Optimized Model for Cloud Load Prediction and Resource Allocation Using Deep Learning

"Dr. Nanasaheb .B. Kadu", "Mr. D. B. Ghorpade", "Miss. K. T. Bhandwalkar"
Department of Information Technology, Pravara Rural Engineering College, Loni
Savitribai Phule Pune University, Maharashtra, India.

***Abstract* — as the cloud computing offers extraordinary processing scale and space, it has completely transformed today's rapidly expanding modern period. One of the most important and difficult issues in the cloud computing industry has been predicting server utilization of workloads. Because of the complicated loading environment in cloud data centers and the inefficiency of feature extraction, load prediction in cloud computing is a challenging endeavor. Because load prediction is based on deep learning models, it significantly lowers the energy used in cloud computing. The reason why most deep learning models that are available to evaluate cloud load are unable to offer the right answer due to the fact that the cloud service providers might run several instances of their software on a single physical server in order to maximize resource consumption. Therefore, a long short term memory (LSTM) deep learning model is being utilized to forecast the cloud workload in order to address this issue. This neural network is a unique variety of recurrent neural network that can solve the RNN's vanishing gradient issue. The resources for different virtualized resources need to be allocated based on the forecast of cloud load. The existence of heterogeneous applications like as content delivery, web apps, and map Reduce makes this a challenging task. Workloads in the cloud that have disputed resource allocation requirements for communication and informational purposes. Hungarian neural network approach is applied on the expected loads to get better outcomes in resource allocation in order to handle all of this in a well-grained manner.**

Keywords: Cloud Computing, Load prediction, resource allocation, Long short term memory, Hungarian neural network, Task allocation.

I. INTRODUCTION

Three different resource services are offered by cloud computing as a commercial computing pattern: Platform as a Service (PaaS), Software as a Service (SaaS), and Infrastructure as a Service (IaaS). Cloud computing still has issues with scheduling tasks and resources even if it offers a variety of services and is focused on different application programs. Regarding the latter issue, user service quality plays a crucial role in determining the reliability of the cloud service, resource usage rate, and operating costs. As a result, the multi-objective task scheduling problem in cloud computing has important theoretical and practical implications. The resources and their loads in the cloud computing

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

environment are subject to several dynamic and unpredictable circumstances. For example, the resource node's load varies with time, and resource requests differ according to the year, quarter, and holiday.

These elements could potentially result in resource waste and decreased service quality. Resources will be squandered if the cloud resource load is too low; on the other hand, the system's service performance would suffer if the cloud resource demand is extremely high. Scholars in the aforementioned domains have conducted study on cloud computing job scheduling in relation to the aforementioned issues. A job scheduling approach for ant colony optimization that is based on load balance, cost, and the shortest task completion time. The task scheduling elements of cloud computing specify the load balancing function and the cost constraint function of task completion time, along with providing the initial pheromone. Next, it enhances the ant colony optimization algorithm's heuristic function and pheromone update technique. Using the ant colony optimization algorithm, it derives the objective constraint function and, ultimately, the global optimal solution. After that, it compares the ACO method with the Min-Min algorithm and runs a cloudsim simulation. This approach outperforms the other two algorithms in terms of job execution time, cost, or load balance, as the experiment demonstrates.

The actual implementation of a cloud environment has been hampered by several issues. Resource discovery, scheduling, security, and privacy are a few of them. Among these, load balancing is one of the most important issues. It describes the distribution of the load across several machines. The delivery and distribution of the necessary workload across many computing platforms is referred to as load balancing. Methods for optimizing system output production, resource usage, and virtual machine (VM) performance factors are proposed by load balancing. In order to optimize resource use, the cloud system makes use of many load-balancing techniques. Recent surveys have presented a few of these algorithms. Reducing reaction times and increasing resource utilization are the goals of load balancing, which raises production while cutting expenses. Furthermore, load balancing strives to offer durability and flexibility for applications that expand in scope and require additional resources. It also gives equal division of complex tasks top priority. This study provides an in-depth analysis of the classification schemes used in the literature to describe load balancing strategies before going any further. Two types of load balancing exist in cloud-based systems:

(1) Static algorithms and (2) dynamic algorithms.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

• In a static method, Location is ignored by the available base stations in a static approach. All of the connections and their traits are known in advance. This type of method has predetermined execution. It doesn't rely on real-time data from the existing system and is easy to use.

• On the other hand, the dynamic balancing procedure takes into account the machine's current state. Its operation is driven by changes in the node design. Although dynamic algorithms are challenging to build, they more effectively balance load by distributing resources in an efficient manner.

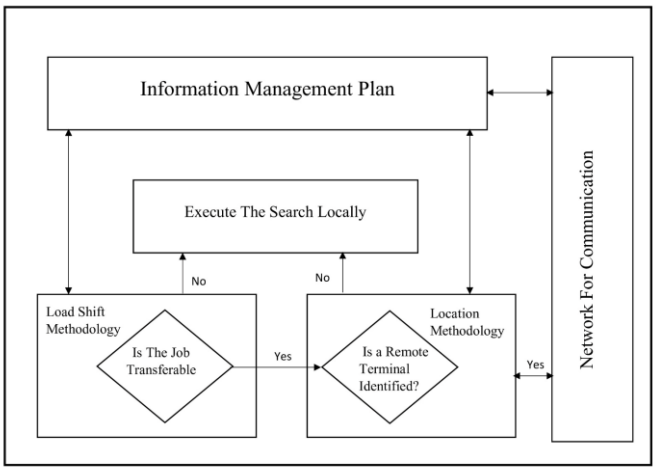

Figure 1: Architectural component of Dynamic Load balancing

Customers can access a vast array of materials from the internet, including technology, services, apps, and other relevant resources. A cutting-edge strategy for getting access to a range of resources, customized to the user's budget, including data, software, power infrastructures, connection, and other commodities. With this method, users can utilize their computer networks as they see fit, regardless of the state of their internet connection. The core idea behind this strategy is that people should be free to use the resources at their disposal whichever best meets their needs. Users can initiate computation in the cloud by requesting it. Three main benefits are provided by cloud computing designs: processing is made easier, service quality is raised, and customer satisfaction is increased.

Compared to traditional technologies, cloud-based services have a lower upfront cost because users are only billed for the specific features they use. Because of this reduction in processing power, programmers may focus on creating new features instead of worrying about buying the necessary computer gear and software. A process scheduler can be used to accelerate software deployment. In order to facilitate concurrent execution, this scheduler acts as a middleman by identifying the best resource to use for workflow tasks and then allocating those tasks to several processing units.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

One essential part of the cloud infrastructure is cloud scheduling. There are three basic steps in the work scheduling process. The first stage is determining the necessary resources and then implementing filtering rules based on the current state of the networked cloud service. Second, it is necessary to decide how a resource will be chosen in accordance with specific objectives. In the end, the selected asset is given assignments to finish. By using static scheduling, the temporal aspects of the work, such as its duration and start and finish timings, are predetermined. As such, it is possible to specify in advance the methodology that will be used to carry out each task.

Scalability is the primary characteristic that characterizes a Cloud computing system's quality. The ongoing issue is to manage data and Cloud infrastructure with optimal task distribution while achieving higher performance and more scalability. Large-scale distributed architectures require the adaptive real-time resource management. The issue is that some systems are unable to accommodate the amount of data that a data-intensive application requires. Determining the relevant indicators for scalability and cloud infrastructure evaluation is another issue. It is necessary to comprehend scalability within the framework of precise optimization techniques. Using the right methods, it is feasible to link a huge number of heterogeneous resources to a system that performs practically linearly and satisfies numerous optimization goals applied to QoS Cloud measurements. The usage of software-defined resource management and artificial intelligence (AI) approaches is growing, opening up opportunities for quick and flexible adaptation and management of workflows' changing requirements. These systems take a clever and centrally managed approach to addressing the best use of resources for dynamically demanded services.

