# OPTIMIZING CLOUD PERFORMANCE USING BIO INSPIRED ALGORITHM IN MICRO SERVICE ERA

M.Indumathi[1], U.Sundhar[2], P.Chitra Devi[3] , D.Kanthasamy[4]

[1]P.G student, Department of CSE, Thiruvalluvar College of Engineering and Technology, Vandavasi.

[2]Head of the Department, Department of CSE, Thiruvalluvar College of Engineering and Technology, Vandavasi.

[3]Assistant Professor, Department of CSE, Thiruvalluvar College of Engineering and Technology Vandavasi

[4]Head of the Department, Department of CSE(AI&ML),Thiruvalluvar College of Engineering and Technology Vandavasi.

**ABSTRACT -** *Cloud computing has transformed the way organizations deploy and manage their IT infrastructure, providing unmatched scalability, flexibility, and cost-effectiveness. As more businesses transition to the cloud, it has become essential to optimize cloud architectures for improved performance. This paper presents a comparative analysis of different cloud architectures, emphasizing performance optimization techniques across various platforms, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). The study investigates architectural patterns such as microservices, serverless computing, and containerization, assessing their influence on performance metrics like latency, throughput, and resource utilization. Key optimization strategies, including load balancing, caching, and auto-scaling, are thoroughly discussed, showcasing their effectiveness in diverse cloud environments. Furthermore, the paper delves into emerging trends and technologies, such as edge computing and AI-driven optimizations, which hold the potential for further improvements in cloud performance. Through empirical analysis and case studies, the research pinpoints best practices and offers recommendations for organizations aiming to enhance their cloud architectures.*

**Key words: Cloud computing, AWS, GCP, AI-driven optimizations**

## 1. INTRODUCTION

The rise of microservice architecture has led to considerable progress in software development, enhancing the scalability and availability of applications. Cloud computing leverages microservice architecture to reduce the risks associated with single points of failure and to ensure adherence to service-level agreements. Nevertheless, implementing microservice architecture introduces two main challenges: 1) the management of network traffic, which can result in latency and congestion; and 2) suboptimal resource allocation for microservices. Current methods fall short in effectively addressing these issues.

To tackle these shortcomings, we suggest an innovative scheduling strategy that utilizes a modified particle swarm optimization algorithm to allocate microservice replicas to the most appropriate physical machines. Additionally, we employ a straightforward round-robin algorithm to distribute the load evenly across the physical machines within the cluster.

Moreover, our scheduling strategy is designed to integrate with Kubernetes, addressing challenges related to resource allocation and deployment. Another vital element of microservices is scalability. Techniques such as scaling and auto-scaling are commonly employed in data centers to improve service availability.

The proposed scheduler is intended to identify physical machines with optimal fitness for replicating microservices. 2) To enhance load balancing, we introduce a modified round-robin (RR) algorithm aimed at significantly improving cloud performance while minimizing latency and resource contention. 3) We propose a plug-in tool that integrates seamlessly with Kubernetes, facilitating the efficient initiation of microservice replications

and the intelligent routing of network traffic to the nearest microservices and their dependent components. We utilize call graph representation to analyze the microservices architecture, which enriches our research by detailing microservices characteristics, including topology, various types of microservices, different communication paradigms, and the quality of microservice deployment.

*Need for Study*

Today's networks must adapt to a broad and swiftly evolving array of business needs, ensuring that employees can access data, applications, and platforms seamlessly from any location. This encompasses the capability to support rapid innovation cycles and the organizational agility fostered by the growing adoption of cloud technologies. It also includes application-to-application connectivity across various cloud services and back to enterprise and partner data centers. Furthermore, applications are becoming increasingly distributed, complex, and dynamic, utilizing modern microservices, containerization, Kubernetes orchestration, and serverless architectures. Additionally, there is a significant rise in traffic from machine-to-machine communications, which may involve substantial volumes from edge devices and IoT sensors.

