International Journal of Advance Research in Science and Engineering
Volume No. 13, Issue No. 06, June 2024
www.ijarse.com
IJARSE
ISSN 2319 - 8354

Cloud management with automatic self-healing services: a comprehensive study

[1,2]Saket Suvalal Bhalgat, Dr. P. B. Kumbharkar

[1,2]Computer Engineering, JSPM's Rajarshi Shahu College of Engineering, Pune, India

ABSTRACT

Now a days all the applications are deployed on the cloud because of the CI/CD (continuous integration and continuous delivery/continuous deployment) capability of cloud computing. In order to manage a large number of applications simultaneously, cloud infrastructure must be massive. In order to keep the cloud running well, it is necessary to keep an eye on the resources and the stream of incoming traffic. This has led to a dramatic increase in the complexity of cloud systems as they continue to grow in size and integrate more and more resources. There is an increasing demand for autonomous systems and services to manage these complex and heterogeneous cloud infrastructures, particularly at higher scales. While there are a number of systems that can analyze the threshold and handle these loads, the majority of them require human intervention to manage the resources and, by extension, the load. Therefore, self-healing or self-management of the cloud load is currently trending heavily. In this research, we examine the different approaches with a primary focus on self-healing as a key component of autonomous cloud recovery management. After that, this study suggests a method that uses a layered structure to self-heal the cloud load to a good amount, which increases reliability and guarantees high availability inside a decentralized, hierarchical cloud architecture.

***Keywords*:** *Cloud Computing, Cloud Management, Resource allocation, Load balancing, Self-healing services*

I. INTRODUCTION

In the realm of online crowdfunding platforms, there exists a pressing need for an alternative methodology to address inherent shortcomings. Presently, crowdfunding websites host myriad projects soliciting funds from contributors who donate at their discretion. However, a glaring flaw lies in the lack of trust engendered by this system. Instances abound where

In this day and age of cloud computing, the integration of diverse resources on an ever-expanding scale has resulted in a level of complexity in cloud settings that has never been seen before. In light of the fact that businesses are increasingly relying on the cloud to fulfil their diverse computing requirements, it is of the utmost importance to manage these complex infrastructures in an effective and dependable manner. There is a rising reliance on systems and services that feature autonomic behaviours, which enable them to adapt and respond autonomously to changing situations. This is done in order to overcome the issues that are provided by the complexity and scale of modern cloud environments.

The Self-Healing Automated Services System is the vehicle through which this project explores the field of autonomic cloud administration. The project has a particular emphasis on self-healing mechanisms. In order to

guarantee the robustness and resilience of cloud infrastructures, self-healing is an essential component that must be included. We intend to improve the overall dependability and availability of these systems by incorporating principles of self-healing into the framework that governs cloud management.

If we want our distributed system to be able to fix itself in the event of a breakdown or malfunction, we need to build in mechanisms that can do so automatically. Distributed systems with built-in self-healing capabilities often employ the following methods and approaches:

1. Redundancy: If important parts or data are duplicated over several nodes, the system can keep running even if some of those nodes fail. Tools like load balancers, server clustering, and data replication can help with this.

2. Enforcing Health Checks and Monitoring: By including ongoing health checks and monitoring of system components, issues or abnormalities can be identified in real-time. For system health and performance monitoring, we can use tools like Prometheus, Grafana, Nagios, or DataDog.

3. Automated Recovery: Establish protocols for automatic recovery that can be activated in the event of a failure. It may be necessary to reload malfunctioning parts, reassign resources, or switch to backup systems.

4. Intelligent Algorithms for Self-Healing: Incorporate algorithms with the ability to assess the current condition of the system, forecast when faults may occur, and proactively take steps to avoid or reduce their impact. Predictive analytics, machine learning, and anomaly detection are all applicable here.

5. Dynamic Configuration: To store and disseminate configuration information across the distributed system, use dynamic configuration management solutions such as Consul, etcd, or ZooKeeper. This paves the way for automated reconfiguration in the event of system breakdown or change.