[1] Xueying Guo et al. developed a fuzzy self-defines algorithm-based multi-objective task scheduling optimization for cloud computing. A construction of the objective function of cloud computing multi-objective task scheduling is done based on the choice of the goal. The multi-objective task scheduling optimization for cloud computing is realized after the objective function is solved using the fuzzy self-defense algorithm through search, yielding the global optimal solution. The outcomes demonstrate that this approach performs better in terms of the maximum completion time, the rate of deadline violations, and the use of virtual machine resources. It is therefore more applicable. As the scheduling aim, the suggested method plans the three objectives: the least amount of time customers must wait, the degree of resource load balancing, and the cost of completing multi-objective tasks as a single objective evaluation function. It therefore achieves the quickest maximum scheduling completion time and a superior scheduling effect. Furthermore, the suggested algorithm's low deadline violation rate—roughly 5%—is mostly

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

compromised when the quantity of cloud computing multi-objective tasks rises. This makes the algorithm's deadline violation rate lower than that of other approaches. Furthermore, the results obtained with the suggested approach under a certain number of multi-objective tasks are relatively balanced in terms of task scheduling, suggesting that the virtual machine's source utilization rate is relatively appropriate, and consistently exceeding 95%.

[2] Without the need for supervision from ground-truth dense point clouds, Xinhai Liu et al. presented a revolutionary self-supervised point cloud up sampling technique to create dense and uniform point sets from sparse inputs. Through point feature extraction and point feature expansion, the author's coarse-to-fine reconstruction architecture efficiently enables point cloud up sampling. Furthermore, the quality of point cloud up sampling is significantly improved in an unsupervised way by the Author's self-projection optimization, which effectively projects noisy points onto the underlying object surface itself. The author's experimental results show that the method can achieve good performance, even comparable to the state-of-the-art supervised method, on both synthetic and real scanned datasets.

[3] Ran Huang et al. create a novel load-balancing system called TAM in order to simultaneously satisfy the performance needs of long and short flows. When the queue length beyond the threshold and both the long and short flows coexist, the long flow is indicated by TAM in the path. In the meantime, each parallel line's status is tracked in order to choose the best transmission path for flows or flowlets. When compared to the most advanced data center load balancing systems, TAM can enhance the performance of both long and short flows by 20% to 60%, according to the results of a series of extensive NS2 simulation tests.

[4] Joanna Kosinska et al. assess the suggested rule-based framework for managing cloud-native apps. AMoCNA contributes to SLA compliance. Specific rules (policies) must be published in order to achieve this need. Our suggested technique is adaptable enough to establish useful policies for environments that are cloud-native. A general framework for the experimental evaluation of cloud-native systems is presented in this study. The author suggests two types of studies to assess the rule-based methodology. The first category assesses the effects of dynamic resource adjustment. The influence during the autonomic management process is assessed in the second category. Experiments undertaken in the meantime demonstrated the significance of observability. The information acquired during this procedure is a useful resource for understanding the development and status of cloud-native apps. Achieving autonomic management can be effectively accomplished by applying this information when establishing management policies. The study that is being given demonstrates the value and use of the suggested MRE-K loop paradigm. The idea of using a rule engine

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

to enforce management policies has been successfully validated. The same experiment also shows that even in the face of frequent and dynamic adjustments, the policy-based approach to autonomic administration of Cloud-native apps remains accurate. To reinforce the effects of dynamic changes, the established policies must encompass a wide range of the execution environment's elements. By contrasting its advantages with those provided by the Kubernetes management options, the author also examined the possibility of the declarative management policy approach to autonomous management of Cloud-native apps.

This paper's primary research question is how to use task scheduling to maximize centralized resource management while taking scalability into account in heterogeneous cloud systems. The performance assessment and system scalability study of heterogeneous Cloud infrastructure carrying out resource-intensive tasks are the main topics of this research.

The second part of this study summarizes previous studies on cloud load prediction and resource allocation models. The deployed approach is explained in full in Section 3. The results are evaluated in Section 4, and the study is concluded in Section 5.

II. LITERATURE SURVEY

[5] Xinliang Wei et al. looked into an edge computing challenge involving the combined allocation of resources and task dispatching over various periods. To tackle this joint optimization in a dynamic edge environment, the author suggested a deep reinforcement learning based approach and a two-stage optimization technique. The author's simulation results demonstrated that: (1) both suggested methods outperform greedy and random algorithms; and (2) the benefit of carrying out resource placement and task dispatching in different timescales is that it not only lowers placement costs but also minimizes the need for extensive task prediction in the future.

[6] A revolutionary load balancer building component that ensures PCC and supports any realizable LB mechanism, CHEETAH was introduced by Tom Barbette et al. The author used programmable ASIC Tofino switches and software switches to implement CHEETAH. The authors view this paper as a first step in resolutely harnessing the power of load balancing techniques. The author leaves open the question of whether it is possible to create new load balancing techniques specifically for Layer 4 LBs and to use them with already-existing middle boxes in future research.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

[7] Kohei Hosono et al. describe cooperative autonomous driving, which aims to maximize efficiency and safety by wirelessly exchanging sensor data from autonomous vehicles. Additionally, a dynamic map system is being researched. This is a platform for information transmission that manages shared sensor data and runs applications. When the number of vehicles that transmit and receive sensor data rises, there is a problem regarding the scalability of the dynamic map system, which runs on the cloud on the Internet. As a result, it is believed that by dispersing numerous edge servers worldwide and managing the data these servers manage in the cloud, applications can be run effectively. The edge server that manages the vehicle information and the edge server that receives the data, however, may not always match depending on the real radio wave conditions. As such, certain applications, such merging arbitration and intersection collision danger warning, are challenging to handle. The author designated a "lane section ID" to each segment of the road where the car goes, and each lane section ID was given an edge server. Additionally, by connecting several edge servers, the author built a dynamic map system that links cars and edge servers. Additionally, in order to facilitate effective communication between the edge server and the car, the edge server used multicast to connect with the vehicle, and the vehicle used any cast to interact with it.

[8] Taking network topology and bandwidth limitations into account, Akito Suzuki et al. developed an ideal task-offloading issue for multi-cloud and multi-edge networks. Cooperative multi-agent deep reinforcement learning (Coop-MADRL) is the foundation of the cooperative task-offloading technique that the author presented. By utilizing deep reinforcement learning beforehand to learn the relationship between network-demand patterns and appropriate task offloading, this strategy can quickly achieve efficient task offloading. In addition, this approach presents a cooperative multi-agent mechanism that enhances task offloading effectiveness. Tests showed that in a variety of network topologies, the suggested approach can reduce network usage and task latency while minimizing constraint violations in less than 1 ms. by pre-training with numerous resource-intensive activities, the author showed that the suggested strategy can exhibit generalization performance for various task types. The authors intend to assess the suggested method's effectiveness and carry out more thorough study in more complex use cases or practical implementations. Additionally, the suggested method's scalability and interpretability will be enhanced by the author.

[9] Thiago Rios et al. describe how evolutionary optimization algorithms can tackle several independent tasks at once multitask. One of the main obstacles to design exploration, though, is translating the various task design spaces to a single representation. It is difficult to find a unified, yet useful representation in form optimization problems. The

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

author of this paper presented an MFEA framework that strikes a balance between universality and practicability by using the latent space that is learned by a 3-D point cloud auto encoder. The network produces a domain-independent representation and transfers the designs straight from the unified space to the CAE representation in the Cartesian space because the auto encoder only learns on geometric data. The transfer of latent features between representations results in designs with interpolated characteristics, as the author demonstrated in a simplified car shape optimization problem. This allows for an MFO to foster common geometric characteristics between designs and increase the diversity of individuals with the same skill factor. In order to reduce the aerodynamic drag of three different automobile shapes in four different MFO scenarios, the author finally used the MFEA for shape optimization to a multitask vehicle optimization issue inspired by real-world situations. Furthermore, the Author's framework makes it possible to transfer latent elements that represented the automobile forms' underbody, making the best designs share a nearly identical platform—a trait that may boost manufacturing and maintenance efficiency.

