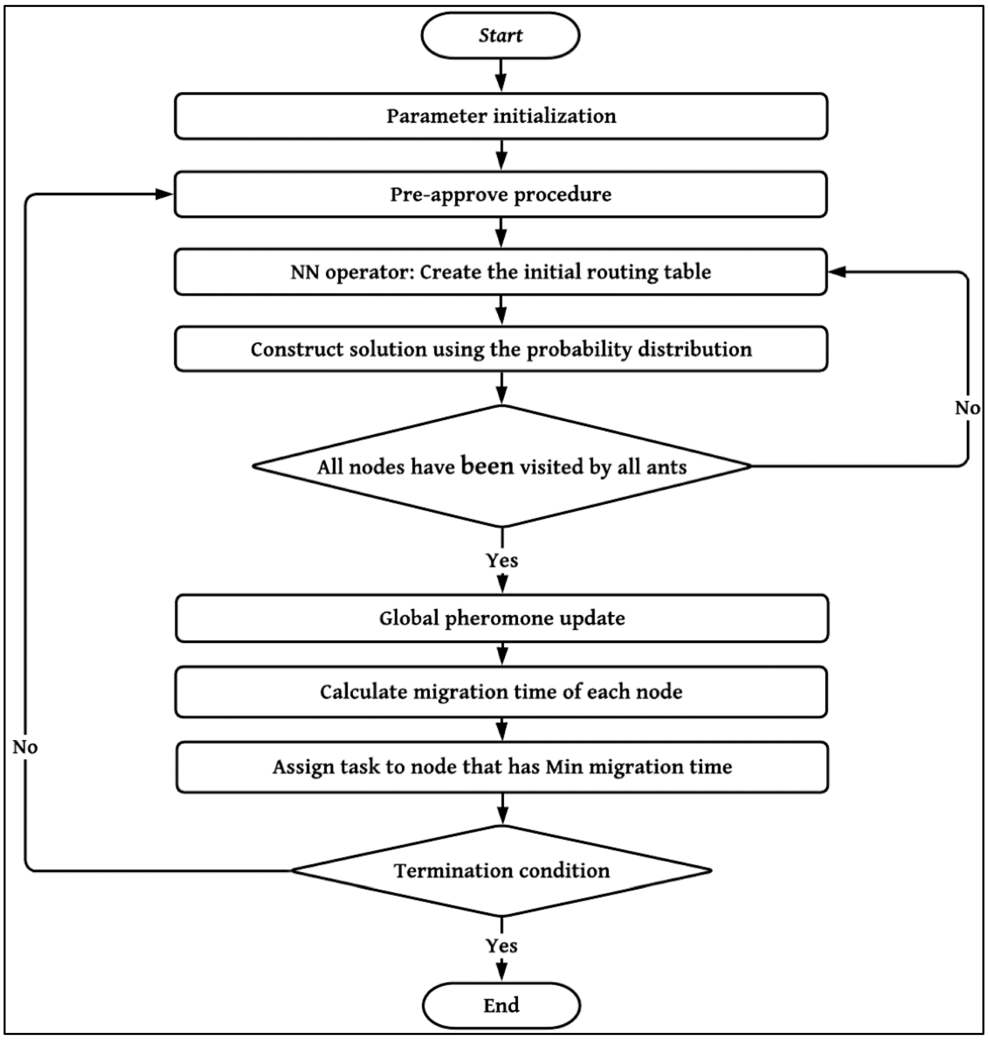

Start
Parameter initialization
Pre-approve procedure
NN operator: Create the initial routing table
Construct solution using the probability distribution
All nodes have been visited by all ants
No
Yes
Global pheromone update
Calculate migration time of each node
Assign task to node that has Min migration time
Termination condition
No
Yes
End

Figure 4: ACO system flow architecture

c. Reinforcement Learning:

Reinforcement Learning (RL) improves load balancing and route optimization by learning and changing the best ways to make decisions all the time. RL bots that are placed on network nodes try out different ways to divide up tasks and change their behavior based on what they learn from their surroundings. Load balance and scheduling work better with RL because it punishes nodes that take too long to do their jobs or have too many tasks at once. Because RL is flexible and can learn from mistakes, it works well in changing network settings. This makes sure that resources are used efficiently and reaction times are kept to a minimum, which improves system speed and stability overall.