Objective of the Paper

To assess the performance of various cloud architectures. To pinpoint essential optimization strategies for cloud architectures. To evaluate the effectiveness of different optimization techniques. Cloud servers are scalable as they leverage virtual resources that can be swiftly modified. This capability allows them to increase or decrease their capacity according to user demand, without requiring physical changes. Users gain from the ability to effectively handle large volumes of work or data during peak times. Cloud servers provide robust security measures to safeguard data. Cloud platforms implement advanced security protocols, encryption, and regular updates to ensure information safety. This helps shield customers from data breaches and unauthorized access, guaranteeing that their sensitive information stays protected. Cloud servers are economical due to their operation on a shared resource model. You only pay for the resources you utilize, which removes the necessity to invest in and maintain costly hardware. This leads to substantial savings for businesses, especially those with variable resource needs. Cloud servers are built for high reliability and availability, featuring redundant infrastructure and automatic failover systems. The strong data centers and disaster recovery plans of cloud providers enable this reliability. For users, this advantage ensures continuous access to applications and data, reducing downtime. Users can connect to cloud servers from any location with internet access. Cloud servers function online, allowing employees to work from various sites and collaborate more effectively. This boosts productivity and flexibility for both businesses and individuals.

## 2.LITERATURE REVIEW

In this paper, Y. Gan and C. Delimitrou discuss "The architectural implications of cloud microservices" in IEEE Comput. Archit. Lett., vol. 17, no. 2, pp. 155-158, Jul. 2018. They explore how cloud microservices affect system bottlenecks and datacenter server design. Initially, they present and characterize an end-to-end application developed using numerous popular open-source microservices that provides a modular and extensible movie renting and streaming service. Subsequently, they utilize this end-to-end service to investigate the scalability and performance bottlenecks associated with microservices, emphasizing their impact on datacenter hardware design. In particular, they revisit the

ongoing debate regarding brawny versus wimpy cores in the context of microservices, quantify the I-cache pressure introduced, and assess the time allocated to computation versus communication between microservices via RPCs.

In this paper, D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan present "A holistic evaluation of Docker containers for interfering microservices" at the Proc. IEEE Int. Conf. Services Comput. (SCC), pp. 33-40, Jul. 2018. They provide an experimental study on the performance evaluation of Docker containers running a heterogeneous set of microservices concurrently. A comprehensive series of experiments were conducted following the Cloud Evaluation Experiment Methodology (CEEM) to assess the interference between containers running either competing or independent microservices. Additionally, they examined the effects of resource constraints on a container by explicitly defining the cgroups. The performance of containers was evaluated in terms of inter-container interference (caused by two concurrently executing containers) and intra-container interference (caused by two microservices running within a single container), which has been largely overlooked in existing literature.

The paper authored by J.-E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, titled "Investigating machine learning algorithms for modeling SSD I/O performance for container-based virtualization", published in IEEE Trans. Cloud Comput., vol. 9, no. 3, pp. 1103-1116, Jul. 2021, presents several contributions. We examined the prediction accuracy, learning curve, feature importance, and training time of the algorithms tested across four distinct SSD models. In addition to detailing the modeling aspect of our framework, this paper seeks to offer insights for cloud providers to develop SLO-compliant container placement algorithms on SSDs. Our machine learning-based framework effectively modeled I/O interference, achieving a median Normalized Root-Mean-Square Error (NRMSE) of 2.5 percent.

In the work by A. Abuabdo and Z. A. Al-Sharif, titled "Virtualization vs. containerization: Towards a multithreaded performance evaluation approach", presented at the Proc. IEEE/ACS 16th Int. Conf. Comput. Syst. Appl. (AICCSA), pp. 1-6, Nov. 2019, the focus is on assessing the performance of these virtual environments and their effects on multithreaded applications. Specifically, we analyze the performance of multithreaded applications and evaluate how it is affected by different virtualization technologies. A multithreaded application was created in both C and Java, with performance metrics collected across various platforms, each supported by either a virtualized system or a containerized environment.