[10] Yuichi Nakatani et al. presented Structured Allocation-based Consistent Hashing (SACH), a hashing algorithm with consistency features that concurrently meets load balancing, consistency, memory utilization, fault-tolerance, and lookup time—all five requirements for cloud architecture. SACH is intended for systems that resemble cloud infrastructure, where backend provisioning is managed and the proportion of concurrently failing back ends is low. Initially, SACH employs an architecture that distinguishes between backend failures/recoveries and planned backend additions/removals using distinct sub algorithms. This architecture allows the cluster to maintain fault-tolerance while updating the allocation quickly enough to allow for adequate time to be spent reallocating the hash space for a planned update. In order to facilitate the formulation of mathematical problems, SACH then models an allocating problem as a backend sequencing problem. This allows for the provision of polynomial-time solutions for the defined load balancing problem. According to the experimental results, as long as the proportion of concurrently failing back ends is low, SACH with comparable memory use outperforms current techniques in terms of load balancing and lookup rate. In this case, SACH outperforms CH and Maglev by a threshold of failure percentage of roughly 10 and 1.5–10 percent, respectively. The results showed that SACH satisfies all five criteria since it is intended to meet consistency and fault-tolerance requirements. SACH will benefit academia and industry alike.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

The issue of load-balanced task scheduling was tackled by Stavros Souravlas et al. in [11]. The Markov process model serves as the foundation for the author's system model, which is paired with a straightforward fair task distribution strategy. The author derives the anticipated virtual machine (VM) utilizations from the balancing state probabilities. The fair work allocation policy of the author is applied on a time slot basis and in a way that ensures that each virtual machine has an equal expectation of utilization. The scheme's constant load balancing guarantee was demonstrated by the author. The suggested plan is multi-objective since it improves several crucial metrics, including response time, degree of imbalance, average utilization, and make span. The author's task allocation strategy outperformed three new, cutting-edge schemes in terms of the aforementioned metrics: the Load Balancing Modified Particle Swarm Optimization (LBMPSO), the Honey Bee Foraging (HBF) with VM pre-selection, and the community-based Particle Swarm Optimization (CPSO) scheme. Optimization is a topic that requires more research. Although the author has demonstrated that their technique ensures load balancing and yields good results for a range of measures under various distribution situations, they have not yet demonstrated optimality.

[12] In order to maintain server uptime, Mana Saleh Al Reshan et al. developed a hybrid fast-converging load balancing technique that achieved globally optimal rapid convergence for balancing loads between VMs. The minimal dependence of GWO on the control parameters makes its implementation simple. Analytical comparisons reveal that whereas BAT deteriorates with increased user retention, ABC, PSO, and SSO algorithms have delayed convergence, i.e., high responsiveness when compared to BAT and GWO. The findings indicate that GWO is preferable for convergence while PSO is suitable for global optimization. Therefore, PSO and GWO are combined to allow for the possibility of accelerating convergence so that the globally optimum solution can be reached. Because of its quick convergence, the scheduling algorithm is adaptable enough to handle competing requests for virtual machines (VMs). This critical thinking is further supported by the simulation, and the proposed method performs as expected. PSO's computational complexity can be roughly expressed as O(k*n), where k is the number of iterations, n is the size of the vector, and the number of particles. Similar to this, k, the vector's size, and the quantity of search agents a determine how complex GWO is. Thus, O(k*a) can be used to approximate the complexity of GWO. The suggested approach has an overall complexity of O (k*n + k*a). The work is concluded in this section with a discussion of the potential of deep learning and machines for load balancing. The results of this study show that many load-balancing mechanism issues are nevertheless eventually overcome by employing a clever and efficient task scheduling algorithm, particularly

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

when it comes to additional QoS metrics and technique complexity assessments. These considerations make machine learning—particularly deep learning—promising for improving both the author's recommended solutions and the ones that are currently in place. Machine learning algorithms have proven useful in a number of fields, such as language modeling, pattern recognition, and manufacturing.

An implementation of the (μ, λ)-Evolution Strategies metaheuristic algorithm is proposed by Petra Loncar et al. [13] in order to address the work scheduling problem in heterogeneous Cloud computing environments. The study that is being presented has used the Cloud Sim framework to perform task scheduling simulations. The workload for the simulation was developed using ALICE jobs from a single production. The effectiveness of the suggested Evolution Strategies technique has been established and validated through a variety of simulated experiments. The results of experiments demonstrate the dependability of the Evolution Strategies task scheduling with the Largest Job First broker policy model implemented. Compared to Evolution Strategies and Genetic Algorithm metaheuristics, it performs noticeably better. The jobs can be assigned to the VMs in an ideal manner by using the suggested task scheduling technique. The solution based on Evolution Strategies minimizes makespan, lowers average execution time and imbalance, boosts throughput and resource usage, and accomplishes scalability. System performance is enhanced in all cases by managing heterogeneous high-capacity data center resources and allocating tasks dynamically. The suggested technique, according to the author, achieves scalability while utilizing cloud resources across various data centers. The Evolution Strategies algorithm holds a significant position in artificial intelligence and holds the ability to address the performance issues with conventional reinforcement learning methods. Future studies aim to include additional workload and resource variables to optimize the algorithm's fitness operator and produce better solutions. Future work will take into consideration the creation of a proactive Evolution Strategies-based job scheduling algorithm that balances load balancing, task assignment time, task completion time, and cost. A thorough comparison between the algorithm and relevant state-of-the-art algorithms will be conducted.

The experimental investigation of the cloud's Load Balancing (LB) mechanism using its Resource Scheduling (RS) algorithms is the main topic of Prathamesh Vijay Lahande et al. [14]. Four stages of task processing and computation were employed in the experiment, using the Work Flow Sim cloud simulation platform. The Sipht task event dataset was used, and the task sizes varied for each step. These tasks were computed using the RS algorithms First Come First Serve (FCFS), Maximum – Minimum (Max – Min), Minimum Completion Time (MCT), Minimum

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

– Minimum (Min – Min), and Round Robin (RR). Each phase included sixteen scenarios with varying numbers of Virtual Machines (VMs) used, ranging from five VMs in the first scenario to a thousand VMs on the last, to allow for a thorough comparison of LB. According to the findings of the experiment, the LB for each phase and scenario was 51.98%, 41.71%, 51.98%, 59.58%, and 52.17% for FCFS, Max-Min, MCT, Min-Min, and RR, respectively. In order to make the LB process dynamic, the intelligence mechanism of the Reinforcement Learning (RL) approach is finally proposed. Improved LB processes with RS algorithms will result in greater resource optimization with RL, giving the cloud the optimal Quality of Service (QoS). The paradigm that is being given can be effectively adopted to enhance current real-world applications like Google Cloud Computing (GCP) and Amazon Web Services (AWS).

[15] Behzad Saemi et al. discuss how mobile devices are capable of running a variety of programs, all of which demand an increasing amount of computing power. Mobile devices frequently leverage cloud computing's offloading functions to do more sophisticated tasks because of their limited resources. The task scheduling problem, which comprises determining where to dump work to optimize its value, is the offloading problem in Mobile Cloud Computing (MCC). Because of the challenges associated with resource relocation and the complexity of the search space needed to identify the optimal scheduler, the task scheduling problem in MCC is NP-hard, making the application of sophisticated search techniques impracticable. To produce a best-case or nearly-best-case situation in terms of project completion time and energy savings, metaheuristic search methodologies are offered. This work addresses the described problem in MCC by offering a non-dominated multi-objective approach termed Hybrid Multi objective Harris Hawks Optimization (HMHHO), which is based on the Harris Hawks Optimization (HHO) technique. Allocating jobs from mobile source nodes to cloud processors, cloud patches, and mobile resource processors were the goals of this study. The suggested method, on average, completes tasks more quickly and consumes less energy than the other four algorithms—the Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Cuckoo Search Algorithm (CSA).