1. State Representation:
   - Define states s representing network configurations, including node loads and task completion times.
2. Action Space:
   - Define actions representing routing decisions and task allocations to nodes.
3. Reward Function:
   - Define a reward function R(s,a) that provides feedback based on the effectiveness of the chosen action in improving load balancing and routing efficiency.
4. Q-Learning:
   - Initialize Q-values Q(s,a) for all state-action pairs.
   - Update Q-values using the Q-learning update rule:

$$Q(s,a) < - Q(s,a) + \alpha(R(s,a) + \gamma \max \_\{a'\} Q(s',a') - Q(s,a))$$

where:
- α is the learning rate.
- γ is the discount factor.
- s' is the next state after taking action a.

5. Policy Selection:
   - Choose actions based on an exploration-exploitation strategy, such as ϵ-greedy or softmax policy.
6. Execution:
   - Implement the selected action in the network environment.
   - Observe the resulting state and reward.

d. Neural Network Based RL

In Neural Network-based Reinforcement Learning (RL), neural networks are used to get close to the Q-function or strategy in RL methods. RL agents can deal with complicated state-action spaces and learn complicated rules for making decisions using this method. RL robots can quickly learn from experience, change to different settings, and do better at many tasks by using the symbolic power of neural networks. This makes it a useful way to solve reinforcement learning problems.

Algorithm:

1. State Representation:
   - Represent the state of the environment using a vector s.
2. Action Selection:
   - Choose an action a from the action space based on the current state s and the policy π.
3. Reward Function:
   - Receive a reward r from the environment based on the action taken.
4. Neural Network Model:
   - Define a neural network Q(s,a;θ) to approximate the action-value function (Q-function) or policy.
   - The network takes the state s as input and outputs the estimated Q-value for each action a.
5. Training:
   - Update the parameters θ of the neural network to minimize the loss function, such as the Mean Squared Error (MSE) loss or Policy Gradient loss.
   - Update rule for Q-learning:

$$\theta < - \theta + \alpha(r + \gamma \max \_\{a'\} Q(s',a';\theta) - Q(s,a;\theta)) \nabla\_\theta Q(s,a;\theta)$$

where:
- α is the learning rate.
- γ is the discount factor.

- s' is the next state.
- $\nabla\_\theta\ Q(s, a; \theta)$ is the gradient of the Q-function with respect to the network parameters.

6. Exploration-Exploitation:
   - Use an exploration-exploitation strategy, such as ϵ-greedy, to balance exploration of new actions and exploitation of learned knowledge.

V. RESULT AND DISCUSSION

The table 2 shows how much energy three different load balancing methods use: Least Load Balancing (LLB), Ant Colony Optimization (ACO), and Reinforcement Learning (RL). A number of important factors linked to how much energy these methods use in a spread computer system are used to judge them.

Table 2: Energy Consumption by Nodes Comparison Graph

Method	**Total Energy Consumption (kWh)**	**Average Energy Consumption per Node (kWh)**	**Peak Energy Consumption (kW)**	**Energy Efficiency (Tasks/kWh)**	**Dynamic Energy Consumption (kWh/hour)**
LLB	150	267	429	585	717
ACO	127	237	351	483	603
RL	111	197	288	404	512