In this paper, A. Saboor, A. K. Mahmood, A. H. Omar, M. F. Hassan, S. N. M. Shah, and A. Ahmadian discuss the topic "Enabling rank-based distribution of microservices among containers for a green cloud computing environment" published in Peer-to-Peer Netw. Appl., vol. 15, no. 1, pp. 77-91, Jan. 2022. This work highlights the flexibility in service delivery while also pointing out the drawbacks of increased energy consumption and reduced service efficiency, leading to higher carbon emissions. To address these challenges, existing technologies were primarily designed for single unit monolithic cloud applications and are not optimized for chain-oriented service delivery. This study focuses on the dynamic provisioning of containers and their associated microservices within a cloud computing environment by creating rank-based profiles that facilitate the allocation of web application microservices alongside containers to cloud data centers. The proposed MicroRanker service is designed to rank all participating microservices and distribute them across various nodes

prior to the execution of cloud services. Additionally, the MicroRanker service is employed to dynamically adjust container placement in response to ongoing DevOps activities.

The contributions of the paper by C. Santana, L. Andrade, F. C. Delicato, and C. Prazeres, titled "Increasing the availability of IoT applications with reactive microservices", published in Service Oriented Comput. Appl., vol. 15, no. 2, pp. 109-126, Jun. 2021, aim to introduce an architecture based on reactive microservices for developing IoT applications. This proposed architecture comprises a collection of software components specifically designed to fulfill the requirements of IoT applications. Furthermore, the proposal includes a software platform that implements several components of the architecture and assists in meeting the availability QoS requirements during runtime. Our proposal was applied in a practical scenario within the Smart Agriculture sector.

In this paper, S. Pallewatta, V. Kostakos, and R. Buyya discuss the "Placement of microservices-based IoT applications in fog computing: A taxonomy and future directions" published in ACM Comput. Surv., vol. 55, no. 14, pp. 1-43, Dec. 2023. The Fog computing paradigm employs distributed, heterogeneous, and resource-constrained devices located at the network's edge to efficiently deploy latency-sensitive and bandwidth-intensive IoT application services. Furthermore, MicroService Architecture (MSA) is increasingly being adopted to address the rapid development and deployment demands of swiftly evolving IoT applications. The fine-grained modularity of microservices, along with their independently deployable and scalable characteristics, positions MSA as a promising approach to leverage Fog and Cloud resources, leading to innovative paradigms such as Osmotic computing. The loosely coupled nature of microservices, supported by container orchestrators and service mesh technologies, facilitates the dynamic composition of distributed and scalable microservices to meet the varied performance requirements of IoT applications utilizing distributed Fog resources. Consequently, the efficient placement of microservices is crucial, necessitating scalable placement algorithms to capitalize on the advantages of MSA while addressing the new challenges posed by this architecture.