The issues of QoS constraint violation and low resource utilization under uncertainty in user access and virtual resource provisioning (i.e., IaaS resource load) are still highlighted in Longchang Zhang et al in [16]. Studies on IaaS resource allocation under service QoS constraints at the IaaS and PaaS levels. The revenue of SaaS providers is not taken into account. With respect to the aforementioned challenges, this research suggests a minimum optimal allocation strategy for IaaS resources that maximizes the anticipated revenue of SaaS providers. In order to accomplish this goal,

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

three minimum optimal allocation strategies are designed: one for IaaS resources with uncertain demand and supply, another for demand and uncertain supply, and a third for both uncertain demand and uncertain supply. The results of the experimental analysis demonstrate that the method used in this study can efficiently allocate IaaS resources and has high operational efficiency. Its benefits include the following: (1) it can efficiently determine the minimum optimal IaaS resource allocation in the three scenarios that maximize the SaaS provider's expected revenue; (2) the SaaS provider determines the IaaS allocation and there is no violation of QoS constraints, which is also conducive to its accurate allocation of IaaS resources and improvement of system resource utilization; and (3) it can successfully handle the issue of complex IaaS configuration brought on by unpredictability in user access and IaaS resource load. Since this study is still in its early phases, there are still a lot of issues that need to be resolved in order to optimize the revenue of SaaS providers through IaaS configuration. (1) The revenue generated by SaaS providers can still be increased; (2) Maintaining the quality of service requirements of users whose services have been transferred; (3) Making the most of idle resources to increase SaaS provider revenue; (4) Allocating resources under multiple resource requirements; (5) Allocating resources under various resource leasing price models; (6) Allocating resources for multiple infrastructure providers; (7) Creating an adaptive resource allocation model under stochastic supply and demand conditions.

[17] Rong Gao et al. explain how the CPU, memory, and bandwidth utilization are used as load factors to create the load assessment for the K-TAHP load balancing approach. It fixes performance loss brought on by load imbalance after extended cluster operation by moving high-load pods from overloaded nodes to nodes with lower loads by utilizing a warning module and a migration module. This improves load balancing in the Kubernetes cluster. The method effectively addresses load imbalance in long-running Kubernetes clusters, as experimental findings show. Additionally, the plan guarantees that Pods will continue to function normally throughout the migration process, protecting the cluster applications' performance. Subsequent research endeavors will center on the problem of Kubernetes pod elastic scaling, with the objective of guaranteeing pod performance via proactive scaling techniques.

[18] Hamirah h. et al. Fuzzy logic is used in a two-phase virtual machine allocation technique created by Alomar et al. for SDN-Cloud heterogeneous networks in cloud data centers. The suggested technique considers a number of factors, including the use of both PM and VM, and multiple resources, including memory and processing power for host-VM mapping. A fuzzification step is included in the first phase, when resources (memory and computing power) are removed from neighboring hosts once they are gathered. Each host has a ratio assigned to it, which is then used to

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

sort the hosts. The next step is called defuzzification, where the requested resource ratio of the requested virtual machine (VM) is calculated to start the process of choosing the optimal suitable host. By comparing the virtual machine's resource ratio to the designated host, an inference rule is constructed. If the allocation is completed successfully, the host's resources are then updated. The author has assessed the author's work in Cloud Sim SDN with an overloaded network by simulating real-world traffic in large-scale data centers through sophisticated applications like Wikipedia workloads. Together with network and CPU serve times, the author uses response time as a gauge of network performance.

[19] Mingxue Ouyang et al. create a resource management platform built on the Band-area Application Container (BAC) as a Service (BaaS). This platform is used to create and implement Internet of Things (IoT) applications as well as enterprise business systems that are already in place. It then combines these applications into a cohesive application system by BAC collaboration. The author proposes a fine-grained application service model called tool service in the microservice framework to adapt to the interface characteristics of the BAC because there isn't a single template or specification for microservices. Pipeline relationships, producer-consumer relationships, and random handshake relationships were the three forms of invocation relationships between tool services that were built. Through the invocation relationship between services in BAC, developers can combine services into service function chains (SFCs) to create new sub-application systems. Moreover, the virtualization of tool services and the layout of the BaaS platform are explained. Three optimization objective functions are built: service transmission overhead, resource load balancing in the CDC, and execution container aggregation in order to solve the deployment problem of the execution container for tool service in the BaaS platform. An accelerated particle swarm optimization (APSO) for the tool service execution container deployment strategy (APSO-TSDS) was developed based on the optimization objective function.

Cloud computing services, as described by Haiyu Zhang et al. [20], offer crucial technical support for the management and integration of online information resources. The inability of conventional cloud technology to optimize multi-task target scheduling, however, results in a drop in the quality of cloud services. By examining current CCs and optimizing task execution time, system operating costs, and system load, a CC MTS model is built to address this problem. To solve the CC MTS model, CSO is introduced. Inertia weight is introduced since the conventional chaotic cat swarm method is prone to entering LOS. Additionally, modifying memory pool fitness improves the accuracy of the model's inadequate training. The suggested CICSO is doing the best in the algorithm model's

 DOI: https://doi.org/10.37896/pd93.4/9342

performance study; it tends to converge after 96 iterations under the Rastigin function. The ideal search value is 0.798, and it outperforms other scheduling models in terms of optimization accuracy and model convergence.

III. PROPOSED MODEL METHODOLOGY

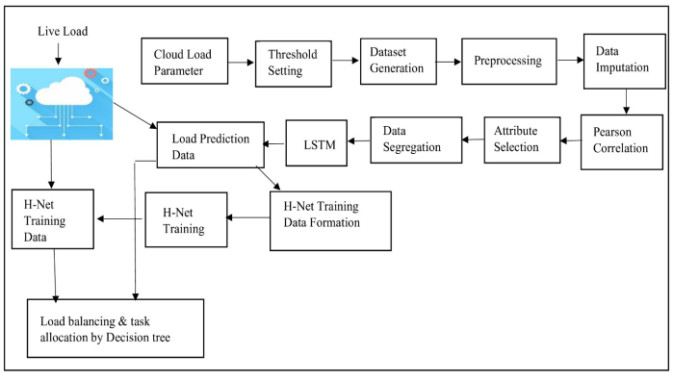

Figure 2: Proposed model for load prediction and Resource allocation

Figure 2 above illustrates the proposed method for resource allocation and load prediction at the cloud. The proposed approach is implemented in the virtual environment to forecast the load at the cloud administrators. A resource allocation method has been developed utilizing the Hungarian neural network and the Genetic algorithm based on this load design. As explained below, several procedures have been implemented to carry out the load prediction and resource allocation processes.

Step 1: *Dataset Generation:* For the purpose of the load prediction the proposed model generates the dataset based on the random number generation based on the threshold maintenance selection process. The Generator generates the dataset for the following fields and finally stores in a workbook file.

num_of_shards - This parameter indicates the total number of shards located on the cloud, which is the same as the number of vectors that will be created.

num_of_samples - This specifies the number of time intervals that are taken into account while creating tasks.

new_period - which specifies the duration associated with a single load vector element. 5.0 is the default value.

Shape - This specifies the gamma distribution's α parameter, from which the number of jobs each period will be selected.

Size - This specifies the gamma distribution's β parameter, from which the number of jobs each period will be selected.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

Step 2: Preprocessing- The cloud load dataset is typically used for pre-processing when it is generated. Following the dataset's reading from the specified path in the form of a workbook sheet, the characteristics are gathered in a double-dimensional list in this pre-processing stage. Python's Pandas module has been used to read the dataset from the specified directory.

The early parameters of the attributes, such as the mean and standard deviation as mentioned in equation 1 and 2, are estimated using this double-dimensional list data in order to characterize the features of the dataset in accordance with the their range as mentioned in the equation 3.

$$\mu = \frac{\sum_{i=0}^{n} xi}{n} \text{ ---------- (1)}$$

$$\alpha = \sqrt{\frac{\sum_{i=0}^{n} (xi-\mu)2}{n}} \text{ ------ (2)}$$

$$(\mu, \alpha) \Rightarrow \{x \in R:(\mu - \alpha) < x < (\mu + \alpha)\} \text{ --------(3)}$$

Where,

μ=mean of the attributes

α= Standard Deviation

n= Number of samples

xi= Instance attribute values

R= Range Factor

Following this procedure, the entropy of the various data types in the dataset is asserted by obtaining the dataset attribute information for various data kinds, such as string and float values. Assigning unique positive integer values to the repeated data in the column sets the properties of the created dataset for labeling. Their frequency is approximated once the dataset has been labeled in order to verify that the dataset balances for the labeled classes. To ensure equitable distribution, these labeled classes are oversampled. In the end, this procedure improves the preprocessing step to produce accurate cloud load prediction results.