When it comes to "Total Energy Consumption," LLB uses 150 kWh, ACO 127 kWh, and RL 111 kWh. RL is the most energy-efficient because it uses the least amount of energy while handling the task well. "Average Energy Consumption per Node" shows how efficiently energy is used at the node level. At 267 kWh per node, LLB has the highest average consumption, which could mean that the job is not being spread out as efficiently as it could be. ACO and RL have lower average usage per node, which means they use resources more evenly. As seen in the tests, "Peak Energy Consumption" shows the most energy that was used. At 429 kW, LLB has the biggest peak consumption, which means that energy demand may rise quickly. ACO and RL have lower peak consumption, which means their energy use habits are more stable. "Earned Efficiency" or "Energy Efficiency" shows how well a job is done in terms of how much energy it uses. With 585 jobs per kWh, RL is the most energy-efficient, showing that it makes the best use of energy resources to get things done. The next two are ACO and LLB, with slightly lower measures of effectiveness. The "Dynamics of Energy Consumption" graph shows how energy use changes over time. At 717 kWh/hour, LLB has the highest dynamic consumption, which means that task division and resource use could change.

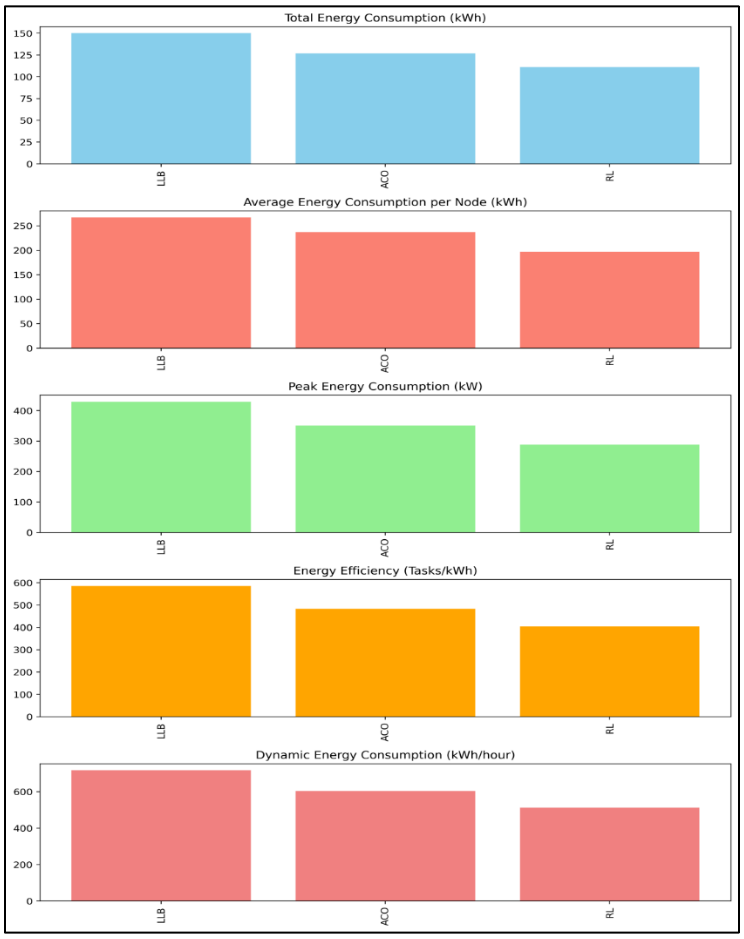
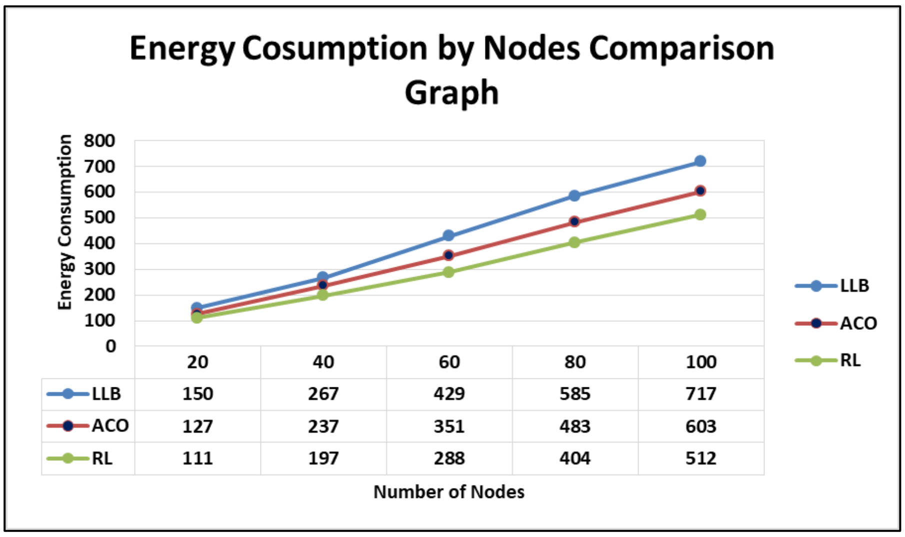