6. Degradation: Design the system such that it gently degrades its usefulness instead of crashing or becoming unavailable when failures occur. In order to keep crucial services running, it may be necessary to reduce the system's burden.

Cloud self-healing approach can be employed in many other factors like

A) First, make sure it can execute our production object or byte code files in a simulated environment with an artificial time basis. Then, we can test our fault handling code by injecting errors, varying the timing, and validating it against a model.

The vast majority of distributed systems fail miserably when put into practice. My first exposure to the issue was a paper by Microsoft called MODIST: Transparent Model Checking of Unmodified Distributed Systems. It provided a quantitative assessment of the problem and revealed that all three of the commercial distributed systems they examined—a 50,000 line system, which is not particularly large—running on 100,000 nodes—a system that is likely very important to the business——had protocol errors.

Although the problem is much simpler at a higher level (send message, receive message, obtain time, schedule timed event, shutdown), they show that we can simulate at a low level by intercepting system and library calls. If a component's messaging and persistence APIs are independent, we can simulate a crash by turning them off and then turning them back on again.

In Drew Eckhardt's response to the question "How does Drew Eckhardt architect/design a distributed system that can be deterministically simulated in a single thread/process in order to test it?" I detail the approach, providing code links on GitHub.

B) Designed to allow for quick recuperation and restarting. Dependability is inversely related to recovery time and directly proportionate to the average interval between failures. Factors beyond our control (hardware, firmware, the thirty third-party libraries we link against, etc.) and things we can't predict (for example, I've witnessed hard drive vibrations interacting with enclosure/mounting hardware to cause premature failures) limit the ability to improve MTBF. We can control and test improvements in the time to recovery, and those are the areas where we will see increases of orders of magnitude.

One of the foundations of David Patterson's Recovery Oriented computing, as outlined in Embracing Failure: A Case for Recovery-Oriented Computing (ROC), is a focus on recovery for these reasons. Our partitioning scheme is also relevant here. If the source and destination are both numerous nodes, as in consistent hashing, then it is significantly faster to build another copy.

C) Make failure domains as tiny as possible and enable restarts inside each. A block storage appliance, for example, may use multiple processes: one for each drive, another for cluster administration, and yet another for translating iSCSI to local protocol. ROC: Motivation, Definition, Techniques, and Case Studies covers all of these and more.

D) Consider incorporating background processes that scan for hidden problems. This way, we may identify and fix them before they cause further damage. On-disk structures that haven't been accessed since last year's restart become unreadable due to bitrot, as I've seen it happen before.

E) Minimize potential failure spots. The largest of them is our program, but we can run diverse configurations to reduce the likelihood of hitting the same edge circumstances on several nodes simultaneously. B+ tree fanouts and checkpoint intervals are two examples of things that can change.

F) Handle failures in a sequential fashion. While many "independent" failures do not really occur at the same time, many "simultaneous" failures do. As an example, consider a primary copy replication setup that experiences a power loss event. One copy goes down first, and then there's IO, which makes the first copy non-authoritative. The second copy also goes down, but this time the second copy is damaged and doesn't survive the power event. Perhaps consensus might provide a temporary home for IOs until we can get replicas or develop new ones, allowing us to avoid that. We can do better than primary copy replication in many circumstances, but it works for minimal sellable products and gets us far with many consumers.

G) Pay close attention to the movement of load when a failure occurs. Instead of straightforward mirroring, which can result in a 100% increase in load, consistent hashing, where the load increase is around n / (n + 1), may be preferable.

H) Obviously, leave out any unnecessary uniformity when designing.