Step 3: Data Imputation – The oversampled data is used to estimate a heat map for each characteristic. To determine the overall amount of missing data and its corresponding percentage, this is accomplished by comparing the

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

transition data obtained during the sorting process. The fillna() function is used to impute the missing data for variables such as ‘num_of_shards’ , ‘num_of_samples’, ‘new_period’, ‘Shape’ and ‘Size’.

For each object in the attributes, a formal parameter called a label encoder is used in conjunction with the fit transform function in order to do imputation. The fit transformer is used to modify the characteristics, and then the IterativeImputer () method is used to deploy multiple imputation using chained equations, resulting in the mouse imputation object. The multiple imputation approach called MICE is used to restore missing data values in a data collection under specific assumptions about the data missingness mechanism. If the model is given a dataset with missing values in one or more of its variables, it may generate multiple copies of the data. The missing values for the particular combination are found after mice imputation is applied, and the missing values are then forecasted using known values from other features in the data as predictors. Next, the acquired imputated values are used to estimate the interquartile range (IQR), which spans from 0.75 to 0.25. This technique eventually yields the dataset comprising the imputation of missing values.

Step 4: Pearson Correlation – A thorough dataset list is produced by the Mice imputation procedure, and it is subsequently put through a Pearson Correlation analysis. The characteristics with the lowest correlation are found using the Pearson Correlation values. The correlation between the qualities is established by Pearson Correlation. Consequently, a correlation matrix is generated that could be useful in selecting the right combination of the attributes. In light of this finding, the traits that exhibit lower correlation are ignored, and new correlation values are calculated. Equation 4 is utilized to perform the Pearson correlation and is displayed below.

$$r = \frac{\sum(x_i-\bar{y})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2 \sum(y_i-\bar{y})^2}} \text{ ------- (4)}$$

Where,

x_i=values of x (independent) variable

y_i= values of y (Dependent) variable

$\bar{x}$= mean of x variable values

$\bar{y}$= mean of y variable values

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

A few unnecessary columns are removed from the list after the correlation phase, which compiles the dataset into a double-dimensional list. As will be shown in the following step, the resulting list is then given to the following data segmentation stage.

Step 5: Attribute Selection - The minMaxscaler function is applied using the imputated and preprocessed data in this attribute selection procedure. Minmaxscaler to alter the features, scale each one to a certain range. This estimator independently scales and translates each feature to fit inside the specified range on the training set, for instance, between zero and one. MinMaxScaler does not minimize the impact of outliers, even if it linearly scales them into a set range where the largest data point corresponds to the maximum value and the smallest one to the minimum value. Seek advice Consider contrasting MinMaxScaler with alternative scalers to get an example of a visualization. The transformation of minmaxscaler can be obtained using equations 5 and 6.

$$x_{std} = \frac{(x - x.min(axis=0))}{(x.max(axis=0) - x.min(axis=0))} \text{----- (5)}$$

$$x_{scaled} = x_{std} * (max - min) + min \text{— (6)}$$

Where min, max = feature_range.

Two lists are generated for the "cloud load" prediction attribute after this process. Next, based on the k highest scores, the characteristics with the given value are selected. Following that, the selected characteristics are fitted for the transform with the random state zero and an estimator's value of 100. This approach yields the critical qualities that will be trained in the next steps. The attributes that were chosen for selection are "num_of_shards," "num_of_samples," "Shape," and "Size."

Step 6: Data Segregation – Here, a list named X has all of the qualities, including "num_of_shards," "num_of_samples," "Shape," and "Size." However, 'new_period' is stored in a list named Y.We use the train_test_split() function to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). Four sections make up the dataframe: X_train, X_test, y_train, and y_test. The X_train and y_train sets are used to fit and train the model. Use the X_test and y_test sets to see if the model is appropriately predicting the outputs and labels. It is possible to test the train's and the test sets' sizes explicitly. It is recommended that we keep our train sets bigger than the test sets.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

Train Configuration: The training dataset is the collection of data that was utilized to fit the model. The model was trained using this dataset. This data is seen by and used to train the model. Test set: A portion of the training dataset is used as the test dataset in order to accurately evaluate the final model fit. 67% of our data are used for training sets and 33% for test sets in the suggested system. Validation set: A validation dataset is a subset of the training set that is used to update the model's hyper parameters. It is employed to gauge the model's effectiveness. Following this procedure, data scaling is used to adjust each feature's minimum and maximum values to zero and one, respectively. MinMax Scaler shrinks the data within the given range, usually between 0 and 1. It alters data by scaling features to a given range. It scales the numbers to a predefined range while preserving the structure of the original distribution. In order to accomplish this, the Sklearn preprocessor defines the MinMaxScaler() method. This MinMaxScaler() method produces results for lists such as train_X, test_X, train_Y, and test_Y. The Long Short Term Memory (LSTM) Neural Network is applied using these four lists, as will be explained in the following step.

Step 7: Long short term memory (LSTM) - The LSTM neural network requires a scalar normalization object in addition to test_x, train_x, and test_y as input parameters. Using ten units of data, a single feature, and a TRUE return sequence, the LSTM model is introduced. Its parameters include train_X1.shape [1], train_X1.shape [2]. Next, a Dense layer with an activation function of "relu" and a kernel size of 1 is added. The thick layer of a densely coupled neural network uses the "sigmoid" activation function on neurons to efficiently learn new information. The classic LSTM neural network consists of two dense layers, but in this case, only one kernel (size 1) is used for one-dimensional data. When creating a neural network, the shuffle option is set to false, and a batch size of 100 and 100 epochs are used.

The ReLU, Sigmoid and Leaky RELU functions are shown in the following equations 7, 8 and 9, respectively.

$$f(x) = max(0, x) \quad (7)$$

Where, x is any positive value

$$S(X) = \frac{1}{(1+e^{-x})} \quad (8)$$

Where,

X is the input to a neuron

$f(x) = Relu\ Activation\ Function$

$S(x) = Sigmoid\ Activation\ Function$

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

e= Euler's Number

ξ denotes leaky ReLU activation function

$$\xi(x) = \begin{cases} \alpha x, x < 0 \\ x, x \geq 0 \end{cases} \quad (9)$$

where $\alpha = 0.2$ denotes the negative slope

Figures 3 and 4 show the created LSTM model summary and the training results, respectively.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10)                1280