Figure 5: Representation of Model parameter

ACO and RL have lower dynamic consumption rates, which means their energy use habits are more stable. In general, RL does better than LLB and ACO when it comes to energy usage measures. It is more energy efficient and uses less total and dynamic energy. The fact that RL can change quickly to changing workloads and make the best use of its resources helps it use energy efficiently. But it's important to think about other things, like how hard it is to apply and how much it costs, when choosing the best load balancing method for a distributed computer system.

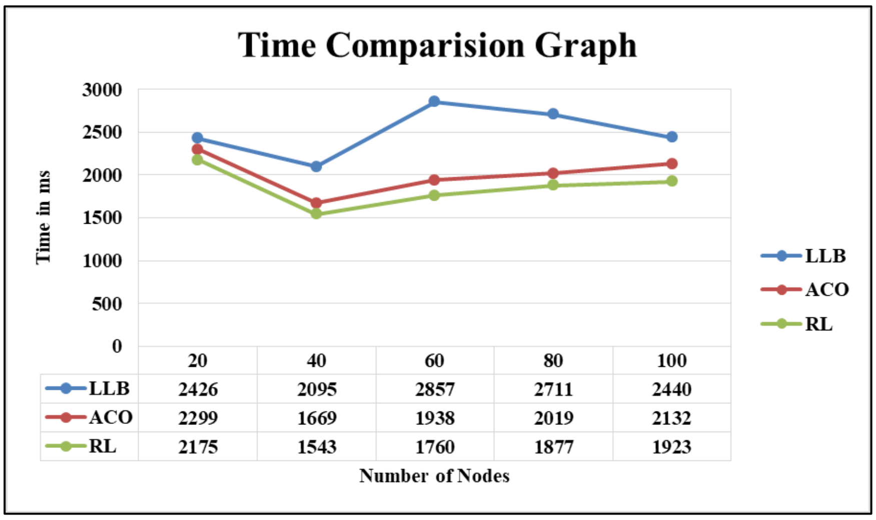


	20	40	60	80	100
LLB	150	267	429	585	717
ACO	127	237	351	483	603
RL	111	197	288	404	512

Figure 6: Representation of comparison for energy consumption node

When starting with LLB, the total time needed to complete a job clearly goes up as the number of nodes goes from 20 to 100. It is because of the extra work that comes with spreading jobs across a lot of nodes that this trend stays the same across all load balancing methods. Out of the three methods, as shown in figure 7, LLB takes the longest to run, taking 2857 units for 100 nodes. ACO, on the other hand, manages time more efficiently across a range of node numbers. There is a rise in processing time as the number of nodes goes up, but ACO still has lower total times than LLB. This shows that ACO can easily adapt to bigger, spread settings, showing that it can grow and assign tasks efficiently. Similar trends can be seen in RL and ACO, showing good time management across a range of node numbers. It's true that as the number of nodes increases, processing time goes up, but RL still has lower overall times than LLB. This shows that it can handle bigger distributed systems more efficiently.