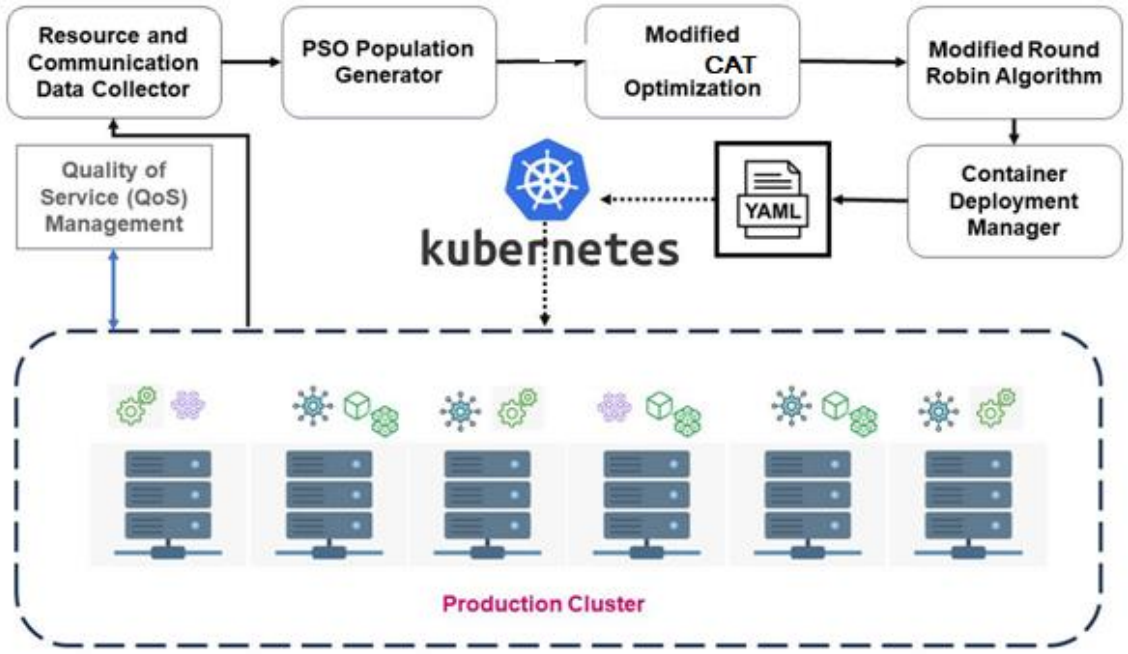
In the paper by G. Yu, P. Chen, and Z. Zheng titled "Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach," published in IEEE Trans. Cloud Comput., vol. 10, no. 2, pp. 1100-1116, Apr. 2022, a novel system called Microscaler is introduced. This system automatically identifies services that require scaling and adjusts them to comply with the Service Level Agreement (SLA) at an optimal cost for microservice applications. Microscaler begins by gathering quality of service (QoS) metrics within the service mesh-enabled microservice infrastructure. Then, it determines under-provisioning or over-provisioning service instances along the service dependency graph with a novel scaling-needed service criterion named service power . The service dependency graph could be obtained by correlating each request flow in the service mesh. By combining an online learning approach and a step-by-step heuristic approach, Microscaler can precisely reach the optimal service scale meeting the SLA requirements.

The paper by M. Xu, L. Yang, Y. Wang, C. Gao, L. Wen, G. Xu, et al., titled "Practice of Alibaba cloud on elastic resource provisioning for large-scale microservices cluster", published in Softw. Pract. Exper., vol. 54, no. 1, pp. 39-57, Jan. 2024, discusses the provision of fine-grained and precise resource allocation to each component while ensuring the overall quality of service throughout the chain. Alibaba Cloud has fully adopted cloud-native and microservice technologies to enhance its core business operations and scenarios,

such as the Double 11 Shopping Festival. This study aims to tackle the research challenge of efficiently provisioning resources for the expanding scale of microservice platforms while maintaining the performance of latency-sensitive microservices.

In the paper authored by B. Cai, B. Wang, M. Yang, and Q. Guo, titled "AutoMan: Resource-efficient provisioning with tail latency guarantees for microservices", published in Future Gener. Comput. Syst., vol. 143, pp. 61-75, Jun. 2023, we introduce a novel resource manager named AutoMan. This manager effectively allocates resources for each microservice while ensuring compliance with the end-to-end tail latency service level objective (SLO). AutoMan accomplishes this by integrating three essential components: (1) utilizing a MADDPG-based approach to allocate the near-optimal quantity of resources that support the desired end-to-end tail latency SLO.

## 3.IMPLEMENTATION




**. Fig 3.1: System Architecture**

As depicted in the figure, our scheduling strategy comprises six key components. Below are the highlights of these components:
*Resource and Communication Data Collector*: This component is tasked with gathering data related to resource utilization and the requirements of microservice resources. Additionally, it collects information on microservice communications. The collected data is utilized by other components to determine the placement of microservice replicas.