dense_2 (Dense)              (None, 1)                 11

=================================================================
```

Figure 3: LSTM Model Summary

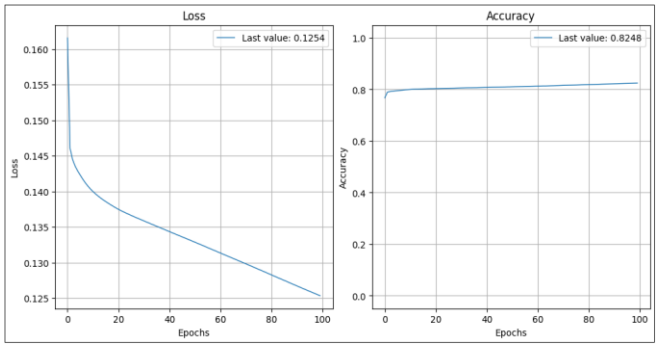

Figure 4: LSTM model training Results

Step 8: Generating Hungarian Neural network (H-net) training Data-

As stated in the previous phase, the LSTM model generated a set of projected load parameter list after training the cloud load dataset. The Hungarian neural network uses this to handle resource allocation. The Hungarian Network (Hnet) is the name of the Hungarian method that uses deep learning to help solve the assignment problem. This method may be applied to additional deep learning tasks that require permutation invariant training (PIT) by using a deep learning neural network. This manner, we can train the deep learning tasks without requiring PIT at all. Permutation invariant training is required for deep learning applications like source separation and multi-source localization.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

In order to train the Hnet for multi-source localization, we must first generate the following data using the load predicted values obtained from LSTM model from the past step. We create a dataset with a training split of specific distance matrices D and their matching association matrices A in order to train the Hnet. 10% of the training split's size is allocated to validation. The highest number in the dataset, Nmax = 2, determines the fixed dimensions of D and A, which are the identical. By selecting reference and forecast load Direction of arrivals at random from spherical equiangular grids with resolutions of 1, 2, 3, 4, 5, 10, 15, 20, and 30 degrees, we sample an equal number of D matrices. The dataset equally represents all possible combinations of (number of predictions, number of reference) such as (0,0), (0,1), (1,0), (1,1), (1,2), (2,1), and (2,2). The distance pairs in D are formed using Euclidean distances as shown in equation 10.

$$\boldsymbol{ED} = \sqrt{(\boldsymbol{x1} - \boldsymbol{x2})^{\boldsymbol{2}} + (\boldsymbol{y1} - \boldsymbol{y2})^{\boldsymbol{2}}} \_\_\_\_\_ \quad (10)$$

Where,

ED- Euclidean distance of a specific row.

x1, x2, y1, and y2 are the labeled value of load direction of arrivals(DoA)

Random high distance values are assigned to the corresponding inactive entries due to padding D to Nmax _Nmax dimensions even when Mt;Nt < Nmax, making it easier for Hnet to determine the precise number of active load Direction of arrivals and their relationships. Thus Hnet obtains an F-score of >99% on any D data produced using the above specified specifications following training.

Step 9: H-net Training - The cornerstone of the proposed training technique based on differentiable tracking is known as Hnet. It estimates a dimension's association matrix {A, which is the same as the input distance matrix D. To train Hnet quickly and effectively, we choose a simpler design, as illustrated in Fig. 5, with three losses, as opposed to the deep Hungarian network. We employ a 128-unit gated recurrent unit (GRU) input layer, which interprets one of the input matrix D's two dimensions as the time-sequence and the other as the feature length. To find the time steps with the right associations, a single-head self-attention network is fed the GRU output time-sequence. A fully-connected network with sigmoid non-linearity processes the self-attention layer's output and estimates ~A as a multiclass, multilabel classification task.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

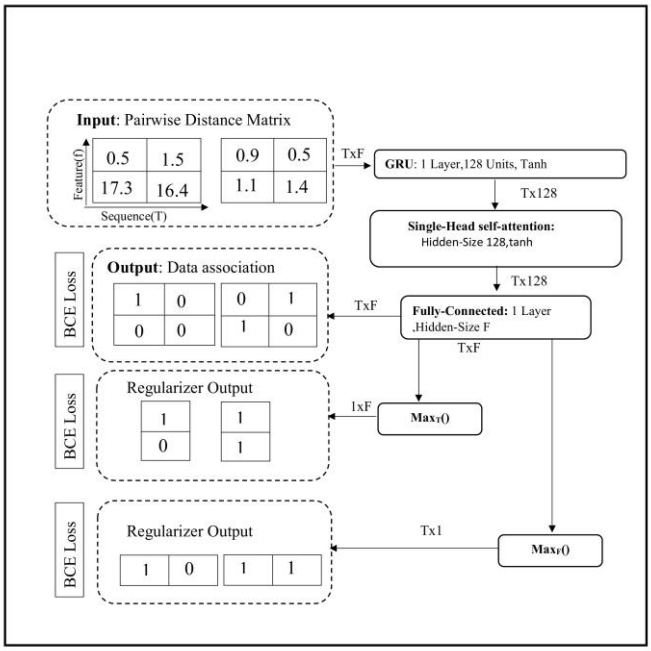

Figure 5: Block Diagram of Hungarian network

Furthermore, we perform max-operation on the fully-connected network's output (prior to the sigmoid non-linearity used to compute "A") along both the temporal (maxT()) and feature (maxF()) axes in order to direct the network to predict a maximum of one association per row and column, as is expected for associations resulting from the Hungarian algorithm. As multiple classes may be active in an output instance, we apply sigmoid non-linearity to their outputs. Lastly, using weighted combinations of the three losses—which are each calculated using binary cross-entropy between the predictions and the goal labels of A, maxT(A), and maxF(A)—the Hnet is trained in a multi-task framework.

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \quad \text{(11)}$$

$$\tanh x = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \text{(12)}$$

Where,

X is the input to a neuron

$f(x) = Tanh\ Activation\ Function$

e= Euler's Number

DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

Step 10: Decision Tree for Resource allocation – A sequence list for the upload process with the load factor is the result of the Hungarian neural network used in the preceding stage. This sequence list is the result of a thorough and innovative fusion of the Hungarian job allocation system in the H-net neural network. During this phase of the procedure, the sequence list will be employed to attain the better resource allocation results. This is an important statistic that will help with the real-time customization of the scheduling technique depending on the cloud resource characteristics. The decision-tree technique will use the decision of adaptive resource allocation scheme in order to achieve the better results to maintain the cloud properties intact.

IV RESULTS AND DISCUSSIONS

The proposed optimized model for cloud load prediction and resource allocation using deep learning runs on a Windows-based computer platform and is implemented in the Python programming language. Throughout the development process, the Spyder IDE has been employed using the Anaconda repository. The development laptop is equipped with a 500GB hard drive, 16GB of RAM, and an Intel Core i5 processor.

The effectiveness of cloud load prediction and resource allocation must be assessed in order to ensure the successful deployment of the system. As stated in the previous phase, this approach uses a dataset that is generated for cloud load prediction using an adoptive generator scheme, and then it is fed to the LSTM model to predict the load classes. The outcome of the load classes is utilized by the Hungarian net model to analyze the resource allocation scheme. The obtained results from both the LSTM model and the Hungarian net are subjected to vigorous testing to accept their outcomes as the novel ones with the below-mentioned evaluation tests.

A. Loss function Convergence trend estimation for Resource allocation using Deep learning model