[1] This work proposes a new paradigm for job loading in cloud data centers based on adaptive scheduling algorithms, as proposed by Dibyendu mukherjee et al. In this paper, author provide ASA-TL, an innovative technique for adaptive scheduling algorithm-based task loading. The cloud provider is provided with the suggested ASA-TL algorithm in this architecture. In order to avoid the over-load problem, the suggested algorithm adaptively schedules the access requests that are delivered to the cloud. By utilizing virtual servers, this technique successfully accomplished load balancing. Minimum task completion time of 16.76 ms, maximum data transmission rate of 89.81%, minimum processing time of 5.25 ms, minimum overall expense rate of 5.36%, and

high load balancing rate of 94.02% are all features of the suggested ASA TL archives. One day, the author hopes to put the suggested method into action by utilizing virtual software resources on a cloud server.

In order to offer industrial automation systems with real-time supervisory control as stated by Wenbin Dai et al. in [2] Suggested a DSS that is cloud-based. In order to make decisions in real-time, the ASM and knowledge base are moved to the cloud from PLCs. Integrating FTA methods into the cloud-based DSS enables one of the self-management capabilities, self-healing. In order to keep the system running, the FTA and the ASM make the required configuration updates. Future study will address the various constraints of the cloud-based DSS. To begin, model-driven engineering techniques will be used to import fault tree models straight from the design documentation. The current method requires engineers to manually create computer models of fault trees. Automatic fault tree generation is possible with the use of model-driven engineering techniques. The second point is that setting appropriate upper and lower bounds for each analog input could drastically cut down on the amount of concurrent simulation runtimes.

One schedule optimization method that improves cloud computing's load balancing is dynamic resource allocation, as described by Ali moazeni et al. [3]. Both cloud service providers and their clients place a high value on the allocation of cloud resources. One of the primary goals of cloud providers is to maximize profit, while customers typically experience low-cost and fast processing of requests. This means that competing goals must be satisfied all at once. Cloud computing resource allocation is the focus of this work, which proposes an optimization technique based on adaptive multi-objective teaching and learning. Goals like decreasing makespan and overall cost while boosting resource usage are met in this method. To top it all off, the suggested algorithm makes use of an adaptive teaching component in the TLBO; this component can change throughout search based on student and instructor results, ultimately leading to better algorithm performance. Furthermore, the non-dominated solutions stored in an external archive are adaptively evaluated using a grid-based approach.

[4] In their discussion of cloud data centres (CDCs), Haitao Yuan et al. highlight how numerous heterogeneous applications with delay constraints can be run cost-effectively. A CDC's energy usage is boosted substantially by its ever-increasing workload. Electricity prices, the maximum number of available servers, and the speeds at which tasks arrive in different CDCs are only a few of the variables that vary from one site to another. Minimizing average energy costs while optimizing tasks for quality of service is a challenging balancing act. In this paper, author offer a way to help CDCs and users achieve a win-win situation by balancing these two competing goals. In particular, the author's recently introduced adaptive bi-objective differential evolution using simulated annealing is applied to a specific objective problem. For the knee, the minimal Manhattan distance method yields the ultimate solution. Its energy cost and average loss probability of tasks are lower than some of its counterparts, according to experimental results. Using generative adversarial networks—which generate dominated and nondominated solutions in an adversarial manner—the authors intend to improve the convergence speed and diversity of solutions obtained in future work. Then, for optimization problems involving many objectives, the distribution of the sets of Pareto-optimal solutions could be learned using approximated non-dominated fronts.

The following is the structure of the literature review that is included in this study: In the second section, which has the form of a literature review and examines past work, the third section outlines the technique that need to

be employed, and the fourth section makes suggestions for areas that require additional examination with the conclusion and future work.