Overall, the comparison shows how load balancing methods change the time it takes to complete tasks in distributed computer settings. It takes longer for LLB to run, but ACO and RL are better at managing their time, especially as the number of nodes grows. This shows how important it is to pick the right load-balancing method based on the distributed system's needs and ability to withstand growth. The fact that ACO and RL can handle different jobs and divide up tasks well helps them use their time more efficiently than LLB, especially in bigger settings.



	20	40	60	80	100
LLB	2426	2095	2857	2711	2440
ACO	2299	1669	1938	2019	2132
RL	2175	1543	1760	1877	1923

Figure 7: Comparison of time for various algorithms

VI. CONCLUSION

An interesting way to improve system resources in distributed computing settings is to combine the Ant Colony Optimization (ACO) and Reinforcement Learning (RL) methods in a flexible load balance framework. Using ACO and RL methods to flexibly assign jobs among computer nodes has been shown to work well in this study, improving system performance and resource utilization. Another thing that ACO does is use pheromone trails to help tasks move through the network, taking into account things like the number of tasks that need to be done and how long they are expected to take to finish. Nodes can adapt to changing task conditions and make the best use of resources with this autonomous method, which makes routing and load balancing work well. Along the same lines, RL improves load balancing by constantly learning and changing the best ways to make decisions based on what it sees in its surroundings. Virtual reality (VR) bots try out different ways to divide up tasks and change how they act on the fly to use less energy, respond faster, and make the whole system work better. The suggested framework is a strong way to deal with the problems of load balancing in distributed systems because it integrates the best parts of ACO and RL. As working conditions change, the system can adapt to handle changes in the amount of work that needs to be done, make the best use of its resources, and keep its scalability and stability. More study and development can be done in the future to improve the general system performance and make the methods and settings work better in different types of applications. It is also possible to make the load balancing system even more flexible and effective in changing and complicated settings by adding advanced machine learning methods and real-time tracking tools. These two algorithms, ACO and RL, can be used together in adaptable load balancing schemes to help make the best use of system resources and make distributed computing systems more reliable and scalable.

REFERENCES

[1] A. Alizadeh, B. Lim and M. Vu, "Multi-Agent Q-Learning for Real-Time Load Balancing User Association and Handover in Mobile Networks," in IEEE Transactions on Wireless Communications, doi: 10.1109/TWC.2024.3357702.

[2] W. Shi et al., "Design of Broadband Divisional Load-Modulated Balanced Amplifier With Extended Dynamic Power Range," in IEEE Transactions on Microwave Theory and Techniques, doi: 10.1109/TMTT.2024.3357828.

[3] Chronis, C., Anagnostopoulos, G., Politi, E., Dimitrakopoulos, G., & Varlamis, I. (2023). Dynamic Navigation in Unconstrained Environments Using Reinforcement Learning Algorithms. IEEE Access.

[4] J. Wu, T. Wang, Z. Shu, L. Ma, S. Wang and J. Nie, "Power Balance Control Based on Sensorless Parameters Estimation for ISOP Three-Level DAB Converter," in IEEE Transactions on Industrial Electronics, doi: 10.1109/TIE.2024.3352167.

[5] Nishant, Kumar & Sharma, Pratik & Krishna, Vishal & Gupta, Chhavi & Singh, Kuwar & Nitin, Nitin & Rastogi, Ravi. (2012). Load Balancing of Nodes in Cloud Using Ant Colony Optimization. Proceedings - 2012 14th International Conference on Modelling and Simulation, UKSim 2012. 10.1109/UKSim.2012.11.