A key component of the resource allocation mechanism for device-to-device communication is algorithm complexity. To demonstrate the benefits of the proposed Hungarian Network (Hnet) paradigm in terms of dependability and complexity. Comparative studies of model loss functions are carried out in the present investigation between our Hnet model and the double-chain deep deciduous MCTS, single-chain deep MCTS, and MCTS as described in [21]. This is done from the standpoint of the model's convergence value and speed. As the three models are tested as described in [21], the Hnet neural network is built for 2500 iterations in the experiment. A comparison graph of the loss function's convergence trend is shown in Figures 6 and 7. The trend graph shows that the Hnet model effectively

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

converges, and the ultimate convergence value is superior to that of the single-chain deep MCTS, double-chain deep deciduous MCTS, and MCTS, demonstrating the validity of the Hnet model put forward in this work.

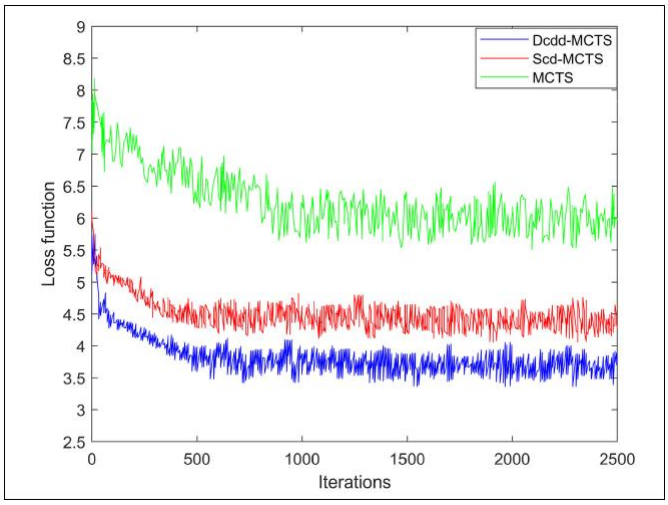

Figure 6. Comparison chart of loss function convergence trend [21]

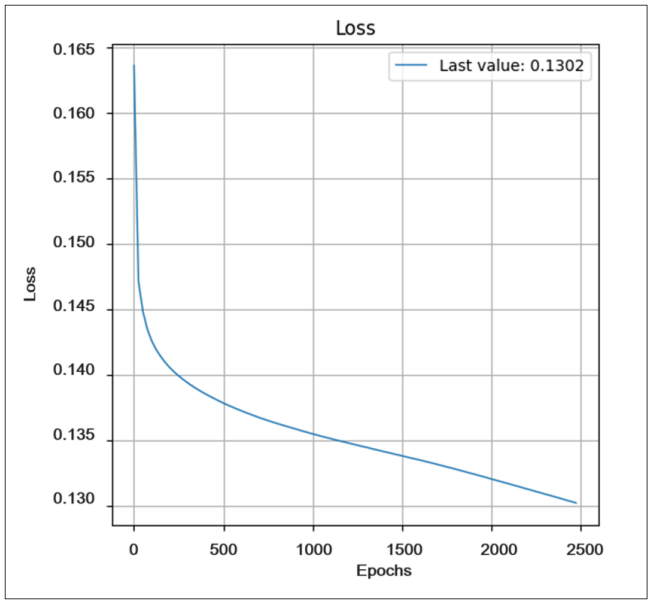

Figure 7: Loss function convergence of Hnet Model

B. Evaluation of Performance Using the Root Mean Square for Load forecasting

Multiple investigations have been carried out in order to quantify the error generated by the long short term neural network-based cloud load forecasting system, as follows. It is easy to examine the performance metrics because of the erroneous identification that the method for precisely determining the cloud load for the given input.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

To evaluate the model performance The Root Mean Square Error(RMSE) and Mean absolute error (MAE), and Mean absolute percentage error(MAPE) is utilized to make it feasible to calculate the error that the approach is being presented achieves. The accuracy of the proposed strategy's performance is demonstrated by the inaccuracy of the previously proposed method as stated in [22]. The RMSE approach simplifies the error evaluation between two continuously correlated metrics. The degree of load prediction accuracy and inaccuracy are the criteria taken into consideration for this procedure. The RMSE, MAE and MAPE formulae can be seen in equation 13, 14 and 15.

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n}(xi - xi(obs))2}{\text{n}}} \text{ ------ (13)}$$

$$MAE = \sqrt{\frac{\sum_{i=0}^{n}|(xi - xi(obs))|}{\text{n}}} \text{ ------ (14)}$$

$$MAPE = \frac{\sum_{i=0}^{n}\left(\frac{|xi - xi(obs)|}{|xi(obs)|}\right)}{n} \text{------ (15)}$$

Where,

$\sum$ - Summation

$(x_1 - x_{(obs)})^2$ - Variations Squared for the total of the predicted number of actual load predictions and the number of load predictions that were obtained.

n - Number of Trails

In [22] The many intricate problems that come up in short-term load forecasting are currently too difficult for a single deep learning model to handle. This research offers a unique multi-scale ensemble approach and multi-scale ensemble neural network to enhance the accuracy of short-term load forecasting.

The fundamental model of this neural network is a temporal convolutional network, gate recurrent units, and long short-term memory. These deep learning networks were formed from single-model scale and multi-model scale, respectively, by combining the stochastic weight averaging ensemble approach and differential evolution ensemble method, to form neural network using LSTM and ANN to call as contrast ensemble model (CEM). This CEM model works well when compare with the other models as mentioned in [22] for the parameter like RMSE and MAE.

On the other hand when this CEM is compared with our ensemble model of LSTM-HNet its performance seems to bit low. This is because LSTM-HNet models are tend to perform forecast load and allocate simultaneously. The Allocated resource information will adds to the next learning process of load forecasting thus it reduces the error

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

probability successfully. Hence, ensembling approach of LSTM- HNet out performs the CEM model mentioned in [22] and the results and the graphs are mentioned in table 1 and figure 8.

Models	RMSE	MAE
CEM1	294.638	221.701
CEM2	288.971	219.742
LSTM-HNet	178.968	161.234

Table 1: Comprehensive performance results of models for RMSE and MAE factors

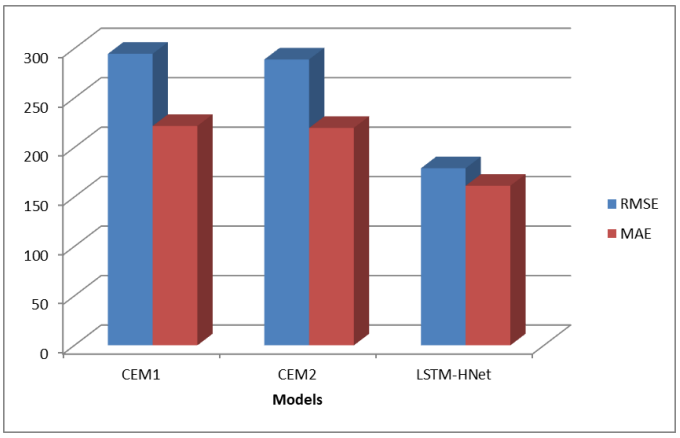

Figure 8: RMSE and MAE analysis results

[23] Uses a hybrid deep learning and beluga whale optimization (LFS-HDLBWO) technique to create a Short-Load Forecasting system. Predicting the load in the cloud's smart grid environment is the main goal of the LFS-HDLBWO approach.

This LFS-HDLBWO performs a little bit inferior for MAPE % when compared to our ensemble model of LSTM-HNet. This is due to the fact that LSTM-HNet models frequently forecast load and allocate at the same time. By adding to the subsequent load forecasting learning process, the allocated resource information successfully lowers the mistake chance percentage. As a result, the LSTM-HNet assembling strategy outperforms the LFS-HDLBWO model described in [23]. Table 2 and Figure 9 present the results and graphs.

DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

	MAPE %	
Data Size (samples)	LFS-HDLBWO	LSTM-HNet
300	0.228	0.199
480	0.266	0.191
600	0.254	0.188
720	0.059	0.031

Table 2: Performance results of models for MAPE %

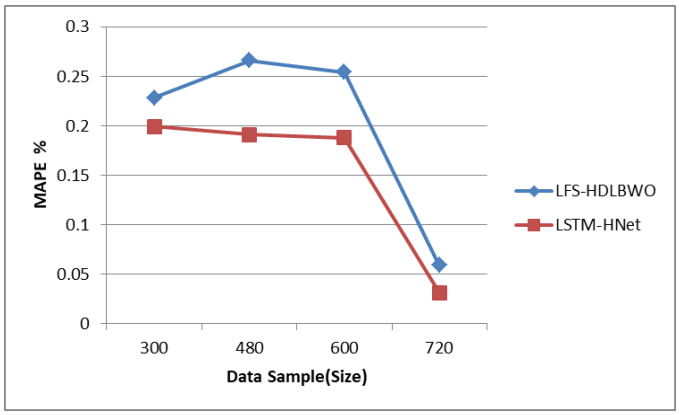

Figure 9: MAPE % analysis results

V CONCLUSION AND FUTURE SCOPE

Using a generator engine, the suggested model in this research first creates the cloud load data for the parameters like num_of_shards,num_of_samples, new_period, Shape, and size attributes. Data segmentation and preprocessing are applied at the time of data generation. Using this dataset, the Long Short Term Memory (LSTM) is used to produce a load forecast. The sequence of the prediction parameter list is produced by this load prediction, and it is utilized on the Hungarian net to determine the optimal resource allocation sequence. The sequence of resource allocation that is obtained facilitates the allocation of resources across n virtual machines.

Using LSTM-HNet models to forecast load and allocate concurrently, the load prediction model is triggered in a recursive way by the sequence of allocated resources. The allotted resource information effectively reduces the error rate % by enhancing the learning process of load forecasting. The result obtained makes it rather evident that the suggested model works better than that of [21] for the parameter convergence of the loss function as a result of the Hungarian Net model's effective deployment. The LSTM-HNet ensembling strategy outperforms the CEM model described in [22] by approximately 26.79% for the parameter of MAE and approximately 37.76% for the parameter of RMSE. The suggested model likewise outperforms [23] by around 24.75% for the MAPE parameter. This shows that