*PSO Population Generator*: This component is responsible for generating two distinct populations. The first population represents a swarm of particles, which corresponds to the resource requirements for each microservice. The second population reflects the resource availability of potential physical machines (PMs).

*Modified Cat*: Our objective is to introduce a scheduler that employs population-based optimization, specifically utilizing the Cat Optimization (Cat) algorithm. This algorithm is based on the tracking and seeking behavior of cats as they explore the solution space to identify the optimal solution.

*Modified Round Robin Algorithm*: This algorithm is employed to schedule replicas across the potential PMs while considering their load balancing. After the modified Cat identifies the optimal PMs, the Round Robin (RR) algorithm is then used to allocate the microservice replicas to those PMs. RR is implemented to ensure load balancing within our strategy.

*Container Deployment Manager*: This component oversees the deployment of replicas to the selected PMs identified by the modified PSO and RR algorithms. It receives outputs from these algorithms, including the number of replicas, the resource requirements for each replica, and the available resources of the target PMs.

*Quality Of Service (Qos) Management*: This component is responsible for ensuring that all QoS requirements are satisfied throughout the deployment process. It involves minimizing the risk of excessive resource utilization, managing the resource thresholds for each PM, monitoring for single system failures, and reporting any breaches of the service level agreement (SLA).

*Data Collection*

There are two widely recognized real-world datasets utilized for evaluating scheduling strategies: Google and Alibaba. The Alibaba data trace outlines a production cluster and is publicly accessible through the Alibaba data center. This dataset encompasses critical information such as node IDs, timestamps, and detailed resource utilization metrics. We determined that this dataset is well-suited for our research as it illustrates the infrastructure of a microservices-oriented data center. Furthermore, it offers extensive insights into job allocation, machine usage, and interdependencies.

The data trace captures the operations of a production cluster over a month. It includes resource utilization data from over 90,000 containers operating on more than 1,300 physical machines (PMs). The dataset provides node IDs, timestamps, and resource utilization specifics for each container. Additionally, it features microservice IDs, types, and communication patterns. We gathered data from over 12,125 microservices that were active for one hour across nearly 100 PMs. The second dataset is a Google trace compiled by the company's system known as Borg. The details of this data are outlined in [37]. It comprises a monthly trace that records all activities, including job scheduling, resource utilization, and job durations. Given the dataset's extensive size and the limitations of our machine, records for approximately 9,000 jobs were replicated to create a simulated workload of 24,000 jobs distributed across 100 physical machines (PMs). The jobs that entered the data center at varying times render it appropriate for simulating scenarios involving new job arrivals at different intervals.

*CAT Population Generator*

COA operates in two main modes:

- Seeking Mode: This mode illustrates a cat's natural curiosity. It prioritizes exploration, allowing the algorithm to search the solution space extensively to prevent getting stuck in local optima.

- ❖ Tracing Mode: This mode simulates a cat pursuing a target. It concentrates on exploitation, enhancing and fine-tuning the current best solutions.