II LITERATURE SURVEY

[5] Shuo Qin et al. This study delves into the cloud workflow scheduling problem with cost minimization and deadline constraints. Keeping the T EC as low as possible while still meeting the deadline is the goal. In order to address the problem at hand, a framework called KADWWO is created, which stands for knowledge-based adaptive discrete optimization. This framework is tailored to the specifics of the situation at hand. To conduct exploration by mining the idle time knowledge edge of resources, KADWWO employs a discrete propagation operator. After then, in order to prevent stagnation, a new adaptive refraction operator is created. In addition, exploitation through mining the block structure knowledge edge is carried out using the dynamic grouping based breaking operator. The DOE method is used to calibrate the parameters. Lastly, five state-of-the-art algorithms that were developed recently are compared to the proposed KADWWO. The evaluation is based on the CyberShake, Montage, Epigenomics, Inspiral, and Sipht processes, which have varied scales and deadlines. Both the experimental and statistical results show that, with a 95% confidence level, the proposed KADWWO performs far better than the state-of-the-art controls. In addition, the proposed KADWWO works well and is resilient enough to handle the issue at hand.

Task scheduling is an important NP-hard problem in cloud computing, as described by Zulfiqar ali khan et al. [6], which affects the effectiveness and use of resources in cloud datacenters. One of the most important factors in improving service quality is efficient scheduling. Algorithms that rely on heuristics, such DRALBA, OG-RADL, and SG-PBFS, etc., to plan operations on a group of resources in a way that minimizes makespan by rating resources or prioritizing activities. On the other hand, meta heuristic algorithms are fair in their treatment of tasks and resources. Nevertheless, for a meta-heuristic algorithm to produce the best possible outcome, it is crucial to strike the correct balance between global and local search. This motivates the development of the Parallel Enhanced Whale Optimization Algorithm (PEWOA), a method for scheduling autonomous work among diverse cloud virtual machines. To boost convergence, eliminate local optima, and increase solution variety, PEWOA uses parallelization, an updated encircling maneuver, and a bubble net assaulting mechanism. By efficiently balancing exploration and exploitation, the solution quality was optimized by the enhanced encircling maneuver and bubble net assaulting mechanism.

[7] Yes. Researchers Shin et al. This study proposes the MARS scheme, an evolution of the CTMC scheme. Although the CTMC method had no problem accurately allocating virtual machines, it was slow to do so, which led to inflexibility. The analysis also shows that ML systems based on MA time-series forecasting do not do well when dealing with bursty scenarios. By accurately deriving the ideal number of VMs using an easily-computable KKT solution value, the proposed MARS approach solves the shortcomings of prior techniques. You can use the MARS system to assign virtual machines in real time based on the processing capability, memory size, and cost of the edge cloud in either a private or public cloud. To accomplish optimal resource allocation with elasticity and precision, future research should concentrate on ways to achieve energy savings by breaking IIoT jobs into

subtasks and offloading those subtasks to adjacent edge clouds. These clouds can support the central cloud, which is often further away, while still achieving energy savings.

Soravlas Stavros et al., [8]. Task scheduling with load balancing was the subject of this work. The Markov process model and a basic fair task allocation method form the basis of the author's system model. The author derives the anticipated VM utilisation from the equilibrium state probabilities. To ensure that all virtual machines (VMs) are anticipated to use the same amount of resources, the author's fair task allocation policy is applied on a time slot basis. This approach consistently ensures load balancing, as demonstrated by the author. Because it improves many critical metrics—makespan, average usage, degree of imbalance, and response time—the proposed system is multi-objective. In terms of the aforementioned metrics, the author's task allocation strategy outperformed three new state-of-the-art schemes: the Load Balancing Modified Particle Swarm Optimization (LBMPSO) scheme, the Honey bee foraging (HBF) with VM pre-selection, and the community-based Particle Swarm Optimization (CPSO) scheme. The optimization problem requires additional research. The work of the author has demonstrated load balancing and yields satisfactory results for many metrics across various distribution situations; nevertheless, the author has not yet demonstrated optimality.