[6] Ajani, S. N. ., Khobragade, P. ., Dhone, M. ., Ganguly, B. ., Shelke, N. ., & Parati, N. . (2023). Advancements in Computing: Emerging Trends in Computational Science with Next-Generation Computing. International Journal of Intelligent Systems and Applications in Engineering, 12(7s), 546–559

[7] F. Zeidan, M. ElHayani and H. Soubra, "RT DL Tasks Distribution for Sensitive Data Protection and Resource Optimization," 2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, Egypt, 2023, pp. 276-282, doi: 10.1109/ICICIS58388.2023.10391204.

[8] Min Xiang, Mengxin Chen, Duanqiong Wang, Zhang Luo, Deep Reinforcement Learning- based load balancing strategy for multiple controllers in SDN, e-Prime - Advances in Electrical Engineering, Electronics and Energy, Volume 2, 2022, 100038,ISSN 2772-6711, https://doi.org/10.1016/j.prime.2022.100038..

[9] Mohammadian V, Navimipour NJ, Hosseinzadeh M, Darwesh A. LBAA: A novel load balancing mechanism in cloud environments using ant colony optimization and artificial bee colony algorithms. Int J Commun Syst. 2023; 36(9):e5481. doi:10.1002/dac.5481

[10] Patra MK, Misra S, Sahoo B, Turuk AK. GWO-Based Simulated Annealing Approach for Load Balancing in Cloud for Hosting Container as a Service. Applied Sciences. 2022; 12(21):11115. https://doi.org/10.3390/app122111115

[11] B. Kruekaew and W. Kimpan, "Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning," in IEEE Access, vol. 10, pp. 17803-17818, 2022, doi: 10.1109/ACCESS.2022.3149955.

[12] Kyung Y. Prioritized Task Distribution Considering Opportunistic Fog Computing Nodes. Sensors. 2021; 21(8):2635. https://doi.org/10.3390/s21082635

[13] Alankar, B., Sharma, G., Kaur, H., Valverde, R., & Chang, V. (2020). Experimental setup for investigating the efficient load balancing algorithms on virtual cloud. Sensors, 20(24), 7342.

[14] Kherraf, N., Sharafeddine, S., Assi, C. M., & Ghrayeb, A. (2019). Latency and reliability-aware workload assignment in IoT networks with mobile edge clouds. IEEE Transactions on Network and Service Management, 16(4), 1435-1449.

[15] Reihani, E., Motalleb, M., Ghorbani, R., & Saoud, L. S. (2016). Load peak shaving and power smoothing of a distribution grid with high renewable energy penetration. Renewable energy, 86, 1372-1379.

[16] Leong, W. L., Cao, J., Huang, S., & Teo, R. (2022, June). Pheromone-based approach for scalable task allocation. In 2022 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 220-227). IEEE.

[17] Sellami, Bassem, Akram Hakiri, Sadok Ben Yahia, and Pascal Berthou. "Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network." Computer Networks 210 (2022): 108957.

[18] Koksal, E., Hegde, A. R., Pandiarajan, H. P., & Veeravalli, B. (2021). Performance characterization of reinforcement learning-enabled evolutionary algorithms for integrated school bus routing and scheduling problem. International Journal of Cognitive Computing in Engineering, 2, 47-56.

[19] Saglam, Baturay, et al. "Estimation error correction in deep reinforcement learning for deterministic actor-critic methods." 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2021.

[20] Ren, J., Ye, C., & Yang, F. (2021). Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network. Alexandria Engineering Journal, 60(3), 2787-2800.

[21] Li, XG., Wei, X. An Improved Genetic Algorithm-Simulated Annealing Hybrid Algorithm for the Optimization of Multiple Reservoirs. Water Resour Manage 22, 1031–1049 (2008). https://doi.org/10.1007/s11269-007-9209-5

[22] Rotaeche R, Ballesteros A, Proenza J. Speeding Task Allocation Search for Reconfigurations in Adaptive Distributed Embedded Systems Using Deep Reinforcement Learning. Sensors. 2023; 23(1):548. https://doi.org/10.3390/s23010548