*Steps:*

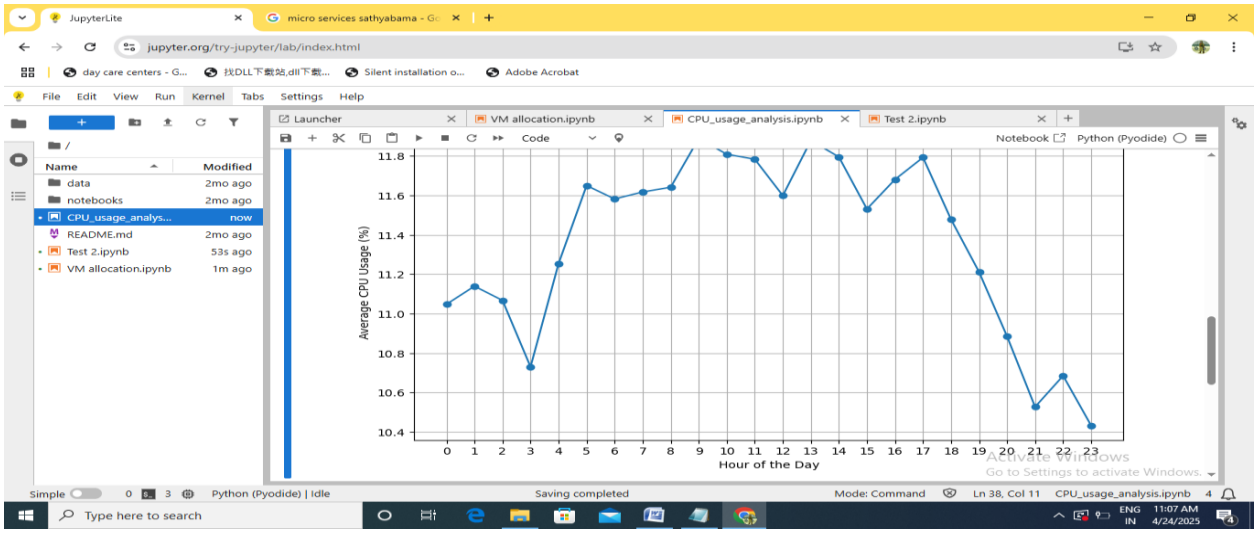
- ➢ Initialization: Represent the initial population (cats) with random configurations of task-resource allocations.
- ➢ Evaluate Fitness: Assess the performance of each configuration based on objectives such as cost and energy consumption.
- ➢ Apply Seeking Mode: Broaden the search by creating variations of the solutions.
- ➢ Apply Tracing Mode: Progress towards promising configurations to refine them.
- ➢ Update Best Solution: Keep the best configurations after each iteration.
- ➢ Repeat: Continue iterating until a stopping criterion is satisfied (e.g., no further improvement).