[9] Mana Saleh Al Reshan et al. In order to maintain server availability, this study presented a hybrid fast-converging load balancing method that achieved globally optimal quick convergence for VM load balancing. Due to GWO's low dependency on the control parameters, its implementation is simplicity itself. When compared to BAT and GWO, ABC, PSO, and SSO algorithms converge slowly, meaning they have a high response rate. However, as user retention improves, BAT's performance degrades. The outcomes demonstrate that GWO is superior for convergence while PSO is suitable for global optimization. Therefore, PSO and GWO are combined to increase the likelihood of faster convergence, which could lead to the globally optimum solution. Because of the rapid convergence, the scheduling method can handle competing requests for virtual machines.

To address the issue of dynamic resource allocation in cloud computing, a scheduling solution that is based on the Symbiotic Organism Search algorithm (MOSOS) was suggested by Ali Belgacem et al. [10]. MOSOS's stated goal was to reduce costs and makespan for better dynamic resource distribution in cloud settings. To that end, the author began by providing definitions of the terms employed in this study: cost and makespan. The author also took into account the fact that virtual machines employ two distinct resources: random access memory (RAM) and central processing unit (CPU). The author also suggested a dynamic model to handle dynamic task scheduling and emphasize the continuity method of resource allocation. The author's suggested method was tested with typical workload logs collected from a distributed system in operation. The author used the pricing structure of the virtual machines found in the Amazon cloud. The results of the experiments demonstrate that MOSOS outperforms and outstabilizes FCFS, PSO, MOGA, and PBACO algorithms. The author is eager to address the dynamic resource allocation issue in a cloud context by taking additional limitations and other considerations into account.

According to Arunkumar arulappan et al. [11], six top-tier algorithms are trained on a thousand episodes and then assessed for speed, accuracy, cost, and incentives. While the PPO+MC agents do increase the overhead for resource efficiency during autoscaling operations, they only improve the cost by 1.7%. With a 3.2% improvement

in cost, TRPO+GAE agents are decent at auto-scaling. On the other hand, the author's suggested DQN/DDQN learning method optimises the pricing and reduces the cost by 18.4%. Compared to the baseline of TRPO+GAE and other comparative state-of-the-art algorithms, the adaptive self-healing of VNFs improves computing performance by around 27%. The TensorForce framework is used to create these RL algorithms in Python, and their performance is compared according to stability and cost.

According to what Kee Kim et al. [12] say, maintaining the performance of cloud applications becomes very difficult when resource storms occur in enterprise cloud systems. Orchestra, a cloud-specific framework for user-space FG and BG control, was introduced by the author to address this issue; it guarantees FG performance while minimizing BG performance penalty. On the fly, Orchestra generates lightweight RT estimation and performance models for both applications, and it measures the RT of a FG in real-time. Orchestra optimizes the distribution of resources to many cloud apps with SLA targets using resource analytics and prediction models. the author has used Amazon Elastic Compute Cloud (EC2) to deploy and assess Orchestra with production workloads. Web services and NoSQL databases (MongoDB) for FGs and backup (AWS Sync) and malware scanning (ClamAV) for BGs are the main workloads for the author.

[13] Boonhatai Kruekaew et al. This article presents the MOABCQ technique as a scheduling strategy for heterogeneous cloud computing that aims to optimize many objectives simultaneously. This approach took into account the selection of appropriate VMs by computing their fitness. Also included were heuristic techniques, like FCFS and LJF. In order to evaluate the suggested algorithms, experiments were carried out using different datasets. Compared to Max-Min, FCFS, Q-learning, HABC_LJF, MOPSO, and MOCS algorithms, the suggested solution improves load balancing jobs with existing system resources and reduces makespan, DI, cost, and boosts throughput and ARUR. Results from the experiments showed that the suggested approach was the best of its kind. The author does not, however, claim that MOABCQ_LJF is the best algorithm available. However, not every test dataset will provide optimal system performance. Work in the future may present fascinating challenges when it comes to task scheduling in multi-cloud, fog cloud, or edge cloud environments. In various settings, the author suggests a timetable arranging strategy. Additional applications of machine learning techniques are also possible. It is also possible to evaluate the suggested method in a real-world setting to see how well the MOABCQ method works.