DOI: https://doi.org/10.37896/pd93.4/9342

the LSTM-HNet model's deployment, using a novel ensembling technique, performs satisfactorily for the load forecasting and resource allocation paradigm cloud.

Future studies aim to include additional workload and resource variables to refine the Deep learning's robustness operator and produce better solutions. Future research will examine the issue of creating a proactive task scheduling algorithm based on Evolution Strategies that balances load balancing, job assignment time, task completion time, and cost by ensembling the Hungarian Neural network with the other techniques like Genetic algorithm and Multi party computation in virtual servers of the cloud.

REFERENCES

[1] Xueying Guo, "Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm," in IEEE Transactions, Volume 60, Issue 6, December 2021, Pages 5603-5609 doi: 10.1016 j.aej.2021.04.051.

[2] X. Liu, X. Liu, Y. -S. Liu and Z. Han, "SPU-Net: Self-Supervised Point Cloud Upsampling by Coarse-to-Fine Reconstruction with Self-Projection Optimization," in IEEE Transactions on Image Processing, vol. 31, pp. 4213-4226, 2022, doi: 10.1109/TIP.2022.3182266.

[3] Ran Huang, Jingliang Zhang, Yuanzhen Hu, Shaojun Zou, Xidao Luan Jinbin Hu, Chang Ruan, Tao Zhang, "Coarse-Grained Load Balancing with Traffic-Aware Marking in Data Center Networks," in IEEE Transactions, Volume 2022, Article ID 9594517, 17 pages.

[4] J. Kosińska and K. Zieliński, "Experimental Evaluation of Rule-Based Autonomic Computing Management Framework for Cloud-Native Applications," in IEEE Transactions on Services Computing, vol. 16, no. 2, pp. 1172-1183, 1 March-April 2023, doi: 10.1109/TSC.2022.3159001.

[5] X. Wei and Y. Wang, "Joint Resource Placement and Task Dispatching in Mobile Edge Computing across Timescales," 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), Tokyo, Japan, 2021, pp. 1-6, doi: 10.1109/IWQOS52092.2021.9521283.

[6] T. Barbette, E. Wu, D. Kostić, G. Q. Maguire, P. Papadimitratos and M. Chiesa, "Cheetah: A High-Speed Programmable Load-Balancer Framework with Guaranteed Per-Connection-Consistency," in IEEE/ACM Transactions on Networking, vol. 30, no. 1, pp. 354-367, Feb. 2022, doi: 10.1109/TNET.2021.3113370.

 DOI: https://doi.org/10.37896/pd93.4/9342

[7] K. Hosono, A. Maki, Y. Watanabe, H. Takada and K. Sato, "Implementation and Evaluation of Load Balancing Mechanism With Multiple Edge Server Cooperation for Dynamic Map System," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 7270-7280, July 2022, doi: 10.1109/TITS.2021.3067909.

[8] A. Suzuki, M. Kobayashi and E. Oki, "Multi-Agent Deep Reinforcement Learning for Cooperative Computing Offloading and Route Optimization in Multi Cloud-Edge Networks," in IEEE Transactions on Network and Service Management, vol. 20, no. 4, pp. 4416-4434, Dec. 2023, doi: 10.1109/TNSM.2023.3267809.

[9] T. Rios, B. van Stein, T. Bäck, B. Sendhoff and S. Menzel, "Multitask Shape Optimization Using a 3-D Point Cloud Autoencoder as Unified Representation," in IEEE Transactions on Evolutionary Computation, vol. 26, no. 2, pp. 206-217, April 2022, doi: 10.1109/TEVC.2021.3086308.

[10] Y. Nakatani, "Structured Allocation-Based Consistent Hashing With Improved Balancing for Cloud Infrastructure," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 9, pp. 2248-2261, 1 Sept. 2021, doi: 10.1109/TPDS.2021.3058963.

[11] S. Souravlas, S. D. Anastasiadou, N. Tantalaki and S. Katsavounis, "A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling," in IEEE Access, vol. 10, pp. 26149-26162, 2022, doi: 10.1109/ACCESS.2022.3157435.

[12] M. S. Al Reshan et al., "A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing," in IEEE Access, vol. 11, pp. 11390-11404, 2023, doi: 10.1109/ACCESS.2023.3241279.

[13] P. Loncar and P. Loncar, "Scalable Management of Heterogeneous Cloud Resources Based on Evolution Strategies Algorithm," in IEEE Access, vol. 10, pp. 68778-68791, 2022, doi: 10.1109/ACCESS.2022.3185987.

[14] P. V. Lahande, P. R. Kaveri, J. R. Saini, K. Kotecha and S. Alfarhood, "Reinforcement Learning Approach for Optimizing Cloud Resource Utilization With Load Balancing," in IEEE Access, vol. 11, pp. 127567-127577, 2023, doi: 10.1109/ACCESS.2023.3329557.

[15] B. Saemi, A. A. R. Hosseinabadi, A. Khodadadi, S. Mirkamali and A. Abraham, "Solving Task Scheduling Problem in Mobile Cloud Computing Using the Hybrid Multi-Objective Harris Hawks Optimization Algorithm," in IEEE Access, vol. 11, pp. 125033-125054, 2023, doi: 10.1109/ACCESS.2023.3329069.

[16] L. Zhang, J. Bai and J. Xu, "Optimal Allocation Strategy of Cloud Resources With Uncertain Supply and Demand for SaaS Providers," in IEEE Access, vol. 11, pp. 80997-81010, 2023, doi: 10.1109/ACCESS.2023.3300735.

 DOI: https://doi.org/10.37896/pd93.4/9342

ISSN: 0369-8963

[17] R. Gao, X. Xie and Q. Guo, "K-TAHP: A Kubernetes Load Balancing Strategy Base on TOPSIS+AHP," in IEEE Access, vol. 11, pp. 102132-102139, 2023, doi: 10.1109/ACCESS.2023.3313643.

[18] A. H. Alomari, S. K. Subramaniam, N. Samian, R. Latip and Z. A. Zukarnain, "Dual-Phase Resource Allocation Algorithm in Software-Defined Network SDN-Enabled Cloud," in IEEE Access, vol. 11, pp. 102301-102315, 2023, doi: 10.1109/ACCESS.2023.3315856.

[19] M. Ouyang, J. Xi, W. Bai and K. Li, "Band-Area Resource Management Platform and Accelerated Particle Swarm Optimization Algorithm for Container Deployment in Internet-of-Things Cloud," in IEEE Access, vol. 10, pp. 86844-86863, 2022, doi: 10.1109/ACCESS.2022.3198971.

[20] H. Zhang and R. Jia, "Application of Chaotic Cat Swarm Optimization in Cloud Computing Multi Objective Task Scheduling," in IEEE Access, vol. 11, pp. 95443-95454, 2023, doi: 10.1109/ACCESS.2023.3311028.

[21] X. Li and G. Chen, "D2D Cooperative Communication Network Resource Allocation Algorithm Based on Improved Monte Carlo Tree Search," in IEEE Access, vol. 11, pp. 72689-72703, 2023, doi: 10.1109/ACCESS.2023.3280604.

[22] Q. Shen, L. Mo, G. Liu, J. Zhou, Y. Zhang and P. Ren, "Short-Term Load Forecasting Based on Multi-Scale Ensemble Deep Learning Neural Network," in IEEE Access, vol. 11, pp. 111963-111975, 2023, doi: 10.1109/ACCESS.2023.3322167.

[23] M. M. Asiri, G. Aldehim, F. A. Alotaibi, M. M. Alnfiai, M. Assiri and A. Mahmud, "Short-Term Load Forecasting in Smart Grids Using Hybrid Deep Learning," in IEEE Access, vol. 12, pp. 23504-23513, 2024, doi: 10.1109/ACCESS.2024.3358182.

 DOI: https://doi.org/10.37896/pd93.4/9342