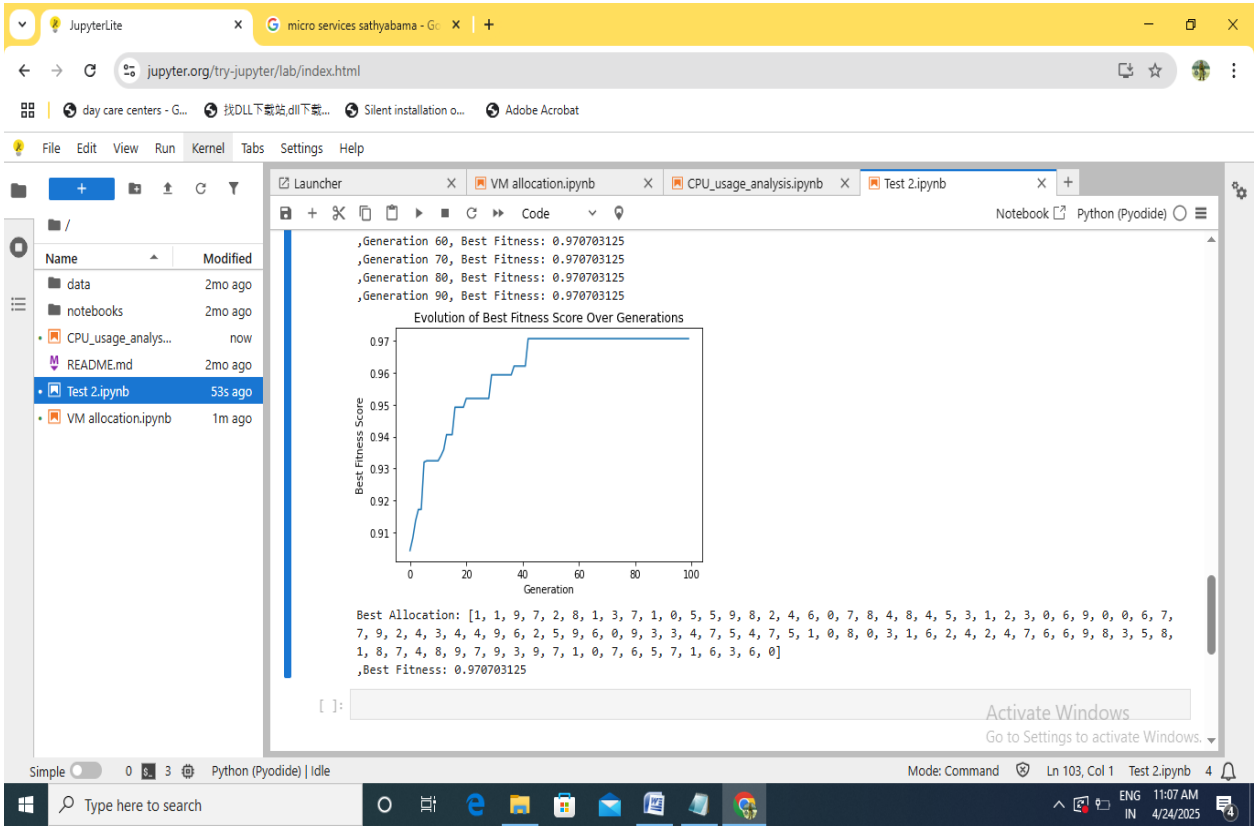
*Container Deployment Manager*

Our scheduling strategy utilizes Kubernetes as a container deployment manager. This aspect of our strategy entails defining a configuration file in 'YAML' format that outlines the image name, version, resource requirements, and number of replicas. It also encompasses additional details related to the deployment of microservices into containers. Kubernetes oversees the deployment and scaling of the microservices. Furthermore, it offers microservice discovery and health-checking mechanisms. Our method includes configuring replica scaling, which aims to enhance management efficiency. It also ensures reliability and resilience in case of failure.

*Load Balancing*

It plays a crucial role in the performance of microservices. We are implementing the RR algorithm to distribute the load among PMs. Our objective is to enhance resource utilization across all PMs within the data center. The scheduling strategy we propose improves the system's performance resiliency by adhering to three primary strategies. Firstly, we guarantee system availability by minimizing the risk of a single point of failure. Secondly, we oversee resource utilization by employing a predefined threshold; thus, resource usage remains within limits, avoiding breaches of service level agreements (SLAs). Lastly, we ensure an equitable load distribution for all PMs.






## CONCLUSION AND FUTURE ENHANCEMENT

In summary, the scheduling strategy we propose offers an innovative and thorough method for tackling the allocation and deployment of resources for microservices in complex and ever-changing environments. This approach enhances the overall performance of the microservice system while reducing network traffic costs, leading to lower latency and greater system efficiency.

The incorporation of a modified CAT facilitates the exploration and exploitation of the search space, which helps to decrease network traffic among microservices, while the RR algorithm contributes to better load balancing. Our proposed strategy enhances resource allocation, system performance, scalability, and availability. Kubernetes ensures strong health-checking capabilities for containers and microservices. Furthermore, by overseeing the system through QoS management, our algorithm enhances QoS.

## REFERENCES

[1] Y. Gan and C. Delimitrou, ‘‘The architectural implications of cloud microservices,’’ IEEE Comput. Archit. Lett., vol. 17, no. 2, pp. 155–158, Jul. 2018.

[2] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan, ‘‘A holistic evaluation of Docker containers for interfering microservices,’’ in Proc. IEEE Int. Conf. Services Comput. (SCC), Jul. 2018, pp. 33–40.

[3] J.-E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, ‘‘Investigating machine learning algorithms for modeling SSD I/O performance for container-based virtualization,’’ IEEE Trans. Cloud Comput., vol. 9, no. 3, pp. 1103–1116, Jul. 2021.

[4] A. Abuabdo and Z. A. Al-Sharif, ‘‘Virtualization vs. containerization: Towards a multithreaded performance evaluation approach,’’ in Proc. IEEE/ACS 16th Int. Conf. Comput. Syst. Appl. (AICCSA), Nov. 2019, pp. 1–6.