[14] Longchang zhang et al. Prior research on infrastructure as a service (IaaS) resource allocation in the context of service quality of service (QoS) constraints at the IaaS and platform as a service (PaaS) levels has focused on issues like low resource utilization due to uncertainty in user access and virtual resource provisioning (i.e., IaaS resource load), but has ignored SaaS providers' revenue. In order to overcome these challenges, this study presents an optimal allocation strategy for infrastructure as a service (IaaS) resources that maximizes the revenue that SaaS providers can expect to earn. It then goes on to design three such strategies: one with demand and supply uncertainties, one with only supply uncertainties, and one with both types of uncertainties. This study's method is highly efficient and can efficiently acquire the optimal allocation of IaaS resources, according to the experimental analysis.

[15] Patryk Osypanka et al. An approach to minimizing expenses associated with cloud resources is laid out in this piece of writing. No external adjustment is required for the author's technique to run autonomously in closed-

loop setup. data derived from a manufacturing system was utilized by the author. By reducing the use and expense of cloud resources, tests demonstrate that the author's solution is effective and that the estimated savings are substantial. By comparing the existing system costs with those after optimization, author can see that the solution would have reduced expenses by 85%, or 6,128 euros, had it been applied to the working system during the 10 months of testing. The author's proposed solution is to decrease the expense of utilizing cloud resources through the use of demand prediction and provisioning adjustments. Thus, the author's solution can be applied to optimize any cloud-based system that makes use of scalable resources, such as Infrastructure as a service, platform as a service, or software as a service initiatives. Although understanding the inner workings of the system being optimized is not necessary for resource allocation optimization, the author's solution will capture any performance gains in this system and lead to future reductions in resource provisioning.

In [16], the authors are Saurabh Singhal and colleagues. For the purpose of cloud load balancing, this article suggests a Rock Hyrax optimization approach. This approach improves energy efficiency and decreases total makespan time by distributing jobs among numerous virtual machines according to availability and current load. Even in widely dispersed data centers, the algorithm effectively controls resource usage. The performance of load balancing methods is affected, but it resolves the issue of local maxima. Both static and dynamic implementations of virtual machines and workloads have shown success with the Rock Hyrax method. The approach saves 8%-13% of total energy usage and 10%-15% of makespan when compared to alternative load balancing methods. This provides more evidence that the Rock Hyrax algorithm works as advertised, boosting data center efficiency by making jobs and virtual machines run more smoothly.

[17] According to S.V. A. Kumer et al., CC isn't complete without Cloud TS, and timely service delivery depends on optimizing data transfer. There has been a lot of study on cloud task scheduling with a single aim, but there has been more recent interest in scheduling difficulties with multiple objectives. In order to optimize bandwidth allocation, decrease network congestion, avoid bottlenecks, and boost system performance, this research suggests hybrid POA algorithms. These algorithms can efficiently schedule cloud jobs. If you want your cloud data transfers and scheduling to run smoothly, you need to optimize make span, throughput, execution time, cost, and latency. A Python experiment proves that the newly developed technique guarantees efficient and stable secure task scheduling. As a result, cloud providers can better prioritize data transfers according to importance and urgency using the suggested method. In sum, the research provides helpful information about cloud job scheduling with many objectives and suggests efficient algorithms to improve cloud work scheduling, data transfer, and data security. To improve the efficacy of cloud computing systems and tackle new issues, future development will center on a few critical areas.

III PROPOSED MODEL

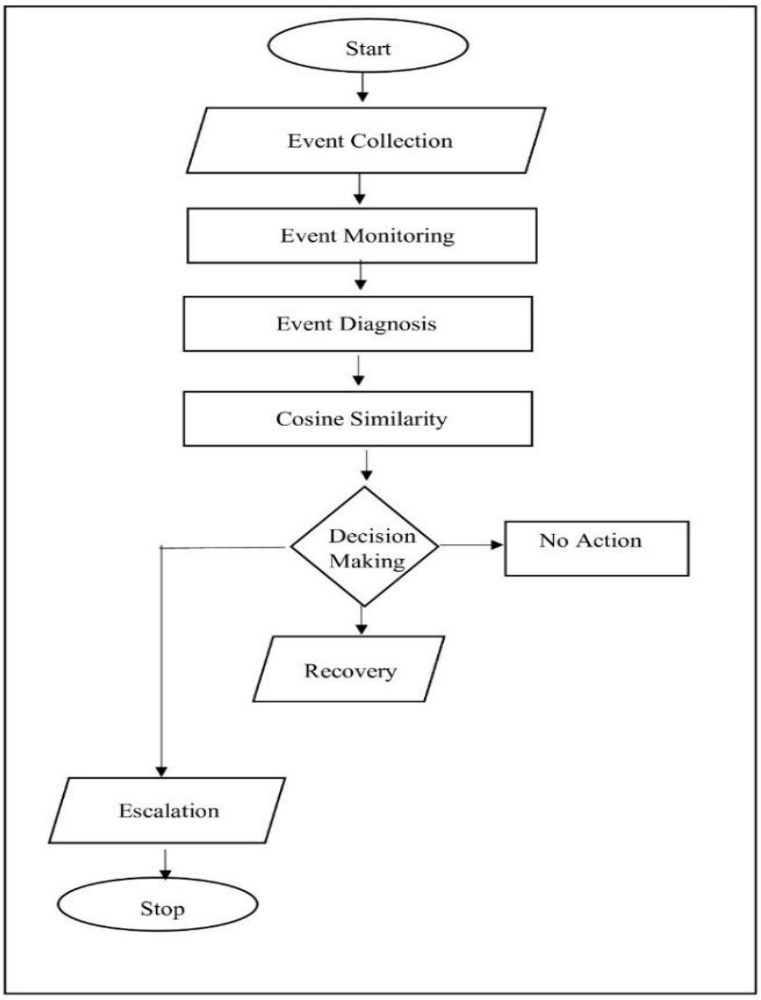

Figure 1: Proposed model

An illustration of the flow of the proposed model in self-healing cloud management is provided by the technique that is depicted in the flowchart of figure 1. In the beginning, the event traffic is collected, which entails the continuous monitoring of the cloud load through the collection of patterns of incoming traffic. Following the completion of the monitoring of the incoming data, we make use of cosine similarity in order to diagnose it and comprehend prior states. The decision about traffic self-healing is also guided by cosine, which determines whether the action should be revoked, whether it should be subjected to recovery, or whether it should be forwarded for escalation.

IV CONCLUSION AND FUTURE SCOPE

This review paper mainly concentrated on maintaining cloud resources using the self-healing technique to enrich load balancing, optimize the resources, and boost the overall performance of cloud services. Many works related to cloud resource management, cloud load balancing, and self-healing services are referred to and studied in depth to understand the requirements of the current scenario. The earlier studies led to the concept of self-healing cloud

services, which involve constant monitoring of the cloud load through the collection of incoming traffic patterns. Once we monitor the incoming traffic, we use cosine similarity to diagnose it and understand its past state. Cosine similarly guides the decision on traffic self-healing, determining whether to revoke the action, subject it to recovery, or forward it for escalation.

We can enhance the proposed model for future deployment by implementing deep learning models to predict traffic patterns more efficiently, learn new traffic patterns automatically, manage resources, and maintain cloud load balancing through enriched self-healing techniques.

REFERENCES

[1] D. Mukherjee, S. Ghosh, S. Pal, A. A. Aly and D. -N. Le, "Adaptive Scheduling Algorithm Based Task Loading in Cloud Data Centers," in IEEE Access, vol. 10, pp. 49412-49421, 2022, doi: 10.1109/ACCESS.2022.3168288.