[5] A. Saboor, A. K. Mahmood, A. H. Omar, M. F. Hassan, S. N. M. Shah, and A. Ahmadian, ‘‘Enabling rank-based distribution of microservices among containers for green cloud computing environment,’’ Peer-to-Peer Netw. Appl., vol. 15, no. 1, pp. 77–91, Jan. 2022.

[6] H. Zhang, S. Li, Z. Jia, C. Zhong, and C. Zhang, ‘‘Microservice architecture in reality: An industrial inquiry,’’ in Proc. IEEE Int. Conf. Softw. Archit. (ICSA), Mar. 2019, pp. 51–60.

[7] C. Santana, L. Andrade, F. C. Delicato, and C. Prazeres, ‘‘Increasing the availability of IoT applications with reactive microservices,’’ Service Oriented Comput. Appl., vol. 15, no. 2, pp. 109–126, Jun. 2021.

[8] S. Pallewatta, V. Kostakos, and R. Buyya, ‘‘Placement of microservicesbased IoT applications in fog computing: A taxonomy and future directions,’’ ACM Comput. Surv., vol. 55, no. 14, pp. 1–43, Dec. 2023.

[9] G. Yu, P. Chen, and Z. Zheng, ‘‘Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach,’’ IEEE Trans. Cloud Comput., vol. 10, no. 2, pp. 1100–1116, Apr. 2022.

[10] A. Kwan, J. Wong, H.-A. Jacobsen, and V. Muthusamy, ‘‘HyScale: Hybrid and network scaling of dockerized microservices in cloud data centres,’’ in Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS), Jul. 2019, pp. 80–90.

[11] M. Xu, L. Yang, Y. Wang, C. Gao, L. Wen, G. Xu, L. Zhang, K. Ye, and C. Xu, ‘‘Practice of Alibaba cloud on elastic resource provisioning for large-scale microservices cluster,’’ Softw., Pract. Exper., vol. 54, no. 1, pp. 39–57, Jan. 2024.

[12] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, and C. Xu, ‘‘Erms: Efficient resource management for shared microservices with SLA guarantees,’’ in Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst., vol. 1, Dec. 2022, pp. 62–77.

[13] B. Cai, B. Wang, M. Yang, and Q. Guo, ‘‘AutoMan: Resource-efficient provisioning with tail latency guarantees for microservices,’’ Future Gener. Comput. Syst., vol. 143, pp. 61–75, Jun. 2023.

[14] X. Wan, X. Guan, T. Wang, G. Bai, and B.-Y. Choi, ‘‘Application deployment using microservice and Docker containers: Framework and optimization,’’ J. Netw. Comput. Appl., vol. 119, pp. 97–109, Oct. 2018.

[15] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, ‘‘Characterizing microservice dependency and performance: Alibaba trace analysis,’’ in Proc. ACM Symp. Cloud Comput., Nov. 2021, pp. 412–426.

[16] J. Shah and D. Dubaria, ‘‘Building modern clouds: Using Docker, Kubernetes & Google cloud platform,’’ in Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC), Jan. 2019, pp. 184–189.

[17] N. Singh, Y. Hamid, S. Juneja, G. Srivastava, G. Dhiman, T. R. Gadekallu, and M. A. Shah, ‘‘Load balancing and service discovery using Docker swarm for microservice based big data applications,’’ J. Cloud Comput., vol. 12, no. 1, pp. 1–9, Jan. 2023.

[18] P. Banerjee, S. Roy, A. Sinha, M. M. Hassan, S. Burje, A. Agrawal, A. K. Bairagi, S. Alshathri, and W. El-Shafai, ‘‘MTD-DHJS: Makespan-optimized task scheduling algorithm for cloud computing with dynamic computational time prediction,’’ IEEE Access, vol. 11, pp. 105578–105618, 2023.