[2] W. Dai, L. Riliskis, P. Wang, V. Vyatkin and X. Guan, "A Cloud-Based Decision Support System for Self-Healing in Distributed Automation Systems Using Fault Tree Analysis," in IEEE Transactions on Industrial Informatics, vol. 14, no. 3, pp. 989-1000, March 2018, doi: 10.1109/TII.2018.2791503.

[3] A. Moazeni, R. Khorsand and M. Ramezanpour, "Dynamic Resource Allocation Using an Adaptive Multi-Objective Teaching-Learning Based Optimization Algorithm in Cloud," in IEEE Access, vol. 11, pp. 23407-23419, 2023, doi: 10.1109/ACCESS.2023.3247639.

[4] H. Yuan, J. Bi and M. Zhou, "Energy-Efficient and QoS-Optimized Adaptive Task Scheduling and Management in Clouds," in IEEE Transactions on Automation Science and Engineering, vol. 19, no. 2, pp. 1233-1244, April 2022, doi: 10.1109/TASE.2020.3042409.

[5] S. Qin, D. Pi, Z. Shao and Y. Xu, "A Knowledge-Based Adaptive Discrete Water Wave Optimization for Solving Cloud Workflow Scheduling," in IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 200-216, 1 Jan.-March 2023, doi: 10.1109/TCC.2021.3087642.

[6] Z. A. Khan, I. A. Aziz, N. A. B. Osman and S. Nabi, "Parallel Enhanced Whale Optimization Algorithm for Independent Tasks Scheduling on Cloud Computing," in IEEE Access, vol. 12, pp. 23529-23548, 2024, doi: 10.1109/ACCESS.2024.3364700.

[7] Y. Shin, W. Yang, S. Kim and J. -M. Chung, "Multiple Adaptive-Resource-Allocation Real-Time Supervisor (MARS) for Elastic IIoT Hybrid Cloud Services," in IEEE Transactions on Network Science and Engineering, vol. 9, no. 3, pp. 1462-1476, 1 May-June 2022, doi: 10.1109/TNSE.2022.3145876.

[8] 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.

[9] 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.

[10] A. Belgacem, K. Beghdad-Bey and H. Nacer, "Dynamic Resource Allocation Method Based on Symbiotic Organism Search Algorithm in Cloud Computing," in IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 1714-1725, 1 July-Sept. 2022, doi: 10.1109/TCC.2020.3002205.

[11] A. Arulappan, A. Mahanti, K. Passi, T. Srinivasan, R. Naha and G. Raja, "DQN Approach for Adaptive Self-Healing of VNFs in Cloud-Native Network," in IEEE Access, vol. 12, pp. 34489-34504, 2024, doi: 10.1109/ACCESS.2024.3365635.

[12] I. K. Kim, J. Hwang, W. Wang and M. Humphrey, "Guaranteeing Performance SLAs of Cloud Applications Under Resource Storms," in IEEE Transactions on Cloud Computing, vol. 10, no. 2, pp. 1329-1343, 1 April-June 2022, doi: 10.1109/TCC.2020.2985372.

[13] 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.

[14] 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.

[15] P. Osypanka and P. Nawrocki, "Resource Usage Cost Optimization in Cloud Computing Using Machine Learning," in IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 2079-2089, 1 July-Sept. 2022, doi: 10.1109/TCC.2020.3015769.

[16] S. Singhal et al., "Energy Efficient Load Balancing Algorithm for Cloud Computing Using Rock Hyrax Optimization," in IEEE Access, vol. 12, pp. 48737-48749, 2024, doi: 10.1109/ACCESS.2024.3380159.

[17] S. V. A. Kumer, N. Prabakaran, E. Mohan, B. Natarajan, G. Sambasivam and V. B. Tyagi, "Enhancing Cloud Task Scheduling With a Robust Security Approach and Optimized Hybrid POA," in IEEE Access, vol. 11, pp. 122426-122445, 2023, doi: 10.1109/ACCESS.2023.3329052.